CS260 – Advanced Systems Security

File System Security

May 7, 2025

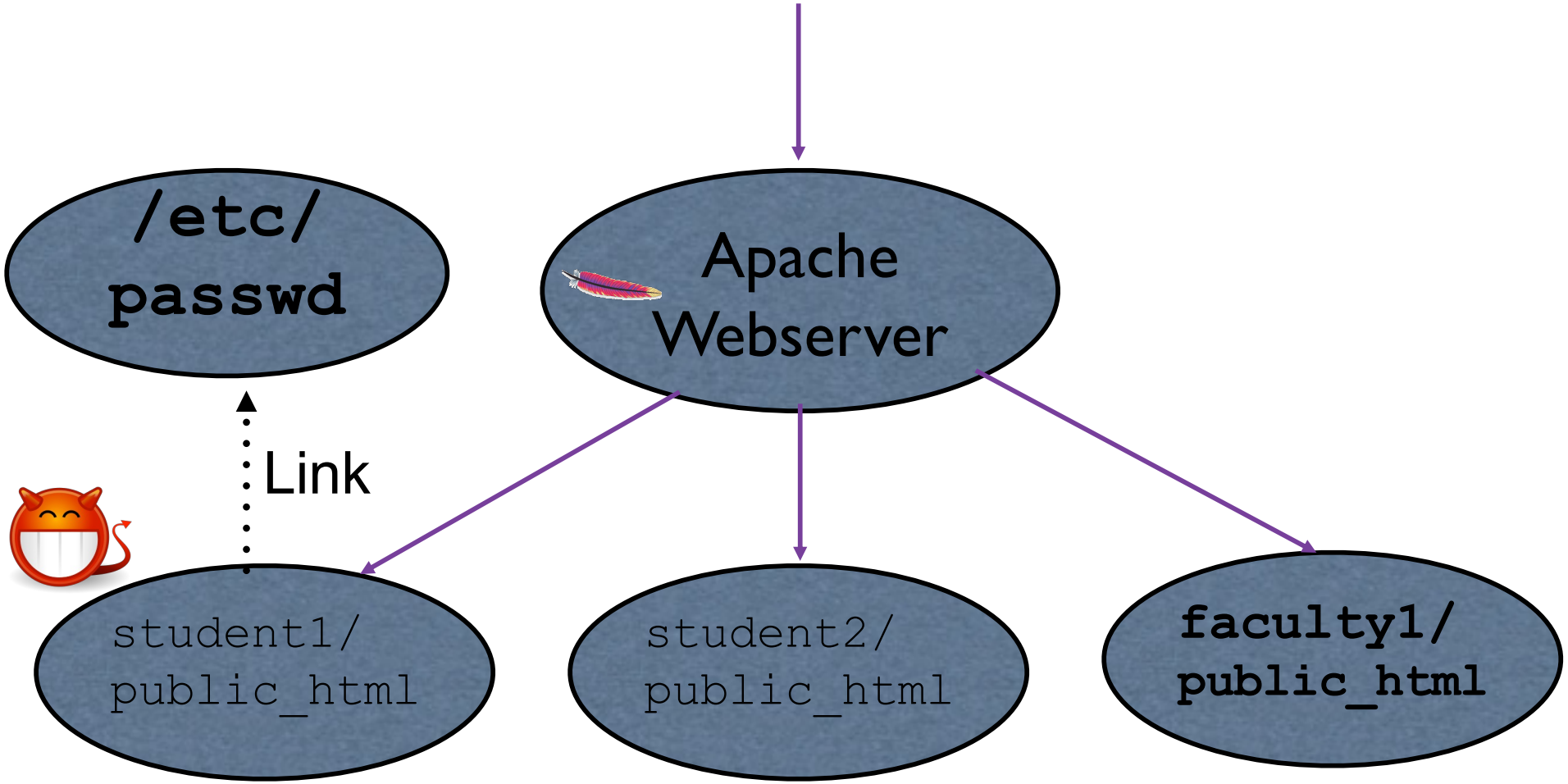
Opening Files

- Problem: Processes need resources from system
  - Just a simple `open(filename, …)` right?
  - But, adversaries can redirect victims to resources of their choosing

