Privilege Escalation

Introduction

- Applications provide services
- Locally (E.g. Word processing, file management)
- Remotely (network service implementation)
- Behavior of an application is determined by
- Code being executed
- Data being processed
- Environment in which application is run
- Attacks against application aims at bringing applications to execute operations that violate security of a system
- Violation of Integrity
- Violation of confidentiality
- Violation of availability

Application Vulnerability Analysis

- Process of identifying vulnerabilities in applications as deployed in a specific operational environment
- Deployment vulnerabilities
- Introduced by a faulty deployment/configuration of the application
- E.g. An application is installed with more privileges than it should have
- E.g. An application is installed on a system with faulty security policy – for instance a file that is read only has write privileges
- Implementation vulnerabilities
- Were introduced because the application is not able to correctly handle some events
- Unexpected input / Poorly formatted input
- Unexpected errors/exceptions
- Unexpected interleaving of events

Local & Remote Attacks

- Local Attack

- Allows one to manipulate behavior of an application through local interaction

- Requires a previously-established presence on the host (e.g. a user account or another application under the control of the attacker)

- Allows one to execute operations with privileges (usually higher) than what the attacker has.

- Are easier to perform since the attacker has better knowledge of the environment.

Remote Attacks

- Allows one to manipulate behavior of an application through network-based interaction.
- Allows one to execute operations with privileges of vulnerable application.
- Are more difficult to perform but are more powerful, as they don’t require prior access to system

Principle of Least Privilege

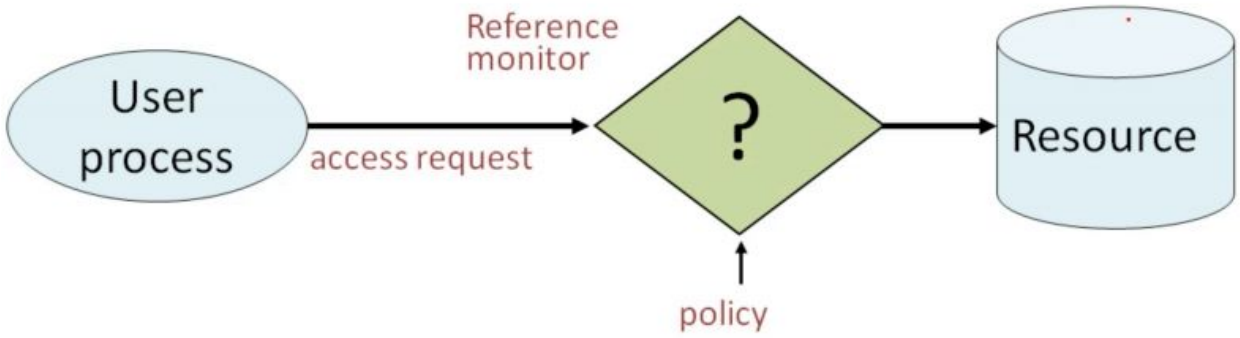
- Privilege
  - Ability to access or modify a resource
- Principle of Least Privilege
  - A system module should only have the minimal privileges needed for intended purposes.
- Requires compartmentalization and isolation
  - Separate the system into independent modules
  - Limit interaction between modules

Privileged Programs

- One that gives users extra privileges than beyond that are already assigned to them
- E.g. Web server allows remote users to access server-side resources
- Set-Root-UID programs allows users to gain access to root privilege during execution of programs

Review: Access Control

- Assumptions
- System knows who the user is
- Authentication via name and password, other credential
- Access requests pass through gatekeeper (reference monitor)
- System must not allow monitor to be bypassed



UNIX File Access Control

- Each process has a user ID based on the user that started the process or inherited from the creating process
- A process's access to file controlled by ACLs
  - Specified permission for owner, group and others
- Only "owner" and "root" can change permissions.
  - Privilege cannot be delegated or shared.
- Privilege management is very coarse grained
  - Root can do anything.
  - Many programs run as root even though they only

need to perform a small number of operations.

- What is the problem?
- Privileged programs are juicy target for attackers.
- By finding bugs in parts of programs that do not need privilege, attackers can gain root privilege

UNIX File Access Control

What can we do?

- Drop privilege as soon as possible.
- Example: A network daemon only needs privilege to bind to low port (#1024) at the beginning.
- Drop privilege after binding the port.
- Advantage:
- Attacker only has a small window to exploit.

- How to drop privilege?
- Setuid programming in UNIX

Permissions

The first item specifies the file type to one of the following:

• d – directory

• - (dash) – a regular file (rather than link or directory)

• l – a symbolic link to another program or file elsewhere on the system

Beyond the first item

• r – file can be read

• w – file can be written to

• x – file can be executed

• - (dash) – specific permission has not been assigned

Permissions

The file's owner (in this case, user) has permission to read and write to the file. The group, user, has permission to read and write to foo.txt, as well. It is not a program, so neither the owner or the group has permission to execute it.

