(Advanced) Computer Security!

Privileged Programs & Program Inputs:

The Set-UID Mechanism & Environment Variables

(part I)

Prof. Travis Peters

Montana State University

CS 476/594 - Computer Security

Spring 2021

https://www.traviswpeters.com/cs476

Today

Reminder!
Please update your Slack, GitHub, Zoom
(first/last name, professional photo/background)

- Announcements
  - Lab 00 recap (look @ a few exemplary README.md files)
  - Lab 01 released

- Learning Objectives
  - Review fundamental ideas in Linux security + related Linux commands
  - Understand the need for privileged programs
  - Understand how the Set-UID mechanism works
  - Examine attack surface of (Set-UID) programs
  - Examine the role of environment variables in shells and Set-UID programs

Some slides and figures adapted from Wenliang (Kevin) Du's
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

How would you protect your computer & its resources? What sorts of things should we consider?

Think. Pair (Break Out Rooms). Share.

Encrypt important data - at rest (secret)

VPNs - data in transit

Back-ups (integrity) loss of data → ransomware

Checksums (crypto hashes)

antivirus sw (active scanning)

passwords / password hygene

plugins / sw block ads (adware) (other things)

Cookies

2-factor auth (multi-factor)

(Public key Crypto)

Access Control

Access Control:

How would you protect your computer & its resources?

who can do **what** to **whom**

users/groups

Need notions of "identity"

objects

Usually *things* on a *filesystem*

File systems

permissions (read/write/execute)

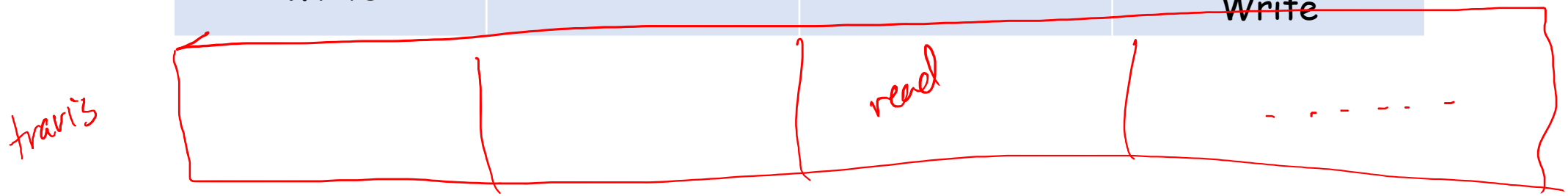
OK, I know the who/whom—what are you permitted to do?

Access Control:

Access Control Matrix

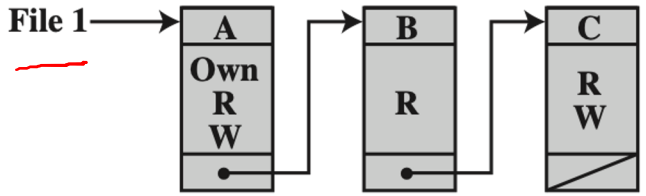
Objects

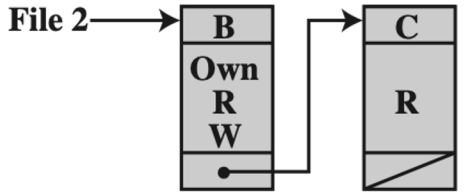
Subjects	File 1	File 2	File 3	File 4
User A	Own Read Write		Own Read Write	
User B	Read	Own Read Write	Write	Read
User C	Read Write	Read		Own Read Write

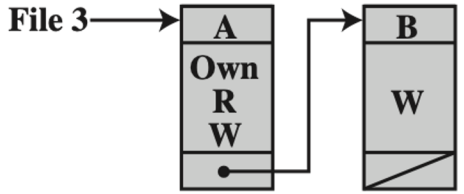


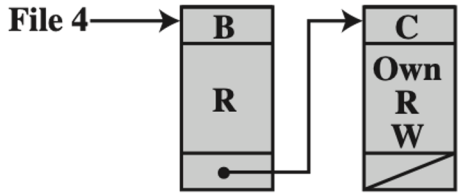
Access Control:

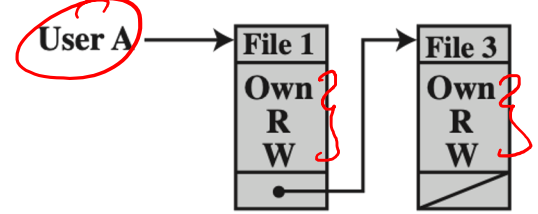
Access Control List (ACL)

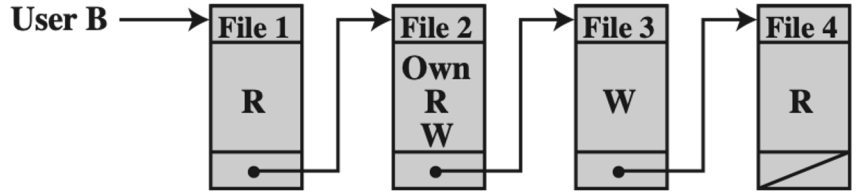


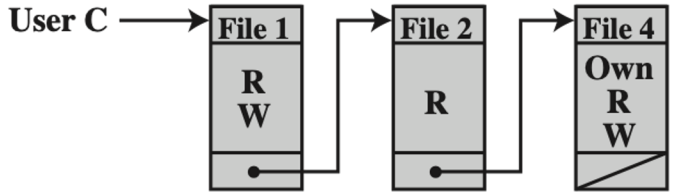






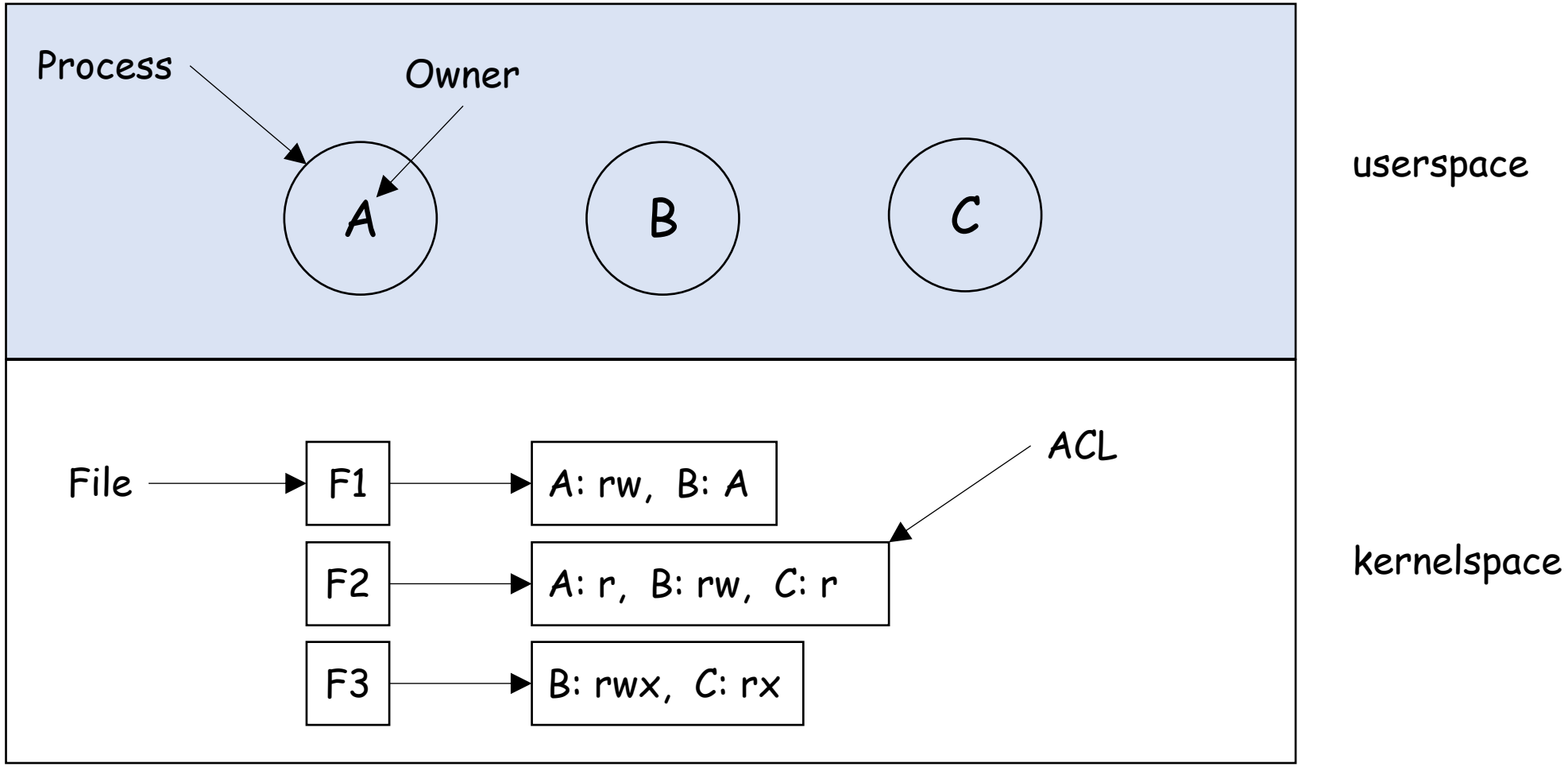






Access Control:

A Unix ACL Example



Access Control:

Unix File Modes & Permissions

- Every Unix file has a set of **permissions** that determine whether someone* can read, write, or run the file.
  - Try: `ls -l ~`
  - Try: `ls -l /dev`

Access Control:

Unix File Modes & Permissions

- Every Unix file has a set of **permissions** that determine whether someone* can read, write, or run the file.
  - Try: ls -l ~
  - Try: ls -l /dev

Every file has...

```
$ ls -l file
-rw-r--r-- owner group date/time file
```

- File mode (4 parts)
  -> [file type][user][group][other]
  - file type
  - ugo -> rwx

- How do we change these settings?
  - Ex: allow other members of my **group** to **write** "file"?

A typical who can do what to whom flow

If **user A** asks to perform **operation O** on a **file object F**, the OS checks:
e.g., A tries to read F, where:

```
$ ls -l F
-rw-rw-r--  B  G  ...  F
```

- Is **A** the owner of **F**?
  - use **owner permissions** to decide whether A can do operation O.
  - e.g., A is not F's owner
- Is **A** a member of **F's** group?
  - use **group permissions** to decide…
  - e.g., A is not F's owner or a member of F's group, G
- Otherwise…
  - use **other permissions** (i.e., "everyone else") to decide…
  - e.g., A can (r)ead the file

You Try!

Think. Pair (Break Out Rooms). Share.

Suppose **user C** asks to **execute** a **file object F2**. Will they be able to do so?

```
$ ls -l F
-rwxrwxrwx   B H   ...   F1
-rwxr-xr--   D G   ...   F2
-rw-r-----   D H   ...   F3
-rw-rw-rw-   B G   ...   F4
```

Note:

