SELinux

Introduction and Security Analysis

Radhika B S

Indian Institute of Technology Bombay

Overview

Access Control in Operating Systems

The Reference Monitor

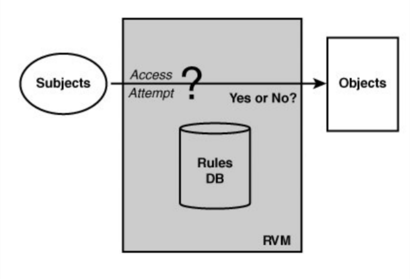

Figure 1: The Reference Monitor

- subjects: Active entities like processes, users ...
- objects: Passive entities like files, sockets ...

Discretionary Access Control

```
-rw-r--r-- 1 r     r       f1
-rw-r--r-- 1 root root     f2
```

- Access decisions are taken based on user identity and the ownership of the object
- Permissions can be changed at owner's discretion
- *root* is omnipotent
- Coarse-grained access control

Password Management in Linux

- Linux provides passwd command to allow regular users to change their own password and root to change any user's password
- passwd needs to access file /etc/shadow which stores password hashes
- /etc/shadow is owned by root and only root can read/write it
- The executable passwd is also owned by root

```
[r@localhost bin]$ ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 27872 usr/bin/passwd
```

Password Management in Linux

- setuid is used with passwd to allow regular users to change their password
- When a regular user executes passwd, it runs with root privilege
- setuid programs are usually small and highly verified

Password Management in Linux

Any process running as root can access /etc/shadow

Linux Security Module

- Realizing the need for a better access control, several MAC based systems were developed
- Many of those solutions had similar design approach
- LSM was developed to provide a framework for supporting variety of custom MAC implementations
- Exposes hooks for labeling and access control decision making
- Some of the systems include AppArmor, TOMOYO, Smack

Linux Security Module

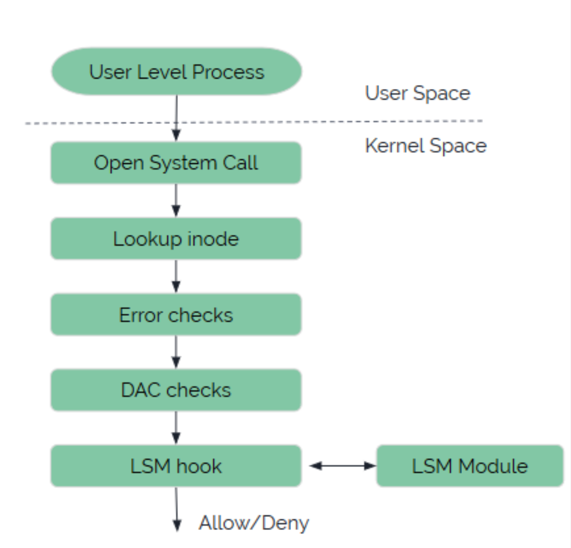

Figure 2: Linux Security Module

SELinux

SELinux

- MAC based access control system developed by NSA which was made open source in 2001
- Provides confinement and helps in proactive security
- Successfully protected systems against several zero-day attacks especially privilege escalation attacks such as DirtyCOW, ShellShock
- Also being used in Android as SEAndroid
- About 75% (1.5 billion) of the Android devices running today are using SELinux in enforcing mode. The Android project estimates that SELinux has reduced the severity of almost half of their kernel bugs

Type Enforcement

- Related entities are grouped into types
- To access an object, the subject's type must be authorized to access the object's type
- Provides flexible and fine-grained access control

SELinux Security Context

```
user:role:type[:levels]
```

SELinux Components

- Object class: Category of kernel resources such as files, directories, sockets etc. Each class has a set of associated actions
- Type: Logical grouping of objects/subjects
- Domain: Common term used for subject types
- Attributes: Collection of SELinux types/domains. Used for ease of rule specification.

SELinux Policy Specification

- By default, every access is denied
- This can be overridden by using allow rules

```
allow source target:class permissions
```

A depiction of an allow rule

```
allow user_t bin_t : file {read execute getattr};
```

