SSTIC 2004

Mesure locale des temps d'exécution : application au contrôle d'intégrité et au fingerprinting

Gaël Delalleau
Zencom Secure Solutions

gael.delalleau+sstic@m4x.org

Contrôle d'intégrité par mesure du temps d'exécution

- **Problématique et travaux existants (EPA)**
- Contrôle d'intégrité d**epuis l'espace utilisateur (userland):**
  - Noyau
  - Programmes en mode utilisateur
- Autres applications :
  - détection d'une machine virtuelle
  - détection d'un environnement de débogage

Les « timing attacks » en sécurité informatique

- Qu'est-ce que le « timing » ?
- Détection du chemin d'exécution
  - authentification réseau ~~id~~entifiants utilisateurs
  - Griffiths : timing noyau **open( )** ~~ex~~istence d'un fichier
- Effets de cache et vie privée
- Analyse du trafic réseau (ex: ssh 1)
- **Cryptanalyse (ex: RSA, OpenSSL)**

Définitions : contrôle d'intégrité, prise d'empreinte

- Contrôle d'intégrité : de quoi ? Pourquoi ?Du matériel aux programmes lancés par les utilisateurs...
- Prise d'empreinte
  - Ce qui nous intéresse : prise d'empreinte **locale de l'environnement d'exécution**

(aucun risque, j'utilise Tripwire !)

- Limites du contrôle d'intégrité disque dur et système de fichiers (tripwire, md5sum...) :
  - backdoors matérielles, BIOS et « ring -1 » (non publiques)
  - Attaques « in memory only » (outils publics) : *shellcode* + *userland exec* ou *syscall proxying* + injection processus ou noyau + détournement d'API
- Survie au reboot : pas toujours nécessaire (longs uptimes, réinfection possible, *mass-rooting*...) ou possible via des moyens détournés.

Execution Path Analysis

- EPA : Joanna Rutkowska (PatchFinder pour Linux, PatchFinder 2 pour Windows)
- Principe : traçage pas à pas («debug») pour mesurer le nombre d'instructions exécutées lors d'un appel système
  →détection des *rootkits* et *backdoors* dans le noyau
- Très bons résultats, mais...
  - Nécessite les droits « ultimes » sur le système : modification du noyau
  - Détectable : modification de l'environnement d'exécution (flag de traçage TF, installation d'un gestionnaire d'exception INT 1), modification du noyau (ajout d'un appel système)...
  - Contournable : Barbosa (attaque l'IDT)

Thématique

- Problématique et travaux existants (EPA)
- **Contrôle d'intégrité depuis l'espace utilisateur (userland):**
  - **Noyau**
  - Programmes en mode utilisateur
- Autres applications :
  - détection d'une machine virtuelle
  - détection d'un environnement de débogage

Mesure du temps d'exécution appliquée à la détection d'une modification non autorisée de code

- Principe : mesure du temps d'exécution d'un morceau de code situé en mémoire, appelé avec certains paramètres.

- Hypothèse (simpliste): si le temps d'exécution a changé, pour les mêmes paramètres d'appel, c'est que le code exécuté a été modifié ! ALERTE →

Notre cahier des charges

- Objectif : savoir mesurer et comparer de manière fiable les temps d'exécution de certaines parties de code en mémoire
- Contraintes :
  - Mesure du temps depuis l'espace utilisateur non privilégié (sur Linux x86 : *ring 3*, *[e]uid > 0*, éventuellement dans une prison *chroot*)
  - Le code mesuré doit pouvoir être situé partout où l'on a un droit d'exécution : dans l'espace utilisateur (processus courant, autres processus), et dans l'espace noyau (*ring 0*).

Utilisations possibles :
les attaques du noyau, mais aussi...

- Détection des *rootkits* et *backdoors* installées dans le noyau (ex: adore-ng) : temps d'exécution des appels systèmes
- Détection des *rootkits* et *backdoors en mémoire en mode utilisateur* : temps d'exécution des fonctions des bibliothèques dynamiques (API) et des procédures critiques des processus
- Détection d'un environnement surveillé : machine virtuelle, débogage du programme

Outils de mesure d'un temps d'exécution

- Alarme et boucle : `alarm()`, `setitimer()`...
- Appel système : `time[s]()`, `gettimeofday()`, `clock()`, ...
- Registre ou instruction dédiée du microprocesseur :
  - Intel x86 : `rdtsc` « **R**ea**D** **T**ime**S**tamp **C**ounter »
  - Sparc v9 : registre `%tick`
- Avantages et inconvénients choix de l'instruction machine

Normalisation, référence et comparaison

- Pour pouvoir comparer des mesures similaires prises sur différents systèmes
- Normalisation (division) du temps mesuré par rapport à un temps de référence type « boucle vide » (temps mis pour exécuter *N* instructions en espace utilisateur) :

```
__asm__ ("rdtsc\n movl %%eax, %0" : "=a" (t1));
for (j=0; j<10; j++)   // N = 10
          ;      // boucle dix fois sans autre action
__asm__ ("rdtsc\n movl %%eax, %0" : "=a" (t2));
ref = t2 - t1;
```

Problèmes !

