Linux Kernel Rootkit

Nathan Castets & Olivier Huge

Université de Bordeaux

20 Février 2019

Overview

Définitions

Rootkit

Utilitaire qui permet d'effectuer différentes actions sur une machine. Le but principal est d'installer un accès privilégié à cette machine pour un pirate de façon persistante dans le temps.

A la différence d'un malware classique, le rootkit se veut discret et dissimule au maximum ses actions à l'utilisateur et aux programmes de surveillance.

Définitions

Il existe 2 types de rootkit :

- Espace utilisateur
  Remplace des fonctions utilisées par un programme
  Injection de librarie dynamique via *LD_PRELOAD*
- Espace noyau
  Remplace des appels systèmes
  Module noyau qui écrase la table des appels systèmes

Définitions

Table des appels systèmes

Tableau contenant les adresses mémoires des fonctions associées aux appels systèmes. Ces appels systèmes permettent aux programmes de l'espace utilisateur de communiquer avec le noyau.

Les appels systèmes sont indispensables pour les programmes de l'espace utilisateur pour utiliser des fonctions que seul le noyau peut exécuter.
On appelle aussi la table des appels systèmes la *sys_call_table*.

Pré Linux Kernel 4.17

/fs/open.c :

```
/* *** */

EXPORT_SYMBOL(sys_close);

/* *** */
```

La fonction associée à l'appel système *sys_close* est accessible par n'importe quel programme présent dans le noyau.
Cet export est présent car le module *mount* a besoin de *sys_close*.

Un brute-force de la mémoire noyau à la recherche des occurences de l'adresse de *sys_close* nous donne la *sys_call_table*.

Post Linux Kernel 4.17

- Suppression de la majorité des appels systèmes dans le code noyau
  L'export de la fonction *sys_close* n'existe plus
- Rajout de fonction avec un comportement similaire *ksys_xyzxyz()*
  Le but étant de dissocier au maximum les appels venants de l'espace utilisateur et noyau

Cela implique :

- Qu'il n'est plus possible de brute-force la *sys_call_table* à l'aide de l'adresse d'un appel système
- Qu'il n'est plus possible d'altérer le comportement de programme présent dans le noyau

Déterminer l'adresse de la table des appels systèmes

L'idée est de s'intéresser au fonctionnement des appels systèmes et plus précisément au code exécuté en préambule pour préparer l'appel système.

Retracer ce code dans la mémoire noyau jusqu'à retrouver un offset vers la *sys_call_table*.

Nous avons principalement travaillé sur les version 4.17 à 4.20 du noyau Linux pour la suite des recherches.

Déterminer l'adresse de la table des appels systèmes

Dès qu'un appel système est levé, le processeur doit exécuter du code pour préparer cet appel système. L'adresse de ce code se trouve dans le registre *MSR_LSTAR*. Voyons à l'initialisation ce que contient ce registre.

/arch/x86/kernel/cpu/common.c (4.17 - 4.19) :

```
if (static_cpu_has(X86_FEATURE_PTI))
  wrmsrl(MSR_LSTAR, SYSCALL64_entry_trampoline);
else
  wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
```

/arch/x86/kernel/cpu/common.c (4.20) :

```
wrmsrl(MSR_LSTAR, (unsigned long)entry_SYSCALL_64);
```

Déterminer l'adresse de la table des appels systèmes

/arch/x86/entry/entry_64.S (4.17 - 4.20) :

```
ENTRY(entry_SYSCALL_64)
    /* *** */
    pushq %rax

    PUSH_AND_CLEAR_REGS rax=$-ENOSYS
    TRACE_IRQS_OFF

    movq  %rax, %rdi
    movq  %rsp, %rsi
    call  do_syscall_64

    TRACE_IRQS_IRETQ

    movq  RCX(%rsp), %rcx
    movq  RIP(%rsp), %r11

    cmpq  %rcx, %r11
    jne swapgs_restore_regs_and_return_to_usermode
```

Hook un appel système

Cacher des fichiers à l'utilisateur

Conclusion