CSci 4271W
Development of Secure Software Systems
Day 11: OS security: access control

Stephen McCamant (he/him)
University of Minnesota, Computer Science & Engineering

Based in large part on slides originally by Prof. Nick Hopper

Operating systems

- The goal of an operating system is to provide a uniform platform for programs to access system resources.
- The security goal of an operating system is to prevent processes from inappropriately accessing resources used by other processes.
- In order to do this, the OS must also protect itself from the processes it manages.

Operating Systems

An OS broadly provides three kinds of security functions:

- Authentication: linking processes to users
- Access Control: making decisions about access to resources
- Protection: enforcing access control policies

Outline

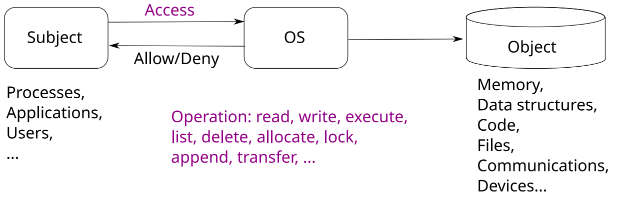
OS security overview

OS security: access control

Announcements, midterm debrief

Access control

The operating system mediates access requests between subjects and objects



Processes, Applications, Users, ...

Operation: read, write, execute, list, delete, allocate, lock, append, transfer, ...

Memory, Data structures, Code, Files, Communications, Devices...

Access control matrix

Objects

Subjects	Obj1	Obj2	Obj3	Obj4	· · ·	ObjN
Subj1	rwl	-	-	rl	· · ·	wx
Subj2	rw	rw	-	-		lx
Subj3	-	l	x	rw		-
⋮					⋱	⋮
SubjM	rl	wl	rl	rw	· · ·	rx

Access control matrix, storage

The matrix is implemented through a combination of subject-stored data and object-stored data

Subject
Object

Identifiers, Capabilities, other Capabilities

Access Control List

Unix subjects

Unix subject = process.
Each process stores:
Several 32-bit user IDs
A list of 32-bit group IDs
A set of capabilities

Unix subjects

Unix subject = process.
Each process stores:
Several 32-bit user IDs
A list of 32-bit group IDs
A set of capabilities

UIDs:
UID:username map: `/etc/passwd`
Real UID (ruid):
Inherited from parent process
Effective UID (euid):
Determines access
Saved UID (suid):
Set after EUID is changed
(FS UID: Linux-only, obsolete)

Unix subjects

Unix subject = process.
Each process stores:
Several 32-bit user IDs
A list of 32-bit group IDs
A set of capabilities

GIDs:
UID:GID map: `/etc/passwd`
groupname:GID:members map:
`/etc/group`
Effective GID (egid):
Allows access
Real GID, Saved GID: analogous to UID
Supplementary GIDs:
Also allow access

Unix subjects

Unix subject = process.
Each process stores:
Several 32-bit user IDs
A list of 32-bit group IDs
A set of capabilities

PCAPs:
Set of capabilities that are subsets of `root`
`CAP_DAC_OVERRIDE`
(skip R/W/X permission checks)
`CAP_FOWNER` (owner on all files)
`CAP_KILL` (signal any process)
`CAP_SYS_TIME` (set clock)
`CAP_SYS_ADMIN` (catchall)

Unix objects

Primarily file system objects, like:
Files
Directories
Device files
Named pipes

Every object has:
owner UID and permissions
group GID and permissions
"other" permissions
possible set(uid/gid)
Permissions include:
(r)ead, (w)rite, e(x)ecute
Only one of the owner, group, or other permissions apply

Directory permissions

- Same R/W/X bits, slightly different interpretation
  - Read: list contents (e.g., `ls`)
  - Write: add or delete files
  - Execute: traverse ("search")
- X is needed on every level of parent directory
- R and W only apply at one level
- X but not R means: have to know the names

Permission examples

Suppose we have:

object	owner	group	permissions
/	0	0	d rwx r-x r-x
/path	101	100	d --x --x --x
/path/f1	101	100	- -wx -wx --x

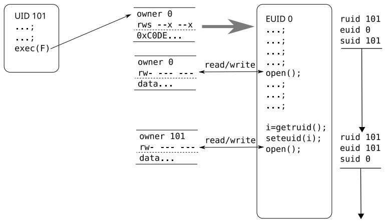
subject	euid	gid	request
proc1	101	100	open("/path/f1",O_RDWR)
proc1	101	100	exec("/path/f1",...)
proc2	1001	100	chdir("/path")
proc3	1001	100	open("/path/f1",O_WRONLY)

Which requests will succeed?

