Systems and Internet Infrastructure Security

Network and Security Research Center
Department of Computer Science and Engineering
Pennsylvania State University, University Park PA

CMPSC 447
Confused Deputy

Trent Jaeger
Systems and Internet Infrastructure Security (SIIS) Lab
Computer Science and Engineering Department
Pennsylvania State University

Android External Storage

- Android has its apps use "external storage" (used to be an SD-card) to store its code and configurations
  - A shared filesystem space for use by apps
- **Problem**: Multiple apps can write files in the same directories
  - Why could that be a problem?

Android External Storage

- Android has its apps use “external storage” (used to be an SD-card) to store its code and configurations
  - A shared filesystem space for use by apps
- Problem: Multiple apps can write files in the same directories
  - Why could that be a problem?
- A malicious app that knows the name of a file that will be created by another app can create that file in advance
  - E.g., for library files
  - E.g., for symbolic links
  - Why could these cause an issue?

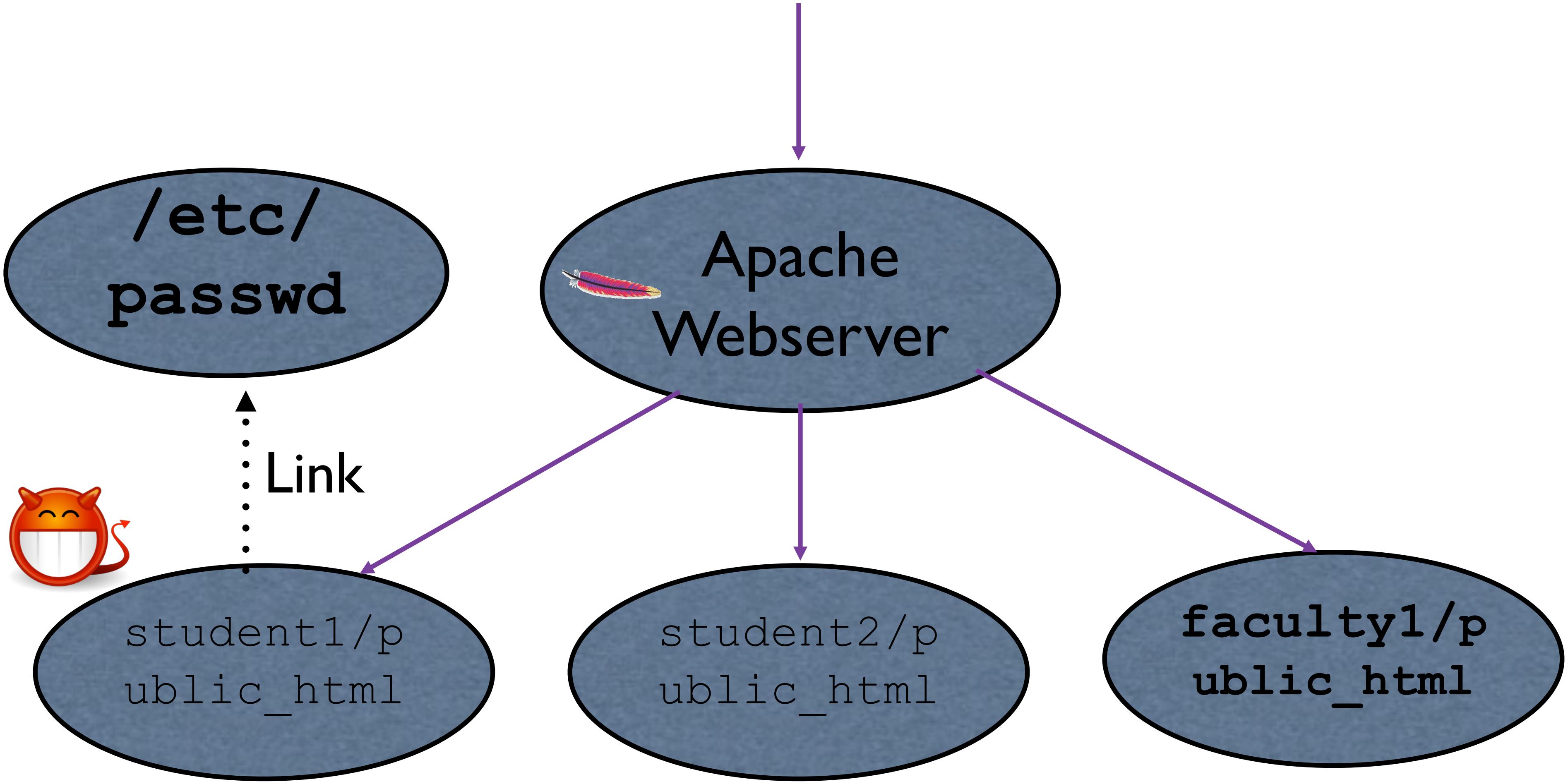
File Open

- 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

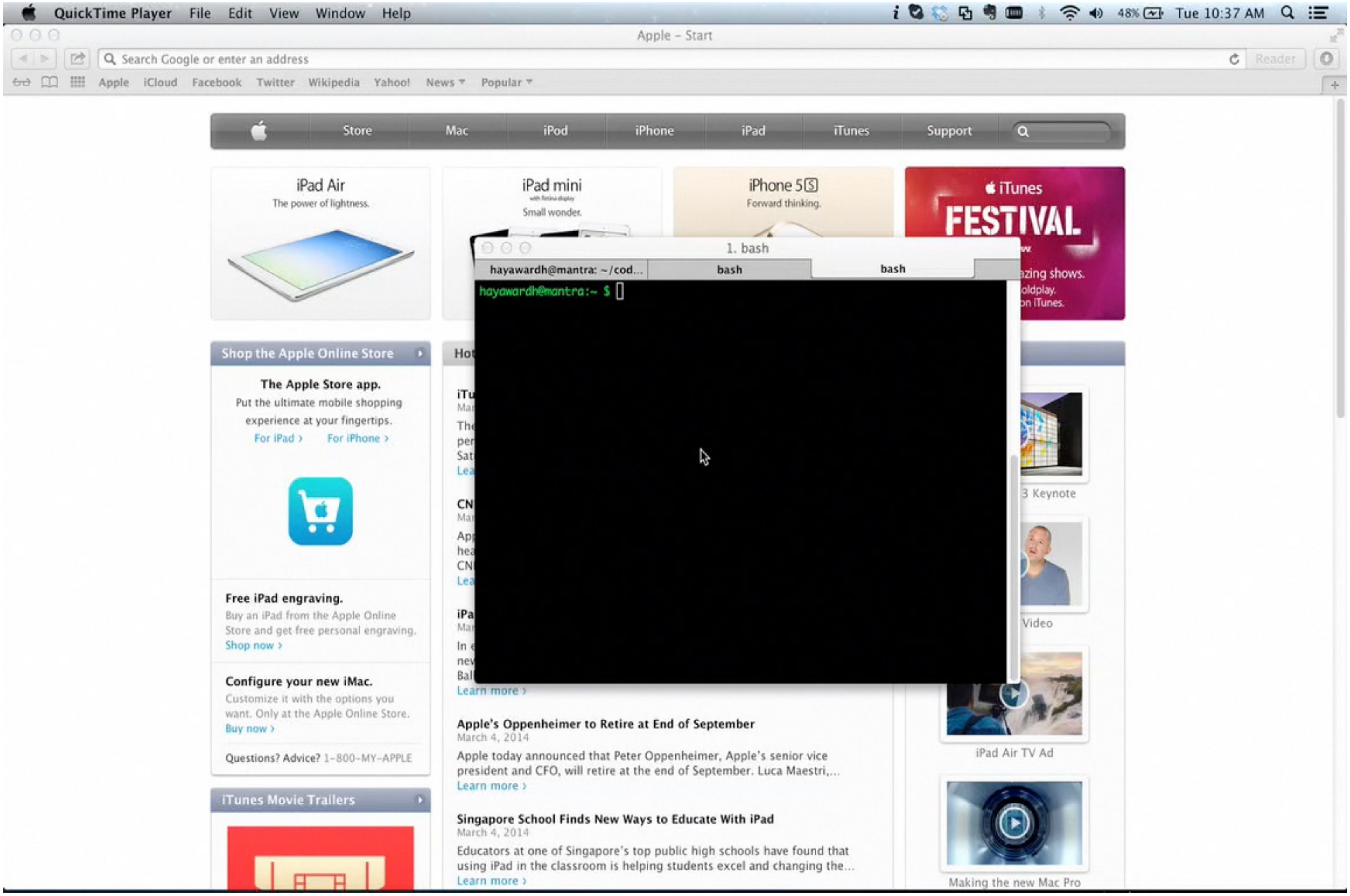
Link

student1/public_html

student2/public_html

faculty1/public_html

Attack Video



```
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 ›

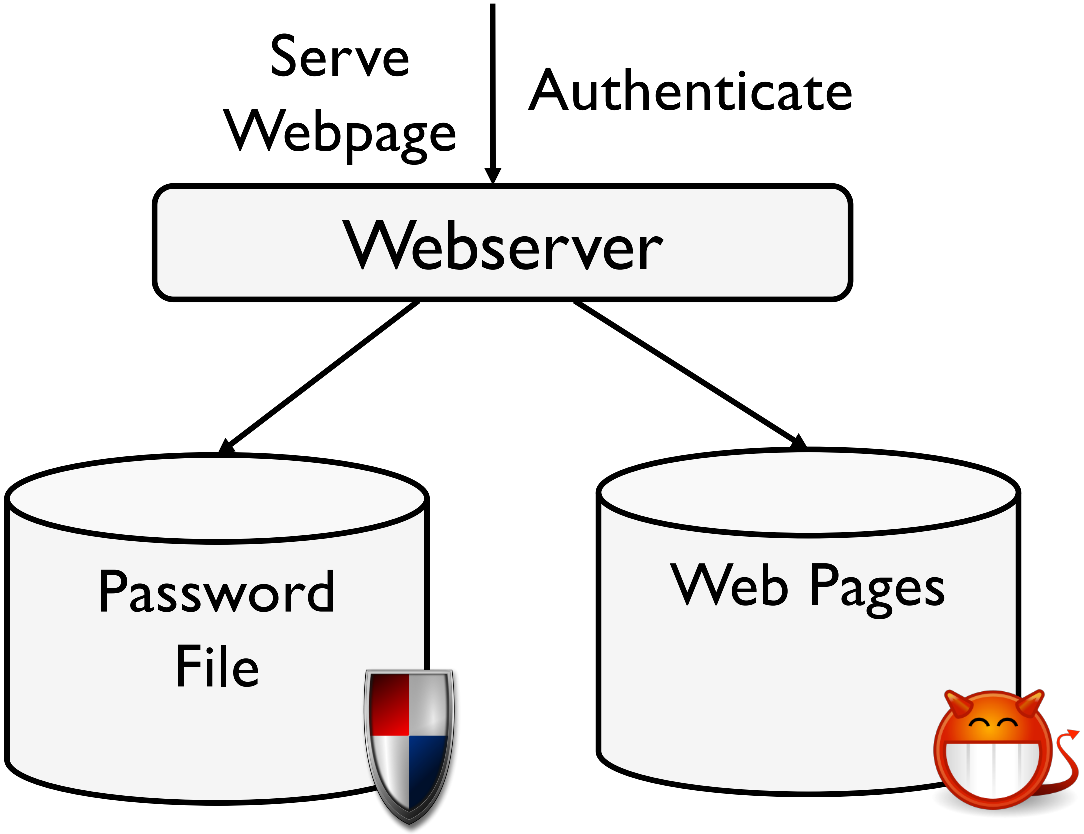
Singapore School Finds New Ways to Educate With iPad