- Standard permissions read, write, execute
- Additional permission
- Set UID, Set GID, Sticky Bit
- 12 bits to set permissions for each file/directory
- 2^12=4096 combinations

QUIZ

To run the *‘cat’* command, does the command cat require permission to the file or is it that the read permissions on the file has to be set??

Setting Permissions

- *chmod* command
- `$ touch foo.txt`
- `$ ls -l foo.txt -rw-r--r-- 1 alice staff 0 Mar 15 09:27 foo.txt`

User has rw permission, group and other users have only read permission

```
$chmod g+w foo.txt $ ls -l foo.txt
-rw-rw-r-- 1 alice staff 0 Mar 15 09:27
foo.txt
```

Setting Permissions

- Removing all permissions

```
$ chmod a-rwx foo.txt

$ ls -l foo.txt

---------- 1 alice staff 0 Mar 15 09:27 foo.txt

$ cat foo.txt cat: foo.txt: Permission denied
```

Identities

- u — the user who owns the file (that is, the owner)
- g — the group to which the user belongs
- o — others (not the owner or the owner's group)
- a — everyone or all (same effect as specifiying **ugo** flags together). If no flags are specified default is a.

Setting Permissions

Permissions

• r/w/x— read, write, execute access

• X — set executable bit only if

• The target is a directory

• Has already atleast one bit set for any of u/g/o

• s — set-user-ID-on-execution permission if **u** flag is specified or implied. Similarly for **g** as well. Not allowed for **o. Implies that x is set**

• S — same as **s,** but x is not set (typically used for directories)

• t — For directories, indicates that only file owners can link/ unlink files specified in that dir. Can only be found in

the **others** triad and implies **x** is set

Setting Permissions

Difference between x and X

$ cd test

$ ls -lR

total 0 -rw-r--r-- 1 renuka staff 0 Mar 15 10:47 file1
drwxr-xr-x 3 renuka staff 96 Mar 15 10:45 test1

./test1:

total 0 -rw-r--r-- 1 renuka staff 0 Mar 15 10:45 file2

$ cd ..

$ chmod -R g+rwx test

Setting Permissions

Difference between x and X

$ ls -lR

total 0

-rw-rwxr-- 1 alice staff 0 Mar 15 10:47 file1

drwxrwxr-x 3 alice staff 96 Mar 15 10:45 test1

./test1:

total 0 -rw-rwxr-- 1 alice staff 0 Mar 15 10:45 file2

Modifies the permission of the files in the directory as well

Setting Permissions

Difference between x and X

$ chmod -R g+rX test1/

$ ls -lR

total 0

-rw-r--r-- 1 alice staff 0 Mar 15 10:53 file1

drwxr-xr-x 3 alice staff 96 Mar 15 10:53 test1

./test1:

total 0

-rw-r--r-- 1 alice staff 0 Mar 15 10:53 file2

Setting Permissions

Difference between x and X

Allows user to make directories executable without changing the permission on files

Setting Permissions

Permissions

- r/w/x— read, write, execute access
- X — set executable bit only if
- The target is a directory
- Has already atleast one bit set for any of u/g/o
- s — set-user-ID-on-execution permission if **u** flag is specified or implied. Similarly for **g** as well. Not allowed for **o. Implies that x is set**
- S — same as **s,** but x is not set (typically used for directories)
- t — For directories, indicates that only file owners can link/ unlink files specified in that dir. Can only be found in the **others** triad and implies **x** is set

Setting Permissions

Actions

- + — adds the permission
- - — removes the permission
- = — Clears the selected permission field and sets it to permission specified. If no permission specified then, chmod removes all permissions from the selected field

$ ls –al

-rwxr-xr-x 1 alice staff 0 Mar 18 16:36 file1

drwxr--r-T 3 alice staff 96 Mar 15 10:53 test1

$ chmod =ur file1

$ ls –al

-r--r--r-- 1 alice staff 0 Mar 18 16:36 file1

drwxr--r-T 3 renuka staff 96 Mar 15 10:53 test1

Common Examples

Here are some common examples of settings with chmod:

- g+w — adds write access for the group
- o-rwx — removes all permissions for others
- u+x — allows the file owner to execute the file
- a+rw — allows everyone to read and write to the file
- ug+r — allows the owner and group to read the file
- g=rx — allows only the group to read and execute (not write)

What does ‘rwx’ on Directory Mean?

