Mandatory Access Control

Sarah Newman
sarah.newman@computer.org

https://prgmr.com/~srn/mac_apparmor.pdf

Pop Quiz!

network traffic : firewall :: process : _____

Pop Quiz!

network traffic : firewall :: process : _____

a) system call

Pop Quiz!

network traffic : firewall :: process : _____

a) system call

b) virtual memory

Pop Quiz!

network traffic : firewall :: process : _____

a) system call

b) virtual memory

c) file permissions

Pop Quiz!

network traffic : firewall :: process : _____

a) system call

b) virtual memory

c) file permissions

d) mandatory access control

Pop Quiz!

network traffic : firewall :: process : _____

~~a) system call~~

~~b) virtual memory~~

~~c) file permissions~~

d) mandatory access control (MAC)

Mandatory Access Control

- “Controls access by subjects to objects”
  - Subjects: active entity such as task or process
  - Object: files, other processes, network, etc.
- Set by policy administrator
- Ordinary user can't make changes

Why use MAC?

- “Defense in Depth” - one layer of many
- Might be easier to specify policy than change program, maintain multi-user permissions, etc.

What can be confined

- File system access
- Network access
- Inter Process Communication
- Use of capabilities (partial superuser privileges)
- Execution
- mount/umount
- Probably some other things

MAC Implementations in Linux

- SELinux – applied to files (inodes)
- AppArmor – applied to file paths
- SMACK – applied to files, mostly used in embedded systems
- grsecurity – not mainlined, stable patches are no longer free
- TOMOYO – from NTT, current maintenance unclear

MAC Selection Considerations

- Whether security policy can be defined at all
- Commercial support – desirable or not?
- Integration with existing infrastructure
- Ease of use

Strengths of Each Implementation

- SELinux – very fine grained control, not going away
- AppArmor – feasible to write policies by hand, usable on FS w/out extended permissions, not going away
- SMACK – policies extremely easy to define
- grsecurity – handles lots of security issues beyond just MAC, stacks with other MAC implementations

Distribution MAC Support

Runs by default:

- SELinux: Redhat, CentOS, Fedora, Android
- AppArmor: Ubuntu, SUSE

Some support:

- SELinux: Debian (CLI only,) Gentoo, Arch, yocto
- AppArmor: Debian
- grsecurity: Debian, Gentoo, Arch
- SMACK: Ostro OS (yocto derived)

Summary

- Mandatory Access Control (MAC) can add another layer of protection around a process
- Different MAC implementations have different strengths and weaknesses
- Make sure the implementation matches your needs

AppArmor

Sarah Newman
sarah.newman@computer.org

https://prgmr.com/~srn/mac_apparmor.pdf

Why AppArmor

- Enabled by default in one of the most common distributions, Ubuntu
- Relatively straightforward syntax
- No separate build process for new policies

AppArmor Components

- Kernel
- Userspace Utilities
- Profiles

Kernel Support

- Check distribution support first
- Most versions 2.6.36+ should be OK, worst case download from https://www.kernel.org/
- Userspace release has kernel patches in kernel-patches/<major version> - add these or some functionality may be missing
- To protect hardlinks, add commit 800179c9b8a1 for versions < 3.6

Kernel Config

- With make menuconfig, under “Security options”:
  - “AppArmor support” - enable
  - Default security module → AppArmor
  - Either set CONFIG_SECURITY_APPARMOR_BOOTPARAM_VALUE=1 and CONFIG_SECURITY_SELINUX_BOOTPARAM_VALUE=0 or add “apparmor=1 selinux=0” to kernel command line

Building Userspace

- Check distribution first for packages (apparmor-utils)
- Start from http://wiki.apparmor.net/index.php/Main_Page#Userspace
- Try to follow README in main directory, except run 'autoreconf –force –install' before configure
- Dependencies: autotool, libtool, perl-Pod-Checker, perl-gettext, perl-Test-Simple, swig, gcc-c++, libstdc++-static, texlive-*
- You may need to do your own init scripts

Before testing...

