UNIX Security

From UNIX SYSTEMS Programming, Robbins & Robbins

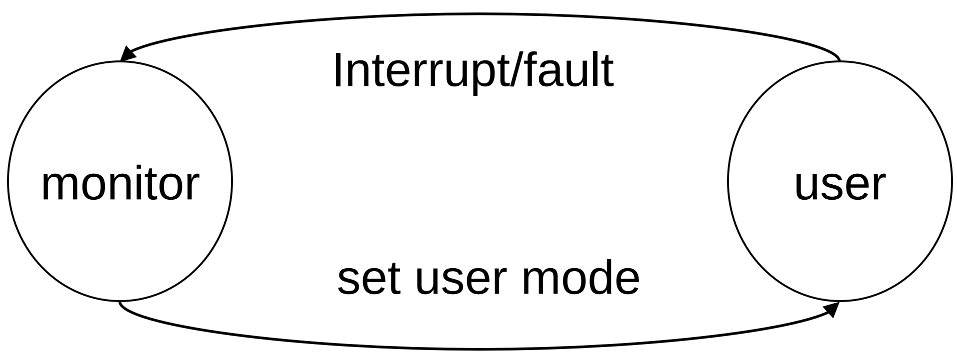
Benjamin Brewster

Dual-Mode Operation

- Sharing system resources requires operating system to ensure that an incorrect program cannot cause other programs to execute incorrectly
- Provide hardware support to differentiate between at least two modes of operations
  1. *User mode* – execution done on behalf of a user
  2. *Monitor mode* (also *supervisor mode* or *system mode*) – execution done on behalf of operating system

Dual-Mode Operation

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1)
- When an interrupt or fault occurs hardware switches to monitor mode
- *Privileged instructions* can be issued only in monitor mode



I/O Protection

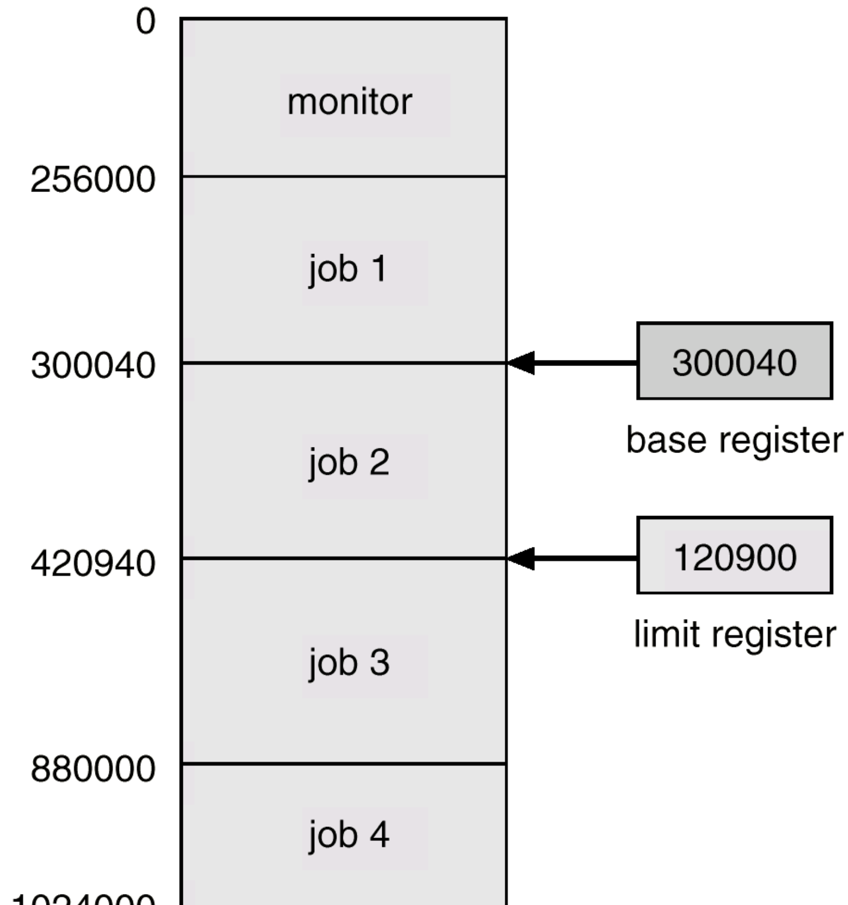
- All I/O instructions are privileged instructions

- Must ensure that a user program could never gain control of the computer in monitor mode (I.e., a user program that, as part of its execution, stores a new address in the interrupt vector)

