Université de Rennes

istic Informatique Électronique

HBO

Pwning a Linux Web server

Gwendal Patat
Univ Rennes, CNRS, IRISA
2025/2026

On today’s schedule

Walkthrough of a simple pentest lab.

Introduction to a more complete pentest overview than jeopardy like CTF.

Recall of Pentest Methodology

Reconnaissance & Info Gathering

Vulnerability Identification

Exploitation

Post-Exploitation & Privilege Escalation

Pivoting to the next target

Reconnaissance & Info Gathering

Concepts

Passive reconnaissance: collect data without touching the target (OSINT, DNS records, public registries, Shodan). Low/no detection risk.

Active reconnaissance: interact with the target (ping, port scan). Higher information yield but detectable and potentially intrusive.

Target profiling: OS, open ports, running services, versions, network topology, exposed protocols, publicly disclosed vulnerabilities.

Port Scanning

Main tool: nmap

Can be used to list:

- Open ports
- Offered services (with version)
- OS version
- etc.

Information gathering on HTTP

HTTP Header can always give you some more information:

```
$ curl -I target.com
```

Fast reconnaissance:

- HTTP status code (200, 301, 401, 404): quick success/redirect/auth check.
- Server / X-Powered-By: possible service/framework fingerprint.
- Location header: follow redirects / canonical host.
- Security headers (HSTS, CSP, X-Frame-Options) and Set-Cookie attributes.

For web server, navigating the interface can be useful to find directories. Can be semi-automated!

Directory Listing

Purpose: Automated brute-force of web paths and filenames to discover hidden directories, admin pages, backups, sensitive files, etc. The brute-force will check for:

- **Status codes**: indicate existence, redirects, auth or errors.
- **Response size & timing**: unexpected sizes often reveal content.
- **Extensions found:** useful to target specific technologies (.php, .aspx, .jar).
- **Redirect targets**: can reveal canonical hosts or staging URLs.

Drawbacks: Pretty noisy with the number of request. Can easily be detected.

Example of tool:

```
$ dirsearch -u https://target.com/
```

Testing inputs and URLs

Purpose: Manipulate query/body parameters to provoke informative server responses (errors, stack traces, different status codes) that reveal backend behaviour (e.g. an SQL error exposed in an argument).

How:

- **Simple malformed values**: ', ", <>, ../../
- **Path manipulation**: trailing slash, directory traversal patterns, appended extensions
- **Change HTTP method and content type**

Next step: If an input reveals backend behavior, move to controlled manual analysis or specific tool to exploit the backend.

Exploitation

Exploiting the SQLi

First option: By hand.

- We could use the found URL parameters to try to understand the database based on error messages.
  - Doable but who got time for this?

Second option: Using tools.

- For instance: `sqlmap` is a tool that, given a vulnerable URL/parameters, can use page code, timing, error message, etc…, to retrieve databases, tables, and even content.

Crack of pass and Further Exploration

Well, well, well… A MD5 hash!

As you now, this could be cracked with tool based on wordlist and rules like hashcat or john.

We the found password and account, we can start the exploration again!

File Upload Vulnerability

Handling file upload is a sensitive concept in web security:

- An insecure file upload allows an attacker to upload files that the application accepts and processes in an unintended way (e.g., arbitrary code, web shells, or sensitive data exposure).

A forged file could be upload and loaded/executed through either from a **Local File Inclusion (LFI)** vulnerability or by the legitimate server usage (e.g., profile picture display).

Q: What could we upload?

A **reverse shell** of course!

UNIX Access Control

Discretionary Access Control (DAC)

With DAC, access rights are determined by the owner of the resource.

Characteristics:

- Flexible, easy to implement
- Prone to misconfigurations by users
- Can be difficult to force a common policy on complex systems

Common Use: File systems like UNIX and Windows.

Example: Alice creates a file, and decides who can read it, modify it, and execute it.

UNIX Access Control Mechanism

UNIX systems use a combination of user and group IDs (UID, GID) and file permissions.

- **User ID (UID)**: A unique identifier assigned to each user.
- **Groupe ID (GID)**: A unique identifier for a group of users.
- **Effective User ID (EUID)**: The UID used by a process when run.
  - By default, processes inherit the uid/gid of the user spawning it.

Specific UIDs:

- 0: root.
- 1-99: Predefined accounts.
- 100-999: Reserved for the system.
- ≥ 1000: Users.

UNIX File Permissions

Permissions are assigned to the **u**ser owner, the **g**roup owner, and **o**thers.

Represented using a combination of letters (r, w, x) or as octal digits.

- **Read** (r=4)
  - File: Read the content of the file
  - Directory: List the files within a directory
- **Write** (w=2)
  - File: Modify or delete content of a file
  - Directory: Add or delete files
- **Execute** (x=1)
  - File: Execute the file
  - Directory: Open files, and make it the current directory

UNIX Access Checks Flow

1. The system checks if the process’s EUID matches the file owner (User permissions apply).
2. If not, it checks if the process’s GID matches the file’s group (Group permissions apply).
3. Finally, if neither match, it checks the permissions for others.