- Set sysctl fs.protected_hardlinks=1 at boot to avoid profile bypass
- Verify securityFS is mounted, otherwise add

```
securityfs /sys/kernel/security securityfs rw,nosuid,nodev,noexec,relatime 0 0
```

Is it working?

```
# aa-status
apparmor module is loaded.
0 profiles are loaded.
0 profiles are in enforce mode.
0 profiles are in complain mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile
defined.

# cat /sys/kernel/security/apparmor/profiles
```

How Profiles Work

- Default deny - empty profile means that process can effectively do nothing at all
- Different permissions (network, capabilities, file access, etc) can be explicitly permitted or denied
- Profiles must be explicitly loaded into the kernel using a userspace tool to become active

Profile Modes

- audit - use aa-audit or flags=(audit) in profile
- enforced - use aa-enforce or no flags
- complain - use aa-complain, apparmor_parser -C or flags=(complain) in profile
- disabled - aa-disable or apparmor_parser --remove

Getting Profiles

- Check distribution (ie apparmor-profiles package)
- Where to find:
  - ubuntu-centric @ https://git.launchpad.net/apparmor-profiles/tree/ubuntu
  - github
- Typically live in /etc/apparmor.d

Let's Confine a Program....

```
#aa-enforce /etc/apparmor.d/bin.ping
Setting /etc/apparmor.d/bin.ping to enforce
mode.
# aa-status
apparmor module is loaded.
1 profiles are loaded.
1 profiles are in enforce mode.
   /{usr/,}bin/ping
0 profiles are in complain mode.
0 processes have profiles defined.
0 processes are in enforce mode.
0 processes are in complain mode.
0 processes are unconfined but have a profile
defined.
```

Let's Confine a Program....

```
#  ( ping 8.8.8.8 &> /dev/null ) &
# ps -efZ | grep -E '(LABEL|ping)'
LABEL                             UID  PID  PPID
/{usr/,}bin/ping (enforce)  root 2229 1394
C STIME TTY      TIME     CMD
0 16:19 pts/0    00:00:00 ping 8.8.8.8
```

And Unconfine It

```
# aa-disable /etc/apparmor.d/bin.ping
Disabling /etc/apparmor.d/bin.ping.
# ( ping 8.8.8.8 &> /dev/null ) &
# ps -efZ | grep -E '(LABEL|ping)'
LABEL      UID        PID  PPID  C STIME TTY
unconfined root      2164  1394  0 15:54 pts/0
TIME    CMD
00:00:00 ping 8.8.8.8
```

Can I confine a running process?

Not anymore. See
http://wiki.apparmor.net/index.php/ReleaseNotes_2_4

```
# ( ping 8.8.8.8 &> /dev/null ) &
# aa-enforce /etc/apparmor.d/bin.ping
Setting /etc/apparmor.d/bin.ping to enforce mode.
# ps -efZ | grep -E '(LABEL|ping)'
LABEL      UID        PID   PPID  C STIME TTY
unconfined root       2234  1394  0 16:20 pts/0
TIME     CMD
00:00:00 ping 8.8.8.8
# echo 'setprofile /bin/ping' \
  > /proc/2234/attr/current
-bash: echo: write error: Permission denied
```

Creating Profile

- Where to RTM
  - Best bet: “man apparmor.d” - or online man pages
  - Quick guide is at http://wiki.apparmor.net/index.php/QuickProfileLanguage
  - Do not use http://wiki.apparmor.net/index.php/AppArmor_Core_Policy_Reference

Creating Profile

- Start with aa-genprof (install apparmor-utils on Ubuntu)

  # aa-genprof /usr/sbin/nginx

  ….

  [(S)can system log for AppArmor events] / (F)inish

  # #other window

  # service nginx restart

- Do some operations then “S”can message logs, save and quit.. this will be inadequate

Creating Profile, Continued

- Set new profile to complain mode with aa-complain <profile> and restart service
- Run tail -f <messages> and look for apparmor messages
- Edit profile to allow required permissions
- Repeat until warnings are gone
- Try aa-enforce <profile> and fix any further errors
- Important to run with exact same setup as production

