CS165 – Computer Security

Filesystem Security

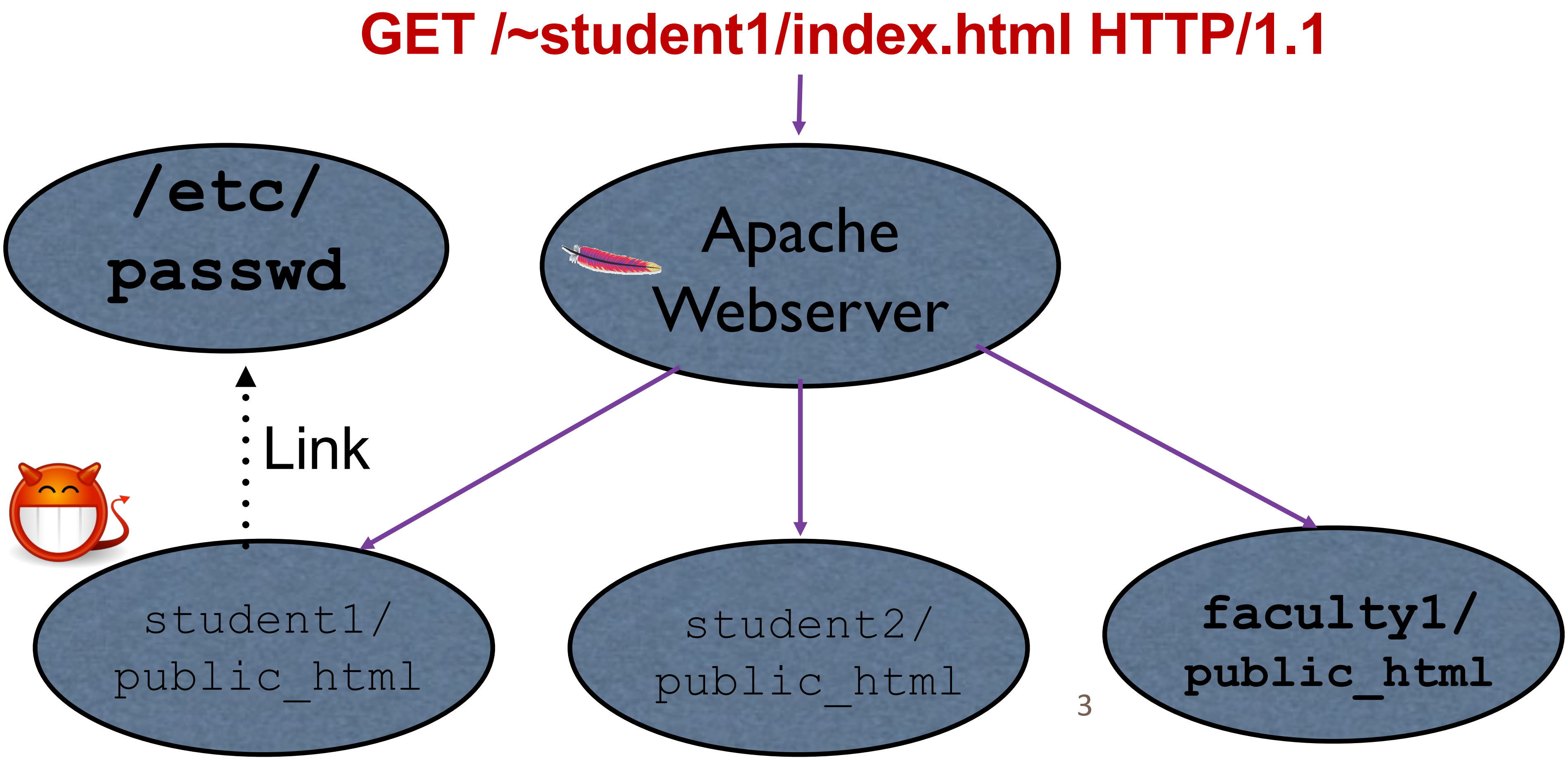
November 21, 2025

File Open

- Problem: Processes need resources from system
  - Just a simple `open(filepath, …)` right?
  - But, adversaries can cause victims to access resources of their choosing
  - And if your program has some valuable privileges, an adversary may want to trick you into using them to implement a malicious operation