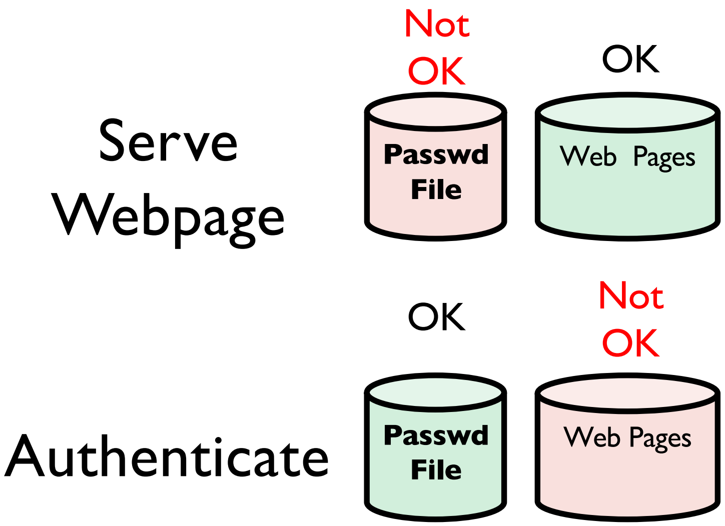
March 4, 2014

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?



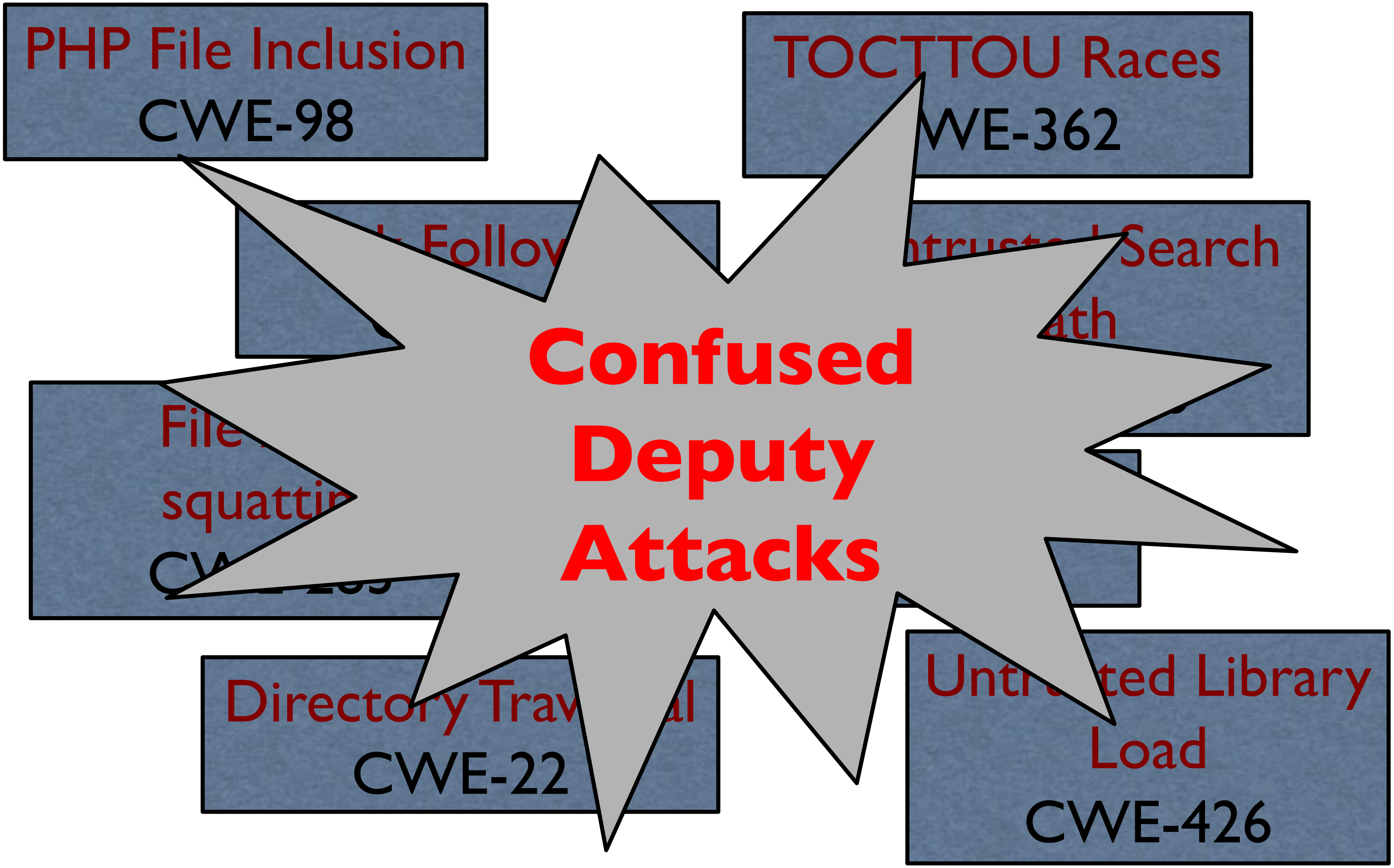


- Program acts as a *confused deputy*
  - 😈 when expecting 🛡
  - 🛡 when expecting 😈

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
WE-362

k Follow

ntrusted Search
ath

File
squatti
CW

Directory Trav al
CWE-22

Untr ted Library
Load
CWE-426

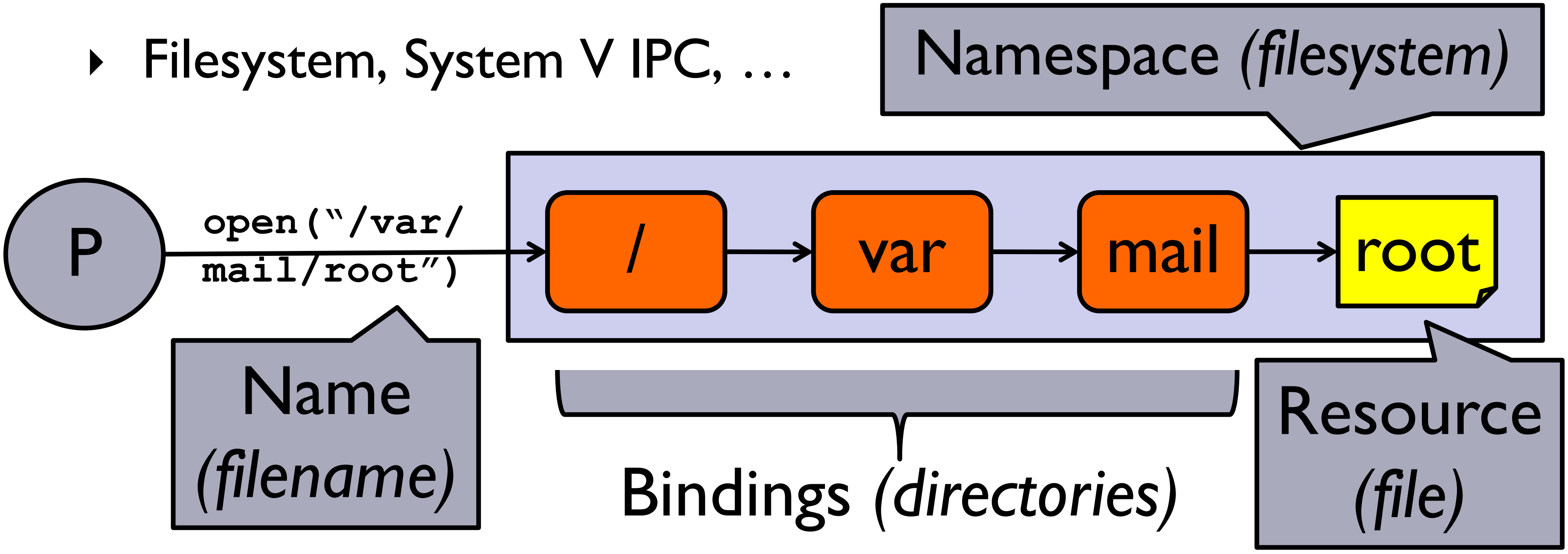
Confused Deputy Attacks

