Securing your system with AppArmor & SELinux

Johannes Segitz
SUSE Security Team

2014/11/27

Introduction

What will we cover?

- Mandatory access control
- AppArmor
- SELinux
- Comparison

50 minutes are not much time, so:

- Mix between concrete examples and higher level concepts
- Complex systems, some statements are simpler than the reality

Who am I?

- SUSE employee since 2014, security engineer, resident in Germany
- Long time interest in IT security
- Long time fan of mandatory access control systems (Rule Set Based Access Control - RSBAC, 1998)
- First time in Stockholm, very nice city

Mandatory access control

Discretionary access control (DAC)

Usual form of access control in Linux

- Typical example:

```
# ls -l /etc/shadow
-rw-r-----. 1 root shadow 1421 /etc/shadow
```

- Discretionary: The owner of an object can control the access of the objects he owns

Discretionary access control (DAC)

Drawbacks:

- Coarse: Basically 3 x rwx
- Prone to (user) error

```
# ls -lah ~/.ssh/id_rsa
-rw-rw-rw-. 1 jsegitz users 1.7K ~/.ssh/id_rsa
```

- Hard to analyze
- root == God (- capabilities)

But it's familiar, easy to use and to understand

Mandatory access control (MAC)

Mandatory (in this context):

- Access control decisions are not made by the owner/user
- Access control rules are managed centrally

Advantages:

- Access control in the hand of people who know what they're doing
- Centralized control and review is possible
- Often very fine grained $\rightarrow$ compartmentalization

Drawbacks:

- (Sometimes) hard to understand
- (Sometimes) complex to administer
- Missing support/experience

AppArmor

History

Linux security module (LSM)

- Since 2.6.36 part of the Linux kernel
- Developed by Immunix, bought by Novell
- Default system in SUSE, openSUSE and Ubuntu

Basic idea

Restrict possible actions of processes

- Map profile to process using the path to the binary as key
- (Often) used only for network facing daemons

Advantages:

- Easy administration
- Good tools are available
- Supported in SUSE products

Disadvantages:

- Can't do everything that SELinux can do
- Smaller community

AppArmor profiles

Live in /etc/apparmor.d

- Named by convention: /bin/ping → /etc/apparmor.d/bin.ping
- Local override via files in /etc/apparmor.d/local
- openSUSE 13.1:
  - Active profiles for 37 programs
  - 100 additional profiles under /usr/share/apparmor/extra-profiles

Profile for /bin/ping

```
#include <tunables/global>

/{usr/,}bin/ping {
  #include <abstractions/base>          # 66 rules
  #include <abstractions/consoles>      # 4 rules
  #include <abstractions/nameservice> # 78 rules

  capability          net_raw,
  capability          setuid,

  network             inet raw,

  /{usr/,}bin/ping   mixr,

  /etc/modules.conf r,
}
```

More complicated profile

```
#include <tunables/global>

/usr/bin/foo {
  #include <abstractions/base>
  capability           setgid,
  network              inet tcp,

  link /etc/sysconfig/foo -> /etc/foo.conf,

  /dev/{,u}random      r,

  /etc/foo/*           r,
```

More complicated profile

```
/lib/ld-*.so*         mr,
/lib/lib*.so*         mr,

/proc/[0-9]**         r,

/tmp/                 r,
/tmp/foo.pid          wr,
/tmp/foo_data.*       lrw,

/@{HOME}/.foo_lock    kw,
```

Execute modes

```
/bin/*                  Px,
/usr/bin/foobar         Cx,
/bin/mount              Ux,
```

Different ways of executing other programs:

- **Px**: Discrete profile execute mode
- **Cx**: Discrete local profile execute mode
- **Ux**: Unconfined execute mode
- **ix**: Inherit execute mode

Lowercase versions (**px**, **cx**, **ux**) do not scrub the environment

More complicated profile

Remember: `/usr/bin/foobar Cx` → discrete local profile

```
  profile /usr/bin/foobar flags=(complain) {
    /bin/bash          rmix,
    /bin/cat           rmix,
    /var/log/foobar*   rwl,
    /etc/foobar        r,

    rlimit data        <= 100M,
    rlimit nice        >= 10,
  }
}
```

Profile creation

Ways of creating new profiles:

- Write them from scratch
- Adapt existing profiles
- Use one of the tools that are shipped for that purpose

aa-autodep

aa-autodep creates a basic framework of a profile in complain mode.

```
# aa-autodep ls
```

yields

```
#include <tunables/global>

/usr/bin/ls flags=(complain) {
  #include <abstractions/base>

  /usr/bin/ls mr,
}
```

So that will be really useful ...

aa-genprof

Next try: **aa-genprof**

- Generates a basic profile, sets it to complain mode
- Execute the application and analyze log events

```
> aa-genprof ls
Writing updated profile for /usr/bin/ls.
Setting /usr/bin/ls to complain mode.
<snip>
Profiling: /usr/bin/ls

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

Work with the application and try to provoke every access pattern

```
# ls /dev
```

aa-genprof

Scan the resulting log entries:

```
Profile:  /usr/bin/ls
Path:     /dev/
Mode:     r
Severity: unknown

 [1 - /dev/]

[(A)llow] / (D)eny / (G)lob / Glob w/(E)xt / (N)ew /
    ↪ Abo(r)t / (F)inish / (O)pts
```

Other tools

- **aa-logprof**: Interactively scan and review log entries
- **aa-easyprof**: Easy to use tool. Results might be less restrictive than with other tools
- **aa-exec**: Launches a program in an AppArmor profile

Always review the result of the tools!

SELinux

History

Security **E**nhanced **Linux**

- Linux security module (LSM), developed by the National Security Agency (NSA)

  Don't panic, it's open source and reviewed thoroughly
- First release 2000, since then integrated in the Linux kernel

Basic idea

- Type Enforcement (TE). Every object has
  - an user: `unconfined_u`
  - a role: `unconfined_r`
  - a type: `unconfined_t`
  - a sensitivity: `s0-s0`
  - a category: `c0.c1023`
- These form the **Security Context (SC)**

```
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Basic idea

(Almost) everything has a SC.

- Files

```
# ls -lZ /etc/shadow
----------. root root system_u:object_r:shadow_t:s0
    ↪    /etc/shadow
```

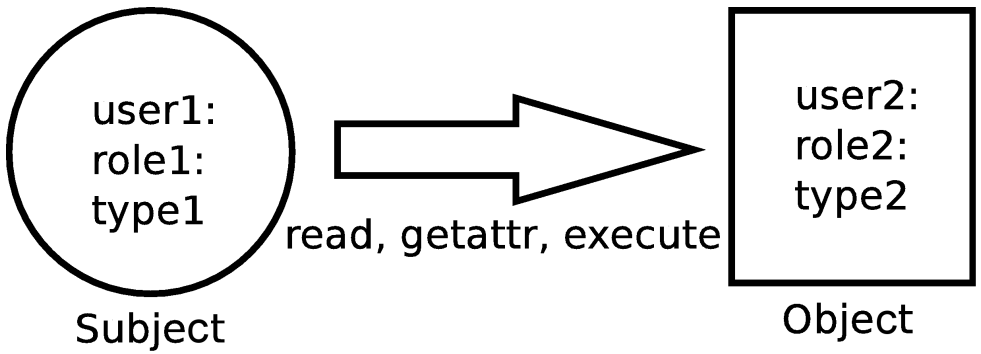
- Processes

```
# ps axZ | grep 'postfix/master'
system_u:system_r:postfix_master_t:s0 1250 ?   Ss
    ↪   0:00 /usr/lib/postfix/master -w
```

- Sockets, packets, ... 83 security classes

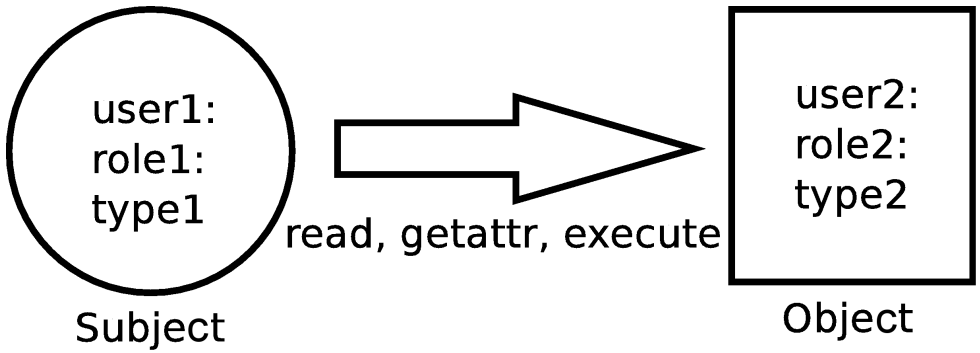
Basic idea

- DAC comes first
- Every access decision is checked against the SC of the source and the target



