CSci 4271W Development of Secure Software Systems Day 13: OS-level Injection Threats

Stephen McCamant University of Minnesota, Computer Science & Engineering

Outline

Injection vulnerabilities: format strings

Shell code injection and related threats

Race conditions and related threats

Injection vulnerabilities

- Common dangerous pattern: interpreter code with attacker control
- 🖲 Interpreted language example: eval
- OS example: shell script injection
- Web examples: JavaScript (XSS), SQL injection
- C library example: printf format string

Format string attack: overwrite

- n specifier: store number of chars written so far to pointer arg
 - Benign but uncommon use: account for length in other formatting
- Advance format arg pointer to other attacker-controlled data
- Control number of chars written with padding
- Net result is a "write-what-where" primitive

Practical format string challenges

- Attacker usually must control format as well as one or more arguments
- Writing a big value requires impractical output size
 - Workaround 1: overwrite two bytes with %hn
 - Workaround 2: use overlapping unaligned write to control byte by byte

Format string defenses

- Compilers will warn for printf that looks like it should just be puts
- Several platforms have decided to just remove %n
 Android Bionic, Visual Studio
- Linux glibc by default will block %n if the format string is writeable
- Major remaining use is information disclosure

Outline

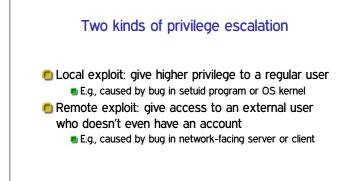
Injection vulnerabilities: format strings

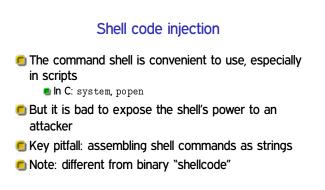
Shell code injection and related threats

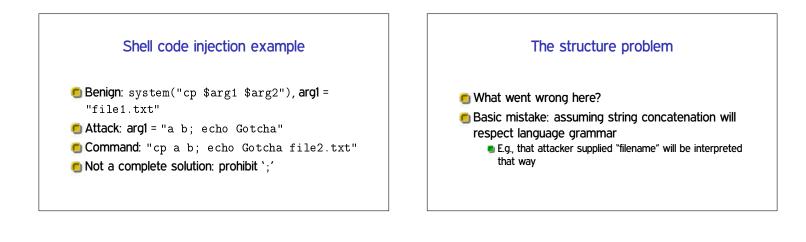
Race conditions and related threats

Demo: first steps of BCLPR format attack

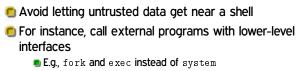
In demo: quick audit, supplying format











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



- 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

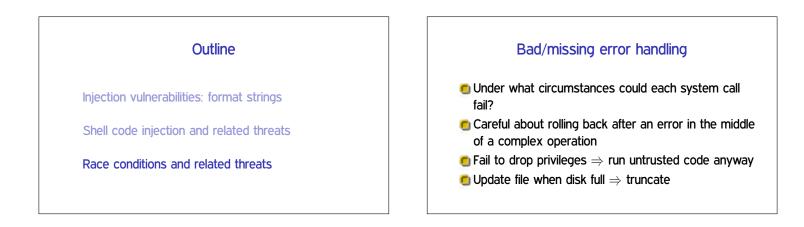
- Complex Unix systems include shells at multiple levels, making these issues more complex
 - \blacksquare Frequent example: ${\tt scp}$ runs a shell on the server, so filenames with whitespace need double escaping
- Other shell-like programs also have caveats with levels of interpretation
 - Tcl before 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

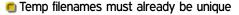
- In Unix, splitting a command line into words is the shell's job
 - String → argv array ■ grep a b c vs. grep 'a b' c
- Choice of separator characters (default space, tab, newline) is configurable
- Exploit system("/bin/uname")
- In modern shells, improved by not taking from environment



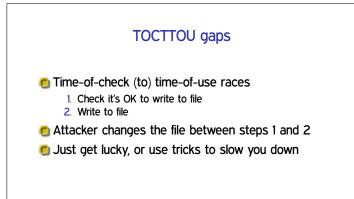
Race conditions

- Two actions in parallel; result depends on which happens first
- 🖲 Usually attacker racing with you
- 1. Write secret data to file
- 2. Restrict read permissions on file
- Many other examples

Classic races: files in /tmp



- But "unguessable" is a stronger requirement
- Unsafe design (mktemp(3)): function to return unused name
- Must use O_EXCL for real atomicity



Read It Twice (WOOT'12)

- Smart TV (running Linux) only accepts signed apps on USB sticks
- 1. Check signature on file
- 2. Install file
- Malicious USB device replaces app between steps
- 🖲 TV "rooted"/"jailbroken"

TOCTTOU example

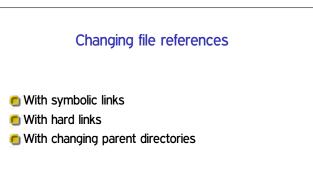
```
int safe_open_file(char *path) {
    int fd = -1;
    struct stat s;
    stat(path, &s)
    if (!S_ISREG(s.st_mode))
        error("only regular files allowed");
    else fd = open(path, O_RDONLY);
    return fd;
}
```

TOCTTOU example

```
int safe_open_file(char *path) {
    int fd = -1, res;
    struct stat s;
    res = stat(path, &s)
    if (res || !S_ISREG(s.st_mode))
        error("only regular files allowed");
    else fd = open(path, O_RDONLY);
    return fd;
}
```



```
int safe_open_file(char *path) {
    int fd = -1, res;
    struct stat s;
    res = stat(path, &s)
    if (res || !S_ISREG(s.st_mode))
        error("only regular files allowed");
    else fd = open(path, O_RDONLY);
    return fd;
}
```



Directory traversal with ...

```
Program argument specifies file, found in directory
files
```

Okat about files/../../../etc/passwd?