CS 447/647

Access Control

A program is generally exponentially complicated by the number of notions that it invents for itself. To reduce this complication to a minimum, you have to make the number of notions zero or one, which are two numbers that can be raised to any power without disturbing this concept. Since you cannot achieve much with zero notions, it is my belief that you should base systems on a single notion.

The UNIX Command Language (1976)
https://github.com/susam/tucl

Access control not security

Standard Unix Access Control

- Access control decisions depend on the user
- Objects (e.g., files and processes) have owners
- You own the objects you create
- “root” can act as the owner of any object
- Only root can perform certain sensitive administrative operations.

Filesystem Access Control

- Each file has an owner and group
  - `chown user:group file`
  - `chmod 700 somedir`
    - `find . -type d -exec chmod 700 {} \;`
- Owner specifies what operations (syscalls) the group can perform (rwx)
  - `open(2)` - open and possibly create a file
    - `O_APPEND` - Append to file
    - `O_CREAT` - Create file if it doesn't exist
    - `int open(const char *pathname, int flags);`
      - EACCES - The requested access to the file is not allowed
  - `read(2)`
  - `write(2)`
  - `close(2)`

open(2)

```
zachn@DESKTOP-P1TE0QI:~$ ipython3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import posix

In [2]: fd = posix.open("foobar", posix.O_RDWR | posix.O_CREAT | posix.O_ASYNC)

In [3]: posix.write(fd,b"test")
Out[3]: 4

In [4]: posix.close(fd)

In [5]: # Read

In [6]: fd = posix.open("foobar",posix.O_RDONLY)

In [7]: buf = bytes(4)

In [8]: buf = posix.read(fd, 4)

In [9]: posix.close(fd)

In [10]: buf
Out[10]: b'test'
```

```
f = open("foo", "w")

f.write('Hello World')

f.close()
```

Process Ownership

- Processes have owners
  - You can look them up with the “ps” command
    - `ps aux | grep bash`
  - You can also check in /proc
    - `ls -la /proc`
- Owners can send signals to their processes
  - `kill(1)`, which calls `kill(2) - man 7 signal`
    - `kill -SIGTERM 2000 - posix.kill(2000, 15) #Python kill(2)`
    - `kill -SIGKILL 2000 - posix.kill(2000,  9) #Python kill(2)`
- Some signals can be used to interact
  - `SIGUSR1 {10}` - User defined signal
  - `SIGINT`

root account

- Superuser account
  - Can perform any file or process operation
  - Can execute restricted syscalls
    - `settimeofday(2)`
- Only root can
  - Create device files
  - Set the system clock
  - Set the system's hostname
  - Configure network interfaces
  - Open privileged network ports (those numbered below 1,024)
  - Shutting down the system
- Can change a processes UID and GID
  - `login` - Begins a session as a different user on the system

setuid and setguid

- Identity substitution
  - Allows marked executables to run with elevated permissions
    - typically root
  - setuid - Set User ID
    - passwd(1), su(1), login(1), ping(1), singularity(1)
    - Big security concern
      - nosuid filesystem option
  - setguid- Set Group ID

```
root@ncr-0:/usr# find . -user root -perm -4000
./local/libexec/singularity/bin/starter-suid
./sbin/exim4
./sbin/mount.nfs
./bin/rltraceroute6
./bin/ndisc6
./bin/chfn
./bin/gpasswd
./bin/ntfs-3g
./bin/passwd
./bin/umount
./bin/mount
./bin/pkexec
./bin/rdisc6
./bin/newgrp
./bin/chsh
./bin/fusermount
./bin/sudo
./bin/su
```

root management

- Assign root a password
  - Forbidden on Ubuntu
  - Bad practice - No logging
- Use the “su” command
  - Prompts for root’s password
  - Logs who su’d
  - Can be used by root to login as other users
    - su newellz2 -c "whoami"
- sudo
  - Superuser Do
  - Often the best solution

sudo

- Configured with the /etc/sudoers file
  - Use visudo to edit it
- Allows running commands as root and other users
  - sudo -U backup /usr/local/bin/backup.sh
- Interactive root session
  - sudo -i
  - sudo su -
- Logging!

```
sudo sh -c "cd /home ; du -s * | sort -rn > USAGE"
```

```
banyan sudo: newellz2 : TTY=pts/13 ; PWD=/home/newellz2 ;
USER=root ; COMMAND=/bin/sh -c cd /home ; du -s * | sort -rn >
USAGE
```