- Comparaison entre systèmes tout de même difficile : influence noyau, type de microprocesseur, mémoires caches, etc...
- Plus grave : le temps d'exécution d'un même morceau de code, sur un même système, n'est pas toujours le même !
- Cause : optimisations processeur, interruptions matérielles (IRQ), mais surtout effets de la mémoire cache.
  - Peu d'influence de la charge du système (changements de contexte)

Thématique

- Problématique et travaux existants (EPA)
- Contrôle d'intégrité d**epuis l'espace utilisateur (userland):**
  - **Noyau**
  - Programmes en mode utilisateur
- Autres applications :
  - détection d'une machine virtuelle
  - détection d'un environnement de débogage

Rappels sur les attaques du noyau par *rootkits*

- Insertion de code en mode noyau (ring 0 sur Intel) : par modules ou écriture mémoire
- Détournement de fonctions (appels systèmes, VFS...) : camouflage d'activité, accès caché, espionnage par interception et/ou détournement des communications
- Survie au redémarrage : fichier de configuration, infection module légitime, infection image du noyau sur le disque...
- Windows, Linux, Solaris, *BSD

Détection de rootkits inconnus

- Outils de détection existants : ne détectent que les techniques connues de détournement de fonctions du noyau ; parfois contournables trivialement
- Détection par timing : susceptible de détecter toute modification du chemin d'exécution d'un appel système, même si le détournement est réalisé par un *rootkit* inconnu à l'aide d'une technique nouvelle
- Inconvénient : plus difficile à mettre en oeuvre...

Les appels systèmes intéressants

- `readdir()` dans le système de fichiers racine
- `readdir()` dans le système de fichiers /proc
- `read()` sur certains fichiers, par exemple `/proc/net/tcp`
- `open()` sur certains fichiers, par exemple `/dev/(k)mem`
- `setuid()`, `kill()`, `recv()/send()`...

Mesures : système sain

- Temps d'exécution de `readdir(«/bin»)` pour un grand nombre de mesures :

```
2068 ************************************************************
2076 ****************************
2077 ************
2088 ************************
2097 ***
```

- Une autre série de mesures donne :

```
2032 ********************************
2037 ************************************************************
2052 ****
```

- Ou encore :

```
2025 ************************************************************
2038 ****************************************
2045 ************
2058 *******
```

Mesures : système infecté

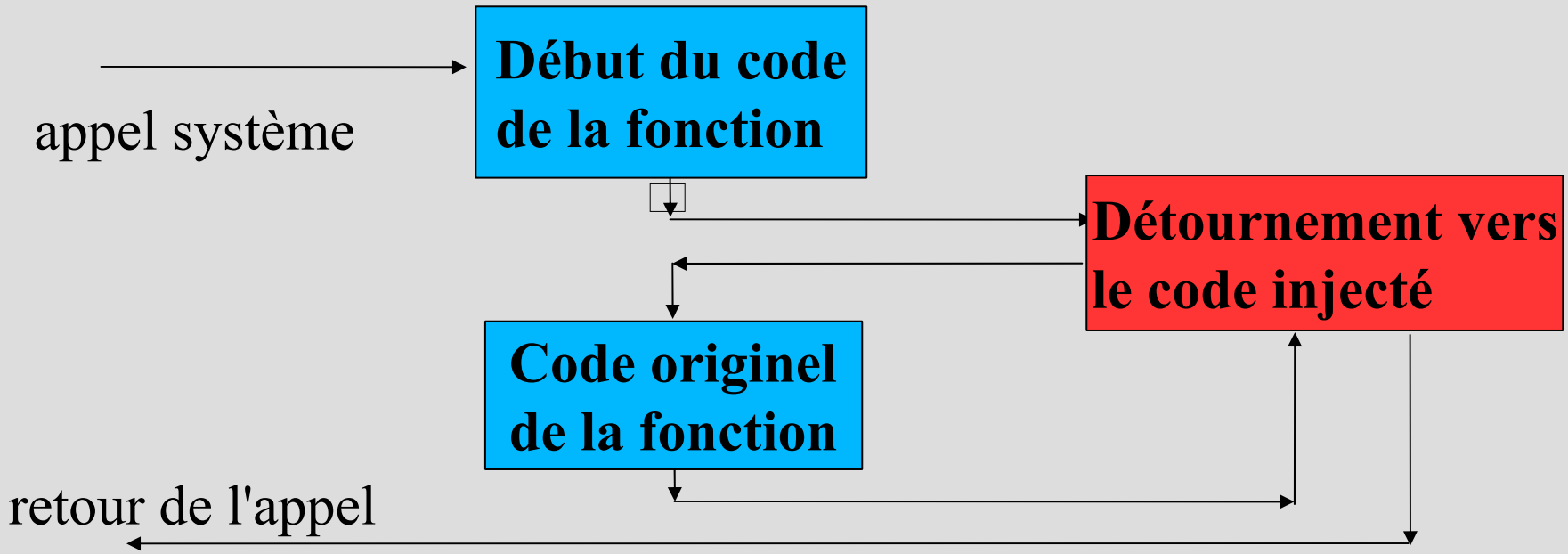
- Après insertion d'un *rootkit* noyau :

```
2418 ********************
2419 ************
2421 ************************************************************
2438 ****
2439 **
2441 ******************************
```

