LINUX SECURITY BASICS

CS-5156/CS-6056: SECURITY VULNERABILITY ASSESSMENT (SPRING 2025)
LECTURE 4

Environment Setup

- Install SEED Ubuntu 20.04 VM
  - Installation Manual:
    - https://github.com/seed-labs/seed-labs/blob/master/manuals/vm/seedvm-manual.md
  - VM Image: https://seed.nyc3.cdn.digitaloceanspaces.com/SEED-Ubuntu20.04.zip
  - Username: seed, password: dees
- This VM can be used for
  - Practicing the example illustrations in this lecture
  - Completing the upcoming lab assignment
    - https://seedsecuritylabs.org/Labs_20.04/Software/Environment_Variable_and_SetUID/

Outline

- Users and groups
- Permissions and access control
- Running commands with privilege
- Authentication

USER AND GROUP

Users

- In Linux, each user is assigned a unique user ID
- User ID is stored in /etc/password

```
root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
```

- Find user ID

```
seed@VM:~$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed)

root@VM:~# id
uid=0(root) gid=0(root) groups=0(root)
```

Add Users & Switch to Other Users

- Add users
  - Directly add to /etc/password
  - Use “adduser” command

- Switch to another user

```
seed@VM:~$ su bob
Password:
bob@VM:/home/seed$
```

Group

- Represent a group of users
- Assigning permissions based on group
- A user can belong to multiple groups
- A user's primary group is in /etc/password

```
root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
bob:x:1001:1001:Bob,,,:/home/bob:/bin/bash
alice:x:1002:1003:Alice,,,:/home/alice:/bin/bash
```

Primary Group ID

Which Group Does a User Belong To?

```
seed@VM:~$ grep seed /etc/group
adm:x:4:syslog,seed
sudo:x:27:seed
plugdev:x:46:seed
lpadmin:x:120:seed
lxd:x:131:seed
seed:x:1000:
docker:x:136:seed
```

```
seed@VM:~$ groups
seed adm sudo plugdev lpadmin lxd docker
```

```
seed@VM:~$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),27(sudo),
46(plugdev),120(lpadmin),131(lxd),136(docker)
```

Group Management

How to add users

```
$ sudo groupadd alpha              # create a group alpha
$ sudo usermod -a -G alpha seed    # add seed to alpha
$ sudo usermod -a -G alpha bob     # add bob to alpha
```

PERMISSIONS AND ACCESS CONTROL

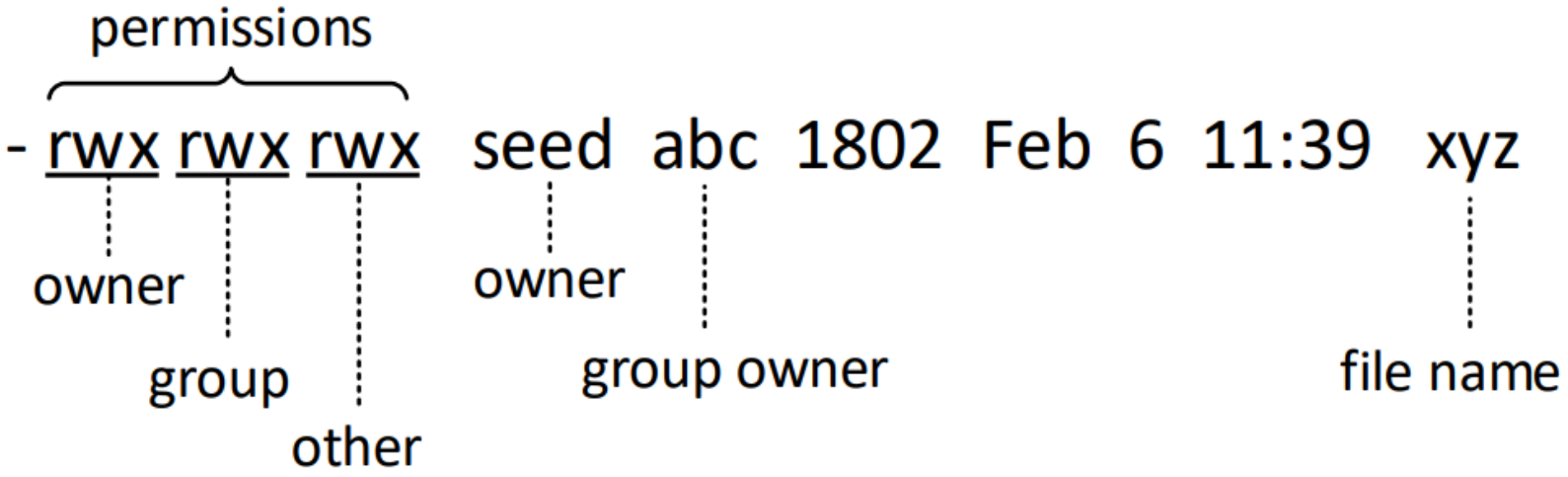
Traditional Permission Model

- An example of **DAC** (**D**iscretionary **A**ccess **C**ontrol)
  - Resources available for use at the discretion of the **user**.
  - Every object has an **owner** (creator)
  - “Owner” decides who accesses object