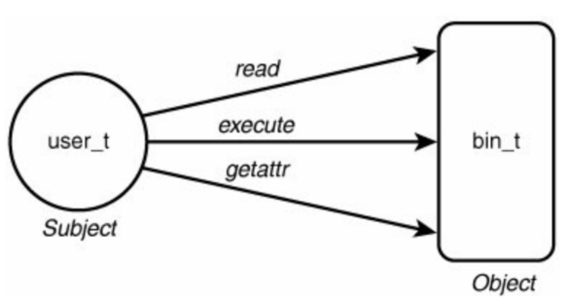

Figure 3: A depiction of an allow rule

Information Flow Analysis of SELinux Policy

Information Flow in SELinux Policy

```
allow httpd_t user_t:file read
```

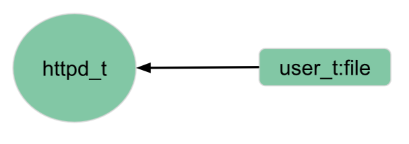

Figure 4: Information Flow in Read

Information Flow in SELinux Policy

```
allow httpd_t user_t:file write
```

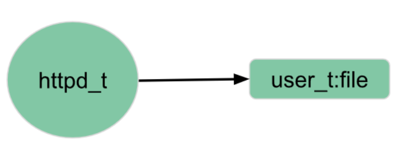

Figure 5: Information Flow in Write

Information Flow in SELinux Policy

```
allow ping_t  user_tty_device_t:chr_file write;
allow updpwd_t user_tty_device_t:chr_file read;
allow updpwd_t shadow_t:file write;
```

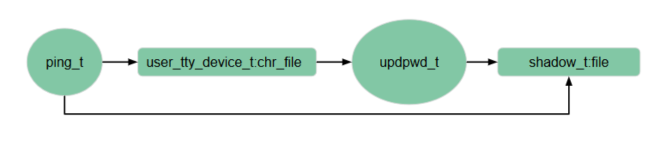

Figure 6: Indirect Information Flow

Neverallow Rules

- Have similar syntax as allow rules
- Enables policy writer to specify certain allow rules that should be added to the policy
- Help in avoiding accidental addition of unintended allow rules
- Used during compilation
- If a policy contains contradictory allow rules, the compilation fails

Readers Writers Flow Model

Readers Writers Flow Model

- Let S and O be the set of subjects and objects in the system respectively.
- An RWFM label, also called as RW Class is defined as a triplet (s, R, W). Where $s \in S$ denotes the owner of the information in the class. R denotes the set of subjects which can read the objects of the class. W denotes the set of subjects which can write or which have influenced the class.
- A subject s is allowed to read an object o if $owner(s) \in R(o)$ and $R(o) \supseteq R(s)$ and $W(o) \subseteq W(s)$
- A subject s is allowed to write an object o if $owner(s) \in W(o)$ and $R(s) \supseteq R(o)$ and $W(s) \subseteq W(o)$

Analysing Inconsistencies in SELinux Policy

Analysing Inconsistencies in SELinux Policy

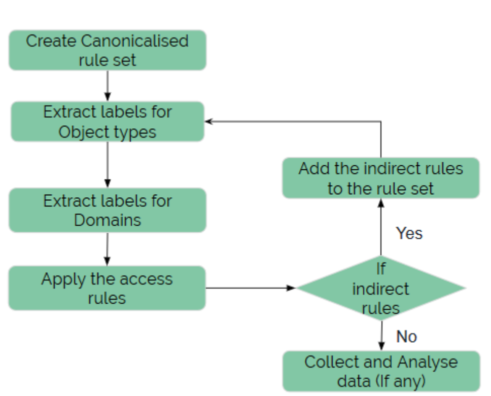

Figure 7: Analysing Inconsistencies

Canonicalization

- A canonical rule corresponds to a single access and made of a single domain, a single object type and a single permission
- Rules in a policy can contain attributes and sets of components
- Split the rules so that each resulting rule corresponds to a single access
- This helps us understand the effect of each individual access on the information flow

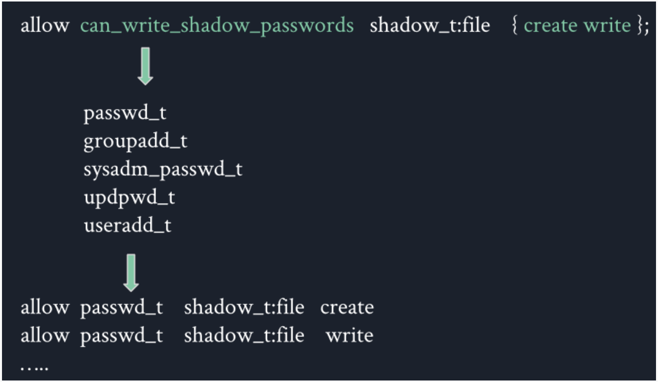

Figure 8: Canonicalization

Labelling the object types

Algorithm 1: ExtractObjectTypeLabel
Input: Canonicalized policy rule set
Output: Labels of all the object types

foreach $t \in T$ do
  $R(t) = W(t) = \{\}$

foreach "*allow d t perm*" do
  if $perm == r$ then
    $R(t) = R(t) \cup d$
  else if $perm == w$ then
    $W(t) = W(t) \cup d$