A Webserver’s Story …

- Consider a university department webserver …



GET /~student1/index.html HTTP/1.1

/etc/ passwd

Apache Webserver

Link

student1/ public_html

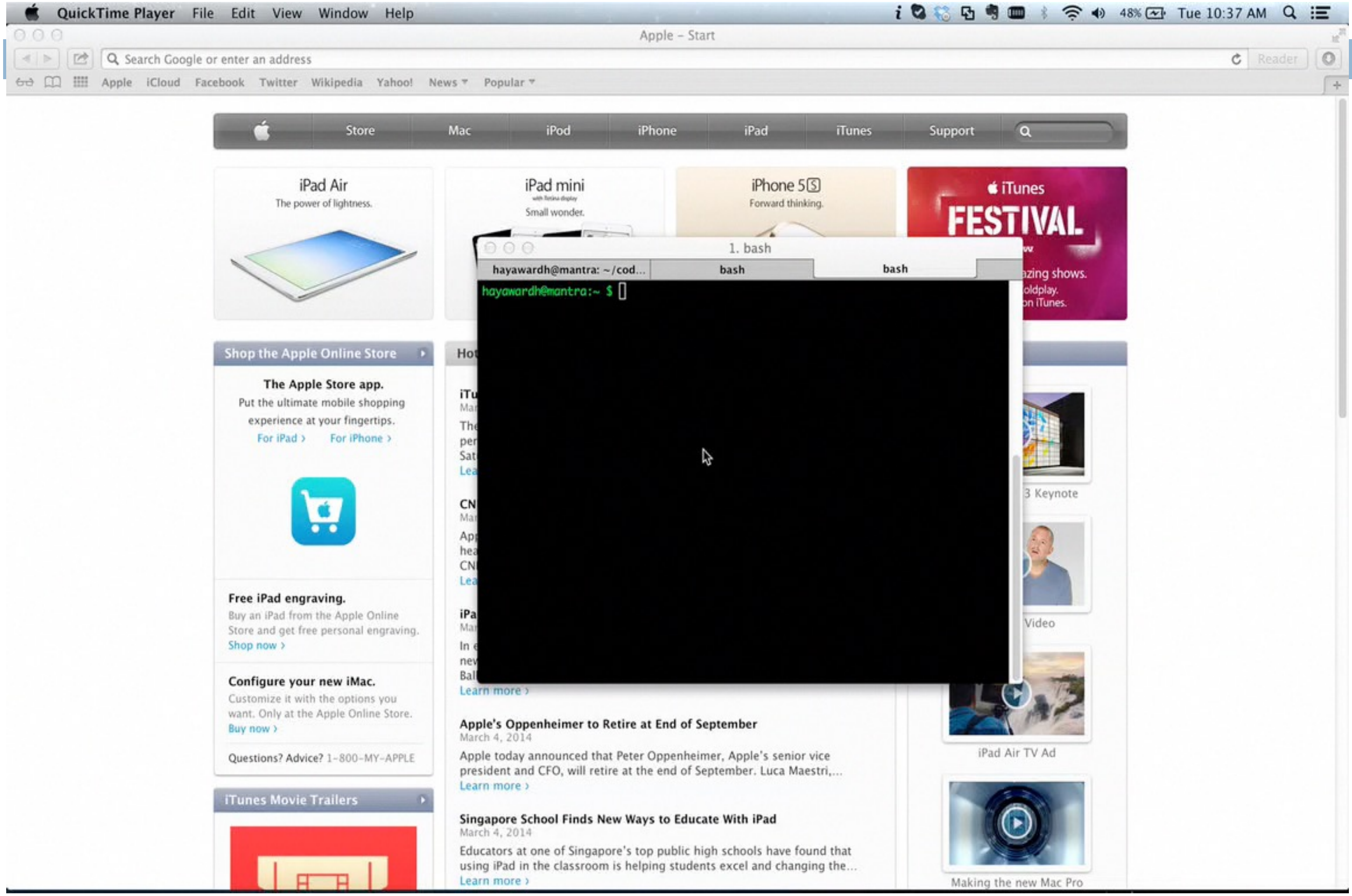
student2/ public_html

faculty1/ public_html

Symbolic Link

- Many file systems allow you to create a "link" to refer another file
  - I.e., file systems are not trees, but graphs
- There is a link command – "ln"
  - `ln -s target linkname`
  - Creates a "link" file named "linkname" in the current directory
- When you "open" the linkname, you actually open the target file
  - `ln -s /etc/passwd mylink`
  - `open("mylink", O_RDWR, …);`
  - Does what?

Attack Video



1. bash

hayawardh@mantra: ~/cod... | bash | bash

```
hayawardh@mantra:~ $ 
```

The Apple Store app.

Put the ultimate mobile shopping experience at your fingertips.