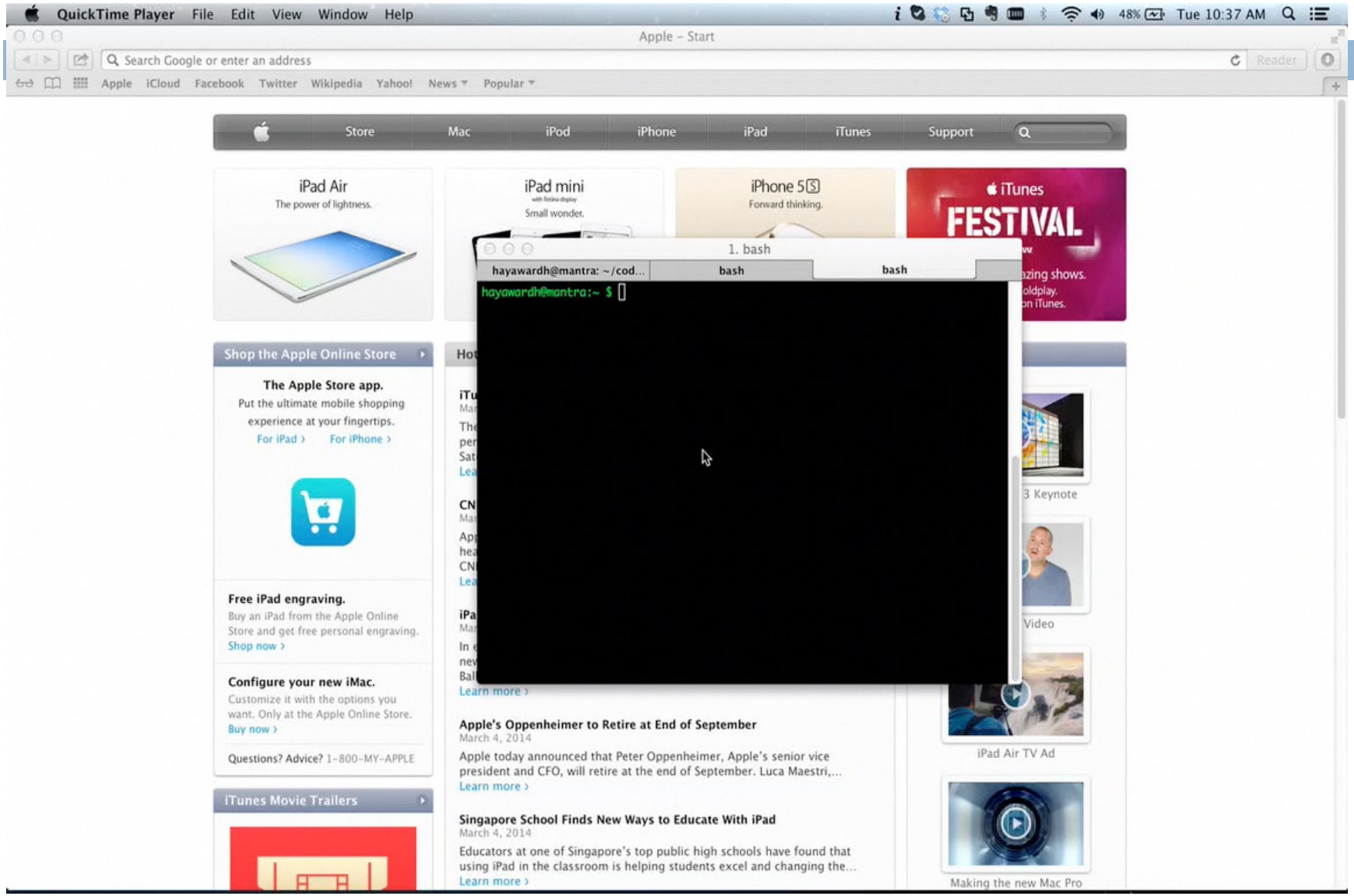
A Webserver's Story …

- Consider a university department webserver …

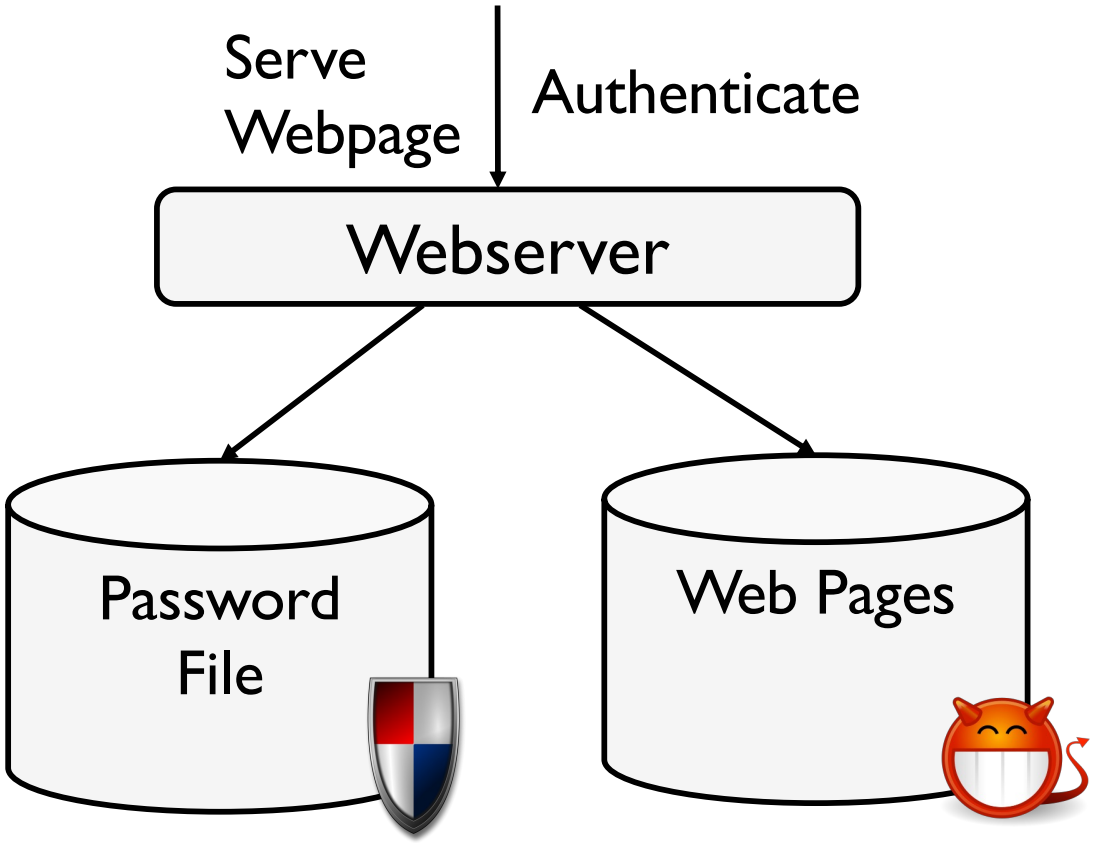
GET /~student1/index.html HTTP/1.1

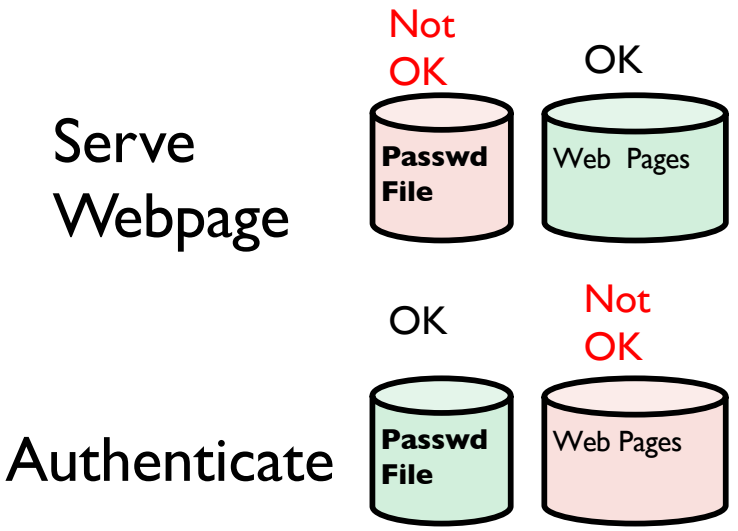


Attack Video



What Just Happened?