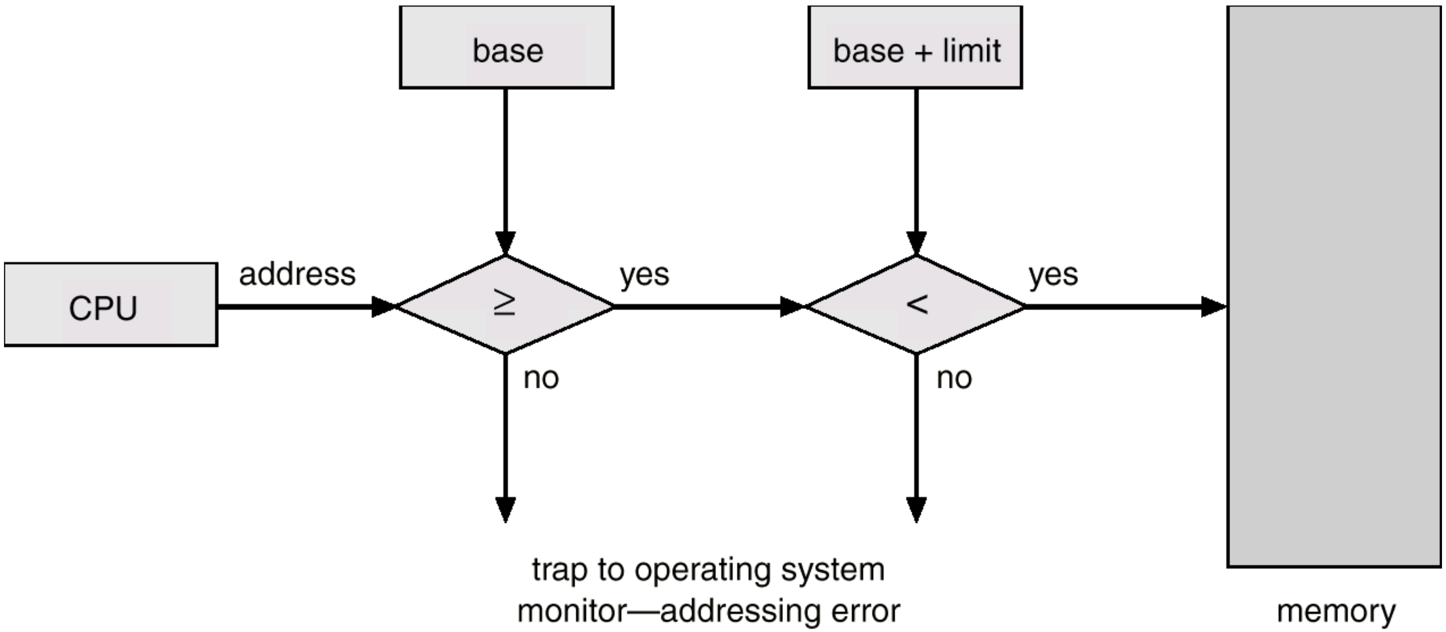
Memory Protection

- Must provide memory protection at least for the interrupt vector and the interrupt service routines
- In order to have memory protection, add two registers that determine the range of legal addresses a program may access:
  - **base register** – holds the smallest legal physical memory address.
  - **Limit register** – contains the size of the range
- Memory outside the defined range is protected

A Base and a Limit Register Define a Logical Address Space



Protection Hardware



- When executing in monitor mode, the operating system has unrestricted access to both monitor and user’s memory
- The load instructions for the *base* and *limit* registers are privileged instructions

CPU Protection

- *Timer* – interrupts computer after specified period to ensure operating system maintains control
  - Timer is decremented every clock tick
  - When timer reaches the value 0, an interrupt occurs
- Timer commonly used to implement time sharing
- Time also used to compute the current time
- Load-timer is a privileged instruction

General-System Architecture

- Given the I/O instructions are privileged, how does the user program perform I/O?
- System call – the method used by a process to request action by the operating system
  - Usually takes the form of a trap to a specific location in the interrupt vector
  - Control passes through the interrupt vector to a service routine in the OS, and the mode bit is set to monitor mode
  - The monitor verifies that the parameters are correct and legal, executes the request, and returns control to the instruction following the system call

Acting as a different user

- User files are protected from other users by defining access based on accounts
- If you are logged in as an account with access (ie, you're the owner, or a group owner), you can manipulate the file
  - chmod
  - group ids

Acting as a different user

- If you want to temporarily act as a different user (but stay logged on as yourself), you can use the `su` command:
  - `su yoog`

- You'll need to know `yoog`'s login credentials

id revisited

- The id command prints out your user and group ids:

```
% id
uid=22026(brewstbe) gid=6009(upg22026)
        groups=6009(upg22026),12028(transfer)
```

id revisited

- The user and group ids are changed when using `su`

- You will now have different *effective ids*, as opposed to your *real ids*, which you still have
  - `id` can display both your real and effective ids

root

- Most UNIX systems have a super-user account, typically called root
  - `su root`