For iPad › For iPhone ›

Free iPad engraving.

Buy an iPad from the Apple Online Store and get free personal engraving.

Shop now ›

Configure your new iMac.

Customize it with the options you want. Only at the Apple Online Store.

Buy now ›

Questions? Advice? 1-800-MY-APPLE

Apple's Oppenheimer to Retire at End of September

March 4, 2014

Apple today announced that Peter Oppenheimer, Apple's senior vice president and CFO, will retire at the end of September. Luca Maestri,...

Learn more ›

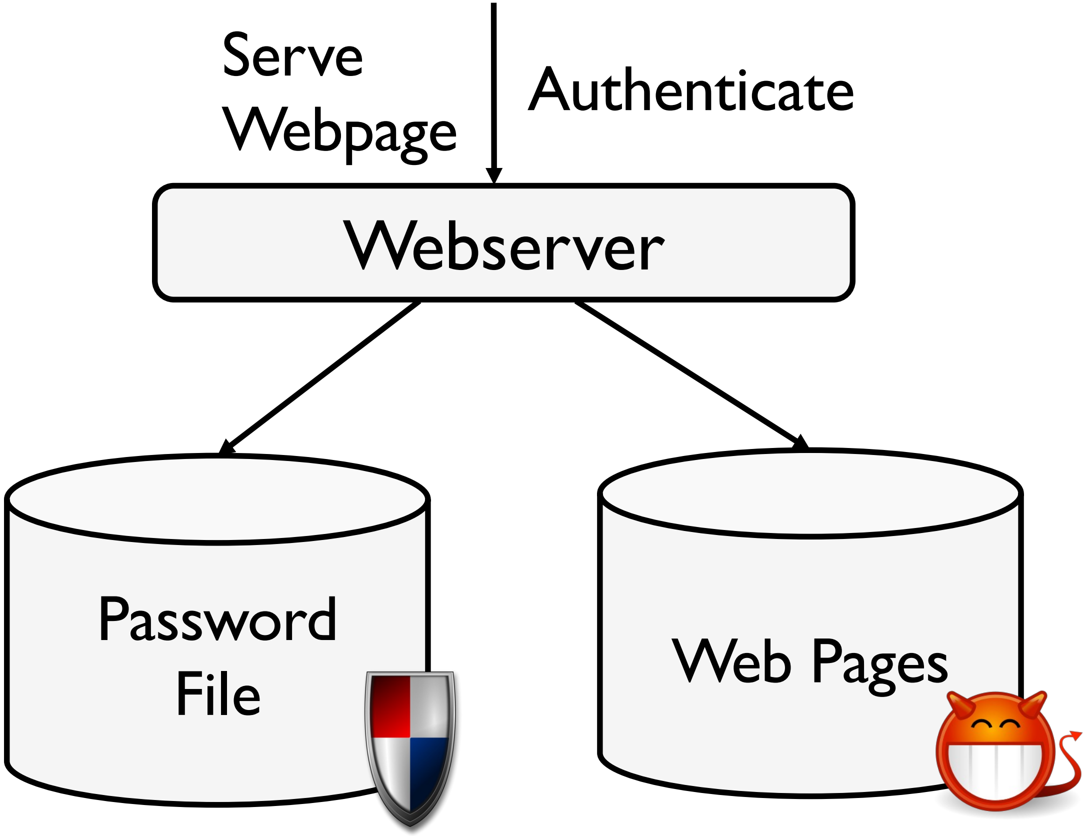
Singapore School Finds New Ways to Educate With iPad

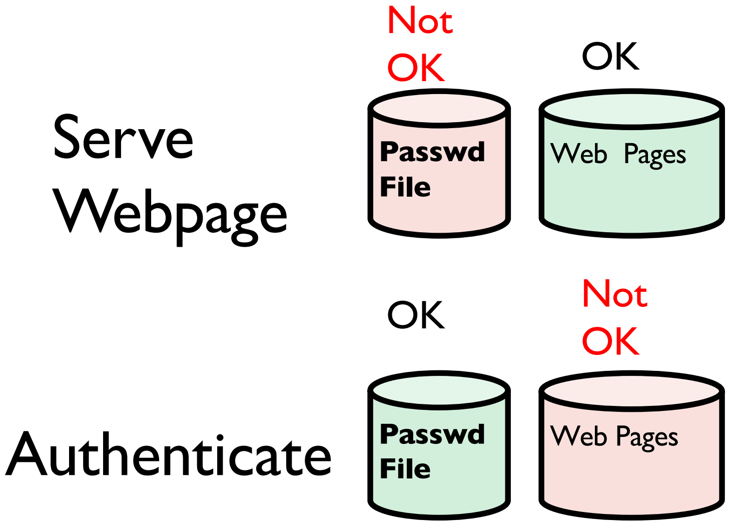
March 4, 2014

Educators at one of Singapore's top public high schools have found that using iPad in the classroom is helping students excel and changing the...

