CSci 4271W
Development of Secure Software Systems
Day 13: More Permissions, and OS-level Injection Threats
Stephen McCamant
University of Minnesota, Computer Science & Engineering

Outline
Legal context for security, cont’d
More Unix permissions
Announcements intermission
Shell code injection and related threats

CFAA
- Computer Fraud and Abuse Act of 1986
- Civil and criminal liability for “unauthorized access” to a computer
- Gradually extended to cover any computer, and many related activities
- Potentially applied to any contract or terms-of-service violation
  - Not always successfully

Example: Randal Schwartz
- Schwartz worked as a contract sysadmin several Intel divisions
- He ran a password cracking program and moved password files between machines in a division he no longer worked for
- He was convicted of three felonies under an Oregon state law
  - Similar to the CFAA, somewhat more vague

DMCA
- Digital Millennium Copyright Act of 1998
- Legally reinforces DRM by criminalizing “circumvention” and tools that perform it
- But, can violate without violating copyright
  - App stores, video game bots, garage door openers
- A narrow exemptions process is growing in application

Example: Sony BMG “rootkit”
- In 2005, sold CDs with software that modified a Windows or Mac OS to interfere with copying
- To prevent removal, the software used techniques usually used by malicious software
  - A “rootkit” is backdoor software installed on a compromised machine
  - Common techniques include hiding files and processes
- Led to a recall, class action suits, FTC settlement, etc.

Outline
Legal context for security, cont’d
More Unix permissions
Announcements intermission
Shell code injection and related threats

Process UIDs and setuid(2)
- UID is inherited by child processes, and an unprivileged process can’t change it
- But there are syscalls root can use to change the UID, starting with setuid
  - E.g., login program, SSH server
Setuid programs, different UIDs

- If 04000 "setuid" bit set, newly exec'd process will take UID of its file owner
- Other side conditions, like process not traced
- Specifically the effective UID is changed, while the real UID is unchanged
- Shows who called you, allows switching back

More different UIDs

- Two mechanisms for temporary switching:
  - Swap real UID and effective UID (BSD)
  - Remember saved UID, allow switching to it (System V)
- Modern systems support both mechanisms at the same time

Setgid, games

- Setgid bit 02000 mostly analogous to setuid
- But note no supergroup, so UID 0 is still special
- Classic application: setgid games for managing high-score files

Special case: /tmp

- We'd like to allow anyone to make files in /tmp
- So, everyone should have write permission
- But don't want Alice deleting Bob's files
- Solution: "sticky bit" 01000

Special case: group inheritance

- When using group to manage permissions, want a whole tree to have a single group
- When 02000 bit set, newly created entries with have the parent's group
  - (Historic BSD behavior)
- Also, directories will themselves inherit 02000

Other permission rules

- Only file owner or root can change permissions
- Only root can change file owner
  - (Historic BSD behavior) "give away chown"
- Setuid/gid bits cleared on chown
  - Set owner first, then enable setuid

Outline

- Legal context for security, cont'd
- More Unix permissions
- Announcements intermission
- Shell code injection and related threats

Midterm 1 grade statistics

- Mean: 78.75, Median: 79
Project 0.5 now available
- Code auditing and attacking against BCBASIC
- Audit and attacks in groups of up to 3, write reports individually
- More realistic code auditing than you've had to do before
- Due a week from Friday, October 27th

Project 0.5 suggestions
- Understand more than 1000 lines of C
- Vulnerabilities are bugs, but easy to miss
- Attack techniques are also more complex
- Group activities will be individual in the next project
- Help everyone in your group understand process and results
- Software/AI tools allowed, but don't expect them to make it easy

BCBASIC basic introduction
- fib.bcbas shown in text editor

Outline
- Legal context for security, cont’d
- More Unix permissions
- Announcements intermission
- Shell code injection and related threats

Two kinds of privilege escalation
- Local exploit: give higher privilege to a regular user
  - E.g., caused by bug in setuid program or OS kernel
- Remote exploit: give access to an external user who doesn't even have an account
  - E.g., caused by bug in network-facing server or client

Shell code injection
- The command shell is convenient to use, especially in scripts
  - In C: system, popen
- But it is bad to expose the shell's power to an attacker
- Key pitfall: assembling shell commands as strings
- Note: different from binary "shellcode"

Shell code injection example
- Benign: system("cp $arg1 $arg2"), arg1 = "file1.txt"
- Attack: arg1 = "a b; echo Gotcha"
- Command: "cp a b; echo Gotcha file2.txt"
- Not a complete solution: prohibit ';

The structure problem
- What went wrong here?
- Basic mistake: assuming string concatenation will respect language grammar
  - E.g., that attacker supplied "filename" will be interpreted that way
**Best fix: avoiding the shell**
- Avoid letting untrusted data get near a shell
- For instance, call external programs with lower-level interfaces
  - E.g., fork and exec instead of system
- May constitute a security/flexibility trade-off

**Less reliable: text processing**
- Allow-list: known-good characters are allowed, others prohibited
  - E.g., username consists only of letters
  - Safest, but potential functionality cost
- Deny-list: known-bad characters are prohibited, others allowed
  - Easy to miss some bad scenarios
- "Sanitization": transform bad characters into good
  - Same problem as deny-list, plus extra complexity

**Terminology note**
- Historically the most common terms for allow-list and deny-list have been "whitelist" and "blacklist" respectively
- These terms have been criticized for a problematic "white=good", "black=bad" association
- The push to avoid the terms got significant additional attention in summer 2020, but is still somewhat political and in flux

**Different shells and multiple interpretation**
- Complex Unix systems include shells at multiple levels, making these issues more complex
  - Frequent example: scp runs a shell on the server, so filenames with whitespace need double escaping
- Other shell-like programs also have caveats with levels of interpretation
  - Tcl before version 9 interpreted leading zeros as octal

**Related local dangers**
- File names might contain any character except / or the null character
- The PATH environment variable is user-controllable, so cp may not be the program you expect
- Environment variables controlling the dynamic loader cause other code to be loaded

**IFS and why it was a problem**
- In Unix, splitting a command line into words is the shell's job
  - String \(\rightarrow\) argv array
  - grep a b c vs. grep 'a b' c
- Choice of separator characters (default space, tab, newline) is configurable
- Exploit system("/bin/uname")
- In modern shells, improved by not taking from environment