UNIX Security

CSE497b - Spring 2007
Introduction Computer and Network Security
Professor Jaeger
www.cse.psu.edu/~tjaeger/cse497b-s07/
UNIX System

• Originated in the late 60’s, early 70’s
  – Bell Labs: Ken Thompson, Dennis Ritchie, Douglas McIlroy

• Multiuser Operating System
  – Enables protection from other users
  – Enables protection of system services from users

• Simpler, faster approach than Multics
UNIX Security

• Each user owns a set of files
  – Simple way to express who else can access
  – All user processes run as that user

• The system owns a set of files
  – Root user is defined for system principal
  – Root can access anything

• Users can invoke system services
  – Need to switch to root user (setuid)

• Q: Does UNIX enable configuration of “secure” systems?
UNIX Challenges

• More about *protection* than *security*
  – Implicitly assumes non-malicious user and trusted system processes

• Discretionary Access Control (DAC)
  – User or their processes may update permission assignments
    • Each program has all user’s rights
    • Must trust their processes to be non-malicious

• File permission assignments
  – Assignment based on what is necessary for things to work
    • All your processes have all your rights

• System services have full access
  – Users invoke *setuid* (root) procs that have all rights
    • Must trust system processes
UNIX Protection State

- **Subjects**
  - Users
  - Groups
  - Processes make accesses on behalf of users belonging to particular groups

- **Objects**
  - Files
  - Directories

- **Operations**
  - Read
  - Write
  - Execute
Subjects

• Users
  – Username
  – User ID (UID)
  – Groups
  – Special User: root

• Process
  – UID, GID
    • Real user ID
    • Effective user ID
    • FS user ID
    • Saved user ID
  – Users run processes
  – Effective UID determines access, generally (FS UID for Linux)
Groups

• Users belong to one or more groups
• Primary group: defined in /etc/passwd
• All other possible groups: defined in /etc/group
  – group_name:group_passwd:GID:list_of_users

• Commands to change group membership
  – e.g., newgrp
• Group membership gives additional permissions beyond UID
UNIX Authentication

• Login process
  – Started at boot time (runs as ‘root’)
  – Takes username and password
  – Applies crypt() to password with stored salt
  – Compares to value in /etc/shadow for that user

• Starts process for user
  – Executes file specified as login in /etc/passwd
  – Identity (uid, gid, groups) is set by login
UNIX UID Transitions

- **UID transitions**
  - For `login` process: UIDs are root
  - After authentication, the shell’s UIDs are: tjaeger
  - Exec su: real is tjaeger; effective is root

- Transitions among UIDs are complex

Figure 5: Three finite state automata describing the `seteuid`, `setreuid`, `setresuid` system calls in Linux respectively.

Ellipses represent states of the FSA, where a notation like “R=1,E=0,S=1” indicates that euid = 0 and ruid ≠ suid ≠ 0.

Each transition is labelled with the system call it corresponds to.

(c) An FSA describing `setresuid` in Linux
UNIX Objects == Files

• UNIX objects are represented as files
  – Regular files
  – Device (character or block) files
  – Socket files
  – FIFO files
  – Link files

• Directories
  – Hierarchical organization of files
  – Paths: Sequence of directories followed by a file name

• Files are stored as *inodes*
  – Inode to data mapping is fixed
  – File name to inode mapping is *not* fixed

• Beyond socket files, there is *no* network access control
UNIX Mode Bits

• Operations
  – Read, write, execute

• Users
  – Owner, Group, World

• File type
  – Semantics of operations
    • Based on file type
  – Different semantics between files and dirs
Changing permissions

• Change permissions of a file
  – chmod
    • chmod 644 file -- owner can read/write, group, others can read only
    • chmod u+x file -- adds execute permission for owner

• Change owner of a file
  – chown
    • chown new_owner file

• Change group of a file
  – chgrp
    • chgrp new_group file
UNIX Authorization

- File’s owner UID == Process’s effective UID
  - fsuid for Linux
  - Check owner mode bits

- File group GID is a member of process’s active group set
  - Check group mode bits

- Otherwise,
  - Check others mode bits
UNIX Permissions

• What UNIX permissions are granted to...
  – An editor process?
  – An editor process that you run?
  – An editor process that someone else runs?
  – An editor process that contains malware?
  – An editor process used to edit a password file?

