Signierte ausführbare Dateien

Vortrag zum Seminar „System- und Netzwerksicherheit“

Johannes Schlumberger
asso@0xbadc0ffee.de
Alexander Würstlein
arw@arw.name

Friedrich-Alexander-Universität Erlangen-Nürnberg

23. Februar 2007

Inhalt

sbits

- erhöhen die Berechtigungen eines Prozesses
- u.U. führen nicht vertrauenswürdige Benutzer Programme mit root-Rechten aus
- übliches Verfahren fuer sogenannte root-Kits
- Programm mit sbit hinterlegen

Idee

- nur signiertes Programm erhält beim `exec` euid des inodes
- einfache Implementierung mit Prüfsumme statt „richtiger“ Signatur
- Ablegen der Signatur im inode in „extended attributes“

erste lauffähige Version

Userspace

- Signierprogramm im Userspace legt MD5-Summe in Attribut „trusted.sns“
- Namespace „trusted“ legt implizit Berechtigungen fuer Attributszugriffe fest
- nur root darf schreiben, niemand darf lesen

Kernelspace

- Prüfroutine errechnet Prüfsumme über geladene Binärdaten im `exec`
- Strukturelement „sns_valid_sig“ im Prozessdeskriptor
- wird zur zusätzlichen Bedingung für das Setzen der euid

Dynamisch gebundene Binärdateien

- Bibliotheksfunktionen enthalten unsignierten Code
- wird auch beim vorgenannten Ansatz ausgeführt
- Lösung durch Signieren und Prüfen der Bibliotheken
- Strukturelement „sns_valid_sig“ im „vm_area_struct“
- Setzen und Prüfen in `mmap` und `mprotect`

Problem: Verlust des Vertrauens

- durch nachgeladene Bibliothek
- Codegenerierung zur Laufzeit
- unnötigerweise ausführbare, schreibbare Speicherbereiche

Lösungsmöglichkeiten

- vorher erhaltene capabilities, Dateideskriptoren, etc. entziehen
- Prozesse, die derartiges Verhalten zeigen nicht signieren
- Verhinderung o.g. Verhaltens
- Prozess töten

HMAC

- einfacher und billiger Ersatz für asymmetrische Kryptographie
- $\mathrm{HMAC}_K(m) = h((K \oplus \mathrm{opad}) \| h((K \oplus \mathrm{ipad}) \| m))$
- ist bereits im Kernel implementiert
- deutlich weniger rechenaufwendig als RSA

Systemaufruf - Implementierung

- Benutzerprozess fragt Kernel nach Vertrauensstatus eines anderen Prozesses

```
asmlinkage int sys_sns_is_trusted(pid_t p)
{
        struct task_struct *t;
        rcu_read_lock();
        t = find_task_by_pid(p);
        if (IS_ERR(t))
                return -EINVAL;
        p = t->sns_valid_sig;
        rcu_read_unlock();
        return p;
}
EXPORT_SYMBOL_GPL(sys_sns_is_trusted);
```

PITA

- libopenssl kann weitaus weniger Hashalgorithmen als der Kernel
- sehr inkonsistente API (Groß/Kleinschreibung aus `/dev/random`)
- dateisystembezogene syscalls in Linux haben nicht den inode, sondern `struct dentry` als Parameter
- aus dem inode rückwärts aufzulösen ist schwer und unerwünscht
- wir haben aber ausschliesslich den inode zur Verfügung
- *Lösung:* Anpassung der API
- viele MMUs setzen `PROT_READ` nur gleichzeitig mit `PROT_EXEC`
- linux-gate.so

contra

- keine Just-in-time-compiler
- keine VMs
- Skriptsprachen sehr problematisch

pro

- auf jeden Fall geeignet fuer suid-Kontrolle
- sehr leichtgewichtig
- geringe Eingriffe im Kernel (ca. 100 Zeilen diff)
- erweiterbar auf asymmetrische Kryptographie
- neue Hashverfahren sind automatisch verfügbar
- ähnliche Prüfungen auch an anderen Stellen des Kernels leicht möglich (capabilities, selinux)


B. W. Kernighan, D. M. Ritchie
The C Programming Language, Second Edition.
Prentice Hall PTR, Upper Saddle River 1988, 39^{th} printing


N.N.
Linux Kernel Documentation.
git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux-2.6.git linux-2.6/Documentation/, Version 9654640d0af8f2de40ff3807d3695109d3463f54