



Corruption de la mémoire lors de l'exploitation

Samuel Dralet, zg@miscmag.com
François Gaspard, kad@miscmag.com
(et accessoirement Kevin)

SSTIC '06 / 31 Mai, 1-2 Juin 2006



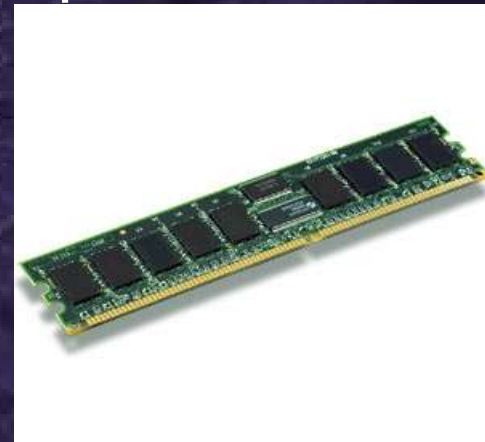
Type de mémoire

2 types :

- Mémoire de masse (disque dur), accès très lent, **information persistante**



- Mémoire vive (RAM), accès rapide, **information non persistante**



Une intrusion informatique classique

5 étapes :

1. Récolter les informations
2. Attaque et pénétration
3. Augmentation de privilèges si nécessaire
4. Installation d'une *backdoor*
5. Effacer les traces

=> peu importe le système visé, ces 5 étapes seront toujours présentes

Récolter les informations

3 types de sources possibles :

- Source humaine
=> utilisation du *social engineering*, envoi de mails, ...
- Source *World Wide Web*
=> utilisation des moteurs de recherche pour obtenir des fichiers de mot de passe, adresses mail, ...
- Source réseau
=> utilisation de logiciels pour connaître la topologie du réseau, les serveurs présents, ... (avec nmap, hping2, scapy, ...)



La source réseau générera des logs qui seront inscrits sur le disque ! C'est inévitable !

Attaque et pénétration (1)

Exploitation : tirer profit d'une faille de sécurité dans un programme vulnérable afin d'affecter sa sécurité (*buffer overflow, format string, ...*)

Utilisation d'un **exploit** :

- Vecteur d'attaque
- Technique d'exploitation
- *Payload*

Payload : suite d'instructions exécutées par le programme vulnérable une fois celui-ci contrôlé

Attaque et pénétration (2)

Types de *payload* :

- Ajout d'un utilisateur
- Ajout d'un service
- *Shellcode* pour obtenir un shell

Types de *shellcode* :

- *Bind socket shellcode*
- *Connect back shellcode*
- *Find socket shellcode*

Augmentation de privilèges

- Augmentation de privilèges pour obtenir un contrôle total de la machine attaquée
- Utilisation le plus souvent d'un exploit local contre un programme avec le bit *suid root* activé

=> Etape parfois inutile si le service exploité possède déjà les droits du super utilisateur

Installation d'une backdoor

Backdoor : dispositif permettant de revenir ultérieurement sur une machine et d'en prendre le contrôle sans reproduire toutes les étapes d'exploitation initialement nécessaires

Types de backdoor :

- Ajout d'un utilisateur
- Ajout d'un service
- Ajout d'une page web
- Modification d'un service existant (noyau, binaire, mémoire)

Modification du binaire et le relancer

=> écriture sur le disque

Modification du service en mémoire

=> aucune écriture sur disque

Effacer les traces

- Analyse des fichiers de logs pour enlever toutes traces qui permettraient de remonter jusqu'à l'attaquant

Exemple: si le serveur exploité est Apache, analyse du fichier *access.log*

- Si un fichier est écrit sur le disque, il faudra soit le cacher soit l'effacer de manière sûre (*shred*, ...)

Intrusion tout en mémoire

Kevin



- Attaque et pénétration
- Augmentation de privilèges
- Installation d'une *backdoor*

Administrateur vérifiant son disque



Internet



Exécution à distance

Rappel : un shellcode est une suite d'instructions déjà préparées pour être exécutées sur la machine distante

MAIS ! Nécessité parfois d'exécuter un binaire complet !