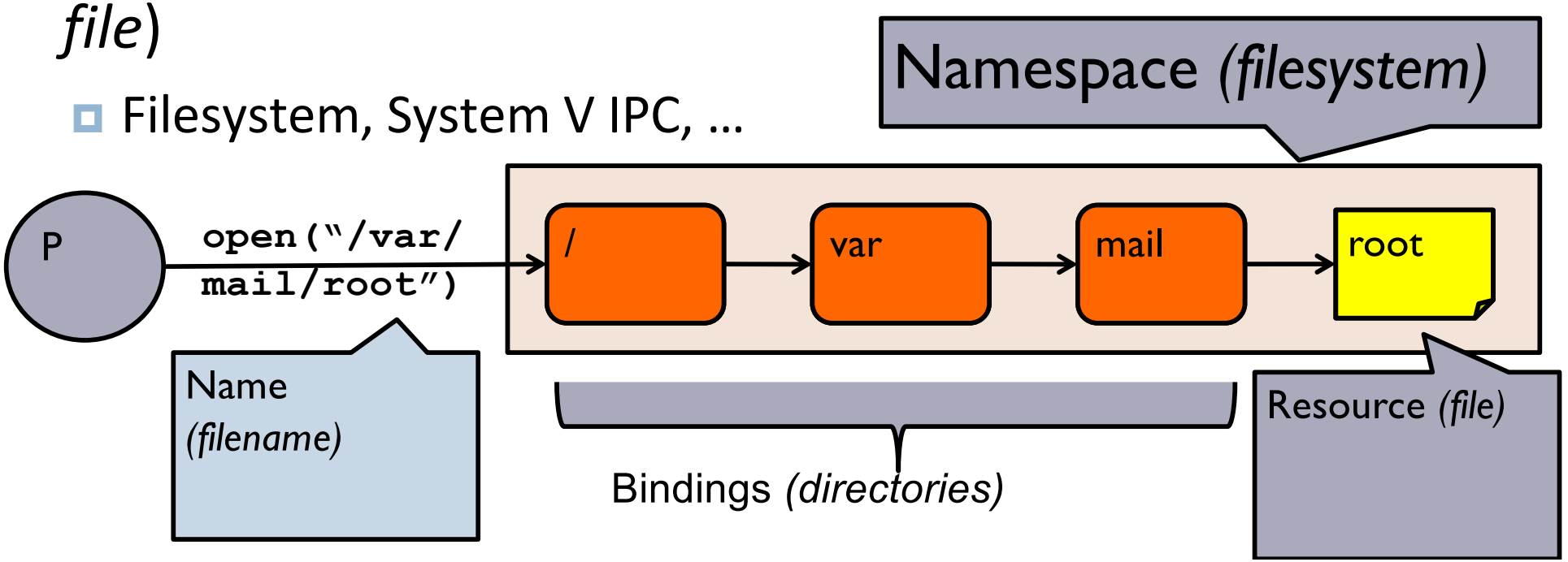




- Program acts as a *confused deputy*
  - when expecting
  - when expecting

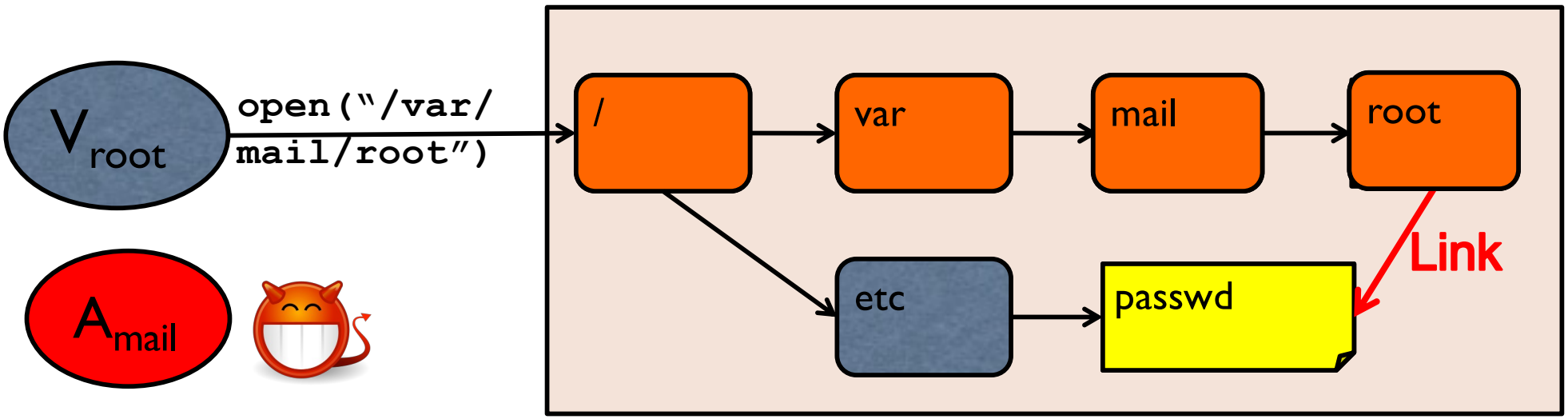
Name Resolution

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



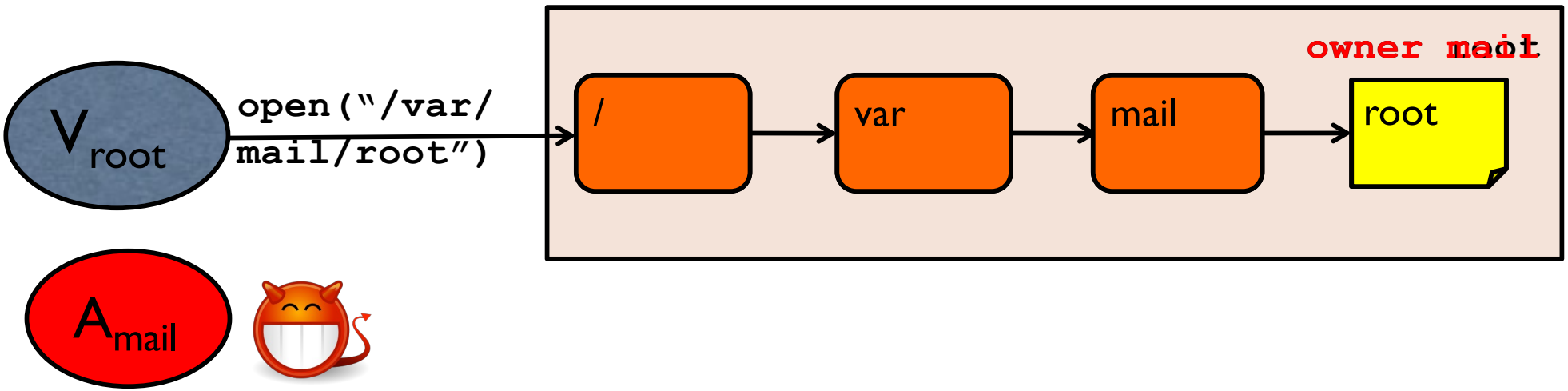
Link Traversal Attack

- Adversary controls bindings to direct a victim to a resource not normally accessible to the adversary
- Victim expects adversary-accessible resource, gets a protected resource instead
  - May take advantage of race conditions (TOCTTOU attacks)