Learn more ›

What Just Happened?



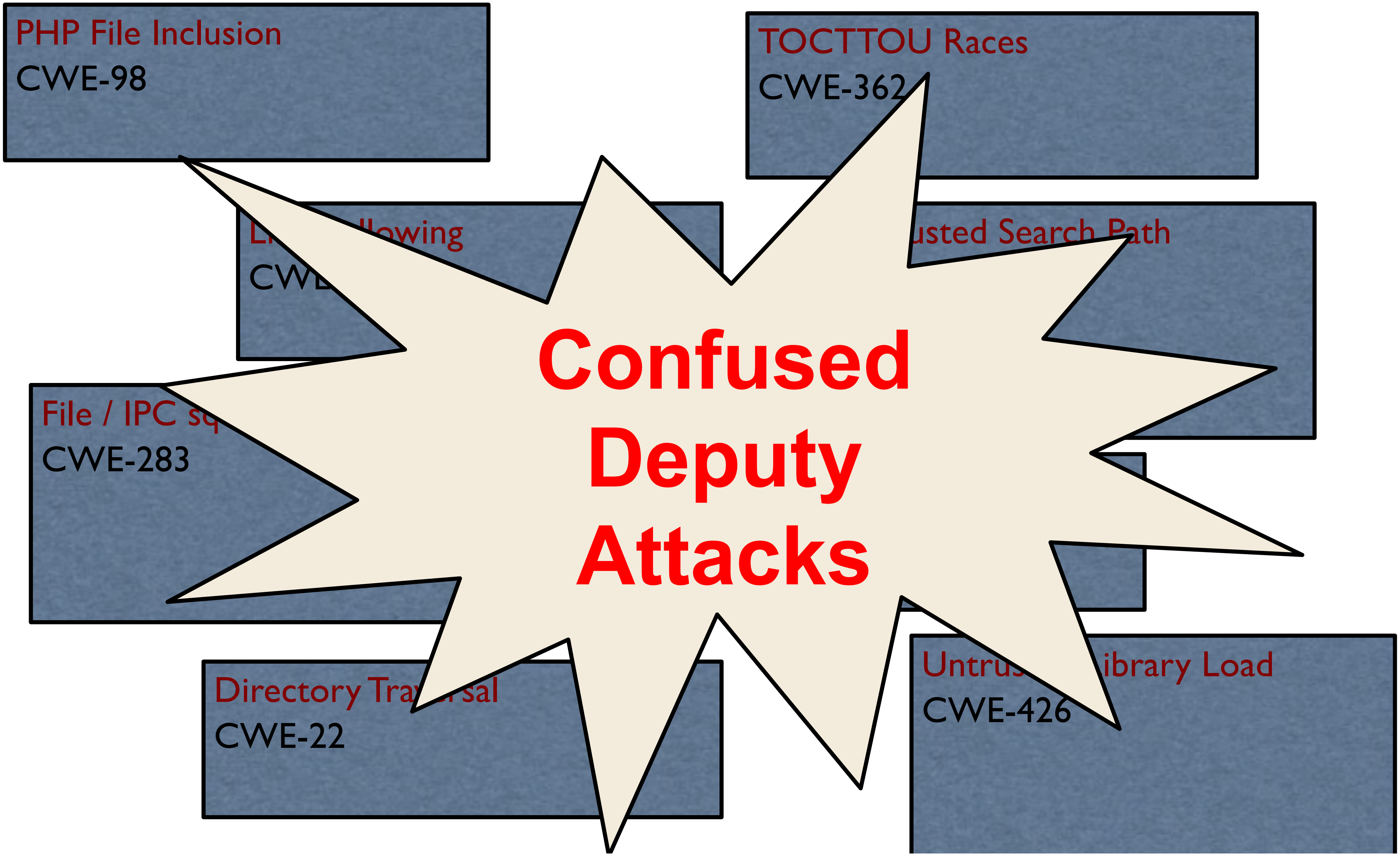


- Program acts as a *confused deputy*
  - 😈 when expecting 🛡
  - 🛡 when expecting 😈