- r — allows you to list the contents of the directory
- w — allows you to create, delete and rename contents of a directory
- x — allows you to traverse through the contents of a directory

$ ls –al

drwxr-xr-x 2 alice staff 64 Mar 18 16:49 testdir

$ chmod -rwx testdir/

$ ls –al

d--------- 2 alice staff 64 Mar 18 16:49 testdir

What does ‘rwx’ on Directory Mean?

```
$ ls testdir/

ls: : Permission denied

$ chmod +r testdir/

$ ls testdir/        <r allows to read the dir> $
cd testdir/

-bash: cd: testdir/: Permission denied

$ chmod +x testdir/

$ cd testdir/       <x allows to traverse into
the dir>

$ touch file
```

```
touch: file: Permission denied

$ cd ..

$ chmod +w testdir/   <w allows to add contents
to dir>

$ cd testdir/

$ touch file

$ ls file
```

Changing Permissions Numerically

Each permission setting can be represented by a numerical value: (defined in include/uapi/linux/stat.h)

- r = 4 w = 2 x = 1 - = 0
- 4000 = Sets user ID on execution (Set UID)
- 2000 = Sets group ID on execution (Set GID)
- 1000 = Sets the t attribute
- 0400 = Permits read by owner.
- 0200 = Permits write by owner.
- 0100 = Permits execute or search by owner.
- 0040 = Permits read by group.
- 0020 = Permits write by group.
- 0010 = Permits execute or search by group.
- 0004 = Permits read by others.
- 0002 = Permits write by others.
- 0001 = Permits execute or search by others.

Changing Permission Numerically

$ touch file

$ ls -l file

-rw-r--r-- 1 alice staff 0 Mar 15 12:43 file

$ chmod 766 file

$ ls -l file -rwxrw-rw- 1 alice staff 0 Mar 15 12:43 file

Sticky Bit

- Used to protect files within a directory
- A file can be deleted only by the owner of the file or root
- Special permission that prevents a user from deleting other user's files from public directories such as /tmp

```
vol@ubuntu:/tmp$ mkdir test

vol@ubuntu:/tmp$ ls -al

drwxrwxr 2 vol vol 4096 Mar 14 23:45 test
vol@ubuntu:/tmp$ chmod 1777 test
vol@ubuntu:/tmp$ ls -al

drwxrwxrwt 2 vol vol 4096 Mar 14 23:45 test
```

How are you as a user able to run a file whose owner is root?

SetUID – Power for a Moment

- Privileged programs widely used by UNIX systems
- By default a process run by a user has the same permissions as that of the user.
- A process that runs a file with setUID enabled is granted access based on the owner of the file (usually root) and not the user running that executable
- Allows users to access files and directories not available to the owner
- Security risk because users maybe able to find a way to maintain the permissions granted to them by setuid process

SetGID

- A process’s effective groud id (EGID) is changed to the group owner of the file and user is granted access based on permissions to that group

- How to turn set-UID bit on?

chmod 4755 file ---> -rwsr-xr-x

Set-UID Concept

- Allow user to run a program with the program owner's privilege.
- Allow users to run programs with temporary elevated privileges
- Example: the passwd program

$ ls -l /usr/bin/passwd

-rw**s**r-xr-x 1 **root** root 41284 Sep 12 2012 **/usr/bin/passwd**

Types of UID (& GID)

- User ID: Integer that identifies a particular user on a system
- Every process has atleast two user IDs
- Real UID
- Effective UI
- For a normal program the real user id and the effective user id is the same.
- Some *NIX systems has a third UID – Saved UID

Types of UID (& GID)

- Real UID
- Used to determine which user started the process
- When setuid bit is not set, process executes with permission that of real uid.
- *unsigned short getuid()*
- Effective UID
- Used to determine what resources the process can access
- When setuid bit is set, real ID remains the same, effective ID becomes that of the user who owns the file
- For access control the kernel checks only the effective user ID.
- E.g. When a process tries to open a file, the effective user ID is checked when deciding whether to let the

process access the file.

- *unsigned short geteuid(), int setuid(uid_t uid)*

- Saved UID
- Used to restore previous EUID
- Used to save UID to drop privileges while switching between UID of user invoking program and ID of the user owning the executable

Attacking SUID Programs

- 99% of local vulnerabilities in UNIX exploit SUID root programs to obtain root privileges.
- 1% target OS itself
- Attacking SUID applications is based on
- Inputs
- Startup: command line, environment
- During execution: dynamic-linked objects, file input
- Interaction with the environment
- File system: creation of files, access to files
- Processes: signals, or invocation of other commands

Set-UID Concept

- Every process has two User IDs.
- **Real UID (RUID)**: Identifies real owner of process
- **Effective UID (EUID)**: Identifies privilege of a process
- Access control is based on EUID
- When a normal program is executed, RUID = EUID, they both equal to the ID of the user who runs the program
- When a Set-UID is executed, RUID ≠ EUID. RUID still equal to the user's ID, but EUID equals to the program **owner**'s ID.
- If the program is owned by root, the program runs with the root privilege.

