Namespacing in SELinux

Linux.conf.au 2018
Sydney, Australia

James Morris
jmorris@namei.org

Introduction

- Who am I?
  - Linux security subsystem maintainer
    - Previously: Crypto API, Netfilter, SELinux, LSM, IPSec, MCS, sVirt
    - Recovering manager
    - blog.namei.org
    - @xjamesmorris
- Overview
  - Briefly review technologies
  - Discuss requirements
  - SELinux namespace prototype
  - Current work: inode labeling
  - Future work

SELinux

- Label-based mandatory access control (MAC)
  - Set security labels on:
    - Subjects
    - Objects
  - Define permissions
  - Centrally managed policy
  - Enforced by kernel
- Generalized
- Separation of policy and mechanism

Linux Security Modules (LSM)

- Kernel API for access control
- Hooks
  - Located at security decision points
  - All security relevant information available
  - Race-free
- Kind of like Netfilter but for the whole kernel
- Pluggable: Smack, SELinux, AppArmor etc.

Linux Namespaces

- Private views of global resources
  - mount, network, ipc, pid, user, uts, cgroup
- APIs: clone(2), setns(2), unshare(2)
- See also: pam_namespace(8)
- Uses:
  - Sandboxes
  - Containers
  - Multi-level security (!)
- **No** namespacing of LSM or other security APIs

Containers

- Not a Thing ™
- Actually namespaces + cgroups + magic
  - Docker, lxc, lxd etc.
- Very popular
- Kernel security APIs not containerized, e.g.
  - Limits functionality for OS-like containers
  - SELinux on Fedora-based distros pretends to be disabled inside container, and yet … !

Use Cases

- Enable SELinux confinement within a container
  - Currently runs as one global label and appears disabled inside container
- Running SELinux enabled within container but not host
  - ChromeOS running Android SELinux container
- Different policy / distro versions on same host
  - Build systems, devops
- Virtual smartphone environment (Cells/Cellrox)
  - Multiple Android instances

Requirements

- Common requirement across several use-cases:
  - Private kernel security APIs inside containers
- OS-like behavior inside container:
  - Load and enforce own policy
  - Independent enforcing mode
  - Isolate from global and other containers
  - Potentially with different versions of policy
  - Fully privileged & not dependent on user namespaces
  - *Potentially with different distros and/or LSMs (**hard**)*

LSM Namespacing

- *Just* create an LSM namespace!
- Presented idea briefly at Plumbers 2017
  - Feedback:
    - Not enough semantic information at LSM layer, no generic solution, would be very complex
    - Real work needs to be done in security modules
    - Implement there
- SELinux prototype already written
  - Just a few problems to solve...

SELinux Namespace Prototype

- Developed by Stephen Smalley
- Published at:
  - https://github.com/stephensmalley/selinux-kernel/tree/selinuxns
- Defines: `struct selinux_ns`
  - Encapsulates global SELinux state
    - Policy database, object label mapping etc.
  - Passed to internal APIs (“security server”)
    - Operates on selinux_ns instead of global state
  - Initial / global namespace:
    - `struct selinux_ns *init_selinux_ns;`

SELinux Namespace Prototype

- Adds per-namespace selinuxfs instances
  - unshare mount ns and mount new selinuxfs
- Move AVC into namespace
- Add per-namespace support for
  - in-memory inodes & superblocks
  - creds
  - SELinux netlink socket
    - requires unsharing network namespace
- Write to selinuxfs unshare node to instantiate

Prototype: Example Use

```
echo 1 > /sys/fs/selinux/unshare
unshare -m -n
umount /sys/fs/selinux
mount -t selinuxfs none /sys/fs/selinux
load_policy
runcon unconfined_u:unconfined_r:unconfined_t:s0:c0.c1023 /bin/bash
setenforce 1
```

Next Step: On-Disk Inodes

- Prototype only supports in-memory inodes
- Implemented for on-disk inodes as xattrs
  - security.selinux = “something”
- Need a way to namespace these labels
  - Answer: extend extended attribute:
    - security.selinux.NS_NAME = “maybe something else”

On-Disk Inodes

- 'NS_NAME' can be anything:
  - container name, UUID, whatever
  - write value to 'unshare' node
- Translated by kernel:
  - Entirely transparent to container
    - No app or policy changes needed
  - Global admin tools need to be aware

On-Disk Inodes: Example Use

- Create a namespace "NS1":

```
echo NS1 > /sys/fs/selinux/unshare
unshare -m -n
umount /sys/fs/selinux
mount -t selinuxfs none /sys/fs/selinux
load_policy
runcon unconfined_u:unconfined_r:unconfined_t:s0:c0.c1023 /bin/bash
setenforce 1
cat /sys/fs/selinux/unshare
NS1
```

On-Disk Inodes: Example Use

- Create a file 'c':

```
touch c
ls -lZ
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 c
```

- The on-disk label actually looks like this:

```
getfattr -d -m . c
file: c
security.selinux.NS1="unconfined_u:object_r:admin_home_t:s0"
```

On-Disk Inodes: v0.1 Feedback

- Track namespace nesting to maintain label provenance
  - e.g. prevent leaking sensitive files
- Label files in ancestor namespaces upon creation
- Nested policy enforcement
- Label inheritance rules
- Read-only shared labels
  - e.g. shared /usr

On-Disk Inodes v0.2: Nested Example

- With v0.2 patchset, create a nested namespace, "NS2":

```
echo NS2 > /sys/fs/selinux/unshare
unshare -m -n
umount /sys/fs/selinux
mount -t selinuxfs none /sys/fs/selinux
load_policy
runcon unconfined_u:unconfined_r:unconfined_t:s0:c0.c1023 /bin/bash
setenforce 1
cat /sys/fs/selinux/unshare
NS1.NS2
```

On-Disk Inodes v0.2: Nested Example

- Create a new file, 'd':

```
touch d
ls -Z
-rw-r--r--. root root system_u:object_r:unlabeled_t:s0 c
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 d
```

- Relabel the file:

```
chcon -t etc_t d
ls -Z d
-rw-r--r--. root root unconfined_u:object_r:etc_t:s0   d
```

- View the on-disk label:

```
getfattr -d -m . d
file: d
security.selinux.NS1.NS2="unconfined_u:object_r:etc_t:s0"
```

Current Work

- listxattr(2) filtering
  - need to translate namespace names
  - xattr list is constructed at fs-level
  - two possibilities, via LSM hooks:
    - fs callback into SELinux
    - rewrite xattr list at SELinux level
  - both ugly
  - Any ideas? Email me...

Future Work

- Label inheritance
- Object inheritance (open fd)
- Nested enforcement
- Label sharing
- Networking
- Non-current namespaces:
  - Not all hooks invoked in process context
  - Operation on behalf of other credentials
- Logging

Discussion

Overflow Slides: Open Issues

- Should kernel enforce required set of namespaces (SELinux + mount + net) ?

- SELinux ns attached to cred and not tied to user ns.
  - Should we: wrap user ns in a more abstract security ns which may include: user ns, SELinux ns, other security ns's ?

- Issues re. inode & superblock blobs:
  - Pins SELinux ns in memory for lifetime of blobs
  - Handling non-sleepable callers when objects accessed for first time in namespace

Overflow Slides: Open Issues

- How to bound resource usage when creating SELinux namespaces
- How to manage nested access control policies and labels
- How to deal with secids (32-bit IDs) which are passed to core kernel and cached there
  - Make them global (Casey's suggestion)
  - Convey namespace info with them
  - Also a major stacking issue
  - 64-bits?