- Group = G = {A, C, K, M, Q, Z}
- Group = H = {A, B, C, Q}

Demos/Activities:
Basic File Ops, Users, and Groups

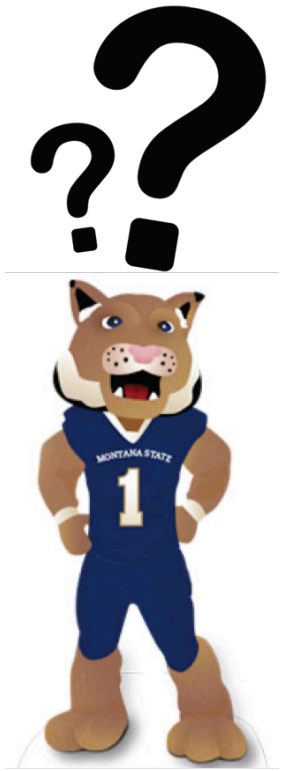
Examples and commands to try can be found here:

https://github.com/traviswpeters/cs476-code/tree/master/00_intro (see README.md)

Limitations of File-Based Access Control

Question: How can a non-privileged user 'champ' change their own password?

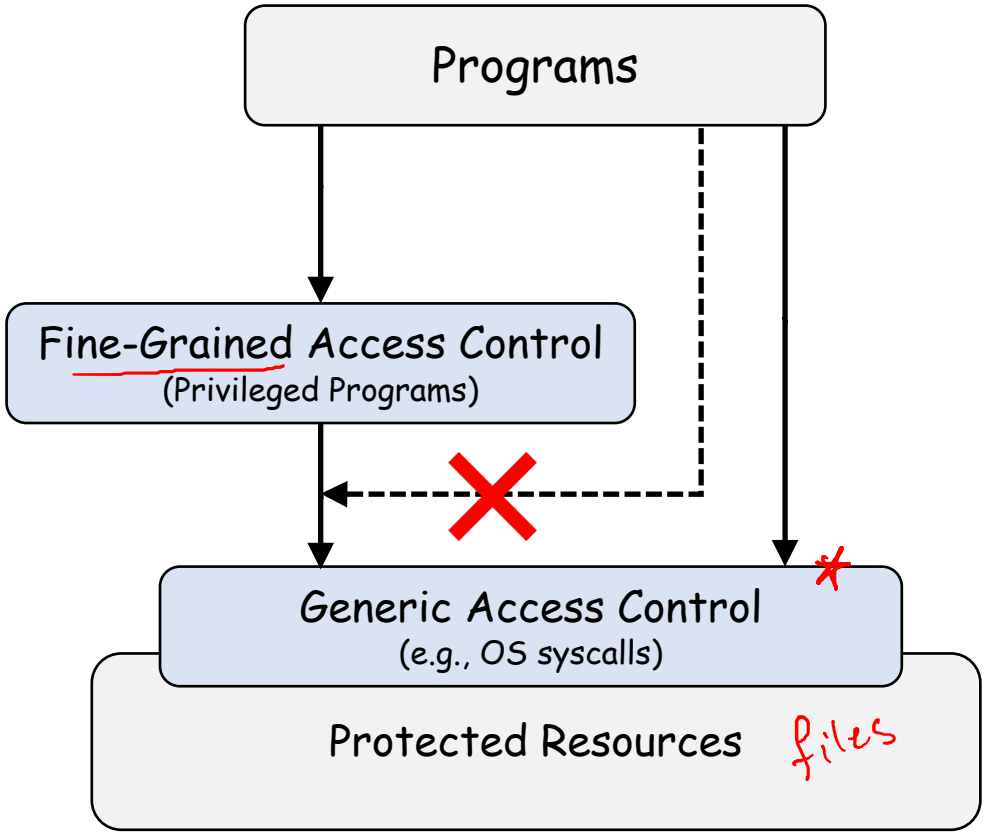
```
[seed@VM][~]$ ls -al /etc/passwd
-rw-r--r-- 1 root root 2886 Nov 24 09:12 /etc/passwd

[seed@VM][~]$ ls -al /etc/shadow
-rw-r----- 1 root shadow 1514 Nov 24 09:12 /etc/shadow
```



Ideas?

