CSE 502: Operating Systems

Security

Access Control Lists

Background (1)

- If everything in Unix is a file…
  - Everything in Windows is an object
- Why not files?
  - Not all OS abstractions make sense as a file
- Examples:
  - Eject button on an optical drive
  - Network card

Windows object model

- Everything (including files) is a generic OS object
- New object types can be created/extended
  - Have arbitrary methods beyond just open/read/write/etc…
- Objects are organized into a tree-like hierarchy
- Try out Windows object explorer (winobj)
  - sysinternals.net

Background (2)

- Windows NT and 2000 centralized administration
  - Create a user account once, can log onto all systems
  - vs. creating different accounts on 100s of systems
- Active Directory: Domain server for user accounts
  - Log on to a workstation using an AD account
  - Ex: CS\lu – Domain CS, user id lu
  - Used by our department, centralizes user management

Active Directory

- Centralize store of users, printers, workstations, etc…
- Each machine caches this info as needed
  - Ex., once you log in, the machine caches your credentials

Big Picture

- Need "language" to express what is allowed or not
- Access Control Lists are a common way to do this
- Structure: "Allowed|Denied: Subject Verb Object"

Unix permissions as ACLs

```
-rw-------@ 1 lu staff  151841 Nov 10 08:45 win2kacl.pdf
```

- Allowed|Denied: Subject Verb Object
- Allowed: lu read win2kacl.pdf
- Allowed: lu write win2kacl.pdf
- Denied: staff read win2kacl.pdf
- Denied: other * win2kacl.pdf

Fine-grained ACLs

- Why have subjects other than users/groups?
  - Not all of my programs are equally trusted
  - Web browser vs. tax returns
  - Want to run some applications in a restricted context
- Still want a unified desktop and file system
  - Don’t want to log out and log in for different applications
- Real goal: Associate restricted context with program

Why different verbs/objects

- Aren’t read, write, and execute good enough?
- Example: Changing passwords
  - Yes, you read and write the password file
  - But not directly
    - I shouldn’t be able to change others’ passwords
  - Administrator gives utility/service permission to write
    - Gives you permission to call a specific service function (change password) with arguments (your own user id/pass)

Fine-grained access control lists

- Keep user accounts and associated permissions
  - But let users create restricted subsets of their permissions
- In addition to files, associate ACLs with any object
  - ACLs can be long, with different rules for each user/context
- And not just RWX rules
  - But any object method can have different rules

Big picture

- ACLs written in terms of enterprise-wide principals
  - Users in AD
  - Objects that may be system local or on a shared file system
  - Object types and verbs usually in AD as well
- ACLs associated with a specific object, such as a file

Complete!

- Assertion: Any policy can be expressed using ACLs
  - Probably correct
- Challenges:
  - Correct enforcement of ACLs
  - Efficient enforcement of ACLs
  - Updating ACLs
  - Correctly writing the policies/ACLs in the first place

Correct enforcement

- Strategy: All policies evaluated by a single function
- Implement the evaluation function once
  - Audit, test, audit, test until you are sure it looks ok
- Keep the job tractable by restricting the input types
- All policies, verbs, etc... expressed in common way
  - Single function must be able to understand
    - Shifts some work to application developer

Efficient enforcement

- Evaluating a single object's ACL is no big deal
- If context matters, work grows substantially
- Example: Linux VFS permission check
  - Starts at current directory (or common parent)
    … traverses each file in the tree
  - Why?
    - To check permissions that you should be allowed to find this file

Efficiency

- Container objects create hierarchy in Windows
- Trade-off
  - Check permissions from top-down on the entire hierarchy
  - Or propagate updates
- Linux: top-down traversal
- Alternative: chmod o-w /home/lu
    - Walk each file in /home/lu and also drop other's write permission

Efficiency, cont

- AD decided propagating updates was more efficient
- Intuition: Access checks more frequent than changes
  - Better to make the common case fast!

Harder than it looks

```
# ls /home/lu
drwxr-xr--x   lu lu 4096 lu
chmod o+r /home/lu/public
# chmod o-r lu
# ls /home/lu
drwxr-x---x   lu l
```

Recursively change all children to o-r.

But do you change public?