- Firewall for system calls

Basic rules



Basic SELinux statements:

```
type type1;
type type2;
role role1 types type1;
role role2 types type2;
allow type1 type2:file { read getattr execute };
```

Reference policy

Basic SELinux statements are not the way to go:

- Way too many rules for even simple programs
- Hard to maintain
- Hard to distribute the work of creating policies

Solution: Reference policy (refpolicy)

- Community project to create a base policy
- Modular, uses M4 macros to create interfaces
- Contains custom modifications for various distributions

Working with refpolicy

What does it take to confine a daemon in SELinux?

- Types for the processes (domain types)
- Types for the files
- Transition rules
- Rules to allow standard interactions (e.g logging, ...)

Ways of creating new modules:

- Write them from scratch
- Adapt existing profiles
- Use one of the tools that are shipped for that purpose

Adapt existing profiles

Good place to start: refpolicy-contrib

`https://github.com/TresysTechnology/refpolicy-contrib`

- Currently 358 modules
- Ranging from $<$ 20 lines to $>$ 1400 (apache)
- Module consists of three files:
  - .fc files: Contain rules that specify types for files
  - .if files: Contain interfaces that the module provides
  - .te files: Contains all rules necessary for this module

Example: arpwatch .te file

One of the smaller profiles, but still only parts of the module

```
policy_module(arpwatch, 1.11.0)

type arpwatch_t;
type arpwatch_exec_t;
init_daemon_domain(arpwatch_t, arpwatch_exec_t)

type arpwatch_data_t;
files_type(arpwatch_data_t)

type arpwatch_var_run_t;
files_pid_file(arpwatch_var_run_t)
```

Example: arpwatch .te file

```
allow arpwatch_t arpwatch_t:capability { net_admin
    ↪ net_raw setgid setuid };

dontaudit arpwatch_t arpwatch_t:capability
    ↪ sys_tty_config;

allow arpwatch_t arpwatch_t:tcp_socket { accept listen
    ↪ };

manage_files_pattern(arpwatch_t, arpwatch_data_t,
    ↪ arpwatch_data_t)

kernel_read_network_state(arpwatch_t)
kernel_read_system_state(arpwatch_t)
```

Example: arpwatch .fc file

```
/usr/sbin/arpwatch        --
    ↪ gen_context(system_u:object_r:arpwatch_exec_t, s0)
/var/arpwatch(/.*)?       --
    ↪ gen_context(system_u:object_r:arpwatch_data_t, s0)
/var/run/arpwatch.*\.pid --
    ↪ gen_context(system_u:object_r:arpwatch_var_run_t,
    ↪ s0)
```

Specifies only initial context. Used by `restorecon` and other SELinux tools

audit2allow

Looked tedious? It is.

audit2allow is a bit like aa-logprof

- Analyzes SELinux denial messages
- Generates rules to allow necessary access
- Is aware of refpolicy interfaces
- Suggests booleans that could allow the access

SELinux log messages

```
type=AVC msg=audit(1416499522.810:77): avc:  denied
    ↪ { transition } for  pid=1282 comm="sshd"
    ↪ path="/usr/bin/zsh" dev="vda2" ino=40462
    ↪ scontext=system_u:system_r:kernel_t:s0
    ↪ tcontext=unconfined_u:unconfined_r:unconfined_t:s0
    ↪ tclass=process
```

audit2allow uses those messages

- But don't use it with every denial, think first. See `scontext` above.
- In this case systemd was running as `kernel_t`, not `init_t`.

audit2allow example

Use all denials since the last boot (`-b`) and create a module named `local` (`-M`) with refpolicy interfaces (`-R`).

```
# audit2allow -b -M local -R
```

- Generates four files: `local.te`, `local.if`, `local.fc` and a compiled module `local.pp`
- Analyze at those files!
- Load with

```
# semodule -i local.pp
```

Comparison

SELinux or AppArmor

Which system is better?

- It depends :)
- Do you know one of those systems? Stick with it
- Do you work in a high security environment? SELinux
- Do you learn from scratch and have some time? SELinux

You can't have both active at the same time, so you must choose.

My personal favorite: SELinux.

Endnote

Thanks

Thank you for your attention. Questions?