File Squatting Attack

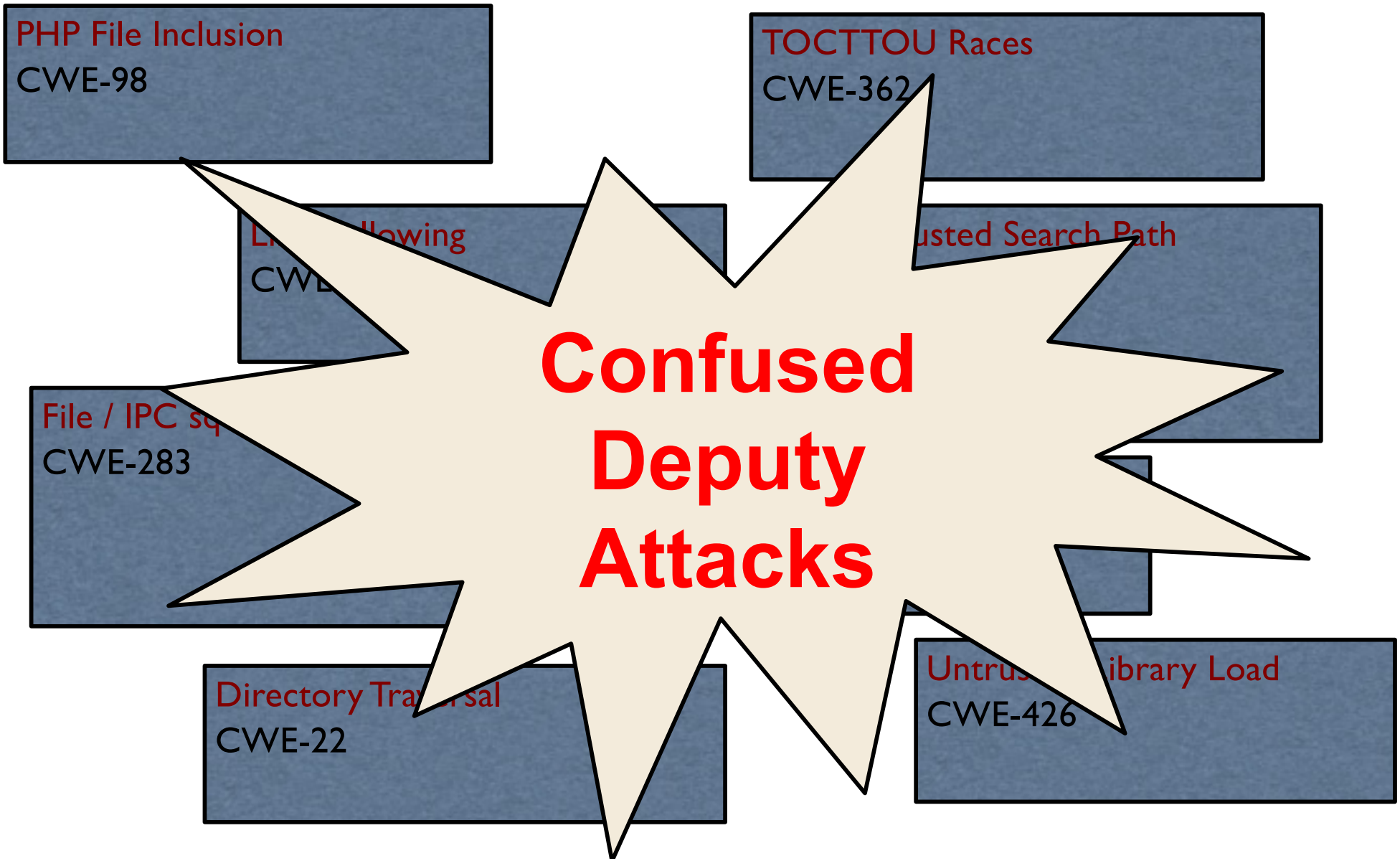
- Adversary controls final resource enabling the adversary to control input that the victim may depend on
- Victim expects protected resource, gets an adversary-controlled resource instead



TOCTTOU Attacks

- Time-of-check-to-time-of-use Attack
- Check System Call
  - Does the requesting party have access to the file? (stat, access)
  - Is the file accessed via a symbolic link? (lstat)
- Use System Call
  - Convert the file name to a file descriptor (open)
  - Modify the file metadata (chown, chmod)

Confused Deputy Attacks



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

Attacks 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

Mandatory Access Control

- Does MAC solve this problem?
  - What does SELinux say?

STING [USENIX 2012]

- We actively change the namespace whenever an adversary can write to a binding used in resolution
  - Fundamental problem: adversaries may be able to write directories used in name resolution

Runtime Analysis

- Run program and detect system call sequences that may be vulnerable
- Still, many false positives
  - Program code might defend itself
  - And may be inaccessible to adversaries
    - In our study, "only" 13% of adversary-accessible name resolutions are vulnerable
- False negatives
  - Attacks require special conditions
    - Current working directory, links, …

```
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, ...)
```

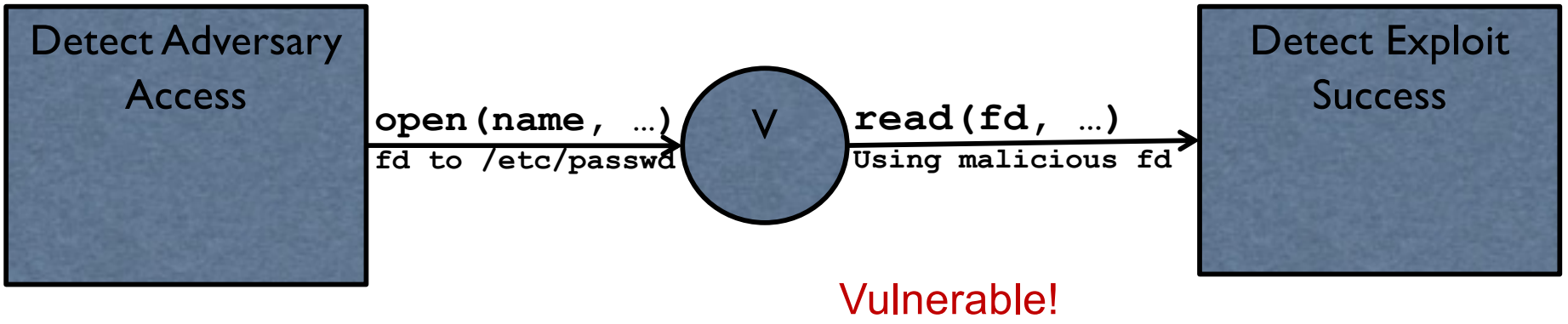
?

?

?

STING [USENIX 2012]

- Use adversary model to identify program adversaries and vulnerable directories [ASIACCS 2012]



STING Detects TOCTTOU Races

- STING can deterministically create races, as it is in the OS

Victim

Adversary

```
SOCKET_DIR=/tmp/.X11-unix

set_up_socket_dir () {
  if [ "$VERBOSE" != no ]; then
    log_begin_msg "Setting up $SOCKET_DIR..."
  fi
  if [ -e $SOCKET_DIR ] && [ ! -d $SOCKET_DIR ]; then
    mv $SOCKET_DIR $SOCKET_DIR.$$
  fi
  mkdir -p $SOCKET_DIR
  chown root:root $SOCKET_DIR
  chmod 1777 $SOCKET_DIR
  do_restorecon $SOCKET_DIR
  [ "$VERBOSE" != no ] && log_end_msg 0 || return 0
}
```

```
ln -s /etc/passwd
      /tmp/.X11-unix
```

Current Defenses

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

System-Only Defenses

- Will have false positives and/or false negatives [Cai et al., Oakland 2009]
  - System lacks information about programmer intent
- Thus, no system-only defenses beyond access control

- What can we do?

