LINUX KERNEL SECURITY

WITH GRSECURITY AND PAX

GRSECURITY AND PAX

GrSecurity is a set of patches for the linux kernel with an emphasis on enhancing security. GrSecurity uses a learning-based approach to determine rulesets.

PAX is a separately developed component that comes bundled with grsecurity. It allows you to flag memory as non-executable and non-writeable.

PAX also provides address space layout randomization (ASLR) to hinder attacks that rely on specific memory locations. It's available independently from GrSecurity.

FEATURES

GrSecurity has a lot of features which are organized into categories based on their intrusiveness.

LOW

- Linking restrictions
  - Following symlinks in +t directories (e.g. temp) disabled
  - No hardlinks to files you don't own
- Enforcing RLMIT_NPROC on execve()
- Restrict dmesg to root
- Disable module loading utilities
- Enforce chdir("/") on chroot

WHY ENFORCE CHDIR("/")?

- chrooting does not change the active directory!

```
mkdir foo
chroot foo
cd ..
```

MEDIUM

- Log failed forks, time changes and signals
- No mounts inside chroot
- No sysctl and mknod in chroot
  - mknod can create a device entry pointing to a harddrive or console, allowing an attacker to write to it
- No access to AF_UNIX sockets outside chroot
- No writing to kmem, mem and port (even as root)
- Restrict /proc
  - Non-root users can only access their own processes
  - Keep normal users from viewing any device information and slabinfo
- ASLR (I'll come back to this later)

ABUSING MKNOD

```
# Find out mknod params
/ ~ cd /dev
/dev ~ ls hda1 -l
brw-r------    1 root root 3,  0 May 17 21:09 hda

# Access hda in chroot
mknod hdacpy b 3 0

# Access console in chroot
mknod console c 5 1 # linux
mknod console c 0 0 # os x
```

HIGH

- No signals, priority, fchdir, ptrache or access to processes outside of a chroot
- Remove all addresses from smaps, maps and stat
- Randomize every task's kernel stack (may break stuff)
- Log mounts
- Restrict sysfs / debugsfs
  - Hint: `chmod 0700 /mpath`
- Active exploit responses
  - Those are triggered on consequent `illegal operation` errors
  - Root? Panic the system!
  - User? Lock the account and save all relevant information

ADRESS SPACE PROTECTION

- Deny writing to kmem, mem and port
  - This will close the four most common ways of inserting malicious code into a running kernel.
  - The only remaining way would be to load a module or use priviliged I/O by ioperm/iopl
  - Hint: Use RBAC or disable privileged I/O
- Deter exploit bruteforcing
  - When a child of a forking daemon is killed or crashes due to an illegal instruction, the parent process is delayed 30 seconds upon every subsequent fork until the daemon is restarted

ADRESS SPACE PROTECTION

- Harden module autoloading
- Hide kernel symbols
  - Makes no sense in precompiled kernels
  - Makes no sense without dmesg protection
  - Restricts syscalls that display symbols
- Awesome feature: Randomize mmap
  - This will make non-specific requests appear at random addresses

PAX

- PAX marks memory as non-executable, making buffer overflows impossible
- This can be enforced system-wide (recommended) or individually per application
- Emulation of executable pages in sandbox-like environments is available for individual processes (e.g xorg)
- Restricts mprotect, makes it smart
- Can also enforce non-executable kernel pages, mitigates code executing kernel exploits (but the same goes for driver BLOBS)

AUDITING

- Execv, Ressources, Signals, Chdir's, fork's
- Append remote-ip to the kernel task struct and log it

RBAC

Role Based Access Control is an approach to selectively limit users by restricting access to certain resources (such as sockets) on a case by case basis

RBAC rules apply to root, so gaining root-access does not necessarily compromise a system (Remember the kernel panic?)

RBAC - LIMITATION

- Limited responsiveness to new software
- Fully compromised system = fully compromised rbac

MISC

- Sanitize all free memory
- Prevent invalid userland pointer dereferences

INSTALLATION AND USAGE

1. Download the patch

```
/root ~ wget grsecurity-2.9.1-3.2.21-201206201812.patch
/root ~ wget grsecurity-2.9.1-3.2.21-201206201812.patch.sig
```

2. Verify the patch

```
/root ~ gpg --import spender-gpg-key.asc
/root ~ gpg --verify grsecurity-2.9.1-3.2.21-201206201812.patch.sig
```

3. Do the same for the userspace utilities

INSTALLATION AND USAGE

1. Patch the kernel (`patch -p1 < gr.patch`)
2. Configure the kernel
3. Boot into the new kernel
4. Install the userspace utilities:

```
make
make install
```

INSTALLATION AND USAGE

- gradm - grsecurity / rbac
- paxctl - PAX flags
- pspax - display pax flags per process
- "sysctl" - runtime configuration

INSTALLATION AND USAGE

```
// enable full system learning
gradm -F -L /etc/grsec/learning.logs

// run administrative tasks while in learning mode
gradm -a admin // ! important

// after a while, disable rbac and create a policy
gradm -D
gradm -F -L /etc/grsec/learning.logs -O /etc/grsec/policy

// enable rbac and test your system
gradm -E
```

NEW PROCESSES?

```
// edit /etc/grsec/learn_config
[command] [path]

inherit-learn - inherit rules from called binaries / libraries
no-learn - use your own rule definition
high-reduce-path - try to restrict as much as possible
dont-reduce-path - do the opposite (pre configured)
protected-path - give own learning subject to path
high-protected-path - same and hide path

// train
gradm -L /etc/grsec/learning.logs -E

// do stuff, then apply rules
gradm -D
gradm -L /etc/grsec/learning.logs -O /etc/grsec/policy
```

RBAC RULES

```
role admin sA # role mode
subject / rvka # subject mode
        / rwcdmlxi

role default G # role mode
role_transitions admin # role attribute
subject / #subject mode
        /               r #object mode
        /opt            rx
        /home           rwxcd
        /mnt            rw
        /dev
        /dev/grsec      h
```

RBAC MODES

```
# role = group / user
u / g / s - user / group / special
l - enables selective learning
N - does not require auth

# subject = process
b - enable process accounting
d - protect all memory access (even from root)
h - hidden
K - kill upon alert
o - protect
k - can kill protected processes

# object = file
r/w/a/c/x - read / write / append / create / exec
i - inherit rules from path
l - allow hardlinks
h - hide
```

CVS ROLE

```
role cvs u
   subject /
           /     h
           -CAP_ALL
           connect   disabled
           bind      disabled

   subject /usr/bin/cvs
           /
           /etc/fstab          r
           /etc/mtab           r
           /etc/passwd         r
           /proc/meminfo       r
           /dev/urandom        r
           /dev/log            rw
           /dev/null           rw
           /home/cvs           r
           /home/cvs/CVSROOT/val-tags  rw
           /home/cvs/CVSROOT/history   ra
           /tmp                rwcd
           /var/lock/cvs       rwcd
```

PAX

```
paxctl -c /usr/bin/vi
 file /usr/bin/vi had a PT_GNU_STACK program header, converted

paxctl -v /usr/bin/vi
 PaX control v0.5
 Copyright 2004,2005,2006,2007 PaX Team [pageexec@freemail.hu]

 - PaX flags: -------x-e-- [/usr/bin/vi]
 RANDEXEC is disabled
 EMUTRAMP is disabled
```

THE END

QUESTIONS?