Traditional Permission Model

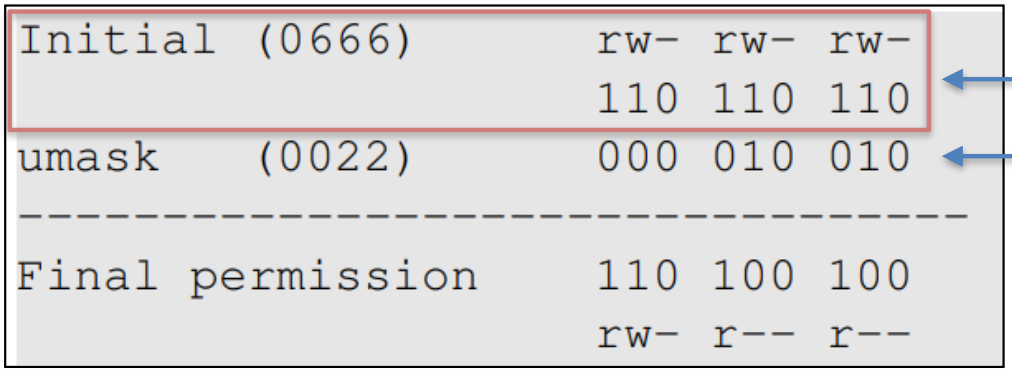
- Types of access on files
  - **read (r)**: user can view the contents of the file
  - **write (w)**: user can change the contents of the file
  - **execute (x)**: user can execute or run the file if it is a program or script
- Types of access on directories
  - **read (r)**: user can list the contents of the directory (e.g., using ls)
  - **write (w)**: user can create files and sub-directories inside the directory
  - **execute (x)**: user can enter that directory (e.g., using cd)

File Permissions



Default File Permissions

- umask value: decides the **default permissions** for **new files**
- Example



Allowing read and write for all users by default

Drop write access for users in group/other

Default umask = 0002 (only dropping write access to users in other)

- Final permission = initial **xor** umask

Examples (umask)

```
$ umask
0002
$ touch t1

$ umask 0022
$ touch t2
$ umask 0777
$ touch t3

$ ls -l t*
-rw-rw-r-- 1 seed seed 0 Feb  6 16:23 t1
-rw-r--r-- 1 seed seed 0 Feb  6 16:24 t2
---------- 1 seed seed 0 Feb  6 16:24 t3
```

Access Control List

- Fine grained ACL
- Assign permissions to individual users/groups
- Coexist with the traditional permission model
- Another example of **DAC**
  - Defines
    - Which users can access resource/object?
    - What privileges (read, write, etc.) authorized users have?

Access Control List

- Example

```
$ getfacl example
# file: example
# owner: seed
# group: seed
user::rw-
group::rw-
other::r--
```

ACL Commands

```
setfacl {-m, -x}  {u, g}:<name>:[r, w, x] <file, directory>
```

```
$ setfacl -m u:alice:r-- example
$ setfacl -m g:faculty:rw- example
$ getfacl example
# file: example
# owner: seed
# group: seed
user::rw-
user:alice:r--
group::rw-
group:faculty:rw-
mask::rw-                    ①
other::r--
```

```
-rw-rw-r--+  1 seed seed  1050 Feb  7 10:57 example
          ↖ indicating that ACLs are defined
```

RUNNING COMMAND WITH PRIVILEGE

Why

- Three command mechanisms
  - sudo
  - Set-uid programs (covered in next lecture)
  - POSIX capabilities

Using sudo

- sudo: Super-user Do
- Run commands as a superuser
- A user must be authorized (/etc/sudoers)
- Here is how the seed user is allowed to run sudo

```
In /etc/sudoers
%sudo ALL=(ALL:ALL) ALL
     ↖ group name

In /etc/group
sudo:x:27:seed  ← the sudo group
```

Getting Root Shell

- In Ubuntu 20.04, the root user account is locked
- Cannot log into the root account
- There are many ways to get a root shell
  - sudo –s
  - sudo bash
  - sudo su
- It is not recommended to run commands using a root shell. Instead, use sudo to run individual commands.

Running Command Using Another User

- Run command using another user (instead of root, default)

```
$ sudo -u bob id
uid=1001(bob) gid=1001(bob) groups=1001(bob),1004(alpha)
```

POSIX Capabilities

- Divide the root privilege into smaller privilege units
  - Known as **capabilities**
- An example of **C**apability **B**ased **A**ccess **C**ontrol
  - Uses non-forgeable, communicable key / token / ticket
  - Key embodies access rights
    - Whoever (user, process, etc) has the key has access to the object
- In Linux, processes (not users) have capabilities
  - Process or executable runs in a certain user context and holds certain permissions to perform the privileged operations guarded by Linux kernel