Example AppArmor Messages

```
# tail -f /var/log/syslog | grep apparmor
Nov  1 19:17:11 localhost kernel: [6070087.303192]
    type=1400 audit(1478053031.105:4257):
    apparmor="STATUS"  operation="profile_replace"
    profile="unconfined" name="/usr/sbin/nginx"  pid=6452
    comm="apparmor_parser"
Nov  1 19:17:13 localhost kernel: [6070089.472159]
    type=1400 audit(1478053033.277:4258):
    apparmor="DENIED" operation="open"
    profile="/usr/sbin/nginx" name="/dev/null"
    pid=6463 comm="nginx" requested_mask="rw"
    denied_mask="rw" fsuid=0 ouid=0
```

nginx example (no ssl)

```
#include <tunables/global>

/usr/sbin/nginx {
#  include <abstractions/base>
  capability net_bind_service,
  capability dac_override,
  capability setgid,
  capability setuid,

  network inet  stream,
  network inet6 stream,

  signal (receive) peer=unconfined,
  signal peer=@{profile_name},

  /proc/cpuinfo r,
  /proc/stat r,
  /proc/sys/kernel/ngroups_max r,
  /dev/null rw,
  /sys/devices/system/cpu/online r,

  /lib/i386-linux-gnu/** mr,
  /usr/lib/i386-linux-gnu/** mr,

  /etc/ld.so.cache r,
  /etc/localtime r,
  /etc/nginx/** r,
  /etc/ssl/openssl.cnf r,
  /etc/nsswitch.conf r,
  /etc/passwd r,
  /etc/group r,

  /var/log/nginx/access.log w,
  /var/log/nginx/error.log w,
  /var/www/** r,

  /run/nginx.pid rw,
}
```

What about a chroot?

```
# cat /etc/apparmor.d/local/bin.ping
deny network inet icmp,

# ping -c1 google.com
ping: icmp open socket: Permission denied

# chroot /var/chroot ping -c1 google.com
PING google.com (216.58.195.78) 56(84) bytes of data.
64 bytes from sfo07s16-in-f14.1e100.net
  (216.58.195.78): icmp_seq=1 ttl=56 time=4.24 ms

--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss,
  time 0ms
rtt min/avg/max/mdev = 4.249/4.249/4.249/0.000 ms
```

Chroot Continued

Profile must refer to pre-chroot path.
Version 1:

```
#include <tunables/global>
/{var/chroot/,}{usr/,}bin/ping
flags=(chroot_relative) {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>

  capability net_raw,
  capability setuid,
  network inet raw,
  /bin/ping mixr,
  /etc/modules.conf r,
  #include <local/bin.ping>
}
```

Chroot Continued

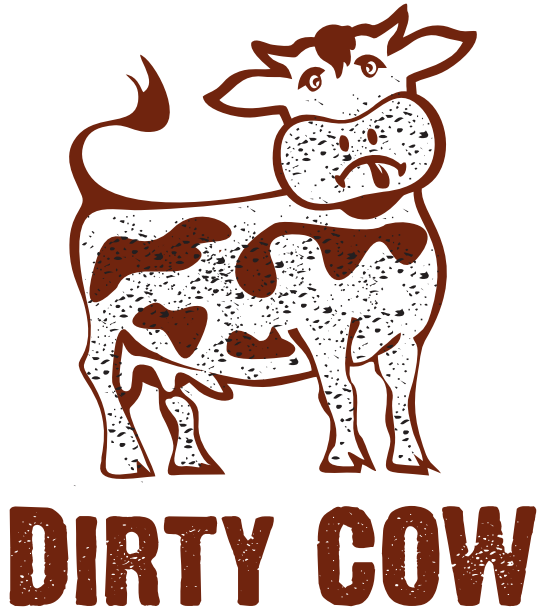
Version 2:

```
/var/chroot/bin/ping {
  #include <abstractions/base>
  #include <abstractions/consoles>
  #include <abstractions/nameservice>
  #include <local/bin.ping>

  capability net_raw,
  capability setuid,
  network inet raw,

  /var/chroot/etc/hosts r,
  /var/chroot/etc/host.conf r,
  /var/chroot/etc/nsswitch.conf r,
  /var/chroot/etc/resolv.conf r,
  /var/chroot/lib/x86_64-linux-gnu/* mr,
}
```

If the program calls another

- By default, child process is confined the same as caller
- Can also use child profile – effectively a nested profile – to further confine
- hats/aa_change_hat can be used from apparmor-aware programs

AppArmor vs.