• Q: Can we restrict an editor to a single file?
Nobody

- Special user with no ownership
  - Belongs to no groups
- Q: What permissions are available to nobody?
Chroot

• Create a *domain* in which a process is *confined*
  – Process can only read/write within file system subtree
  – Applies to all descendant processes
  – Can carry file descriptors in ‘chroot jail’
Chroot Vulnerability

• Unfortunately, chroot can trick its own system
  – define a passwd file at <newroot>/etc/passwd
  – run su
    • su thinks that this is the real passwd file
  – gives root access
    • Use mknod to create device file to access physical memory

• Setup requires great care
  – Never run chroot process as root
  – Must not be able to get root privileges
  – No control by chrooted process (user) of contents in jail
  – Be careful about descriptors, open sockets, IPC that may be available
UID Transition: Setuid

• A special bit in the mode bits

• Execute file
  – Resulting process has the effective (and fs) UID/GID of file owner

• Enables a user to escalate privilege
  – For executing a trusted service

• User defines execution environment
  – e.g., Environment variables

• Service must protect itself or user can gain root access
Setuid Execution

• Process A running as
  – UID=X

• Fork process A to create process B
  – Both running with UID=X

• The exec file C in process B with setuid bit set and owner of root
  – process A has UID=X
  – process B has UID=root
Confused Deputy Problem

• Situation
  – A program has authority (setuid root file)
  – Is confused into using that authority incorrectly

• Example
  – Call httpd supplied libexecdir
  – Add your own libraries to overwrite passwd (if httpd runs as root)

• Also a concern for network daemons
  – Why?

• A motivation for capability systems
  – Discuss later
UNIX System Vulnerabilities

• Several UNIX functions present security problems
  – Provide a function
    • Mount a CD-ROM
  – That provides some (formerly) privileged function
    • Mount a filesystem
  – That has some potential side-effects
    • CD-ROM filesystem’s permissions

• Some are system problems
  – Quite a few have been fixed

• Some require careful programming
  – Harder to fix in general (like preventing a buffer overflow)
Mount Vulnerabilities

- Multiple file systems on different physical devices under `/`
- Mounting file systems
  - mount and automount
- What are the rights of such file systems?
  - Defined by their inodes
  - Get root
    - Mount a file system with a setuid program
- Can use mount options to disable
  - `nosuid` -- turns off setuid bits
  - `noexec` -- no binaries can be executed
  - `nodev` -- no (block or character) devices can be accessed
Link Vulnerabilities

• Add new path to an inode
• Multiple names for a single inode
• Run
  – `ln -s /etc/passwd file`
  – `trusted_dump file < *passwd-entry*`
  – To overwrite `/etc/passwd`
• Programs have to be aware of which files they are using
• `open(file, O_NOFOLLOW, mode)`
  – Prevents open from following a link
• Also, problems with `access`, then open
  – File name to inode mapping causes problems
Device File Vulnerabilities

• Devices are represented as files
  – /dev/tty -- terminal
  – /dev/mem -- physical memory
  – /dev/kmem -- virtual memory
  – /dev/mouse -- mouse

• Create using mknod (only accessible by root)

• Can bypass access control by getting access to memory
  – /dev/kmem or /dev/mem used to be world accessible

• Can get access to user inputs
  – /dev/tty is world readable -- see password, set keys
  – mesg n -- prevents write access to current terminal
/tmp Vulnerability

• `creat(pathname, mode)`
• `O_EXCL` flag
  – if file already exists this is an error
• Potential attack
  – Create file in shared space (/tmp)
  – Give it a filename used by a higher authority service
  – Make sure that service has permission to the file
  – If `creat` is used without `O_EXCL`, then can share the file with the higher authority process
Take Away

• UNIX security originally aimed at protection in multiuser systems
  – DAC over files

• UNIX DAC security model cannot express security requirements
  – Lots of rights accessible by default
  – Means for limiting rights are impractical

• The use of UNIX mechanisms has evolved over time
  – Resulting in vulnerabilities