

CSci 4271W  
Development of Secure Software Systems  
Day 14: OS Attacks and Protection

Stephen McCamant  
University of Minnesota, Computer Science & Engineering

## Outline

Shell code injection related threats, cont'd  
Race conditions and related threats  
Announcements intermission  
Secure OS interaction  
OS: protection and isolation

## 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

## Outline

Shell code injection related threats, cont'd  
Race conditions and related threats  
Announcements intermission  
Secure OS interaction  
OS: protection and isolation

## Bad/missing error handling

- Under what circumstances could each system call fail?
- Careful about rolling back after an error in the middle of a complex operation
- Fail to drop privileges ⇒ run untrusted code anyway
- Update file when disk full ⇒ truncate

## Race conditions

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

## Classic races: files in /tmp

- Temp filenames must already be unique
- But "unguessable" is a stronger requirement
- Unsafe design (`mktemp(3)`): function to return unused name
- Must use `O_EXCL` for real atomicity

## TOCTTOU gaps

- Time-of-check (to) time-of-use races
  - Check it's OK to write to file
  - Write to file
- Attacker changes the file between steps 1 and 2
- Just get lucky, or use tricks to slow you down

## Read It Twice (WOOT'12)

- Smart TV (running Linux) only accepts signed apps on USB sticks
  - Check signature on file
  - 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;
}
```

## 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;
}
```

## Changing file references

- With symbolic links
- With hard links
- With changing parent directories

## Directory traversal with ..

- Program argument specifies file, found in directory files
- What about files/../../../../etc/passwd?

## Outline

- Shell code injection related threats, cont'd
- Race conditions and related threats
- Announcements intermission
- Secure OS interaction
- OS: protection and isolation

## First reading assignment posted

- The external reading on today's topics is chapters from a web-hosted book by David A. Wheeler
- 5 multiple-choice reading questions are a repeatable auto-graded Canvas quiz, due by Thursday after break, 3/14.

## Outline

- Shell code injection related threats, cont'd
- Race conditions and related threats
- Announcements intermission
- Secure OS interaction
- OS: protection and isolation

## Avoid special privileges

- Require users to have appropriate permissions
  - Rather than putting trust in programs
- Dangerous pattern 1: `setuid/setgid` program
- Dangerous pattern 2: privileged daemon
- But, sometimes unavoidable (e.g., email)

## Prefer file descriptors

- Maintain references to files by keeping them open and using file descriptors, rather than by name
- References same contents despite file system changes
- Use `openat`, etc., variants to use FD instead of directory paths

## Prefer absolute paths

- Use full paths (starting with `/`) for programs and files
- `$PATH` under local user control
- Initial working directory under local user control
  - But FD-like, so can be used in place of `openat` if missing

## Prefer fully trusted paths

- Each directory component in a path must be write protected
- Read-only file in read-only directory can be changed if a parent directory is modified

## Don't separate check from use

- Avoid pattern of e.g., `access` then `open`
- Instead, just handle failure of `open`
  - You have to do this anyway
- Multiple references allow races
  - And `access` also has a history of bugs

## Be careful with temporary files

- Create files exclusively with tight permissions and never reopen them
  - See detailed recommendations in Wheeler (q.v.)
- Not quite good enough: reopen and check matching device and inode
  - Fails with sufficiently patient attack

## Give up privileges

- Using appropriate combinations of `set*id` functions
  - Alas, details differ between Unix variants
- Best: give up permanently
- Second best: give up temporarily
- Detailed recommendations: Setuid Demystified (USENIX'02)

## Allow-list environment variables

- Can change the behavior of called program in unexpected ways
- Decide which ones are necessary
  - As few as possible
- Save these, remove any others

## Outline

Shell code injection related threats, cont'd

Race conditions and related threats

Announcements intermission

Secure OS interaction

OS: protection and isolation

## OS security topics

- Resource protection
- Process isolation
- User authentication (will cover later)
- Access control (already covered)

## Protection and isolation

- Resource protection: prevent processes from accessing hardware
- Process isolation: prevent processes from interfering with each other
- Design: by default processes can do neither
- Must request access from operating system

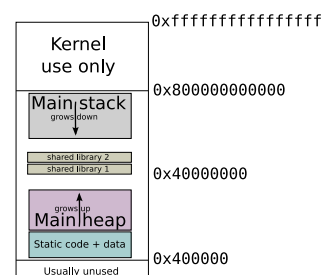
## Reference monitor

- Complete mediation: all accesses are checked
- Tamperproof: the monitor is itself protected from modification
- Small enough to be thoroughly verified

## Hardware basis: memory protection

- Historic: segments
- Modern: paging and page protection
  - Memory divided into pages (e.g. 4k)
  - Every process has own virtual to physical page table
  - Pages also have R/W/X permissions

## Linux example



## Hardware basis: supervisor bit

- Supervisor (kernel) mode: all instructions available
- User mode: no hardware or VM control instructions
- Only way to switch to kernel mode is specified entry point
- Also generalizes to multiple "rings"