2 grandes techniques :

- *Syscall Proxy*
- *Remote Userland Execve*

Le binaire n'est jamais écrit sur le disque de la machine cible

Syscall Proxy (1)

```
$ cat example.c
int main(void)
{
printf("Fonction printf !
\n");
return 0;
}
$ gcc -o example
example.c
$ ./example
Fonction printf !
$
```

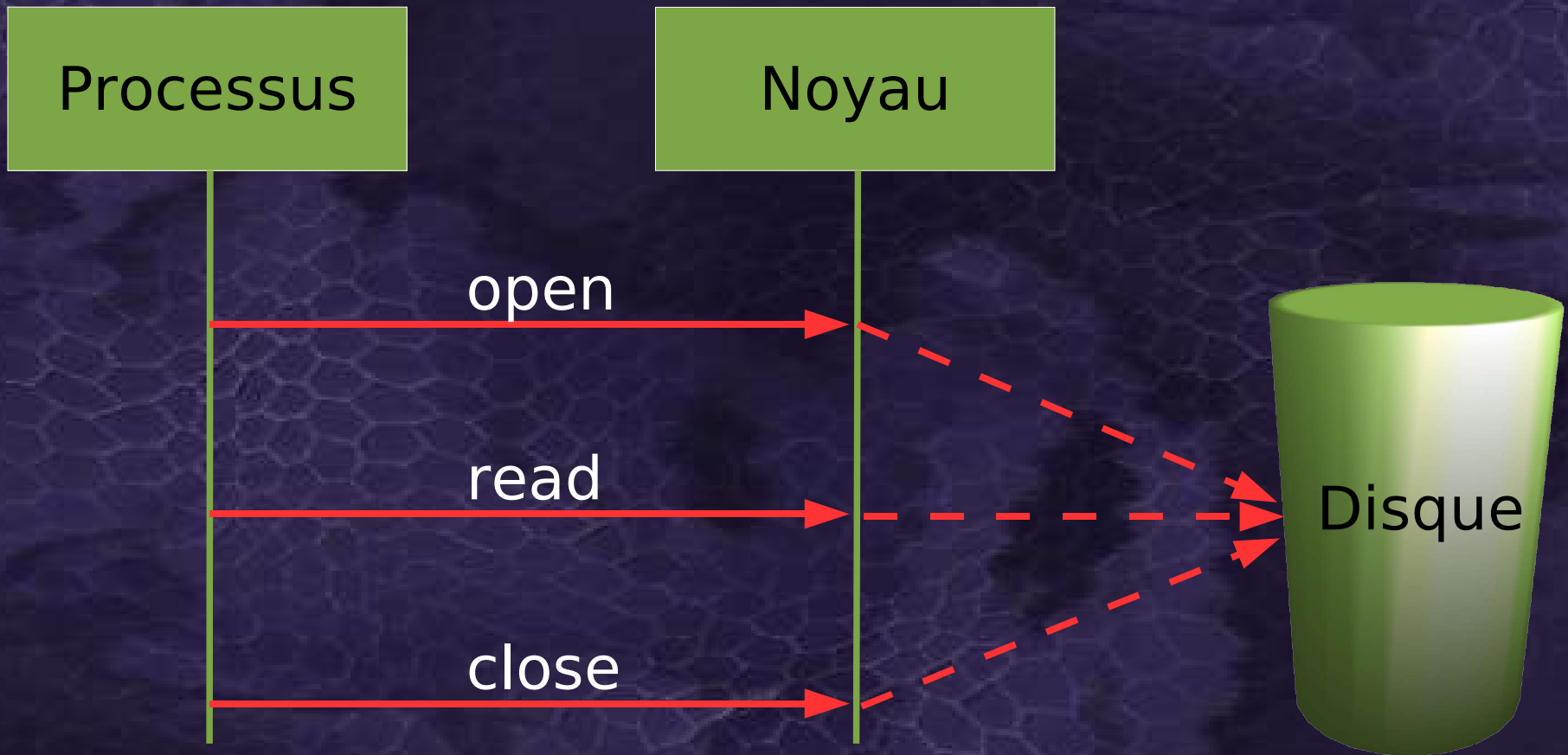
```
$ strace ./example
1: execve("./example", ["/example"], [/* 34 vars
*/]) = 0
2: uname({sys="Linux", node="Joshua", ...}) = 0
...
9: open("/lib/libc.so.6", O_RDONLY) = 3
10: read(3, "\177ELF\1\1\1...") = 512
...
21: mmap2(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_ANONYMOUS,
-1, 0)
= 0x40017000
22: write(1, "Fonction printf ! \n", 19) = 19
23: munmap(0x40017000, 4096) = 0
24: exit_group(0) = ?
$
```