Integrity (and Secrecy) Threat

- Confused Deputy
  - *Process is tricked into performing an operation on an adversary's behalf that the adversary could not perform on their own*
    - Write to (read from) a privileged file

Confused Deputy Attacks



PHP File Inclusion
CWE-98

TOCTTOU Races
CWE-362

Li llowing
CWE

usted Search Path

File / IPC sq
CWE-283

Directory Tra rsal
CWE-22

Untrus ibrary Load
CWE-426

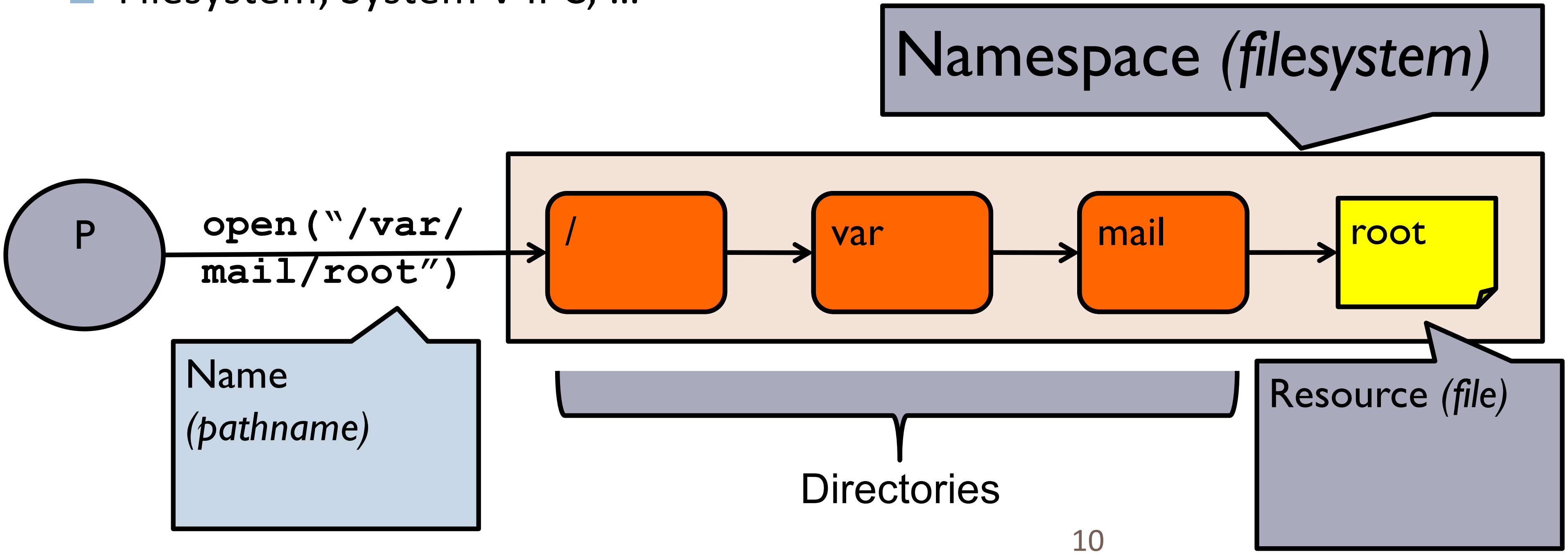
Confused Deputy Attacks

Lesson

- Opening a file is fraught with danger
  - We must be careful when using an input that may be adversary controlled when opening a file
    - Or anything else…