Two-Tier Approach to Access Control

- Implementing fine-grained access control in the OS makes OS code really complicated… (we generally try to avoid this…)
- OS relies on other extensions/features to enforce fine-grained access control
  -> "Privileged programs"



Types of Privileged Programs

- **Daemons**
  - Computer program that runs in the background
  - Needs to run as root or other privileged users

- **Set-UID Programs**
  - Widely used in UNIX systems
  - A normal program… but marked with a special bit

Superman's Past (The Stories You Never Heard…)

- **Superman's 1st Attempt—The Power Suit**
  - **Superman** got tired of saving *everyone*
  - **Super*people:*** give normal people superman's power!
  - **Problem:** not all superpeople are good……

Superman's Past (The Stories You Never Heard…)

- **Superman's 2nd Attempt—Power Suit 2.0**
  - Power suit w/ a sweet computer in it
  - Power suit can only perform a specific task
  - No way to deviate from the pre-programmed task……

The Set-UID mechanism is a lot like superman's power suit 2.0
(but implemented in UNIX systems…)

Set-UID In A Nutshell

There is also a Set-GID (Set Group ID), which works in basically the same way, but applies to group privileges

- Allow user to run a program with the program ***owner's*** privilege
  - i.e., a UNIX mechanism for changing user/group identity
  - Allows users to run programs w/ temporarily elevated privileges

- Created to deal with inflexibilities of UNIX access control
  - *Why might this be useful?*
  - *Why might this be a bad idea?*

- Example: the **passwd** program

```
[seed@VM][~]$ ls -al /usr/bin/passwd
-rwsr-xr-x 1 root root 68208 May 28  2020 /usr/bin/passwd
```

Set-UID In A Nutshell (cont.)

There is also a Set-GID (Set Group ID), which works in basically the same way, but applies to group privileges

- Every process has two User IDs
  - **Real UID (RUID)** — identifies the **owner** of the process
  - **Effective UID (EUID)** — identifies **current privilege** of the process
  - Access control decisions are based on EUID!

- When a normal program is executed,
  - RUID == EUID
    EUID == RUID == user who *runs* the program

- When a Set-UID program is executed,
  - RUID != EUID
    EUID == ID of program's owner

If program owner == root, the program runs with root privileges

Demos / Activities

If we have time… else look at some slides

So Uh… How Do You Set… The Set-UID Bit Thingy

Change the owner of a file to root

```
[seed@VM][~]$ cp /bin/cat ./mycat
[seed@VM][~]$ sudo chown root mycat
[seed@VM][~]$ ls -al mycat
-rwxr-xr-x 1 root seed 43416 Jan 25 21:15 mycat
```

Before enabling the Set-UID bit

```
[seed@VM][~]$ mycat /etc/shadow
mycat: /etc/shadow: Permission denied
```

After enabling the Set-UID bit

```
[seed@VM][~]$ sudo chmod 4755 mycat
[seed@VM][~]$ ls -al mycat
-rwsr-xr-x 1 root seed 43416 Jan 25 21:15 mycat
[seed@VM][~]$ mycat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
...
```

How It Works

Try It!

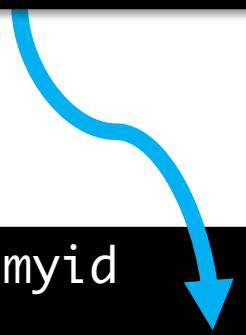
```
[seed@VM][~]$ cp /usr/bin/id ./myid
[seed@VM][~]$ sudo chown root myid
[seed@VM][~]$ ./myid
???
```

```
[seed@VM][~]$ sudo chmod 4755 myid
[seed@VM][~]$ ./myid
???
```

How It Works

A Set-UID program is just like any other program,
except that it has a special bit set
(the Set-UID bit)

```
[seed@VM][~]$ cp /usr/bin/id ./myid
[seed@VM][~]$ sudo chown root myid
[seed@VM][~]$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),...
```



```
[seed@VM][~]$ sudo chmod 4755 myid
[seed@VM][~]$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom)
```