SELinux

Don Porter
CSE 506

MAC vs. DAC

+ By default, Unix/Linux provides **Discretionary Access Control**
  + The user (subject) has discretion to set security policies (or not)
  + Example: I may 'chmod o+a' the file containing 506 grades, which violates university privacy policies
+ **Mandatory Access Control** enforces a central policy on a system
  + Example: MAC policies can prohibit me from sharing 506 grades

---

SELinux

+ Like the Windows 2k ACLs, one key goal is enforcing the principle of least authority
  + No 'root' user
  + Several administrative roles with limited extra privileges
  + Example: Changing passwords does not require administrative access to printers
    + The principle of least authority says you should only give the minimum privilege needed
  + Reasoning: if 'passwd' is compromised (e.g., due to a buffer overflow), we should limit the scope of the damage

---

SELinux

+ Also like Win2k ACLs, a goal is to specify fine-grained access control permission to kernel objects
  + In service of principle of least authority
  + Read/write permissions are coarse
  + Lots of functions do more limited reads/write

SELinux + MAC

+ Unlike Win2k ACLs, MAC enforcement requires all policies to be specified by an administrator
  + Users cannot change these policies
+ Multi-level security: Declassified, Secret, Top-Secret, etc.
  + In MLS, only a trusted declassifier can lower the secrecy of a file
  + Users with appropriate privilege can read classified files, but cannot output their contents to lower secrecy levels
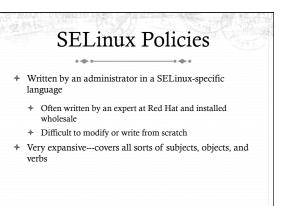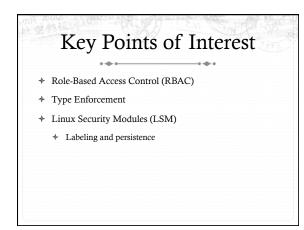

Example

+ Suppose I want to read a secret file
+ In SELinux, I transition to a secret role to do this
  + This role is restricted:
    + Cannot write to the network
    + Cannot write to declassified files
  + Secret files cannot be read in a declassified role
+ Idea: Policies often require applications/users to give up some privileges (network) for others (access to secrets)

General principles

- ✦ Secrecy (Bell-LaPadula)
  - ✦ No read up, no write down
  - ✦ In secret mode, you can't write a declassified file, or read top-secret data
- ✦ Integrity (Biba)
  - ✦ No write up, no read down
  - ✦ A declassified user can't write garbage into a secret file
  - ✦ A top-secret application can't read input/load libraries from an untrusted source (reduce risk of compromise)

SELinux Policies

- ✦ Written by an administrator in a SELinux-specific language
  - ✦ Often written by an expert at Red Hat and installed wholesale
  - ✦ Difficult to modify or write from scratch
- ✦ Very expansive---covers all sorts of subjects, objects, and verbs

Key Points of Interest

- ✦ Role-Based Access Control (RBAC)
- ✦ Type Enforcement
- ✦ Linux Security Modules (LSM)
  - ✦ Labeling and persistence

Role-Based Access Control

- ✦ Idea: Extend or restrict user rights with a **role** that captures what they are trying to do
- ✦ Example: I may browse the web, grade labs, and administer a web server
  - ✦ Create a role for each, with different privileges
  - ✦ My grader role may not have network access, except to blackboard
  - ✦ My web browsing role may not have access to my home directory files
  - ✦ My admin role and web roles can't access students' labs

Roles vs. Restricted Context

- ✦ Win2k ACLs allow a user to create processes with a subset of his/her privileges
- ✦ Roles provide the same functionality
  - ✦ But also allow a user to **add** privileges, such as administrative rights
- ✦ Roles may also have policy restrictions on who/when/ how roles are changed
  - ✦ Not just anyone (or any program) can get admin privileges
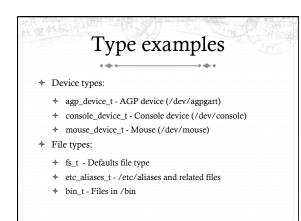
The power of RBAC

- ✦ Conditional access control
- ✦ Example: Don't let this file go out on the internet
  - ✦ Create secret file role
    - ✦ No network access, can't write any files except other secret files
    - ✦ Process cannot change roles, only exit
    - ✦ Process can read secret files
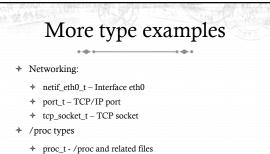- ✦ I challenge you to express this policy in Unix permissions!