Program Defenses – Tell the System

- 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 Program 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 by libc for opens

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

to

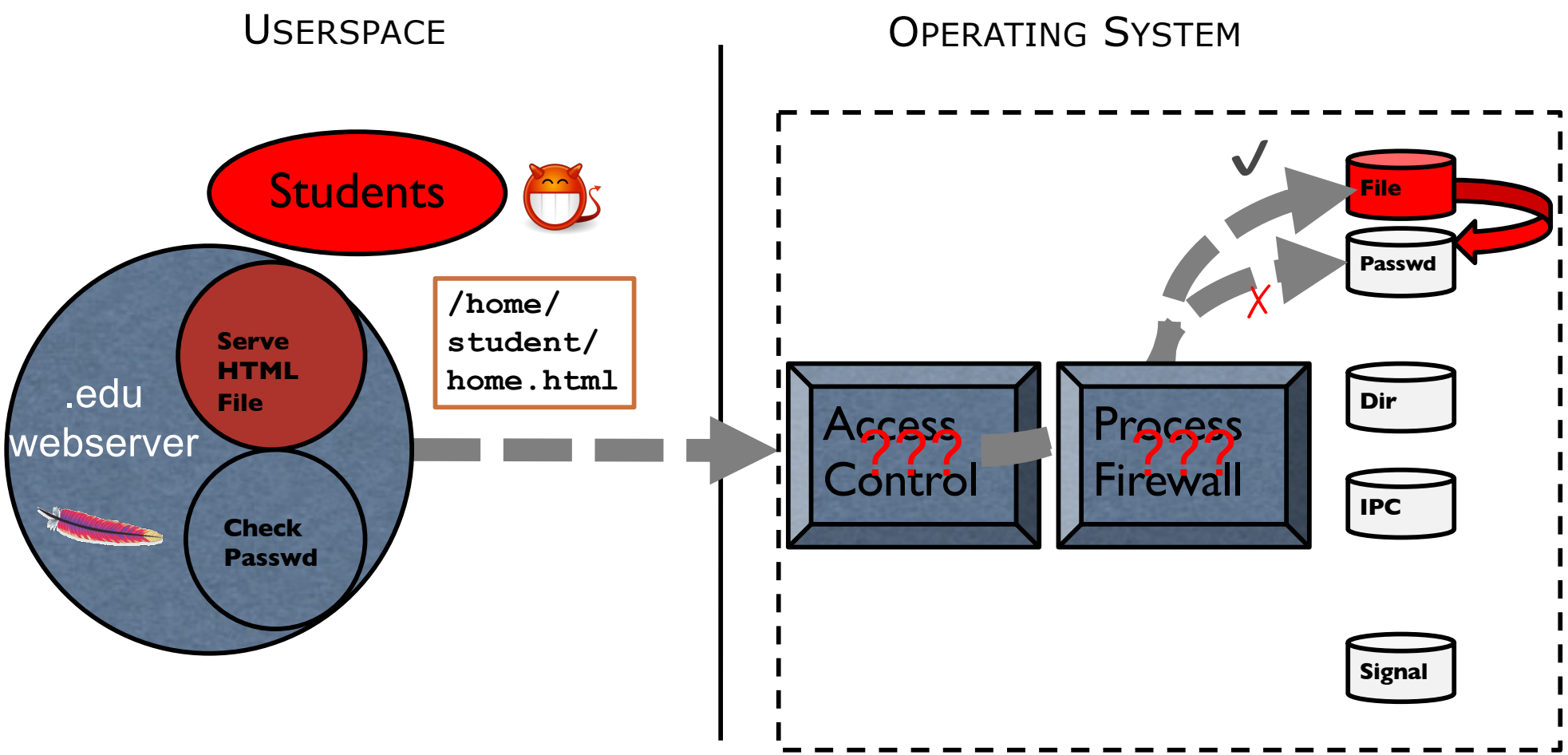
```
return SYSCALL_CANCEL (openat, AT_FDCWD, file, oflag, …);
```

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

Program-Aware System Defense

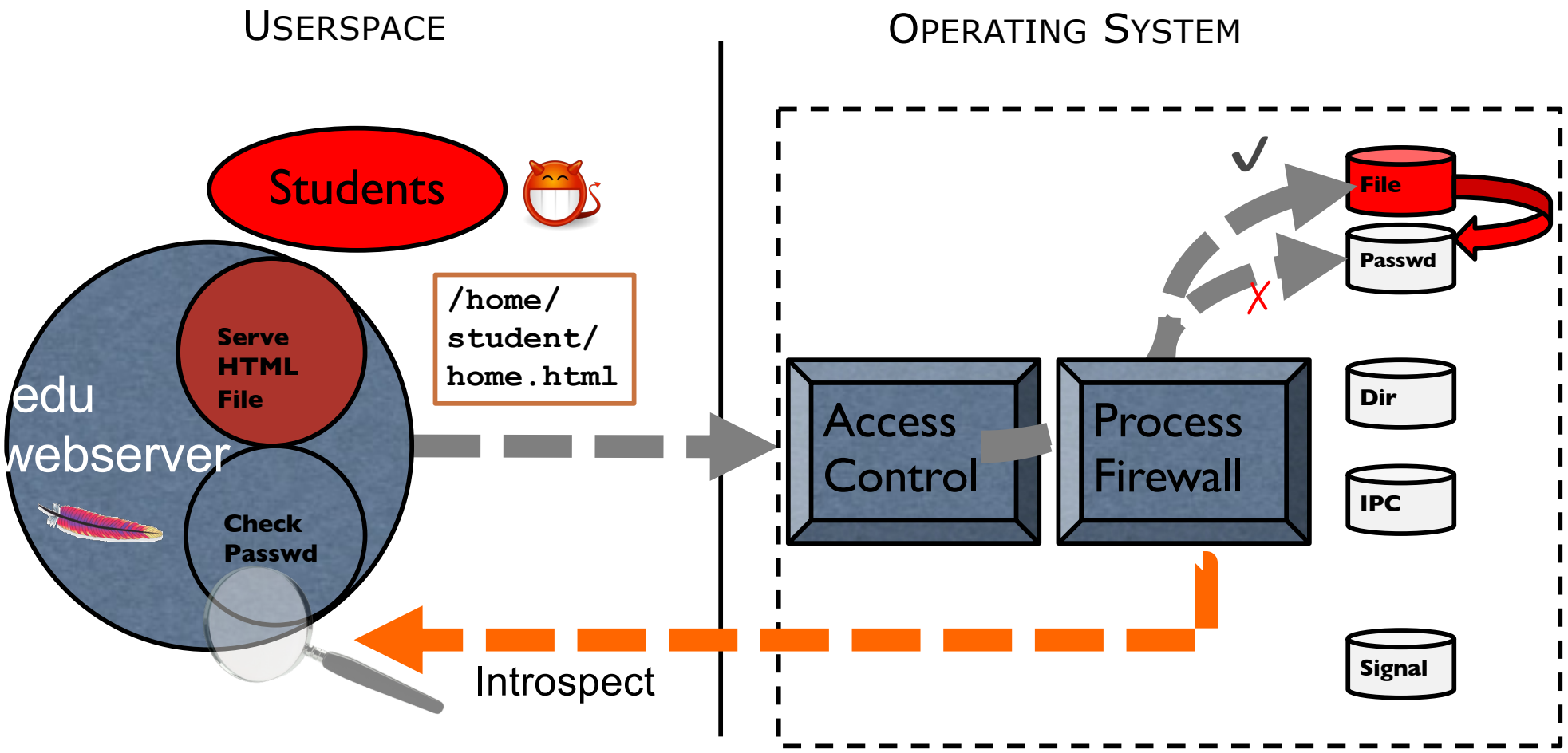
- Can the system do better with some further knowledge about the program?