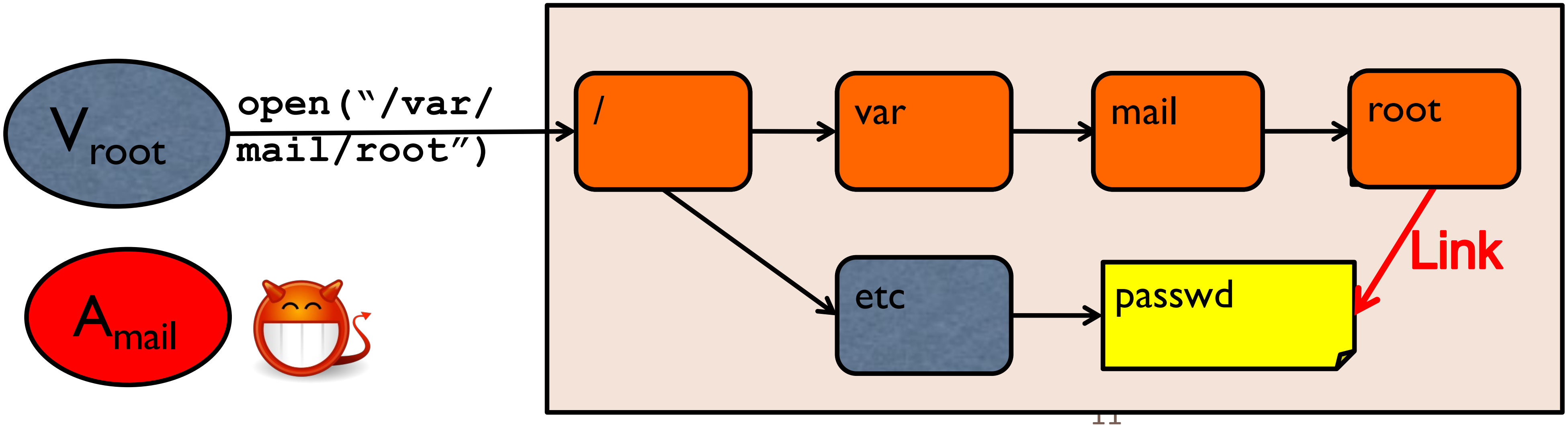
Name Resolution

- Processes often use *names* to obtain access to *system resources*
- A *nameserver* (e.g., OS) performs *name resolution* using a *namespace* (e.g., *directories*) to convert a *name* (e.g., *pathname*) into a *system resource* (e.g., *file*)
  - Filesystem, System V IPC, …



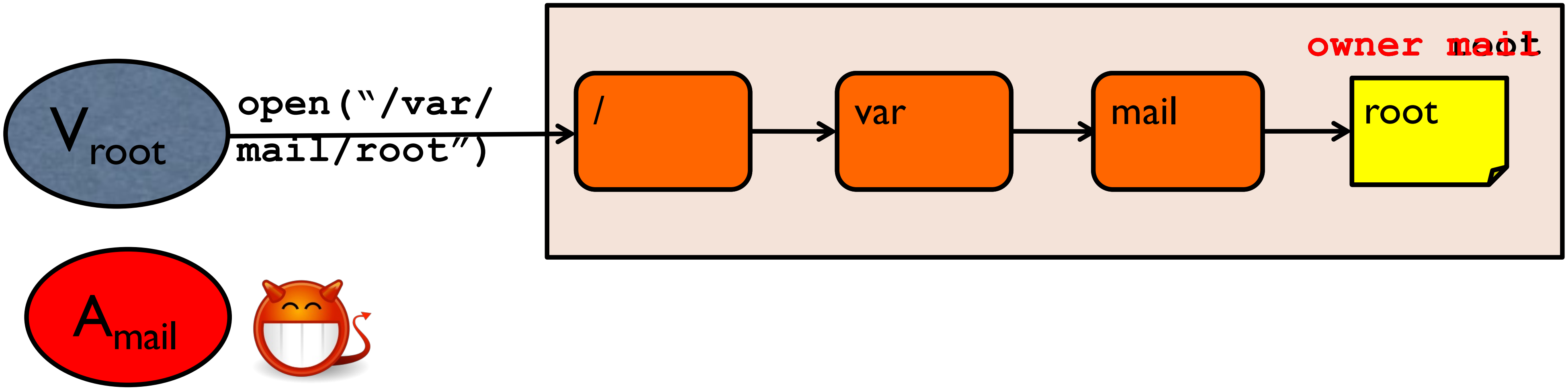
Link Traversal Attack

- Adversary controls links to direct a victim to a resource not normally accessible to the adversary
- Victim expects one resource, gets another instead



File Squatting Attack

- Adversary predicts a resource to be created by a victim – creates that resource in advance
- Victim accesses a resource controlled by an adversary instead



Common Threat

- What is the threat that enables link traversal and file squatting attacks?
  - Common to both

Common Threat

- What is the threat that enables link traversal and file squatting attacks?
  - Common to both
- In both cases, the adversary has write permission to a directory that a victim uses in name resolution
  - Could be any directory used in resolution, not just the last one
  - Enables the adversary to plant links and/or files/directories where they can write

Threat Example

- An adversary may be authorized to write to a directory you use in resolving a file path
- E.g., groups and others may have write permission to a directory
  - Consider the directory /tmp
  - `ls -la /tmp`
    - `drwxrwxrwx --- root root ---  .`
    - Means?

Threat Example

- Suppose your program wants to create a new file at “/tmp/just_a_normal_file_here”
  - What file will you create/open?