Roles vs. Specific Users

+ Policies are hard to write
+ Roles allow policies to be generalized
  + Users everywhere want similar restrictions on their browser
+ Roles eliminate the need to re-tailor the policy file for every user
  + Anyone can transition to the browser role

Type Enforcement

+ Very much like the fine-grained ACLs we saw last time
+ Rather than everything being a file, objects are given a more specific type
  + Type includes a set of possible actions on the object
    + E.g., Socket: create, listen, send, recv, close
  + Type includes ACLs based on roles

Type examples

+ Device types:
  + agp_device_t - AGP device (/dev/agpgart)
  + console_device_t - Console device (/dev/console)
  + mouse_device_t - Mouse (/dev/mouse)
+ File types:
  + fs_t - Defaults file type
  + etc_aliases_t - /etc/aliases and related files
  + bin_t - Files in /bin

More type examples

+ Networking:
  + netif_eth0_t – Interface eth0
  + port_t – TCP/IP port
  + tcp_socket_t – TCP socket
+ /proc types
  + proc_t - /proc and related files
  + sysctl_t - /proc/sys and related files
  + sysctl_fs_t - /proc/sys/fs and related files

Detailed example

+ ping_exec_t type associated with ping binary
+ Policies for ping_exec_t:
  + Restrict who can transition into ping_t domain
    + Admins for sure, and init scripts
    + Regular users: admin can configure
  + ping_t domain (executing process) allowed to:
    + Use shared libraries
    + Use the network
    + Call ypbind (for hostname lookup in YP/NIS)

Ping cont.

+ ping_t domain process can also:
  + Read certain files in /etc
  + Create Unix socket streams
  + Create raw ICMP sockets + send/recv on them on any interface
  + setuid (Why? Don't know)
  + Access the terminal
  + Get file system attributes and search /var (mostly harmless operations that would pollute the logs if disallowed)
    + Violate least privilege to avoid modification!

Full ping policy

```
01 type ping_t, domain, privlog;
02 type ping_exec_t, file_type, sysadmfile, exec_type;
03 role sysadm_r types ping_t;
04 role system_r types ping_t;
05
06 # Transition into this domain when you run this
program.
07 domain_auto_trans(sysadm_t, ping_exec_t, ping_t)
08. domain_auto_trans(initrc_t, ping_exec_t, ping_t)
09
10 uses_shlib(ping_t)
11 can_network(ping_t)
12 general_domain_access(ping_t)
13 allow ping_t { etc_t resolv_conf_t }:file { getattr
read };
14 allow ping_t self:unix_stream_socket
create_socket_perms;
15
16 # Let ping create raw ICMP packets.
17 allow ping_t self:rawip_socket {create ioctl read
write bind getopt setopt};
18 allow ping_t any_socket_t:rawip_socket sendto;
```

```
19
20 auditallow ping_t any_socket_t:rawip_socket
sendto;
21
22 # Let ping receive ICMP replies.
23 allow ping_t { self icmp_socket_t }:rawip_socket
recvfrom;
24
25 # Use capabilities.
26 allow ping_t self:capability { net_raw setuid };
27
28 # Access the terminal.
29 allow ping_t admin_tty_type:chr_file
rw_file_perms;
30 ifdef(`gnome-pty-helper.te', `allow ping_t
sysadm_gph_t:fd use;')
31 allow ping_t privfd:fd use;
32
33 dontaudit ping_t fs_t:filesystem getattr;
34
35 # it tries to access /var/run
36 dontaudit ping_t var_t:dir search;
```

Linux Security Modules

+ Culturally, top Linux developers care about writing a good kernel
  + Not as much about security
  + Different specializations
+ Their goal: Modularize security as much as humanly possible
  + Security folks write modules that you can load if you care about security; kernel developers don't have to worry about understanding security

Basic deal

+ Linux Security Modules API:
  + Linux developers put dozens of access control hooks all over the kernel
    + See include/linux/security.h
  + LSM writer can implement access control functions called by these hooks that enforce arbitrary policies
  + Linux also adds opaque "security" pointer that LSM can use to store security info they need in processes, inodes, sockets, etc.

SELinux example

+ A task has an associated security pointer
  + Stores current role
+ An inode also has a security pointer
  + Stores type and policy rules
+ Initialization hooks for both called when created

SELinux example, cont.

+ A task reads the inode
  + VFS function calls LSM hook, with inode and task pointer
  + LSM reads policy rules from inode
+ Suppose the file requires a role transition for read
  + LSM hook modifies task's security data to change its role
  + Then read allowed to proceed

Problem: Persistence

+ All of these security hooks are great for *in memory* data structures
  + E.g., VFS inodes
+ How do you ensure the policy associated with a given file persists across reboots?

Extended Attributes

+ In addition to 9+ standard Unix attributes, associate a small key/value store with an on-disk inode
  + User can tag a file with arbitrary metadata
  + Key must be a string, prefixed with a domain
    + User, trusted, system, security
  + Users must use 'user' domain
  + LSM uses 'security' domain
+ Only a few file systems support extended attributes
  + E.g., ext2/3/4; not NFS, FAT32

Persistence

+ All ACLs, type information, etc. are stored in extended attributes for persistence
+ Each file must be *labeled* for MAC enforcement
  + Labeling is the generic problem of assigning a type or security context to each object/file in the system
  + Can be complicated
+ SELinux provides some tools to help, based on standard system file names and educated guesses

Summary

+ SELinux augments Linux with a much more restrictive security model
  + MAC vs. DAC
+ Understand Roles and Types
+ Basic ideas of LSM
  + Labeling and extended attributes