DirtyCow Exploit #1 dirtyc0w.c

- Uses /proc/self/mem

```
#include <tunables/global>
/home/builder/dirtyc0w {
  /etc/ld.so.cache r,
  /lib/x86_64-linux-gnu/libpthread-2.19.so mr,
  /lib/x86_64-linux-gnu/libc-2.19.so mr,
  /home/builder/dirtyc0w mr,
  /home/builder/foo r,
  /proc/*/mem r,
}
```

DirtyCow Exploit #1 Contd.

```
# ls -lah foo
-r-----r-- 1 root root 19 Nov  2 03:50 foo
~$ cat foo
this is not a test

# tail -f /var/log/syslog
Nov  2 04:06:02 pigtails kernel: [  151.274662]
   type=1400
   audit(1478059562.663:18):  apparmor="DENIED
   operation="open"
   profile="/home/builder/dirtyc0w"
    name="/proc/1149/mem"
   pid=1151 comm="dirtyc0w" requested_mask="w"
   denied_mask="w"
   fsuid=1000 ouid=1000
```

DirtyCow Exploit #1 Contd.

```
builder@pigtails:~$ ./dirtyc0w foo mooooo
mmap 7fa09cfb8000
procselfmem -100000000
madvise 0
builder@pigtails:~$ cat foo
this is not a test
```

Winner: AppArmor

DirtyCow Exploit #2 pokemon.c

- Uses PTRACE_POKETEXT
- profile:

```
#include <tunables/global>
/home/builder/d {
  /etc/ld.so.cache r,
  /lib/x86_64-linux-gnu/** mr,
  /home/builder/d mr,
  /home/builder/pokeball r,
  signal (receive) peer=unconfined,
  signal peer=@{profile_name},
}
```

DirtyCow Exploit #2 Contd.

```
# ls -l pokeball
-rw-r--r-- 1 root root 8 Nov  2 04:29 pokeball
# cat pokeball
pikachu
# tail -f /var/log/syslog | grep -Po "apparmor.+"
apparmor="DENIED" operation="ptrace"
   profile="/home/builder/d"
   pid=1267 comm="d" requested_mask="trace"
   denied_mask="trace" peer="/home/builder/d"
apparmor="DENIED" operation="ptrace"
    profile="/home/builder/d"
   pid=1267 comm="d" requested_mask="tracedby”
   denied_mask="tracedby"peer="/home/builder/d"
```

DirtyCow Exploit #2 Contd.

```
$ ./d pokeball miltank
pokeball
    (___)
    (o o)_____/
     @@ `     \
      \ _____, /miltank
      //    //
     ^^    ^^
mmap 7f2f4e9c5000
# cat pokeball
pikachu
```

Winner: AppArmor

Summary

- AppArmor is a reasonable tradeoff between ease of use and effectiveness
- AppArmor can run on systems other than Ubuntu and should be usable without python
- AppArmor documentation needs a lot of work – try rather than assume

Backup Slides

More AppArmor Commands

- apparmor_parser – core executable
- aa-autodep – generate profile skeleton
- aa-cleanprof – remove superfluous rules, reorganize, and remove comments
- aa-decode: decodes hex strings in kernel log (if present)
- aa-logprof – interactively change profiles from system log warnings
- aa-mergeprof – merge profiles

Docker

- Add “--security-opt=apparmor=<profile name>” after loading profile to override docker-default profile. Example:

```
docker run \
  --security-opt=apparmor="/{usr/,}bin/ping" \
--rm ping ping google.com
```

- Not possible
- Run 'man docker-run' and search for security-opt to check availability
- Not available in 1.6.2 (ubuntu trusty), present in 1.11.2 (ubuntu xenial)

With aa-status, if you see....

- Could not open /sys/kernel/security/apparmor/profiles: No such file or directory
  - Not all patches were applied or userspace/kernel mismatch
- apparmor module is not loaded.
  - Check correct kernel was booted
- apparmor filesystem is not mounted.
  - Check for /sys/kernel/security securityfs mount

Discretionary Access Control

- Every file has an owner
- Owner has complete control over that file
- Owner can decide what other users can do

Other Security Mechanisms

- PaX
- seccomp(-bpf) – filtering for system calls
- capabilities (instead of setuid)
- sysctl settings
- systemtap (patching the kernel)
- compiler features (gcc stack-protector, etc.)
- namespaces
- cgroups

Traditional Permissions in Linux

- Permissions = file permissions (?)
- File has an owner, owner can grant/revoke access to other users -
- Process has owner, only root can mess with that process

apparmor.d directories

- local – contains custom overrides
- disable (for initscripts only)
- force-complain (for initscripts only)
- abstractions (includes)
- libvirt
- namespaces
- tunables