File Squatting

- Suppose your program wants to create a new file at “/tmp/just_a_normal_file_here”
  - What file will you open?
    - An adversary could have created this file already (file squat) and given you permissions, so that you can use it
      - Can be difficult to verify the origins of a file
  - Causes your program to use a file under adversary control when you expect your own file

Threat Example

- Suppose your program is asked to open the file path “/tmp/just_a_normal_file_here”
  - What file will you open?

Link Traversal

- Suppose your program is asked to open the file path "/tmp/just_a_normal_file_here"
  - What file will you open?
    - An adversary could have created this as a symbolic link to any file in the system that you can access
    - And it is difficult/expensive to verify that this is not a symbolic link
      - stat – provides file system information – e.g., permissions
      - lstat – provides file system information (like "stat") for the link, rather than the file/directory the link refers to
  - Causes your program to access an adversary-chosen file

Prevent File System Attacks

- How would you prevent such attacks?

Check and Use

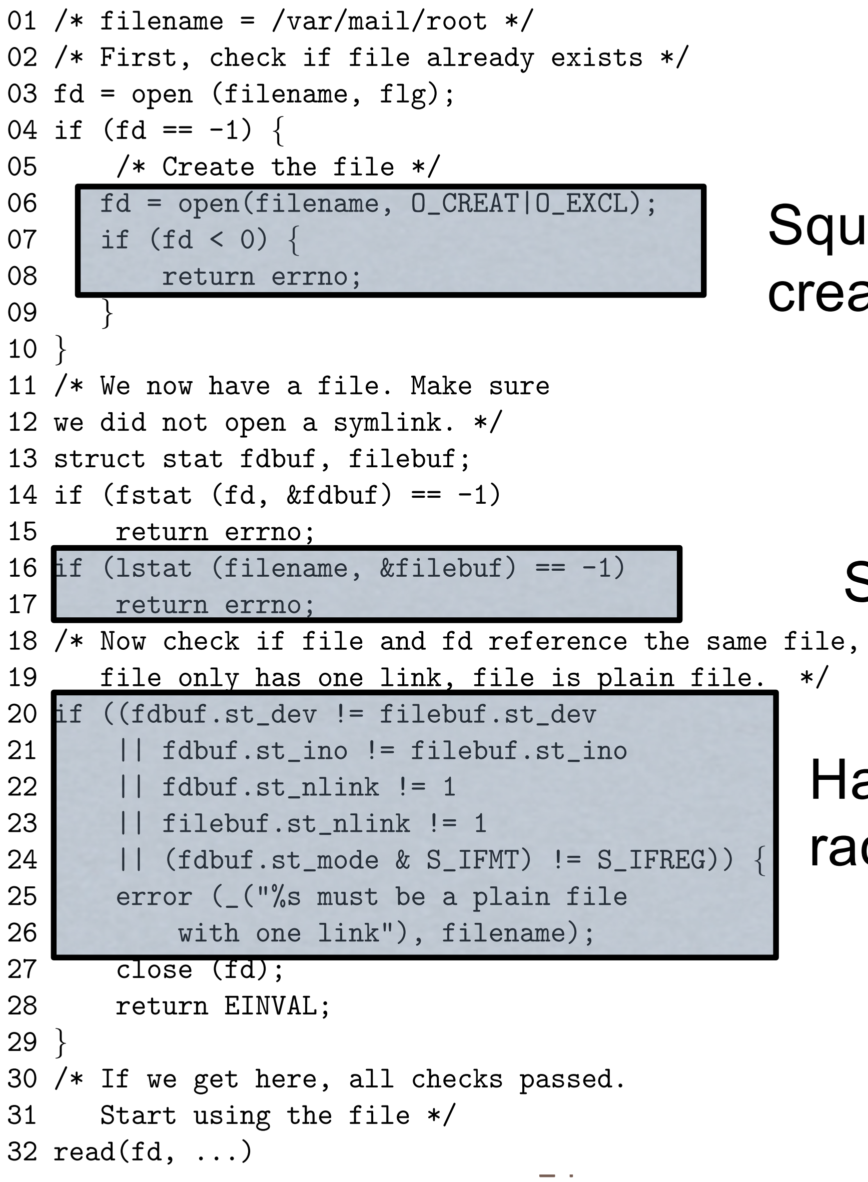
- Some system calls enable checking of the file (check)
  - Does the requesting party have access to the file? (stat, access)
  - Is the file accessed via a symbolic link? (lstat)
- Some system calls use the file (use)
  - Convert the file name to a file descriptor (open)
  - Modify the file metadata (chown, chmod)
- Can an adversary modify the filesystem in between check and use system calls?