UID management

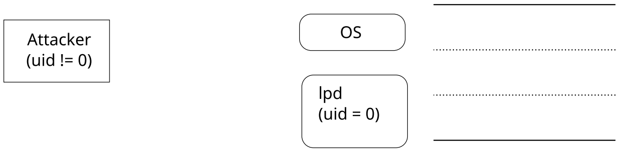
- A process with UID 0 is a "superuser" or `root` process and generally can access all objects and change UID/GIDs.
- Processes created by `fork` inherit parent's UID/GIDs
- If a file F has the setuid bit on, a successful `exec` of F will set the process UID to the file's UID
  - And respectively setgid with GID
- Processes can manipulate UIDs using `set*uid` system calls:
  - $\texttt{seteuid(newid)}$ will succeed if $\texttt{newid} \in \{\texttt{suid}, \texttt{ruid}\}$ or $\texttt{euid} = 0$

Saved UID temporary change



Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.

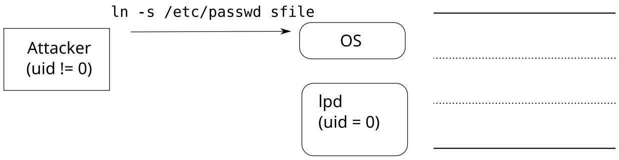

Attacker (uid != 0)

OS

lpd (uid = 0)

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

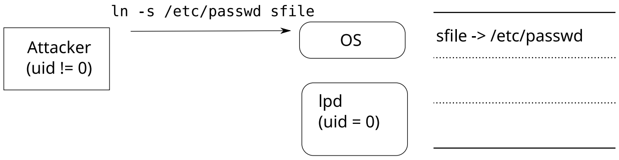
`ln -s /etc/passwd sfile`

OS

lpd (uid = 0)

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

`ln -s /etc/passwd sfile`

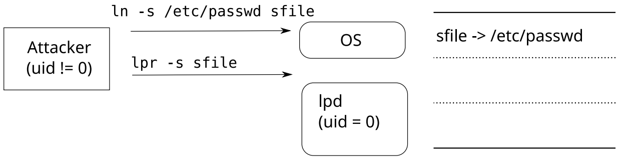
OS

lpd (uid = 0)

sfile -> /etc/passwd

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

`ln -s /etc/passwd sfile`

`lpr -s sfile`

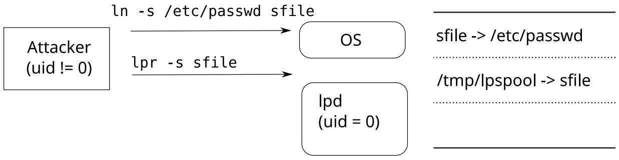
OS

lpd (uid = 0)

sfile -> /etc/passwd

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

`ln -s /etc/passwd sfile`

`lpr -s sfile`

OS

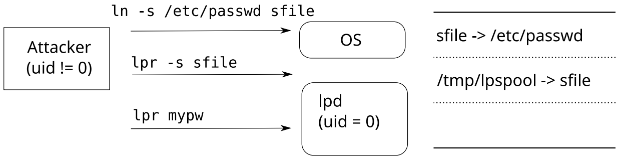
lpd (uid = 0)

sfile -> /etc/passwd

/tmp/lpspool -> sfile

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

`ln -s /etc/passwd sfile`

`lpr -s sfile`

`lpr mypw`

OS

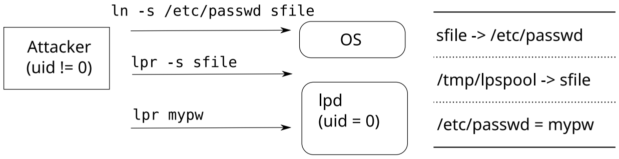
lpd (uid = 0)

sfile -> /etc/passwd

/tmp/lpspool -> sfile

Confused deputy

When a process needs some privileges (e.g., of a UID), and can be confused into using other privileges the UID.


Attacker (uid != 0)

`ln -s /etc/passwd sfile`

`lpr -s sfile`

`lpr mypw`

OS

lpd (uid = 0)

sfile -> /etc/passwd

/tmp/lpspool -> sfile

/etc/passwd = mypw

Outline

OS security overview

OS security: access control

Announcements, midterm debrief

Coming up next week

- Homework 3 due Tuesday, finished relevant material today
- One-per-person section drafts due Thursday

More project-related

- “No BS” policy: don’t claim vulnerabilities you haven’t confirmed
- The BCBMC binary is compiled with standard mitigations
  - For full credit, your PoC exploits must work against these
  - Attacks may use multiple vulnerabilities
- Now available, Piazza is the best place for project questions

Midterm score distribution

I’ve made a $+5$ point difficulty adjustment on Canvas

```
Before adjust.:   After:
  5 | *           5 | *
  6 | ****        6 | **
  7 | *           7 | ***
  8 | **          8 | *
  9 | **          9 | **
                 10 | *
  Mean: 73        Mean: 78
Median: 68      Median: 73
```

Q2: defensive programming

(Code shown outside slides)

Q3: memory corruption

(Code shown outside slides)