

Université de Bordeaux
Faculté de Sciences et Techniques

rkduck : un kernel rootkit pour Linux 4.x.x

Thomas Le Bourlot, Maxime Peterlin, Martial Puygrenier



Bordeaux, le 21 Mars 2016

Sommaire

1	Introduction	2
2	Définition et évolution des rootkits	2
3	Injection - Comment le rootkit est injecté en mémoire ?	3
3.1	Injection via /dev/mem	3
3.1.1	Explication	3
3.1.2	Contre-mesures	3
3.2	Injection via un LKM	3
3.2.1	Explication	3
3.2.2	Contre-mesures	4
4	Détournement de l'exécution du noyau	4
4.1	Détournement des appels systèmes	4
4.2	Détournement du Virtual File System	4
5	Persistance du rootkit	4
6	Fonctionnalités du rootkit	4
6.1	Cacher des dossiers et fichiers	4
6.1.1	Dossiers	4
6.1.2	Fichiers	4
6.2	Backdoor	5
6.2.1	Bind shell	5
6.2.2	Reverse shell	5
6.2.3	SSH backdoor	5
6.2.4	Cacher les connexions réseaux	5
6.3	Contrôle du rootkit depuis userland	5
6.4	Keylogger	5
7	Détection du rooktit	5
8	Conclusion	6
9	Annexes	6
10	Bibliographie	6

1 Introduction

Cet article traite des rootkits. Ce sont des programmes permettant de cacher la compromission d'une machine à ses utilisateurs légitimes. Plus précisément, nous expliquerons le développement de notre rootkit nommé `rkduck`, un LKM (Linux Kernel Module) rootkit pour Linux v4.x.x.

Nos propos seront développés de la manière suivante. Dans un premier temps, nous rappellerons en quoi consiste un rootkit et comment ils ont évolué au fil du temps. Une partie sera consacrée aux différentes méthodes d'injection de code dans le noyau Linux. Suite à cela, nous expliquerons comment notre rootkit détourne le fonctionnement du système pour cacher une activité malveillante et de quelle manière il reste persistant sur la machine. Enfin, nous expliquerons les fonctionnalités mise à disposition par notre rootkit.

2 Définition et évolution des rootkits

Afin de compromettre une machine, un attaquant passe généralement par la même série d'étapes. Une fois que des vulnérabilités dans le système ont été identifiées, l'attaquant les exploite afin de prendre la main sur le système jusqu'à compromettre entièrement ce dernier afin d'avoir les droits de super-utilisateur. Cependant, l'exploitation de failles dans un système ne peut être considéré comme un moyen sûr pour accéder à une machine, car le simple fait de corriger les vulnérabilités empêcherait d'y accéder à nouveau. C'est pourquoi il est nécessaire de faire persister l'accès après la compromission, ce qui est rendu possible par l'utilisation de rootkits.

Les premiers types de rootkits utilisés étaient des ensembles de programmes qui permettaient de substituer les outils d'administration standard tels que `ls`, `ps`, etc. Ainsi, l'intrus avait le contrôle sur les sorties de ses programmes et pouvait cacher l'activité malicieuse qui se passait sur la machine. Cependant, il devenait fastidieux de cacher la compromission de la machine à l'administrateur légitime. Ce dernier pouvait utiliser plusieurs programmes effectuant la même action pour recouper les informations et ainsi découvrir des différences entre les résultats retournés révélant ainsi la présence d'un rootkit.

Les attaquants se sont alors rendus compte qu'il était inutile de réécrire toute une suite de programmes, ce qui peut devenir très conséquent en terme de taille et en temps de développement. Il suffit de remonter aux fonctions utilisées par tous ces programmes d'administration : les appels systèmes. En supposant que l'intrus ait accès au kernel, il est possible d'intercepter les appels systèmes pour les rediriger vers des fonctions malveillantes afin de cacher la compromission de la machine. A partir du moment où l'attaquant a le contrôle sur le noyau du système, il devient difficile de détecter de manière fiable un rootkit, car les systèmes de sécurité opèrent au même niveau que ce dernier et des fonctions que l'on peut penser digne de confiance peuvent retourner des résultats falsifiés.

C'est ce type de rootkit que nous allons étudier par la suite, notamment les rootkits s'injectant dans le système sous forme de modules kernel.

3 Injection - Comment le rootkit est injecté en mémoire ?

Il existe plusieurs méthodes d'injection de code dans un noyau. Nous étudierons dans cette partie les deux méthodes les plus répandues à savoir l'utilisation du périphérique `/dev/mem` et des modules noyau. A titre informatif, il est également possible d'utiliser des vulnérabilités du noyau ou bien l'entrée `FireWire` qui permet d'avoir un accès direct à toute la mémoire de la machine, ce qui nécessite cependant un accès physique à la machine.

Afin de développer `rkduck` nous nous sommes, dans un premier temps, tournés vers l'injection basée sur `/dev/mem`. Cependant, un patch de sécurité empêche son utilisation sur les versions 4.x.x du noyau Linux, ce qui fait que nous nous sommes finalement tournés vers la construction d'un LKM rootkit.

