Université de Bordeaux
Faculté de Sciences et Techniques
Ducky the LKM Rootkit

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 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 se programme 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 afin de les rediriger vers des fonctions malveillantes afin de cacher la compromission de la machine.

3 Injection - Comment le rootkit est injecté en mémoire ?

Nous allons expliquer dans cette partie deux méthodes utilisées pour injecter un rootkit au sein du kernel. Nous avons d'abord tenté une approche avec /dev/kmem mais comme vous pourrez le lire plus en détaille, un patch empêche maintenant sont utilisation. Nous détaillons quand même cette méthode car elle fait partie du processus qui nous à amené à construire 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 patché le système rapidement sans avoir à écrire un driver kernel. Comme on peut très vite l'imaginer, /dev/mem a été un point d'entré pour injecter du code malicieux. L'attaquant va pour cela ciblé la table des appels systèmes en utilisant l'IDT (Interupt Descriptor Table). Il va ensuite changé les entrées de la table système pour qu'elle sur

les fonctions du rootkits. Une autre technique consiste à copier la tables des appels système, la modifier et faire pointé le gestionnaire des appels système vers cette nouvelle table et ainsi laisser la table original inchangé.

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 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 options activé 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 besoins 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 rooktit 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 pendre soin de renommer aussi le rootkit compilé en un nom qui parait 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 ou 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. Évidement 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 ou les autres ne fonctionnerais 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 peux 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

-¿ honey pote page 12

8 Conclusion

Lorem

9 Annexes

Lorem

10 Bibliographie

Lorem