- Sur ces mesures :
  - temps **maximal** ayant un poids significatif pour le système sain : 2097
  - Temps **minimal** ayant un poids significatif du système infecté : 2418

Cause : détournement du chemin d'exécution d'une fonction

- Le détournement de l'appel système, ou d'une fonction appelée au cours de son exécution, ***ajoute*** un certain nombre d'instructions :



Critères de comparaison, seuil d'alerte

- Temps minimal, maximal, moyen (sauf aberrations)
- Quantification de la « ressemblance » entre deux séries de mesures : produit scalaire des transformées de Fourier (basses fréquences)
- Phase d'apprentissage : enregistrement des séries de mesure ne se « ressemblant » pas assez (seuil), prises ensuite comme références / bases de comparaison.

Résultats : cas réels

- Voir actes de la conférence page 229 (adore-ng) et 230 (sebek)
- Adore-ng readdir() sur / :
  - Système sain : min=1987 ress=0.96
  - Système infecté: min=2250 ress=0.02

Démonstration

- test en direct !

Thématique

- Problématique et travaux existants (EPA)
- Contrôle d'intégrité d**epuis l'espace utilisateur (userland):**
  - **Noyau**
  - **Programmes en mode utilisateur**
- Autres applications :
  - détection d'une machine virtuelle
  - détection d'un environnement de débogage

Rappels sur les attaques ELF et PE

- Similitude *rootkits* noyau : injection de code + détournement de fonction
- Injection : statique (ELF, PE, bibliothèque dynamique) ou dynamique (ptrace, API hooking)
- Détournement : PLT, GOT, IAT...
- Utilisation : accès caché, espionnage, plus rarement camouflage et furtivité

Principe de la détection

- Outils existants : *elfcmp* = inefficace
- Mesure du temps d'exécution d'une fonction au sein de l'espace mémoire d'un autre processus :
  - Injection du code de mesure (ptrace, API hooking...)
  - Mesure : 1000 appels successifs de la fonction, résultats écrits dans un fichier : valeur minimale, valeur moyenne
  - Reprise du processus

Résultats expérimentaux

- Serveur OpenSSH tournant en mémoire: injection d'un code détournant la fonction `read()` pour enregistrer les données lues dans un fichier (keylogger).
- Injection du code de mesure, appelant la fonction `read()` via la PLT :

temps minimum	temps moyen	
800	822	avant
1332	1339	après

Thématique

- Problématique et travaux existants (EPA)
- Contrôle d'intégrité d**epuis l'espace utilisateur (userland):**
  - **Noyau**
  - **Programmes en mode utilisateur**
- **Autres applications :**
  - détection d'une machine virtuelle
  - détection d'un environnement de débogage

Détection d'une machine virtuelle

- Utilisation des machines virtuelles : honeypots, étude de code potentiellement hostile
- Émulation x86 : bochs
- UML : gros délai des appels systèmes
- Vmware : timing des instructions privilégiées déclenchant une exception
  (cf slide suivant)

Détection de Vmware par mesure du temps

SYSTEME LINUX NORMAL :
Reference time... 21 mms [1]
Timing various instructions : 24 mms = 114 % [1]
Timing void syscall : 50 mms = 238 % [2]
Timing illegal instructions with signal handlers : 776 mms = 3695 % **[36]**
Timing int 3 with signal handler : 386 mms = 1838 % **[18]**

SYSTEME LINUX SOUS VMWARE :
Reference time... 21 mms [1]
Timing various instructions : 24 mms = 114 % [1]
Timing void syscall : 106 mms = 504 % [5]
Timing illegal instructions with signal handlers : 4978 mms = 23704 % **[237]**
Timing int 3 with signal handler : 2530 mms = 12047 % **[120]**
> special x86 instructions take a long time [VMWARE]

Détection d'un outil d'analyse du flot d'exécution

- strace, ktrace, truss
- ltrace
- Donc : danger !

Conclusion

- Méthode de prise d'empreinte « floue »
- Difficultés de mise en oeuvre
- Futur : détection de petites modifications de code ? Prouver la faisabilité sous Windows
- Eric Detoisien au Social Event :
  « c'est très bien, mais ça ne sert à rien ! »
- Mais champ d'application beaucoup plus large !

Bon appétit !

- Remerciements : Nora, D.M.P.
- Questions ?