3.1 Injection via `/dev/mem`

3.1.1 Explication

`/dev/mem` est un fichier qui fournit un accès à une image de la mémoire physique de la machine. L'intérêt principale est de pouvoir par exemple patcher le système rapidement sans avoir à écrire un driver kernel. Comme on peut très vite l'imaginer, `/dev/mem` a été un point d'entrée pour injecter du code malicieux. L'attaquant va pour cela cibler la table des appels systèmes en utilisant l'IDT (Interrupt Descriptor Table). Il va ensuite changer les entrées de la table système pour qu'elle sur les fonctions du rootkits. Une autre technique consiste à copier la table des appels système, la modifier et faire pointer le gestionnaire des appels système vers cette nouvelle table et ainsi laisser la table originale inchangée.

L'avantage de cette méthode est qu'elle est très discrète par rapport à une injection LKM, en effet le rootkit ne se situe pas directement sur le disque de la machine mais il est présent dans la mémoire volatile, ce qui fait qu'une analyse forensic doit pousser son investigation jusque dans la mémoire RAM pour trouver le rootkit.

3.1.2 Contre-mesures

Depuis les versions 2.6.26 du kernel linux une option activée par défaut, `CONFIG_STRICT_DEVMEM` qui limite l'accès à `/dev/mem` au premier megabyte. Cela permet d'accéder aux périphériques PCI et certaines régions du BIOS ce qui est suffisant pour les applications qui ont besoin d'utiliser `/dev/mem` et empêche les applications l'injection de code malicieuses.

3.2 Injection via un LKM

3.2.1 Explication

Lorem

3.2.2 Contre-mesures

Lorem

4 Détournement de l'exécution du noyau

4.1 Détournement des appels systèmes

Lorem

4.2 Détournement du **Virtual File System**

Lorem

5 Persistance du rootkit

La persistance du rootkit correspond en quelque sorte à sa durée de vie une fois injecté dans le kernel. Elle est très importante, le but du rootkit étant de récupérer des informations sensibles etc... Sa persistance au sein du système est donc primordiale.

Lors d'un redémarrage les modules ajoutés manuellement ne sont pas rechargés, il faut l'indiquer manuellement au système. Nous avons donc écrit un script bash qui va écrire le nom de notre module dans un fichier spécifique '/etc/modules/' (le nom et chemin du fichier n'est pas le même suivant l'OS) et placer notre rootkit compilé dans /lib/modules/./kernel/drivers/directory/. Pour perdre un peu plus l'utilisateur de la machine qui pourrait chercher manuellement dans la liste des dossiers drivers un nom suspect qui pourrait le mettre sur la piste d'une activité malicieuse, le script s'occupe de placer le rootkit compilé au sein d'un dossier légitime exemple : /kernel/drivers/tty/rkduck.ko. Le nom du dossier légitime peut être changé par l'attaquant avant l'injection du module, il pourra prendre soin de renommer aussi le rootkit compilé en un nom qui paraît légitime du style audio.ko etc.

Lorsque le rootkit est chargé pour la première fois le script s'exécute et à chaque démarrage de la machine notre rootkit sera maintenant chargé au sein du kernel, la persistance est donc acquise et nous pouvons maintenant présenter les différentes fonctionnalités de notre rootkit.

6 Fonctionnalités du rootkit

6.1 Cacher des dossiers et fichiers

6.1.1 Dossiers

lorem

6.1.2 Fichiers

lorem

6.2 Backdoor

6.2.1 Bind shell

lorem

6.2.2 Reverse shell

lorem

6.2.3 SSH backdoor

Afin de permettre à l'attaquant d'avoir encore plus la main sur la machine corrompu et dans le cas où le défenseur désactiverait tout les ports non conventionnels, nous avons mis en place une backdoor ssh que l'attaquant est libre d'activer ou non. Elle va tout simplement mettre sa clé publique dans le fichier `/.ssh/authorized_keys` de root, il peut ainsi tout simplement se connecter en ssh avec l'utilisateur root. Évidemment cela suppose que la machine cible est accessible depuis l'extérieur et que root puisse se connecter en ssh (option qui peut être désactivé dans les fichiers de configuration ssh).

Nous considérons cette "backdoor" comme options de secours dans le cas où les autres ne fonctionneraient plus. Elle n'est pas très robuste car il suffit au défenseur de supprimer la clé public dans le fichiers *authorized_keys* mais encore faut il qu'il regarde se fichier et détermine qu'elle clé est la mauvaise. Si nous avons rajouté cette option ssh c'est parce qu'elle est utile si on sait que le défenseur a détecté nos traces (fermeture des ports, filtrage réseaux, commandes suspects netstat etc), il y a peu de chance qu'il ferme aussi le port ssh. L'attaquant peut alors se connecter directement en ssh et supprimer le rootkit avant que la victime n'est le temps de le récupérer pour une analyse forensic. Ou si la discrétion n'est pas la priorité, l'attaquant peut décider changer les ports, le rootkit de place, supprimer le filtrage etc. Le but étant comme nous l'avons évoqué plus haut de "perdre" un maximum le défenseur ou de rester le plus discret possible...

6.2.4 Cacher les connexions réseaux

lorem

6.3 Contrôle du rootkit depuis userland

lorem

6.4 Keylogger

lorem

7 Détection du rooktit

-¿ honeypot page 12

8 Conclusion

Lorem

9 Annexes

Lorem

10 Bibliographie

Lorem