Process Firewall



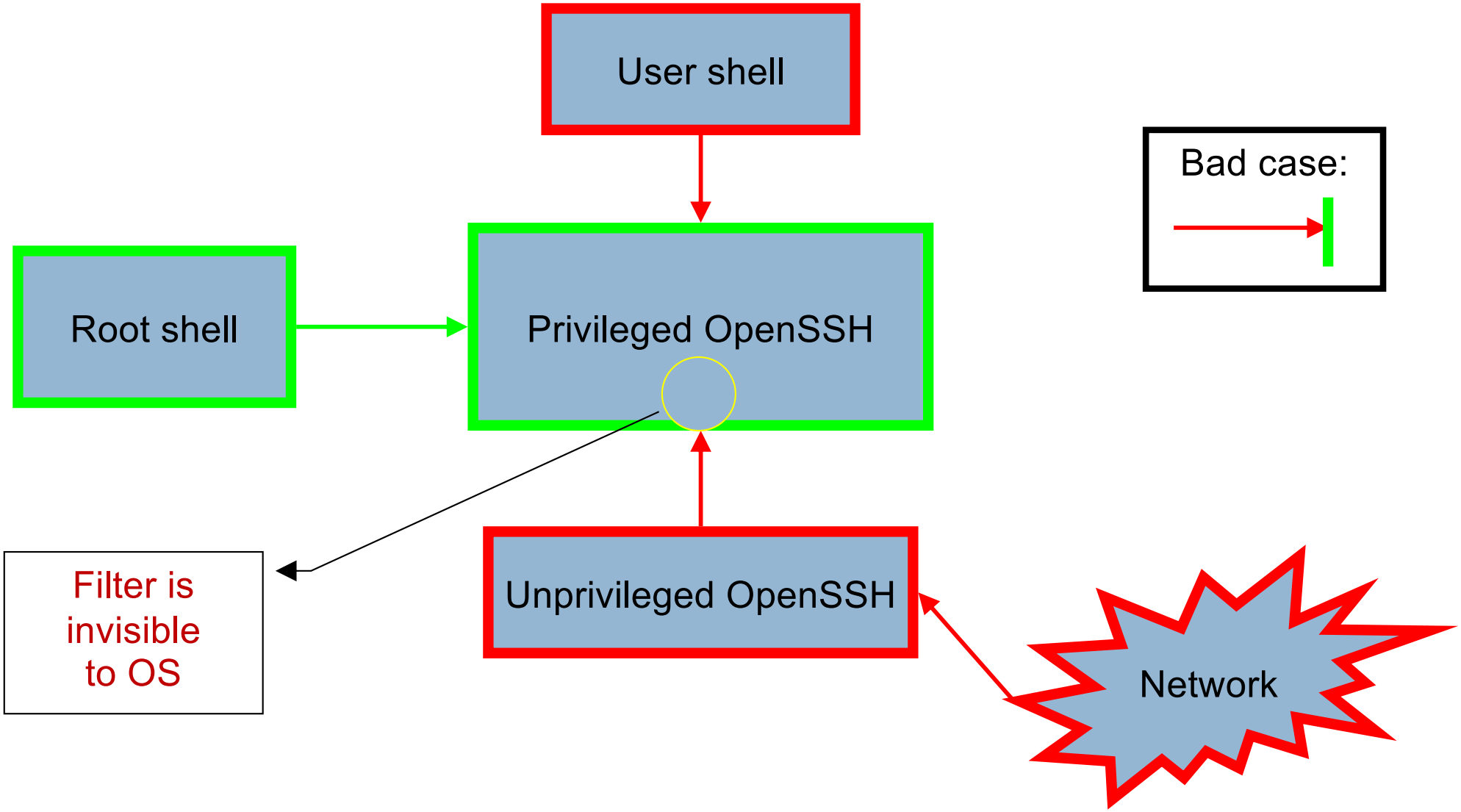
System defense per system call

Identify System Call



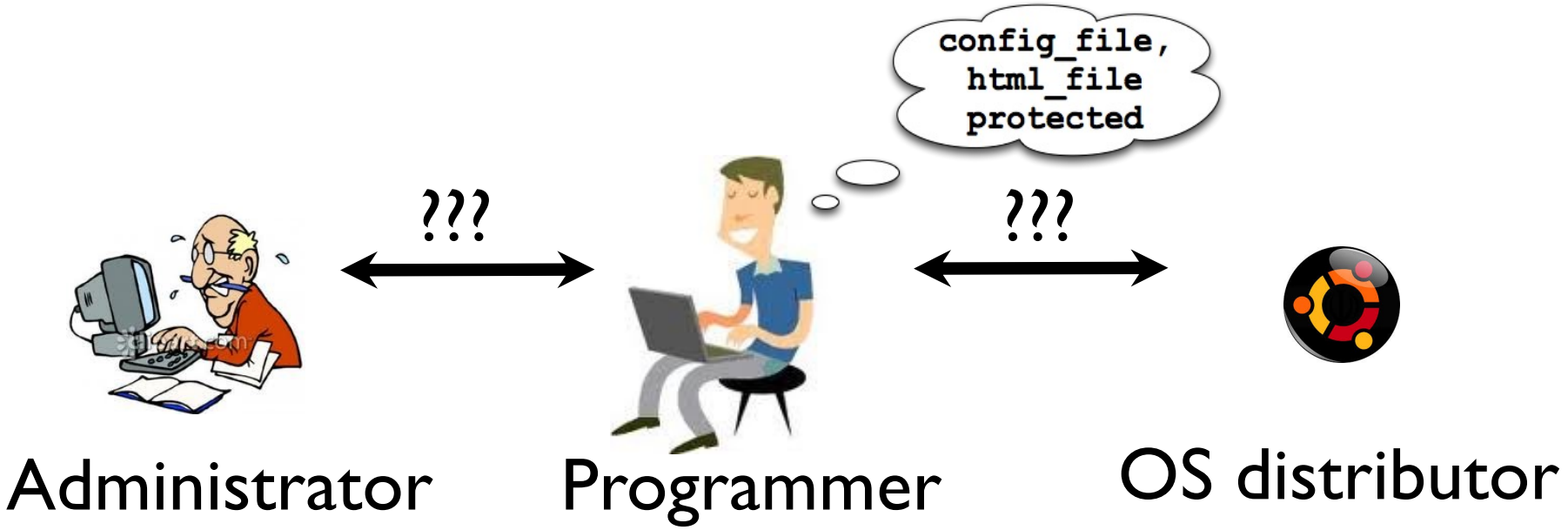
How do we distinguish among system calls?

