CS183

Instructor: Ali Davanian
(Slides were adopted from Brian Crites and Alireza Abdoli)

Access Control & Root

Outline

- Standard Unix Access Control
  - Root
  - Filesystem Access Control
  - Process Ownership
  - su and sudo
- Alternative Access Controls

Please take the survey

https://freeonlinesurveys.com/s/SAwMRvZp

Standard UNIX Access Control

- Access control decisions depend on which **user** is attempting to perform an operation (or possibly what **group** they belong to)
- Objects (files, processes, etc.) have owners and owners generally control their own objects
- You own the objects you create
- The special “root” user can act as the owner of any object (!)
- Only root can perform certain administrative operations

Root Account

- Can run any valid command on any file in the system (UID 0)
- Can also run privileged commands, such as:
  - Creating device files
  - Setting the system clock
  - Raising resource usage limits and process priorities
  - Setting the system’s hostname
  - Configuring network interfaces
  - Opening privileged network ports (< 1024)
  - Shutting down the system
- Root keeps no logs of its actions and has infinite power, so you should probably disable login to it whenever possible

Filesystem Access Control

- Files are only owned by a single person, but other users may have permissions to work with a file
- Permissions are set on a file on a user, group, and all users granularity

```
drwxr-xr-x   4 briancrites  staff  128 Mar 27 11:47 IO
```

- (1) special permissions character, (3) user characters, (3) group characters, (3) all users characters
- d: special bit (this one means its a directory)
- r: read, w: write, x: execute

Change Mode (chmod)

- Command used to modify the permissions of a file
- Allows you to set **u**ser, **g**roup, **o**ther, and **a**ll permissions to **r**ead, **w**rite, **e**xecute, or do nothing

```
chmod u=rwx,g=rx,o=r myfile

chmod +x myfile

chmod a-x myfile
```

Process Ownership

- Processes are owned by those who start them
- Process owners can send them signals and modify their priority (niceness)
- Every process has the following identities associated with them
  - Real UID/GID: used for UID/GID accounting when switching permissions
  - Effective UID/GID: used to determine access permissions (usually the same as real)
  - Saved UID/GID: parking spot allowing the process to switch in and out of privileged mode

setuid and setgid

- Special permissions (denoted by an `s` rather than `x` in permissions list)
- Allows for execution as another UID or GID (usually root, UID 0) rather than the user who is trying to invoke it
- Allows users to execute *some* functionalities that would normally require root permissions (changing passwords for example)
- This creates security problems, and only program that were specifically designed with setuid in mind (and a minimum number of those) should be allowed
- This functionality can be turned off by adding `nosuid` parameter when mounting a file system (`mount` command)

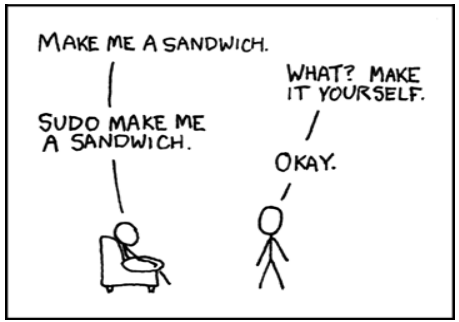
Problem

Assume that “find” command has the special permission (s) set by root:

- Can a malicious user exploit this to escalate their permission? If yes, how?

su: substitute user identity

- Allows you to switch user identifies (and therefore access and powers)
- Usually used to switch to root (although you can switch to anyone else)
- Users must supply passwords to accounts they switch to (including root)
- Root can switch to anyone else's account using su without a password (!)
- Records the account switch, but not commands run by root
- `su - username` allows you to switch to a user and use their bash profile, can be very useful for diagnosing user problems you don’t see as an admin



sudo: limited su (preferred method)

- Users need access to administrative permission levels to do normal tasks (change passwords, create backups, etc.)
- Giving them broad permissions creates security hazards
- Limit the number of commands that they have access to run, and try and choose as small a selection as possible for them to do their jobs
- Sudo allows you to dictate what commands that require permissions a user can run, rather than giving them broad permissions and asking them to relinquish them
- Standard in all distributions, but technically a stand alone product which can be downloaded from source here https://www.sudo.ws/sudo/sudo.html

sudoers File

- Describes who has access to what commands
- Allows for grouping of machines (Host_Alias) and grouping of commands (Cmnd_Alias) to make assigning permissions easier
- sudo commands are logged
- Use visudo to edit (checks for multiple editors, opens editor, and checks sudoers file is valid)

```
# Define aliases for machines in CS & Physics departments
Host_Alias  CS = tigger, anchor, piper, moet, sigi
Host_Alias  PHYSICS = eprince, pprince, icarus

# Define collections of commands
Cmnd_Alias  DUMP = /sbin/dump, /sbin/restore
Cmnd_Alias  WATCHDOG = /usr/local/bin/watchdog
Cmnd_Alias  SHELLS = /bin/sh, /bin/dash, /bin/bash
            User_Alias  ADMINS = alice, bob, charles
# Permissions
mark, ed    PHYSICS = ALL
herb        CS = /usr/sbin/tcpdump : PHYSICS = (operator) DUMP
lynda       ALL = (ALL) ALL, !SHELLS
%wheel      ALL, !PHYSICS = NOPASSWD: WATCHDOG


            ADMINS      ALL = (ALL) ALL
```

