Privileges

Acquiring Privileges

- How can privileged operations be performed?

- More precisely, how can an unprivileged process request that something privileged take place?

- What is privilege?

Types of Privilege

- Hardware restriction—ask the OS (or maybe it just isn't allowed)

- OS restriction—use a privileged process

- Application restriction—application-dependent

What is a Privileged Process?

- One that has access to some resource not generally available

- Doesn't necessarily correspond to "root" or "Administrator"

- More secure systems have many types of privilege

File System-Related Privileges

- Who can write to certain devices?

- Example: print spooler

- Example: Mailer daemon

Mailer Daemon: Sending Mail

- In principle, an unprivileged operation

- For convenience, have one well-written mail daemon per system

  - Accept mail from mail user agents (MUAs)

  - Add some header lines (MessageID, Received, maybe From)

  - Attempt delivery; queue and retry if necessary

- Use "privilege" to protect its own queue files.
  Which security attribute is invoked here?

Mailer Daemon: Sender Security

Confidentiality Protect the confidentiality of the email itself

Integrity Prevent mail from being tampered with

Availability Prevent mailer crashes and email deletion; deliver mail

The Lessons of Mailer Security

- There's no hardware or "privilege"

- The mailer has no resources unavailable to other users

- Mailer protection is just another case of protecting one user from another

What if Sending Mail is Privileged?

- Many sites block direct outbound access to port 25

- Why? Firewalls, spam senders, and "bots"

- How can we restrict which users can connect to which port?

- Either make network access go through the file system —`/dev/tcp/25/another.host`—or add a separate permission mechanism

What Are Other Privileges?

What Are Other Privileges?

- Override DAC (or override it for specific purposes)

- Mount a file system in a restricted fashion

- Mount a file system with no restrictions

- Operate on file as owner

- Change MAC label

- Set time

- Assign privileges

- Many more—IRIX (an old Unix variant) had 48 different privileges

What is the Principle of Least Privilege

- No subject should have more privileges than it needs

- Obvious reason: it can't misuse abilities it doesn't have
  ☞ Very important in case of application compromise

- Difficult to do properly, since one privilege often implies another

- Example: if I can overide the DAC "write" privileges, I can overwrite an executable that a more privileged user will invoke, and thus get that user's privileges

How Processes Get Privileges

- Inheritance

- File attributes

- Ask a privileged process to perform the operation for you

Inheritance

- Many privileges are inherited from parent process (necessary in Unix, where almost every command is run in a separate process)

- Example: Unix uid

- Sometimes associated with username: the login mechanism sets initial privileges, and your shell inherits them

- Obviously, child processes cannot inherit privileges the parent doesn't possess

File Attributes: SetUID (SUID)