- As root, you can change file ownerships, in addition to many other things

- You effectively can change anything

Limits, for example

- As root, you can change /etc/security/limits.conf

```
flip % cat /etc/security/limits.conf
# ...
# ...
*                   hard    nproc           30
```

- On flip, everyone is strictly limited to 30 processes

SUID, SGID

- Each executible has two security bits associated with it: SUID, and SGID
  - If SUID is set, the executible runs with effective user ID of the *owner* of the file
  - If SGID is set, the executible runs with effective user ID of the *group owner* of the file

SUID, SGID

- This is different from before – we're now talking about specific executibles that have bits that enable them to run as different users
  - As opposed to *being* a different user, and then running programs, as `su` allows

S[U|G]ID Example

```
% ls -pla /bin
...
-rwsr-xr-x   1 root root  53024 Jun 20  2005 su
...
```

s here means that the SUID bit is set

An s here would mean that the SGID bit was set

S[U|G]ID Example

```
% ls -pla /bin
...
-rwsr-xr-x   1 root root  53024 Jun 20  2005 su
...
```

- In this example, su runs with root permissions
  - therefore, it can change things that only root can change, while not allowing the user to be root!

`chmod` revisited

- It turns out that there are twelve mode bits:
  - 4000 - Setuid on execution
  - 2000 - setgid on execution
  - 1000 - set sticky bit
  - 0400 - read by owner
  - 0200 - write by owner
  - 0100 - execute by owner
  - 0040 - read by group
  - 0020 - wr
  - 0010 - execute by group
  - 0004 - read by others
  - 0002 - write by others
  - 0001 - execute by others

What if…

- What if you replace the real su, which has SUID set and is owned by root, with your own code?
  - It would have the same permissions, but could do anything you want to the system

What if…

- What if you could set the SUID bit on your own file?
  - It would still be owned by you, and thus would run as you
    - Not interesting

- Can you give your file to root?
  - No – this is specifically why you have to *be* logged in as root to change file ownership!

What to secure

- Following we'll list some things you can do to make your UNIX system more secure
- There's tons more than this… but what's the maximum security we could provide?

Strongest Security

- The strongest form of security involves:
  - Network isolation
  - Physical isolation

Physical Isolation

- Why is physical Isolation so important?

- Even if you disable local shell access, you still have to worry about:
  - Bootable devices (live CDs, flash drives, etc.) can boot a different OS that can access the hard drive of your computer
  - Hard drive could be stolen
  - etc.

Seriously, though

- Physical and Network isolation makes for a not-so-useful computer
  - Maybe you could use it for cryptography, or for storing really sensitive data

- Here are some other ways to secure your system, but still retain real functionality

Password Security

- Don't let users write them down
- Age the passwords
- Enforce stronger (but more annoying) passwords
  - 1337: @nte@te|2
  - random: Z1#3s8u*h
  - long: Ho\/\/YouTypeMeF@stFooL
- Restrict use of previous passwords
- Password dictionary check

Login Failures

- What happens if you don't lock a user account if too many failures happen?
  - A account can be brute forced
    - How?

Password Encryption

- Pork sausage model (one-way):
  - username: Stonesand
  - password: lamepasswd
    - a3R7nito5fo%r

- Store the pair Stonesand / a3R7nito5fo%r
- This encrypted pair is public knowledge, but the encryption method is one-way

Password Encryption

- If anyone knew how to reverse the password method, then they could go:
  - a3R7nito5fo%r -> lamepasswd
- Fortunately it is very hard to crack the one-way encryption
- Problem: why is storing the password file publicly still dangerous?
  - Brute force crack approach on a fast compy
    - Hence non-public password file, and long passwords

Monitoring and Logs

- Finally, monitor everything with logs
  - Network
  - Account login/logout
  - Program usage
  - Others

Insecure Protocols

- Telnet, FTP

- Secure versions of these are SSH, and SFTP

Getting root access

when you're not supposed to have it…

- Try the front door first:

ACCOUNT: *PASSWORD*

- **root: root**
- sys: sys / system / bin
- bin: sys / bin
- **mountfsys: mountfsys**
- adm: adm
- uucp: uucp
- nuucp: anon
- anon: anon
- user: user
- games: games
- **install: install**
- demo: demo
- **umountfsys: umountfsys**
- **sync: sync**
- admin: admin
- guest: guest
- daemon: daemon

Getting root access

when you're not supposed to have it…

- After that, and assuming social engineering didn't work, you'll have to use fancy stuff
  - Port scans + port/program insecurities
  - Buffer overflows (with system access)
  - Boot Hacking (with physical access)

- Why are we talking about this stuff?
  - So you can protect yourself against it