Issues with propagating

- Children distinguish explicit and inherited perms.
  - Ex 1: If I take away read permission to my home directory, distinguish those files with an explicit read permission from those just inheriting from the parent
  - Ex 2: If I want to prevent the administrator from reading a file, make sure the administrator can't countermand this by changing the ACL on /home

AD's propagation solution

- When an ACL is explicitly changed, mark it as such
  - vs. inherited permissions
- When propagating, delete&reapply inherited perms
  - Leave explicit ACLs alone

Example Policy

- “Don’t let this file leave the computer”
- Ideas?
  - Create restricted process that disables network access
  - Only give read permission to this context
- What if process writes contents to a new file?
  - Or over IPC to an unrestricted process?
  - Does the ACL propagate with all output?
    - What if program has legitimate need to access other data?

SELinux

MAC vs. DAC

- Unix/Linux default: Discretionary Access Control
  - User (subject) has discretion to set policies (or not)
  - Example: May 'chmod o+a' the file containing 506 grades
    - This violates university privacy policies
- Mandatory Access Control enforces a central policy
  - Example: MAC policies prohibit sharing 506 grades

SELinux

- Like Win2k ACLs, enforce privilege of least authority
  - No ‘root’ user
  - Several administrative roles with limited extra privileges
  - Example: Changing passwords does not require administrative access to printers
    - Principle of least authority says to give minimum privilege needed
  - Reasoning:
    - If ‘passwd’ is compromised (e.g., due to a buffer overflow)
      - Limit the scope of the damage to printers

SELinux

- Also like Win2k ACLs, 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,…
  - In MLS, only trusted declassifier can lower 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)

SELinux Policies

- Written by administrator in SELinux-specific language
  - Often written by expert at Red Hat and installed wholesale
  - Difficult to modify or write from scratch
- Very broad: covers all 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
    - grader role may not have network access, except to blackboard
    - web browsing role may not have access to my home directory files
    - admin role and web roles can't access students' labs

Roles vs. Restricted Context

- Win2k ACLs allow user to create processes with a subset of his/her privileges
- Roles provide the same functionality
  - also allow a user to add privileges, such as admin 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
  - Can’t be expressed in Unix permissions!

Roles vs. Specific Users

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

Type Enforcement

- Very much like fine-grained ACLs
- 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 any interface
  - setuid (legacy security, only admin can send RAW packets)
  - 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

```
type ping_t, domain, privlog;
type ping_exec_t, file_type, sysadmfile, exec_type;
role sysadm_r types ping_t;
role system_r types ping_t;
# Transition into this domain when you run this program.
domain_auto_trans(sysadm_t, ping_exec_t, ping_t)
domain_auto_trans(initrc_t, ping_exec_t, ping_t)
uses_shlib(ping_t)
can_network(ping_t)
general_domain_access(ping_t)
allow ping_t { etc_t resolv_conf_t }:file { getattr read };
allow ping_t self:unix_stream_socket create_socket_perms;
# Let ping create raw ICMP packets.
allow ping_t self:rawip_socket {create ioctl read write bind getopt setopt};
allow ping_t any_socket_t:rawip_socket sendto;
auditallow ping_t any_socket_t:rawip_socket sendto;
# Let ping receive ICMP replies.
allow ping_t { self icmp_socket_t }:rawip_socket recvfrom;
# Use capabilities.
allow ping_t self:capability { net_raw setuid };
# Access the terminal.
allow ping_t admin_tty_type:chr_file rw_file_perms;
ifdef(`gnome-pty-helper.te', `allow ping_t sysadm_gph_t:fd use;')
allow ping_t privfd:fd use;
dontaudit ping_t fs_t:filesystem getattr;
# it tries to access /var/run
dontaudit ping_t var_t:dir search;
```

Linux Security Modules

- Culturally, Linux devs care about writing good kernel
  - Not as much about security
  - Different specializations
- Their goal: Modularize security as much as possible
  - Security folks loadable modules if you care about security
    - kernel devs don’t have to worry about understanding security

Basic deal

- Linux Security Modules API:
  - Linux devs put 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 adds opaque "security" pointer that LSM can use
    - Store security info needed 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

- Security hooks work for in memory data structures
  - e.g., VFS inodes
- How to ensure policy persists across reboots?

Extended Attributes

- In addition to 9+ standard Unix attributes
  … associate small key/value store with 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