POSIX Capabilities

- Use “man capabilities” to find all the capabilities
- Examples

```
CAP_CHOWN:         Make arbitrary changes to file UIDs and GIDs.
CAP_DAC_OVERRIDE: Bypass file read/write/execute permission checks.
CAP_DAC_READ_SEARCH: Bypass file read permission checks ...
CAP_NET_RAW:       Use RAW and PACKET sockets ...
```

Setting File Capabilities (1)

- Before

```
$ cp /bin/bash ./mybash
$ ./mybash
$ cat < /etc/shadow
mybash: /etc/shadow: Permission denied     ← Failed
```

  - The redirect standard input symbol < tells the shell/bash that you want a file to be read as input for a command ‘here cat’

Setting File Capabilities (1)

- Setting the capabilities

```
$ sudo setcap CAP_DAC_READ_SEARCH=ep mybash
$ ./mybash
$ getpcaps $$     # List the capability of the current process
65331: = cap_dac_read_search+ep  ← The process has the capability
```

  - sudo setcap CAP_DAC_**READ**_SEARCH=**ep** mybash
    - Bypass file **read** permission checks for process **mybash**
    - **e** = **effective**
    - **p** = **permitted**
    - **More info found in the man pages**

Setting File Capabilities (2)

- After

```
$ sudo setcap CAP_DAC_READ_SEARCH=ep mybash
$ ./mybash
```

```
$ cat < /etc/shadow      # Bash will open this file for read
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
sys:*:18474:0:99999:7:::
...

$ cat > /zzzz            # Bash will open this file for write
mybash: /zzzz: Permission denied
```

Case Study 1: Wireshark

- Wireshark
  - Sniffing tool, needs privilege
  - The graphic part is not privileged
  - The **sniffing part** is done by **dumpcap** process/program, and is **privileged**

```
$ getcap /usr/bin/dumpcap
/usr/bin/dumpcap = cap_net_admin,cap_net_raw+eip
```

  - **i = inherited**

Case Study 2: ping

- The ping program
  - Uses raw socket
  - Has the CAP_NET_RAW capability

```
$ getcap /usr/bin/ping
/usr/bin/ping = cap_net_raw+ep
```

AUTHENTICATION

Authentication Methods

- Verifying a user's identity
- Typical authentication methods
  - based on something the **user knows**: password
  - based on something the **user has**: ID card
  - based on something the **user is or does**: fingerprint
- Multi-factor authentication

The Password File (/etc/passwd)

- Each entry contains a user account information
- Password is not stored here (**used to be**)

```
root:x:0:0:root:/root:/bin/bash
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
bob:x:1001:1001:Bob,,,:/home/bob:/bin/bash
alice:x:1002:1003:Alice,,,:/home/alice:/bin/bash
```

First Command After Login

- The last field of each entry

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
bob:x:1001:1001:Bob,,,:/home/bob:/bin/bash
alice:x:1002:1003:Alice,,,:/home/alice:/bin/bash
```

```
$ sudo su bin
This account is currently not available.
```

The Shadow File (/etc/shadow)

- Store password, why not use /etc/passwd anymore?
  - /etc/passwd is readable by **others** (i.e., world-readable)
    - Necessary for any process to, for instance, determine user id based on username

```
bob@VM:/home/seed$ ls -l /etc/passwd
-rw-r--r-- 1 root root 2926 Jan 22 22:01 /etc/passwd
bob@VM:/home/seed$ ls -l /etc/shadow
-rw-r----- 1 root shadow 1644 Jan 22 22:01 /etc/shadow
```

    - Hashed passwords were harder to crack in the past, not anymore

The Shadow File (/etc/shadow)

- Structure for each entry

- Complete breakdown of each entry
  - https://www.cyberciti.biz/faq/understanding-etcshadow-file/

The Shadow File (/etc/shadow)

- Hashing Algorithms
  1. **`$1$`** is MD5
  2. **`$2a$`** is Blowfish
  3. **`$2y$`** is Blowfish
  4. **`$5$`** is SHA-256
  5. **`$6$`** is **SHA-512**
- You can manually create password hashes using mkpasswd command
  - mkpasswd -m md5 password123 12345678 # 12345678 is salt
  - mkpasswd -m md5 password123 # uses random salt

The Purpose of Salt

- Users can choose same password. Easier to crack if no salt
  - These 3 accounts have the same password

```
seed:$6$n8DimvsbIgU0OxbD$YZ0h1EA...(omitted)...wFd0:18590:0:
alice:$6$.1CMCeSFZd8/8QZl$QhfhId...(omitted)...Sga.:18664:0:
bob:$6$NOLhqomO3yNwyFsZ$K.Ql/KnP...(omitted)...b8v.:18664:0:
```

- Defeat brute-force attacks
  - dictionary attack, **rainbow table attack**

The Purpose of Salt

- Rainbow table attack
  - Rainbow tables are huge files that store existing hashes (no salt) to speed up password cracking. Salting helps prevent such speedy process
  - Stores chain of hashes to reduce space
  - Example Password Cracker: **John the Ripper**
    - On ubuntu: sudo apt update; sudo apt install john

Locking Account

- Putting an invalid value in the password field (e.g., !)
- The root account is locked

```
root:!:18590:0:99999:7:::
```