- Fundamental privilege acquisition mechanism in Unix

- Invented in 1973 by Dennis Ritchie

- Patented—U.S. Patent 4,135,240, issued January 1979

- Major step towards *principle of least privilege*

- A serious security risk if used improperly

FIG. IB
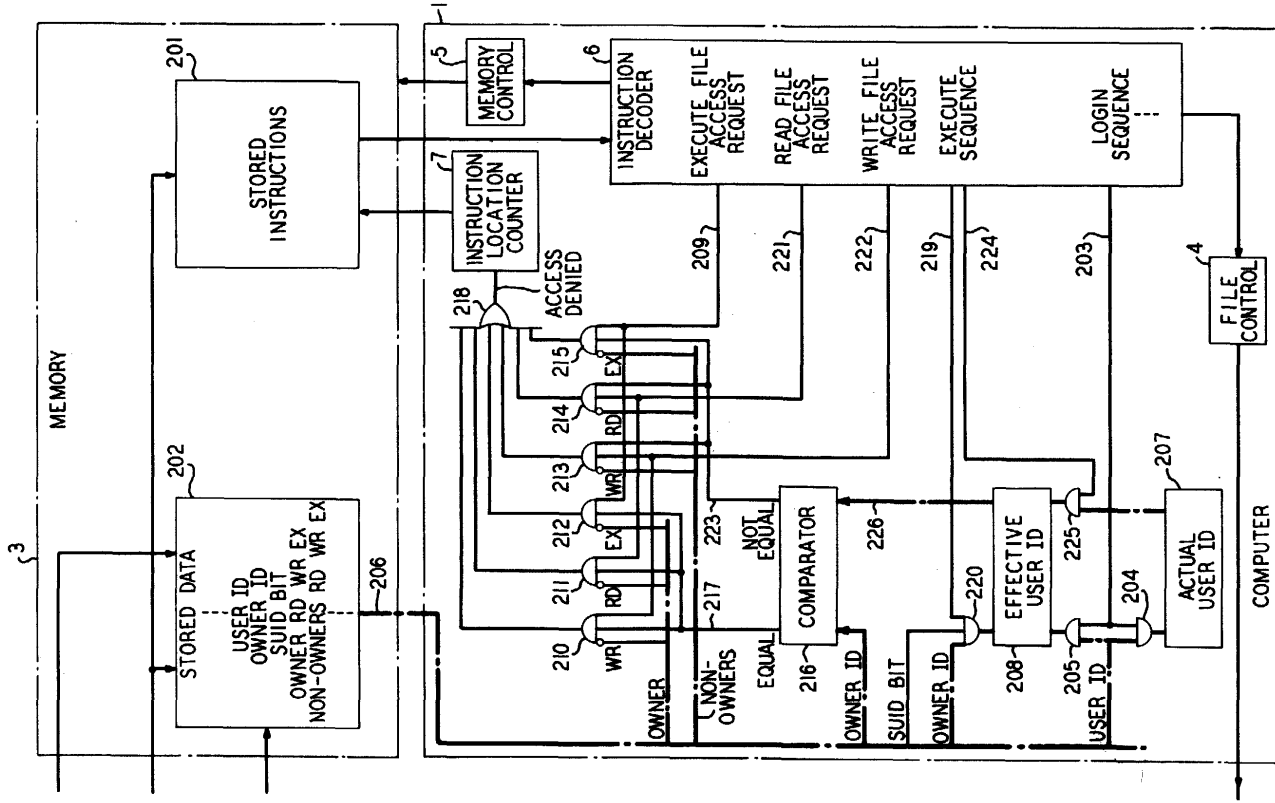
Setting and Querying the SUID Bit

- Set:

  ```
  chmod u+s file
  ```

- Query:

  ```
  ls -l file
  ```
  The "x" for owner is replaced by a "s"
  ```
  $ ls -l /bin/su
  -rwsr-xr-x 2 root root 71288 Feb 27 2013 /bin/su
  ```

What is SUID?

- When the program is executed, it acquires the privileges of the owner

- This feature is available for all uids

- Similar feature for groups: setgid

- If a file is setuid "root", it executes with "root" privileges

What Does This Do?

```
$ cp /bin/sh .
$ ls -l sh
-r-xr-xr-x  1 smb   smb   109768 Sep 15 22:49 sh
$ chmod u+s sh
$ ls -l sh
-r-sr-xr-x  1 smb   smb   109768 Sep 15 22:49 sh
```

Oops!

- It created a setuid shell

- Anyone who executed that shell would have all of my privileges

- Not a good thing to do…

How Did I Do That Safely?

```
$ (umask 077; mkdir f)
$ cd f
$ ls -ld .
drwx------   2 smb   smb   512 Sep 18 22:49 .
```

Safely Doing Dangerous Things

- Create a directory that no one else can access

- Use `umask` to do it atomically

- Create the dangerous file in a locked directory

- Only I (and "root") can get to that directory

Combining Permission Settings

- Use some permissions to restrict acccess

- Use SUID or SetGID to grant more authority to invoker

- What does this do?

```
$ ls -l shutdown
-r-sr-xr--  1 root  operator  14463 Sep  2 01:38 shutdown
```

CS
@CU

The NetBSD `shutdown` Command

Note the permissions:

`-r-sr-xr-- 1 root operator`

r-s SUID "root"

r-x Executable by group "operator"

r-- Readable but not executable by others

The command runs with "root" permissions, but only a select few can get those permissions

Why is SUID Good?

- Available to all users; does not require special privilege

- Used by mailers, printer daemons, games, etc

- Conceptually simple way to provide controlled interface to some resources

Why is SUID Bad?