Turn a Program into Set-UID

- Change the owner of a file to root :

```
seed@VM:~$ cp /bin/cat ./mycat
seed@VM:~$ sudo chown root mycat
seed@VM:~$ ls -l mycat
-rwxr-xr-x 1 root seed 46764 Nov  1 13:09 mycat
seed@VM:~$
```

- Before Enabling Set-UID bit:

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

- After Enabling the Set-UID bit :

```
seed@VM:~$ sudo chmod 4755 mycat
seed@VM:~$ mycat /etc/shadow
root:$6$012BPz.K$fbPkT6H6Db4/B8cLWbQI1cFjn
h/pDyc5U1BWOzkWh7T9ZGu.:15933:0:99999:7:::
daemon:*:15749:0:99999:7:::
bin:*:15749:0:99999:7:::
sys:*:15749:0:99999:7:::
```

How it Works

A Set-UID program is just like any other program, except that it has a special marking, which a single bit called Set-UID bit

How is Set-UID Secure?

- Allows normal users to escalate privileges
- This is different from directly giving the privilege (sudo command)
- Restricted behavior

- Unsafe to turn all programs into Set-UID
- Example: /bin/sh
- Example: vi

Vulnerabilities of SUID program

- The password can be changed by a helpdesk user or by root.
- Relative Path – use absolute path wherever possible
- A privileged program must conduct sanity check on all inputs , including *implicit inputs*
- E.g. implicit input – environment variables . Shell programs use env variables extensively
- They are controlled by users and hence a user can cause a change in the behavior of a program

Use absolute path

Vulnerabilities of SUID program

- E.g. PATH environment variable
- C Program can use getenv to explicitly request environment vars
- What if a setUID program implicitly uses env var
- When running a command in a shell, the shell searches for the command using the PATH environment variable, which consists of a list of directories
- The directories are searched in the order in which they are found
- The first program that matches with the name of the command will be executed

If the user executes the following system(“mail”) How can an attacker cause undesirable behavior?

- System internally calls /bin/sh –c <command> and lets the shell program execute the command
- The attacker can change PATH to the following, and cause “mail” in the current directory to be executed.

PATH=".:$PATH"; export PATH

Vulnerabilities of SUID Programs

• LD_LIBRARY_PATH Environment Variable

• Unless explicitly specified via the *-static* option during compilation, all Linux programs are incomplete and require dynamic linking to link libraries at run time.

• The dynamic linker/loader (ld.so/ld-linux.so) loads the shared libraries needed by a program, prepares the program to run, and then runs it. You can use the following command to see what shared libraries a program depends on:

% ldd /bin/ls

• LD LIBRARY PATH is an environment variable used by the the dynamic linker/loader. It contains a list of directories it searches for shared libraries. Multiple directories are listed colon (:) separated.

Vulnerabilities of SUID Programs

- LD LIBRARY PATH Environment Variable
- Attacker can modify this variable, and force the library loader to search for libraries in the attacker's directory

% setenv LD_LIBRARY_PATH .:$LD_LIBRARY_PATH

- To make sure Set-UID programs are safe from the manipulation of the LD LIBRARY PATH environment variable, the runtime linker/ loader (ld.so) will ignore this environment variable if the program is a Set-UID program.
- Secure applications can also be linked statically with a trusted library to avoid this.

Vulnerabilities of SUID Programs

- LD_PRELOAD Environment Variable
- Allows you to ”preload” user-specified shared libraries before all others.
- This can be used to selectively override functions in other libraries.
- % export LD_PRELOAD=./libmylib.so.1.0.1
- E.g. If libmylib.so.1.0.1 contains a function *sleep*, which is a standard libc function, when a program is executed and calls sleep, the one in libmylib.so.1.0.1 will be invoked

Vulnerabilities of SUID Programs

- Program that override the sleep() function in libc (sleep.c): #include <stdio.h> void sleep (int s) {

printf("I am not sleeping!\n"); }

Compile the program

gcc -fPIC -g –c sleep.c gcc -shared -o libmylib.so.1.0.1 sleep.o –lc

Main Program: int main() {

sleep(1); return 0;

}

```
$ export LD_PRELOAD=./libmylib.so.1.0.1
$ gcc main.c
$ ./a.out
I am not sleeping!
```

Notes for Previous Slide

- Scenario : A process with real user ID 25 is invoking a program whose owner's User ID is 18. The program has the setUID bit set. The process with real UID 25 is running exec and is spawning a new process that executes this program. The program executes and is trying to read-write a file one of which has Owner ID 18 and the other has owner ID 25 and the permissions as specified.
- SetUID & Exec: When a process is spawned from another process it inherits the RUID, but since the owner of the program has user ID 18 the effective user id become 18. This process, now, even though was started using a different process with RUID 25 is now able to access files owned by user ID 18.
- Fix: The process can call getruid and set ruid to 25. The setuid call sets the euid of the process . So now both ruid=euid=25