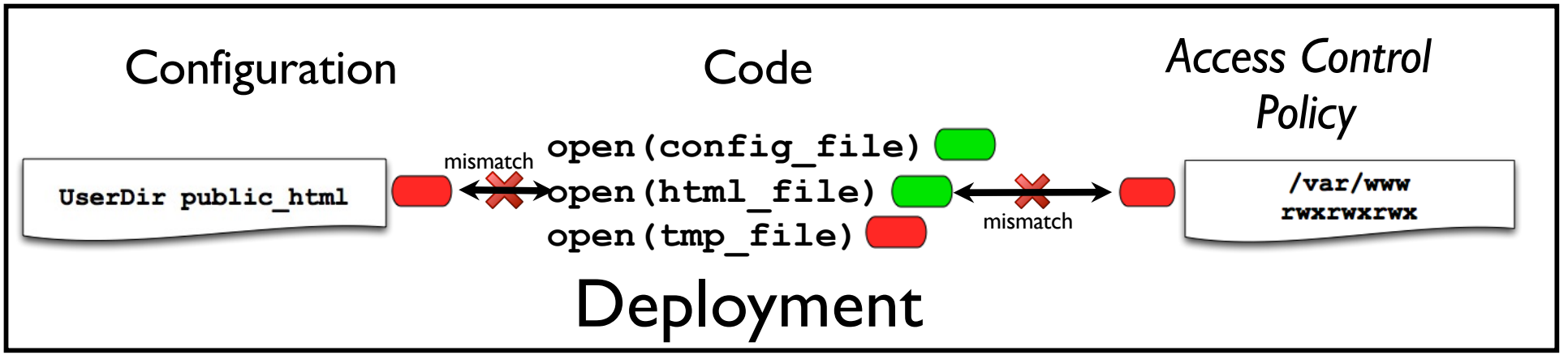
CW-Lite Enforcement



Cause - Multiple Parties

Expectations mismatch, blame each other





Capturing Expectations

- Match programmer expectation onto system
  - Irrespective of OS access control or admin configuration
  - If programmer expects to access only  , then they should not access 
    - Unexpected attack surface
  - If programmer expects  , then they should not access 
    - Classic confused deputy

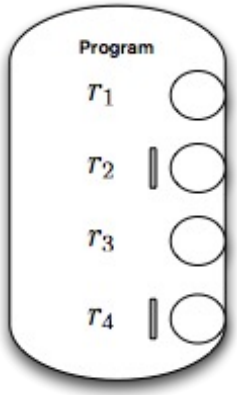
Solution Overview

- {*P*} - System calls where programmer expects adversary control
- {*S*} - System calls in deployment that adversaries actually control
- {*R*} - System calls in deployment that retrieve adversary-accessible resources

- When programmer expects no adversary control, block adversary-controlled system calls
  - Prevent unexpected adversary control: $S \subseteq P$
- When adversary control happens, limit adversary to accessible resources:
  - Prevent confused deputy: $for\ all\ x,\ if\ x\ in\ S \rightarrow x\ in\ R$

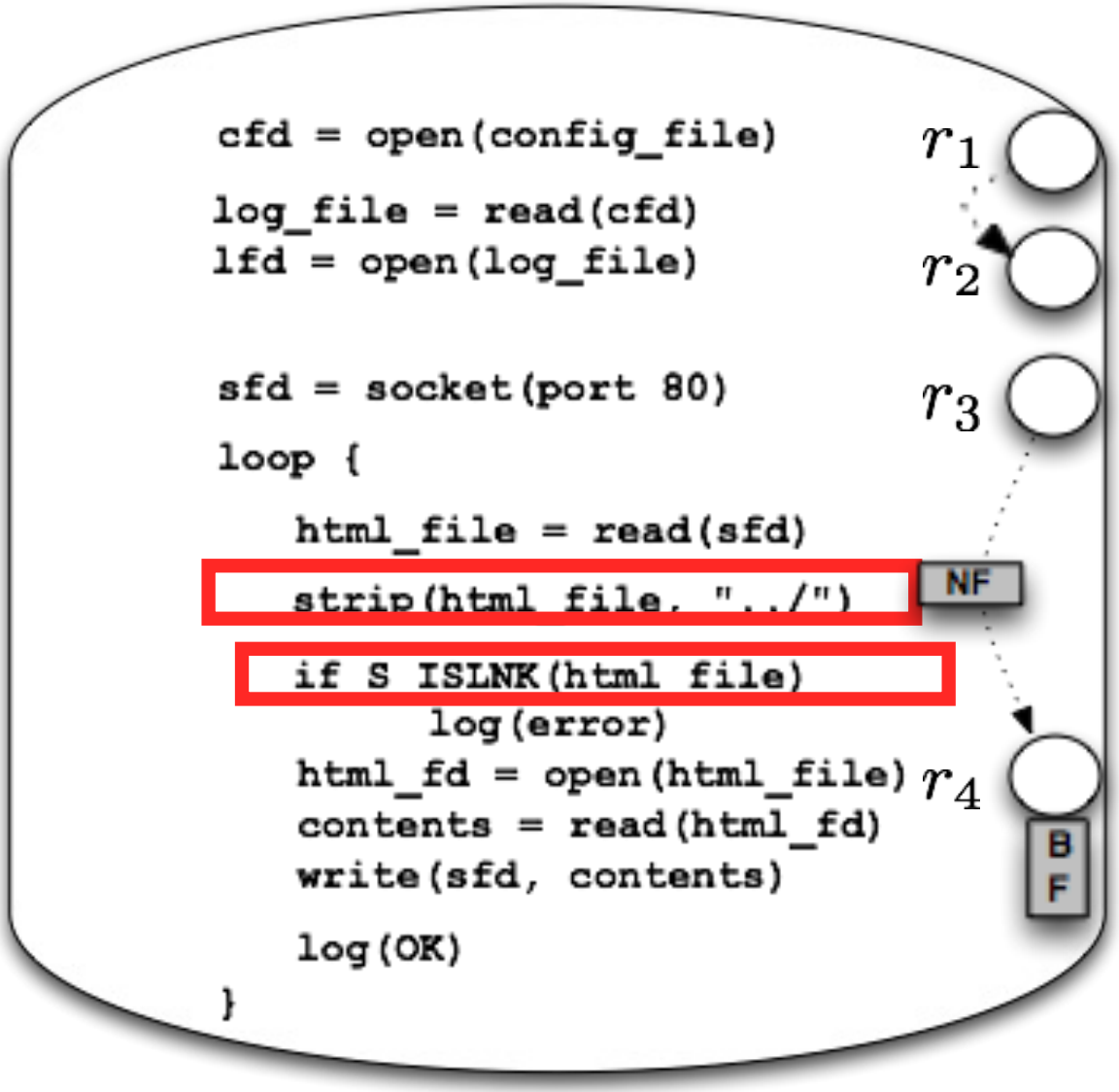
Programmer Expectations

- Can we determine where a programmer expects adversarial control of resource access?
- Strawman solution
  - Ask programmers to add annotations in code
- Insight: There are already annotations (sort of) --
  - *Filters (defensive code)*!