/etc/sudoers

```
Defaults        env_reset
Defaults        mail_badpass
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

USER\%GROUP HOSTS=(USER:GROUPS) COMMANDS

sudo advantages

- Accountability
- Limiting scope
- Revoke privileges without changing root’s password
- Canonical list of users with privileges
- Lessen the likelihood of unattended root shell
- Single file for multiple hosts
- Lock the root account
  - `passwd -l root`

sudo disadvantages

- Breach of a user account equivalent to breach of root
- Logging can be subverted with shell escapes
- Flat file management breaks down with
  - Large number of hosts
  - Large number of users

News

- A major vulnerability in sudo
  - https://lwn.net/Articles/844789/
  - buffer overflow
- Inside the utility since 2011
- Test with
  - sudoedit -s '\' `perl -e 'print "A" x 65536'`
- C programming error leading to a complete compromise of a setuid-root program

Advanced Access Control - Capabilities

- Divides the privileges traditionally associated with superuser into distinct units
  - CAP_KILL - Has the abilities to kill a process
  - CAP_FOWNER - Filesystem owner permissions
  - CAP_NET_ADMIN - Network administration permissions
  - CAP_NET_BIND_SERVICE - Bind to ports < 1024
  - CAP_REBOOT - Reboot the system
  - CAP_SYS_ADMIN - Sweeping root privileges (mount, unmount, create namespaces, etc.)

```
[Service]
Type=forking
PIDFile=/run/nginx.pid
ExecStartPre=/usr/sbin/nginx -t -q -g 'daemon on; master_process on;'
ExecStart=/usr/sbin/nginx -g 'daemon on; master_process on;'
ExecReload=/usr/sbin/nginx -g 'daemon on; master_process on;' -s reload
ExecStop=-/sbin/start-stop-daemon --quiet --stop --retry QUIT/5 --pidfile /run/nginx.pid
Restart=always
RestartSec=5
TimeoutStopSec=5
KillMode=mixed
CapabilityBoundingSet=~CAP_KILL
```

Advanced Access Control - Namespaces

- Segregate processes into hierarchical partitions
  - `clone(2)` system call
  - Files (mount+chroot)
  - Networking
  - PIDs
  - UIDs
  - Mount
- Foundation of containers!

```
ip netns add test
ip netns exec test ip addr #Nothing
```

What is SELinux?

- Security-Enhanced Linux (SELinux): A mandatory access control (MAC) mechanism for Linux.
  - DAC - Discretionary Access Control (traditional UNIX Access Control)
  - MAC - Mandatory Access Control (removes discretionary powers of file owners in favor of centralized policies)
- Developed by the NSA, integrated into the Linux kernel.
- Policies define how processes and resources interact, enforced at the kernel level.

DAC vs MAC

- Access control decisions are made by the resource owner
- Users can modify permissions of their own files
- Relies on traditional Unix file permissions (owner, group, others) to control access
- Root has ultimate control

- Access control is enforced by the system based on security policies.
- Users cannot override system-enforced security policies.
- Security policies are enforced regardless of user ownership to control access
- Root is also subject to policies

MAC Benefits and Drawbacks

- Improving security
- Improving scalability of multi-user systems
  - Keeping user processes isolated from each other
  - Prevents unauthorized resource access
- Compliance requirements
  - Some regulatory frameworks, such as PCI-DSS (Payment Card Industry Data Security Standard) require strict access controls that are much easier to implement with MAC

- Configuration complexity
  - Policies can be time-consuming to configure correctly
  - Improper configurations can easily lock you out of the system
- Some applications require custom security policies
  - Default behavior of some applications may make them try to read/write to directories blocked by MAC
  - Some legacy (pre-SELinux) applications run with the assumption that they can access any file as root