TOCTTOU Races

- Time-of-check-to-time-of-use (TOCTTOU) Race Attacks
- Some system calls enable checking of the file (check)
  - Does the requesting party have access to the file? (stat, access)
  - Is the file accessed via a symbolic link? (lstat)
- Some system calls use the file (use)
  - Convert the file name to a file descriptor (open)
  - Modify the file metadata (chown, chmod)
- Can an adversary modify the filesystem in between check and use system calls? Yes. Pretty reliably.

Vulnerabilities Easily Overlooked

- □ Manual checks can easily overlook vulnerabilities
- □ Misses file squat at line 03!



```
01 /* filename = /var/mail/root */
02 /* First, check if file already exists */
03 fd = open (filename, flg);
04 if (fd == -1) {
05     /* Create the file */
06    fd = open(filename, O_CREAT|O_EXCL);
07    if (fd < 0) {
08        return errno;
09    }
10 }
11 /* We now have a file. Make sure
12 we did not open a symlink. */
13 struct stat fdbuf, filebuf;
14 if (fstat (fd, &fdbuf) == -1)
15     return errno;
16 if (lstat (filename, &filebuf) == -1)
17     return errno;
18 /* Now check if file and fd reference the same file,
19    file only has one link, file is plain file.  */
20 if ((fdbuf.st_dev != filebuf.st_dev
21     || fdbuf.st_ino != filebuf.st_ino
22     || fdbuf.st_nlink != 1
23     || filebuf.st_nlink != 1
24     || (fdbuf.st_mode & S_IFMT) != S_IFREG)) {
25     error (_("%s must be a plain file
26         with one link"), filename);
27     close (fd);
28     return EINVAL;
29 }
30 /* If we get here, all checks passed.
31    Start using the file */
32 read(fd, ...)
```

Squat during create (resource)

Symbolic link

Hard link, race conditions

Current Defenses

- Are there defenses to prevent such attacks?
  - Yes, but the defenses are not comprehensive

Defenses

- Variants of the “open” system call
  - Flag “O_NOFOLLOW” – do not follow any symbolic links (prevent link traversal)
    - Does not help if you may need to follow symbolic links
    - May not be available on your system
  - Flag “O_EXCL” and “O_CREAT” – do not open unless the new file is created (prevent file squatting)
    - Does not help if you if your program does not know whether the file may need to be created
- These lack flexibility for protection in general

More Advanced Defenses

- The “openat” system call
  - Can open the directory (dirfd) separately from opening the file (path) to check the safety of that part of the name resolution
    - *int openat(int dirfd, const char *path, int oflag, ...);*
  - Control some aspects of opening “path” (e.g., no links)
    - E.g., used in libc

```
libc_open (const char *file, int oflag, …)
```

to

```
return SYSCALL_CANCEL (openat, AT_FDCWD, file, oflag, …);
```

- The “openat2” system call
  - More flags limiting “how” name resolution is done for “path”
  - Not standard

Openat Usage Example

- Suppose you want to open “/var/mail/root” safely with “openat”
  - How would you do it?

```
int openat(int dirfd, const char *path, int oflag, ...);
```

- Three steps
  - (1) Open “/var/mail” to obtain a “dirfd”
  - (2) Validate that the resulting file descriptor refers to “/var/mail”
  - (3) Open the file “root” using “openat” using options to protect the open from attacks
    - O_NOFOLLOW to prevent use of symbolic links (i.e., prevent link traversal)
    - O_EXCL with O_CREAT to ensure a fresh file is created (i.e., to prevent file squatting)

Validating Directories

- How do you validate a directory for “dirfd”?
- Three steps
  - (1) Open “/var/mail” to obtain its “fd”
  - (2) Collect the “stat” structure for this “fd”
    - From the file descriptor using fstat

    ```
    int fstat(int fd, struct stat *buf);
    ```
  - (3) Check that this “fd” refers to expected directory inode

    ```
    S_ISDIR(mode_t buf.st_mode);  // see ”struct stat” format
    Check value of st_ino field
    ```
- openat is a limited defense
  - Does not impact absolute pathnames or help if validation is imperfect

Conclusions

- Adversaries can attack your use of the filesystem
- Local exploit on shared access to the filesystem that your program may use in name resolution
  - If an adversary has write permission to any directory used
    - File squatting can control file content used by your program
    - Link traversal can redirect your program to other files
- Can use available system calls, such as openat, to prevent most forms of these attacks, but not all

Questions