Linux Kernel Rootkit

Castets Nathan, Huge Olivier

***Abstract*—Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.**

I. Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

II. Notions et résumé de l'état de l'art

Un rootkit est un 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'autres programmes malveillants, un rootkit se veut discret et dissimule au maximum ses actions à l'utilisateur et aux programmes de surveillance.

Il y a deux types de rootkit. Les rootkits qui opèrent dans l'espace utilisateur et ceux qui opèrent dans l'espace noyau. Dans la suite de ce rapport nous nous concentrerons sur le deuxième groupe de rootkits. La majorité des rootkits qui opèrent dans l'espace noyau utilisent la table des appels systèmes (sys_call_table) afin d'altérer le comportement de la machine.

La table des appels systèmes est un tableau qui contient toutes les adresses mémoires des différents appels systèmes. Ces appels systèmes permettent aux programmes de l'espace utilisateur de communiquer avec le noyau. Ils sont indispensables pour les programmes de l'espace utilisateur pour utiliser des fonctions que seul le noyau peut exécuter. Un rootkit a la possibilité de retrouver cette table pour modifier certaines adresses et remplacer les appels systèmes par ses propres fonctions. Avec ce procédé, le rootkit peut choisir quelles informations retourner aux programmes de l'espace utilisateur et altérer le fonctionnement de la machine.

Pour éviter ce type d'attaque une sécurité majeure existe, la KASLR (Kernel Address Space Layout Randomization). C'est à dire que les différentes parties du code du noyau son répartie aléatoirement dans la mémoire. Ceci à chaque démarrage du système. Elle existe depuis la version 3.14 du noyau Linux mais nécessitait d'être activée et de recompiler le noyau. Récemment, depuis la version 4.12, elle est activée par défaut.

Jusqu'à la version 4.17 du noyau Linux, la majorité des rootkits utilisaient une seule et même tehnique pour passer outre la KASLR et accéder à la table des appels système. Le noyau Linux exportait l'adresse de l'appel système *sys_close()*. C'est à dire que l'adresse de cet appel système était accessible directement par n'importe quel module du noyau Linux. Il suffisait par brute-force de trouver dans la mémoire du noyau à quels endroits étaient présente cette adresse. Ainsi on retrouvait facilement la table des appels systèmes.

III. Linux kernel 4.17

La faille qui permettait aux rootkits de retrouver la table des appels système résidait dans l'export de l'adresse de l'appel système *sys_close()*. Cela était nécessaire car le module *mount*, qui permet de gérer le système de fichier, avait besoin de cet appel système. Un correctif a été appliqué au noyau Linux dans la version 4.17 afin d'empêcher les rootkits de l'espace noyau de retrouver l'adresse de la table des appels systèmes à l'aide de l'adresse de *sys_close()*.

Tout d'abord l'export de l'adresse de l'appel système a été supprimé. On a donc perdu le point d'accroche. Pour que le module *mount* puisse continuer à fonctionner normalement il a été prévu une fonction pour remplacer l'appel système. Cette fonction remplace l'appel système *sys_close()* pour les programmes de l'espace noyau.

Plus généralement dans le noyau, plusieurs modules utilisaient les appels systèmes avant la 4.17. Les développeurs noyaux veulent éviter au maximum cela car ces appels systèmes sont avant tout destinés à l'espace utilisateur. Si jamais il était absolument nécessaire d'utiliser un appel système dans l'espace noyau, un fonction "handler" a été mise en place pour le remplacer. Elle fonctionne de façon similaire à l'appel système qu'elle remplace. Ceci évite que l'adresse d'un appel système soit présente dans le noyau.

Il est donc maintenant nécessaire de trouver une autre façon de faire que l'adresse de l'appel système *sys_close()* pour retrouver l'adresse de la table des appels système.

IV. Déterminer l'adresse de la sys_call_table

L'idée pour atteindre la sys_call_table tout en ayant la KASLR activée, est de trouver un point d'accroche avec du

code qui utilise la sys_call_table. Le premier endroit où regarder est du côté des routines qui initialisent la sys_call_table ou qui gèrent les appels systèmes.

REFERENCES

[1] Linux kernel patch 4.17
https://github.com/torvalds/linux/commit/2ca2a09d6215fd9621aa3e2db7cc9428a61f2911#diff-acc7893dfa77092a11e1b53e98a34d44
[2] System calls in the Linux kernel
https://0xax.gitbooks.io/linux-insides/content/SysCall/
[3] Linux kernel
https://github.com/torvalds/linux/blob/master/arch/x86/entry/entry_64.S
[4] Linux kernel
https://github.com/torvalds/linux/blob/master/arch/x86/entry/common.c
[5] Linux kernel
https://github.com/torvalds/linux/blob/master/arch/x86/kernel/cpu/common.c