Figure 9: Labeling the Object types

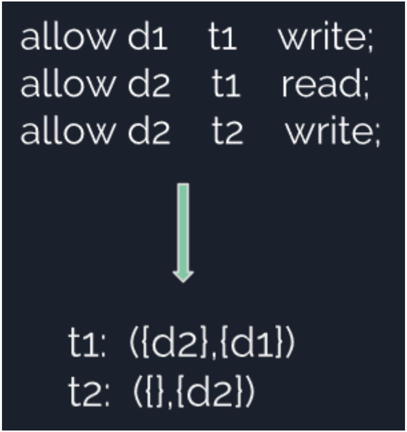

Figure 10: Labeling the Object types

Labelling the Domains

Algorithm 2: ExtractDomainLabel
Input: Canonicalized policy rule set and labels of object types
Output: Labels of all the domains

foreach $d \in D$ do
    $R(d) = W(d) = \{D\}$

foreach $t \in T$ do
    foreach $d \in R(t)$ do
        $R(d) = R(d) \cap R(t)$
    foreach $d \in W(t)$ do
        $W(d) = W(d) \cap W(t)$

Figure 11: Labeling the Domains

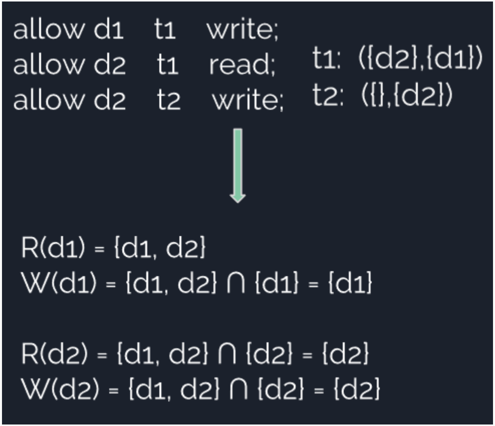

Figure 12: Labeling the Domains

Access Rules Check

Algorithm 3: AccessRuleCheck
Input: Canonicalized policy rule set and labels of object types and labels of domains
Output: Set of indirect rules

IndirectRuleSet $= \{\}$
foreach "*allow d t perm*" do
    if $perm == r$ AND $W(t) \not\subseteq W(d)$ then
        foreach $d1 \in W(t) - W(d)$ do
            foreach $t1$ s.t $d \in W(t1)$ do
                IndirecrRuleSet $\cup=$ "*allow d1 t1 w*"

    if $perm == w$ AND $R(t) \not\subseteq R(d)$ then
        foreach $d1 \in R(t) - R(d)$ do
            foreach $t1$ s.t $d \in R(t1)$ do
                IndirecrRuleSet $\cup=$ "*allow d1 t1 r*"

Figure 13: Access Rules Check

Access Rules Check

- A subject s is allowed to read an object o if $R(d) \subseteq R(t)$ and $W(t) \subseteq W(d)$
- A subject s is allowed to write an object o if $R(d) \supseteq R(t)$ and $W(d) \subseteq W(t)$

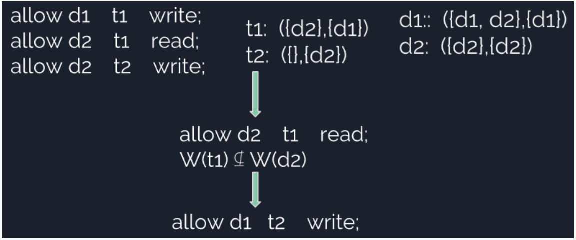

Figure 14: Access Rules Check

Collecting and Analysing Data

- Number of contradictions generated by each rule
- Number of contradictions generated by each domain
- For any given indirect rule, generate the sequences of accesses that can lead to the indirect flow

- Android platform manifest.
  https://android.googlesource.com/platform/manifest/.
  Accessed: Jan, 2018.
- Selinux reference policy.
  https://github.com/TresysTechnology/refpolicy.
  Accessed: Jan, 2018.
- C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wagle, and V. D. Gligor.
  Subdomain: Parsimonious server security.
  In *LISA*, pages 355–368, 2000.

T. Harada, T. Horie, and K. Tanaka.
Task oriented management obviates your onus on linux.
In *Linux Conference*, volume 3, page 23, 2004.

N. N. Kumar and R. Shyamasundar.
Realizing purpose-based privacy policies succinctly via information-flow labels.
In *Big Data and Cloud Computing (BdCloud), 2014 IEEE Fourth International Conference on*, pages 753–760. IEEE, 2014.

F. Mayer, D. Caplan, and K. MacMillan.
SELinux by example: using security enhanced Linux.
Pearson Education, 2006.

C. Schaufler.
The simplified mandatory access control kernel.
White Paper, pages 1–11, 2008.

C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman.
Linux security modules: General security support for the linux kernel.
In *USENIX Security Symposium*, volume 2, pages 1–14, 2002.