Resource Access Filters

- Write defensive checks (filters) to protect resource accesses
  - Name filters
  - Binding filters



Evaluation

Program	*Dev Tests?*	$\|V\|$	$\|E\|$	$\|V_f\|$	$\|E_f\|$	$\in P$	$\notin P$	*Impl. Exp. %*	*Missing*	*Redundant*	*Vulns.*	*Inv. 1s*	*Inv. 2s*
Apache v2.2.22	Yes*	20	23	7	5	7	13	65%	2	0	3	13	12
OpenSSH v5.3p1	Yes	17	17	14	0	14	3	17.6%	0	3	0	3	2
Samba3 v3.4.7	Yes	210	84	78	19	78	132	62.8%	0	5	0	132	40
Winbind v3.4.7	Yes	50	38	19	13	19	31	63.3%	0	0	0	31	28
Postfix v2.10.0	No	181	15	79	7	79	102	56.32%	0	9	0	102	15

- In 4/5 programs, programmers implicitly expect > 55% of resource accesses to never be adversary controlled in any deployment
  - OpenSSH most secure
- We found 2 missing checks that corresponded to 2 previously-unknown vulnerabilities and 1 default misconfiguration in the Apache webserver

.htpasswd Vulnerability

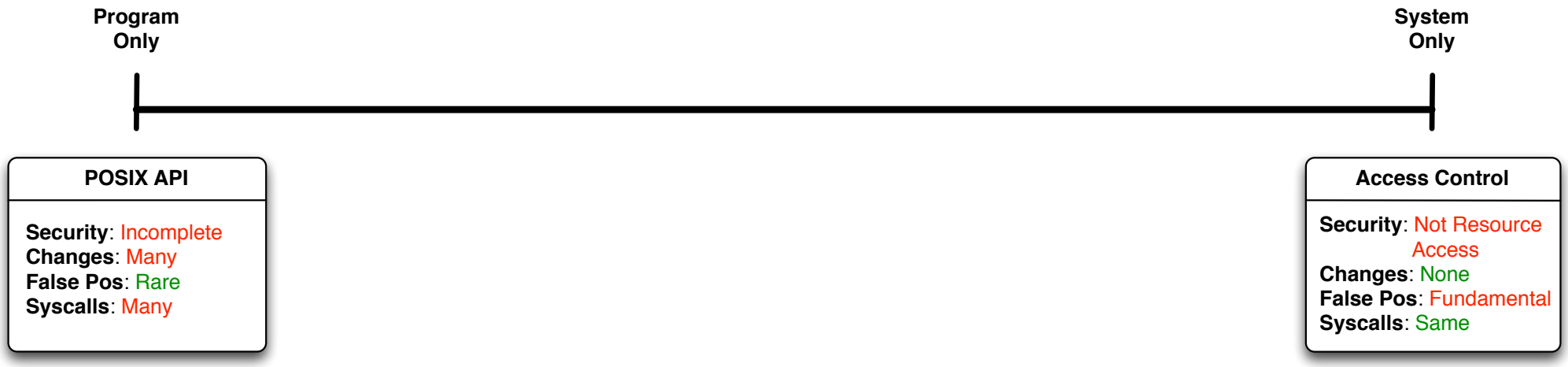
- Apache allows users to specify a password file to control access in .htaccess

```
AuthUserFile /home/userh/.htpasswd
AuthType Basic
AuthName "My Files"
Require valid-user
```

- Neither name flow nor binding is filtered
  - User can specify any password file, even of other users, or the system-wide /etc/passwd (if in proper format)
- Can be used to brute-force passwords
  - No rate limit on HTTP auth (unlike terminal logins)
- Vulnerability hidden all these years, showing importance of automated and principled reasoning of resource access

Alternatives for Defense

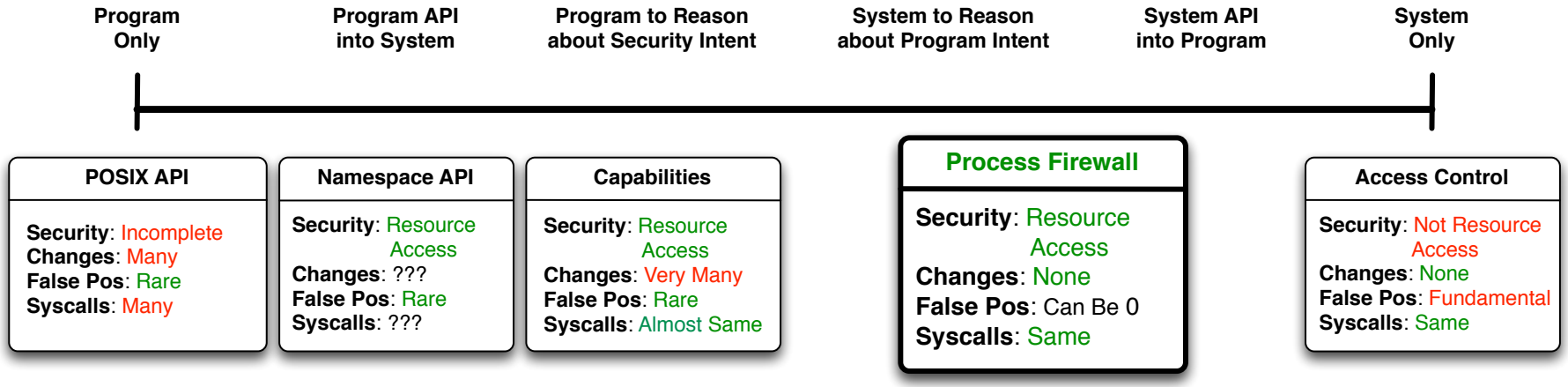
- Program-only (even with POSIX API extensions) and system-only defenses are not effective



- What are the intermediate options?

System Defense with Program Intent

- Process Firewall fits a niche between system-only defenses and the program extensions to reason about security – Key Insight: only protects processes



- Block adversaries from tampering with unprotected resource accesses and causing confused deputy attacks – Challenge: still need to extract intent

Questions