- Available to all users; does not require special privilege

- Writing *secure* SUID programs is *hard*

- Too easy to give away permissions

- Per-user permissions aren't granular enough

Peter Neumann on SUID

It is precisely BECAUSE it allows easy implementation that it is so frequently misused—by people who don't know better. Use of "setuid" opens up the possibility of a variety of security flaws, including Trojan horses, search-path traps, etc., and tends to substantially widen the perimeter of trust. I'm not sure that anyone knows how to characterize "proper use" completely—if it is indeed possible at all.

RISKS Digest 4:53, 1 March 1987

Fred Grampp and Robert Morris on SUID

SUID programs should be used only when there is no other way to get a desired result. On most *UNIX* systems, perhaps a dozen SUID programs, excluding games, are really needed. A lax attitude about SUID programs, combined with a 'quick and dirty' programming style, can produce disasters. . .

It is difficult, when users are writing all but the most trivial programs, to determine in advance that the program will be correct. Programs sometimes do the most amazing things in unforeseen circumstances.

UNIX Operating System Security

AT&T Bell Laboratories Technical Journal 63:8, Part 2, October 1984

Linux and SUID

- Grampp and Morris: " On most *UNIX* systems, perhaps a dozen SUID programs . . . are really needed."

- A CLIC machine has about 29 (plus games, plus 19 setgid). . .

- But—about 10 are for network utilities. Is a new privilege mechanism needed instead?

What is the Problem with SUID?

- The bad guy is running the program and supplying the inputs

- The bad guy controls the environment

- Many subtle traps!

Confusing a Program

```
$ PS1='% ' ksh
% ulimit -f 0
% echo foo >/tmp/foo
File size limit exceeded
$ ls -l /tmp/foo
-rw-r--r--  1 smb  wheel  0 Sep 19 00:04 /tmp/foo
```

What if this happens to the `passwd` command?

We'll talk a lot more about others during the semester

The Alternative: Message-Passing

- A program runs with certain permissions

- It sets up some sort of local communications channel

- Other programs send messages to that channel, and receive responses

- Used by Windows, some Unix subsystems

Practical Difficulties

- How does the initial program start?

- What sorts of channels are available?

- Can you control access to those channels?

- What are the messages and responses like?

Initial Startup

- Very much OS-dependent

- On some systems, any user can have a program started at boot time:

```
$ crontab -l
@reboot echo `hostname` reboot | mail smb
```

- Sometimes program is invoked automatically when channel is contacted

A "Daemon Daemon"

- Have an always-running "daemon daemon"

- It listens for requests for other daemons, and fires up the proper program with the proper privilege

- (Also used for network daemons)

- Must be very privileged, since it can run programs as any other user

Types of Channels

- Local sockets ("UNIX-domain sockets")

- Message-passing queues

- Controlled RPC

Access Control

- Different channels have different permission mechanisms

- Very much OS-dependent

- Getting these right is just as important as file permissions

System V Shared Memory

```
$ ipcs -b
Message Queues:
T         ID       KEY          MODE          OWNER     GROUP QBYTES


Shared Memory:
T         ID       KEY          MODE          OWNER     GROUP   SEGSZ


Semaphores:
T         ID       KEY          MODE          OWNER     GROUP NSEMS
s   19398656         0 --rw-------         www       www     1
```

UNIX-Domain Sockets

- Appear in file system

- *Not* accessed like regular files; use "socket" primitives instead

- Permissions on this "file" not always honored—version-depenent!

- Solution: set directory permissions instead

```
# ls -ld private
drwx------  2 postfix  wheel  512 Sep 10 23:31 private
# ls -ld private/maildrop
srw-rw-rw-  1 postfix  wheel  0 Sep 10 23:31 private/maildrop
```

Why is Message-Passing Good?

- Bad guys can't invoke the privileged commands

- No opportunity to control the environment

- Less opportunity for certain harmful programming mistakes

Why is Message-Passing Bad?

- Fundamentally, you're writing network servers

- We know from experience how hard it is to get them right!

- You have to design a language for the channel, and perhaps marshall/unmarshall arguments

Capabilities

- Capability: a bit-string that gives access to some resource

- Sometimes, a cryptographically protected string

- Can be copied

- Effectively attribute-based access control

- Issues: acquisition, forgery, revocation

- Rarely used in practice, except for Web cookies

What is the Real Issue with Acquiring Privileges?

- Granting selective access is *hard*

- *Never* trust anything that can be controlled by the enemy

- Make sure you know the enemy's powers...