There is also or User_Alias command, can you guess how it can be used?

sudoers File

```
# Define aliases for machines in CS & Physics departments
Host_Alias  CS = tigger, anchor, piper, moet, sigi
Host_Alias  PHYSICS = eprince, pprince, icarus

# Define collections of commands
Cmnd_Alias  DUMP = /sbin/dump, /sbin/restore
Cmnd_Alias  WATCHDOG = /usr/local/bin/watchdog
Cmnd_Alias  SHELLS = /bin/sh, /bin/dash, /bin/bash

# Permissions
mark, ed     PHYSICS = ALL
herb         CS = /usr/sbin/tcpdump : PHYSICS = (operator) DUMP
lynda        ALL = (ALL) ALL, !SHELLS
%wheel       ALL, !PHYSICS = NOPASSWD: WATCHDOG
```

- Host_Alias creates groups of hosts which will allow us to break up who (users) can do what (commands) where (hosts)
- Cmnd_Alias creates groups of commands, usually related by functional need rather than user need
- Each permission line is made up of the following pieces
  - The user(s) to whom the line applies
  - The hosts on which the line applies
  - The commands the specified user can run
  - The users the commands should be executed as

Is there a way for Lynda to still run the shell?

sudoers File

```
# Define aliases for machines in CS & Physics departments
Host_Alias  CS = tigger, anchor, piper, moet, sigi
Host_Alias  PHYSICS = eprince, pprince, icarus

# Define collections of commands
Cmnd_Alias  DUMP = /sbin/dump, /sbin/restore
Cmnd_Alias  WATCHDOG = /usr/local/bin/watchdog
Cmnd_Alias  SHELLS = /bin/sh, /bin/dash, /bin/bash

# Permissions
mark, ed     PHYSICS = ALL
herb         CS = /usr/sbin/tcpdump : PHYSICS = (operator) DUMP
lynda        ALL = (ALL) ALL, !SHELLS
%wheel       ALL, !PHYSICS = NOPASSWD: WATCHDOG
```

- The user “herb” can run tcpdump on CS machines and dump-related commands on PHYSICS machines, but the dump commands can only be run as the “operator” user, not as root
- The user “lydia” can run all commands on all machines as any user, but isn’t allowed to run some common shells
- Note lydia can still gain shell access by copying a shell to another location and executing it, so trying to exclude behavior isn’t really possible using sudo

Additional sudoers File Tips & Tricks

- The default sudoers file is perfectly acceptable for most cases
- Sudo commands use a reduced and sanitized environment, you can add things to the default with:

```
Defaults env_keep += “ADDITIONAL_PATH”
```

- If there are multiple lines that could match, sudo will choose the last matching line
- You can use a single sudoers file to administer multiple hosts, either through configuration management tools or through a cron job which pulls the file from a (heavily watched) server

sudo pros and cons

- Accountability is much improved because of command logging
- Users can do specific chores without having unlimited root privileges
- The real root password can be known to only one or two people
- Using sudo is faster than using su or logging in as root
- Privileges can be revoked without the need to change the root password
- A canonical list of all users with root privileges is maintained
- The chance of a root shell being left unattended is lessened
- A single file can control access for an entire network
- Sudo command logging can be subverted (but the subversion is usually logged)
- A breach in a single account can be similar to a breach in root

Disabling root

- Root (or any other account) can be made inaccessible by changing its password to *
- This disallows logging into the account, but still allows sudo and root related systems to function (root still exists)
- Generally you want a root account on physical machines, as hardware or configuration problems may require root to rescue the machine
- It is also customary to disable system (UID < 10) and pseudo-system (10 - 100) accounts by replacing their passwords with *
- Use /etc/shadow or /etc/master.passwd file, and set their shells to /bin/false or /bin/nologin

sudo vs. advanced access control

- You decide exactly how privileges will be subdivided
- Simple configurations are easy to set up, maintain, and understand
- sudo runs on all UNIX/Linux systems
- You can use a single config across all machines
- You get default syslogging for free
- All user accounts are potentially exploitable to gain root access
- Not a safe way to define limited domains of autonomy
- Not a safe way to place certain operations out of bounds

Advanced Access Controls

- Pluggable Authentication Modules (PAM)
  - Module for allowing multiple types of authentication (not technically access control)
- Kerberos (Network Cryptographic Authentication)
  - Authentication service used in conjunction with PAM, a single server can provide authentication for an entire network of machines
- Access Control Lists (ACL)
  - Allows you to set permissions for multiple users and groups at once

Modern Access Control

- Mandatory Access Control (MAC)
  - Allows administrators to dictate access control which overrides user set permissions
- Role-based Access Control (RBAC)
  - Users belong to role(s) which form permission hierarchies
- SELinux: Security-Enhanced Linux
  - Security implementation combining flavors of MAC and RBAC
  - Becoming more popular in security intensive fields (government, healthcare, finance)
  - Fairly difficult to administer correctly

Questions?

Additional Resources

Sudoers File Official Documentation