UNIX File Information

```
$ ls -l file.txt
-rwxr-xr-- 1 bob bob 3215 Sept 01 10:21 file.txt
```

File types

- - file
- d directory
- b block device
- s socket

- l symbolic link
- p FIFO pipe
- c I/O files

File permission

- 3 blocks, 3 permission per block
- User, Group, Other

UNIX File Information

```
$ ls -l file.txt
-rwxr-xr-- 1 bob bob 3215 Sept 01 10:21 file.txt
```

- Only owners and root can change permissions
- Only root can change ownership
  - Owner can change the group to one of its own.

UNIX File Permissions

- Permissions are represented using a combination of letters (r, w, x) or as octal digits.
- Using chmod to modify basic permissions:

  chmod 754 file

  - 7 (rwx) for the owner.
  - 5 (r-x) for the group.
  - 4 (r--) for others.

rwx	r-x	r--
111	101	100
7	5	4

Special Permissions: SUID, SGID, Sticky Bit

SUID (Setuid): Executes a file with the EUID of the file's owner.

SGID (Setgid): Similar to Setuid, but applies to group permissions. It also affects directory.

Sticky Bit: Applied to directories to prevent users from deleting or renaming files they do not own.

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 3215 Sept 01 10:21 /usr/bin/passwd

$ ls -ld /tmp
drwxrwxrwt 25 root root 12288 Sept 01 10:21 /tmp
```

Security Risk of SUID with root

- When the setuid is used in executables owned by root, it allows regular users to execute those files with **root privileges**.
  - With root, the process's **EUID is 0**.
- This can pose a significant security risk if not managed properly, as it could lead to privilege escalation.

UNIX Capabilities

What if we don’t want suid process to have all root’s privileged?

Capabilities:

- Privileges broken down into distinct units.
- They provide a more secure alternative to the all-or-nothing approach of full root privileges.

Example: ping needs to create sockets, which is not allowed for users. Should we:

- Give ping (which is owned by root) a suid?
- Give ping the capability to create network sockets?

Setting and Getting Capabilities

Use the setcap and getcap commands to manage capabilities on executables.

Example: Setting a capability to allow binding to low ports:

```
$ setcap cap_net_bind_service=+ep /usr/bin/ping
```

Getting Capabilities:

```
$ getcap /usr/bin/ping
```

Capability Management

```
$ setcap cap_net_bind_service=+ep /usr/bin/ping
```

File capabilities:

- **Permitted** (p): The set of capabilities a process can use.
- **Effective** (e): The subset of permitted capabilities that are currently active.
- **Inheritable** (i): The capabilities that can be inherited by a child process (fork).

Understanding these distinctions is key to manage and audit capabilities.

Example of Capabilities

- **CAP_NET_BIND_SERVICE**: Allows binding to ports below 1024.
- **CAP_SYS_ADMIN**: Broad capability granting many administrative functions (to avoid).
- **CAP_CHOWN**: Make arbitrary changes to file UIDs and GIDs.
- **CAP_SYS_MODULE**: Allows to load/unload kernel modules.
- **CAP_SETUID**: Allows setting user IDs.
- **CAP_SETFCAP/CAP_SETPCAP**: Allows setting new capabilities to file/process.

Each capability addresses specific needs, minimizing the security risks associated with full root access.

cf. *man capabilities*

Privilege Escalation

Principle

Privilege escalation (PE) is the process by which an attacker (or a misused account) gains higher privileges than originally granted: typically moving from an unprivileged user to root.

Common vectors:

- Misconfigured file permissions (sensitive files or writable scripts).
- SUID/SGID binaries with unsafe behaviour or writable by non-privileged users.
- Weak or exposed credentials (service accounts, sudoers entries).
- Insecure services or daemons running as root that can be influenced by unprivileged users.
- Unpatched kernel or local software vulnerabilities enabling privilege escalation.

A lot can be used to achieve PE on a linux system, searching through all file can be cumbersome.

LinPEAS: Linux Privilege Escalation Awesome Script

LinPEAS: Automated local enumeration script that collects system artefacts and highlight potential privilege escalation vectors.

What it does:

- Gathers environment info (kernel, distro, users, groups, sudoers).
- Enumerates file permissions, SUID/SGID binaries, services, cron jobs, and writable configs/keys.
- Checks package versions, common misconfigurations, and known weak patterns (credentials, world-writable files, leftover keys).

```
$ curl -L https://github.com/peass-ng/PEASS-ng/releases/latest/download/linpeas.sh | sh
```

Exploiting the found SUID

Here trivial example: /bin/bahs with SUID using the root EUID.

This example is obviously artificial but such problem can be found in intermediate users, which can open access to new binary with specific capabilities etc…

What's next?

Pivoting

Pivoting

Once on a compromised host inside a remote network, we can re-apply our methodology to discover and reach additional hosts/services that were not directly accessible from your original location.

Based on the topology, you might be behind a NAT: **meaning we have a new network to scan!**

```
$ nmap -sn 172.18.0.0/24
```

Based on the found hosts, a new scan over the open service can disclose potential entry points:

- Web servers (HTTP/HTTPS).
- SSH, RDP, SMB, DNS, ...
- Database/listener services, RPC, custom daemons.
- Binary endpoints that could be exploited using software vulnerabilities
  - Will be seen in S2 with ISOS, SE.

Resources and Acknowledgements

- https://book.hacktricks.wiki/en/
- https://github.com/bayufedra/MBPTL/
- Linux man pages
- External materials from Daniel De Almeida Braga.