- La fonction *printf()* a été transformée en un appel système *write()*
- Chaque binaire exécuté fait ainsi appel à plusieurs appels système

Syscall Proxy (2)

Lecture d'un fichier par un processus



Idée : exécuter les appels système sur la machine distante

Syscall Proxy (3)

Kevin

Innocente machine



Processus

Syscall Proxy client

Syscall Proxy serveur

Noyau

open

read

close

send_open

send_read

send_close

open

read

close


Disque

Internet

Syscall Proxy (4)

3 étapes :

1. Numéro d'appel système et arguments à fournir
2. Exécution de l'appel système
3. Valeur de retour à récupérer



Seuls les appels système sont exécutés sur la machine distante, traitement arithmétique, boucles, conditions, allocation mémoire, etc, sont réalisés sur la machine locale.

- Le serveur est représenté la plupart du temps par un *shellcode* lancé après exploitation
- Le client est représenté la plupart du temps sous forme de librairie. Possibilité avec LD_PRELOAD

Remote Userland Execve (1)

Execve : appel système permettant d'exécuter un programme

- Lors d'un appel à `execve()` dans un programme, **l'image du processus est remplacé** par l'image du binaire à exécuter (segment de code, de données, la pile et le tas)



`execve()` ne retourne pas à l'ancien processus !

- Pour retourner à l'ancien processus, il faut utiliser un `fork()`, comme la fonction `system()` de la GNU C Library (`glibc`).

Remote Userland Execve (2)

Problème ! Le binaire à exécuter doit être présent sur le disque

Userland execve : simuler le comportement de l'appel système `execve()` pour exécuter un binaire déjà présent en mémoire

6 étapes :

1. *Unmapper* les pages contenant l'ancien processus
2. Si le binaire est dynamique, charger l'éditeur de liens dynamiques
3. Charger le binaire en mémoire
4. Initialiser la pile
5. Déterminer le point d'entrée
6. Transférer l'exécution au point d'entrée

Remote Userland Execve (3)

Remote Userland Execve : exécuter un binaire sur la machine cible sans rien écrire sur le disque

1. Le binaire est envoyé via une socket
2. Le binaire est réceptionné dans l'espace d'adressage du processus
3. Le binaire est exécuté

Mode client-serveur, avec serveur représenté soit sous forme de librairie (`ul_exec`) ou de *shellcode* (SELF)

Récapitulatif

- Présentation de deux techniques pour exécuter un binaire à distance sans jamais l'écrire sur le disque : *syscall proxy* et *remote userland execve*
- Techniques pouvant être utilisées lors de l'étape 2 « attaque et pénétration ».
- Techniques pouvant également être utilisées lors de l'étape 3 « élévation de privilèges » (article MISC 26)
- Ces deux techniques peuvent être utilisées à partir d'un exploit dans la partie *payload*, le *payload* envoyé étant soit le serveur du *syscall proxy*, soit le serveur du *remote userland execve*
- Reste l'étape 4, l'installation d'une *backdoor* ...

Injection de bibliothèques (1)

Rappel : Modification d'un service existant pour que rien ne soit écrit sur le disque !

- Ancienne méthode, l'injection de code assembleur dans la mémoire du processus à l'aide de *ptrace()* (voir article MISC 25)

Plus intéressant d'injecter du code C directement !

Injection de bibliothèques en local :

1. Injection d'un petit code assembleur exécutant *dlopen()* à l'aide de l'appel système *ptrace()*
2. Rediriger une fonction à l'aide d'une *GOT redirection*, *PLT redirection* ou méthodes avancées *EXTSTATIC*, *CFLOW* (MISC 26)

Injection de bibliothèques (2)

Injection de bibliothèques à distance :

Comme pour *Remote Userland Execve*, on envoie la bibliothèque via une socket et on la stocke dans l'espace d'adressage du processus

Problème ! *dlopen()* nécessite que la bibliothèque soit sur le disque

2 solutions :

- Utiliser une partition de type *tmpfs* ou *ramdisk* (partition en RAM)
- Recoder *dlopen()* pour qu'il charge une bibliothèque déjà présente en mémoire et non sur le disque (MISC 26)

Constatations (1)

Constatations générales :

- Le serveur exploité sur la machine cible est condamné à mourir
- Le serveur ne peut pas être relancé juste avant de mourir
- Le serveur ne peut être relancé par manque de privilèges

Constatations (2)

Constatations à propos du *userland execve* :

- Le binaire ne doit pas être recompilé d'une manière (trop) spécifique
- Utiliser des binaires compilés en dynamique n'est pas une bonne idée
- Un *userland execve* doit *forker* pour ne pas tuer le processus utilisé
- Les pages ne doivent pas être swappées sur le disque
- Le binaire à exécuter sur la machine cible doit être le plus petit possible

Constatations (3)

Constatations à propos du *syscall proxy* :

- Il y a beaucoup trop de communications réseaux
- Il est nécessaire de réimplémenter tous les appels système
- L'appel système *fork()* ne peut pas être utilisé

Constataions (4)

Syscall Proxy ou Remote Userland Execve ?

Syscall Proxy :

Essai	Début	Fin	Différence en seconde
1	1144575001.544480	1144575001.844930	0.300452
2	1144575222.945630	1144575223.264960	0.319324
3	1144575389.394810	1144575389.702290	0.307482
Moyenne			0.309086

Remote Userland Execve :

Essai	Début	Fin	Différence en seconde
1	1144576301.802520	1144576301.921960	0.119442
2	1144576566.364180	1144576566.489810	0.125633
3	1144576756.478590	1144576756.639000	0.160407
Moyenne			0.135160

Constataions (5)

Constataions à propos de l'injection de librairies à distance :

- La librairie ne pourra pas être injectée dans le processus exploité
- L'injection de librairies pour des binaires compilés en statique ne fonctionnera pas

Plato (1)

En partant de ces constatations ...

Plato : logiciel permettant de réaliser une intrusion informatique en ne restant rien qu'en mémoire

Contraintes :

- Le processus exploité ne *fork()* pas et ne se relance pas automatiquement
- Le logiciel doit être capable de fonctionner avec n'importe quel exploit

Plato (2)



Mémoire du système cible

Process 1

Process 70

Process 30

Process 110

Etape 2 : ptrace injecting shellcode

Process 710

Etape 3 : connect-back + read shellcode

Etape 4 : relancer Apache

Apache

Etape 1 : multi-stage shellcode stage 1 -> findsocket + read

Plato

Etape 5 : remote library injection



Super Kevin

1

2

5

3

4

Protections

Protections classiques : *firewall*, audit des applications, protections contre les failles de type *buffer overflow*

Protections spécifiques :

- Contrôler (ou interdire) l'utilisation de l'appel système *ptrace()* (module noyau, grsecurity, ...)
- Audit des processus en cours d'exécution



Une analyse forensic de la mémoire vive est aussi importante qu'une analyse du disque !!!!!!!!!!!!!



Conclusions

- La mémoire vive est aussi importante, si pas plus, qu'une mémoire de masse
- *Remote Userland Execve* > *Syscall Proxy*
- Beaucoup de choses ont été présentées, mais beaucoup d'autres n'ont pas pu l'être par manque de place (*remote DHIS*, détails sur plato, *remote dlopen*, élévation de privilèges, ...) MISC 26 ! :-)
- Platon > Aristote (cf article SSTIC) :-)

Merci pour votre attention !

Des questions ?



Encore une
victoire de Kevin !