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

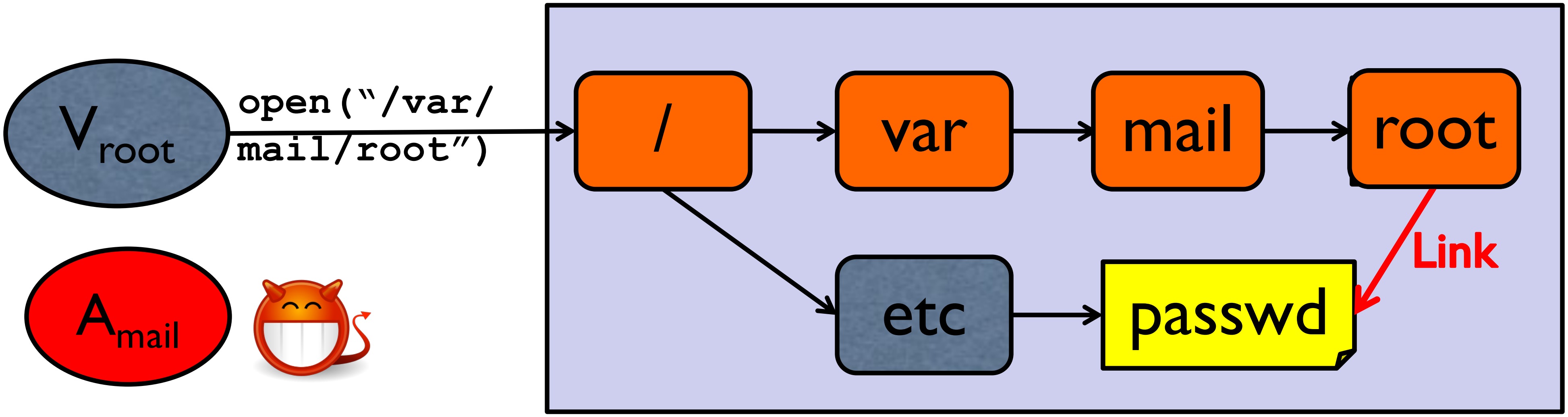
Name Resolution

- Processes often use *names* to obtain access to *system resources*
- A *nameserver* (e.g.,OS) performs *name resolution* using *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 links to direct a victim to a resource not normally accessible to the adversary
- Victim expects adversary-accessible resource, gets a protected resource instead



V_{root}

`open(“/var/mail/root”)`

A_{mail}

/

var

mail

root

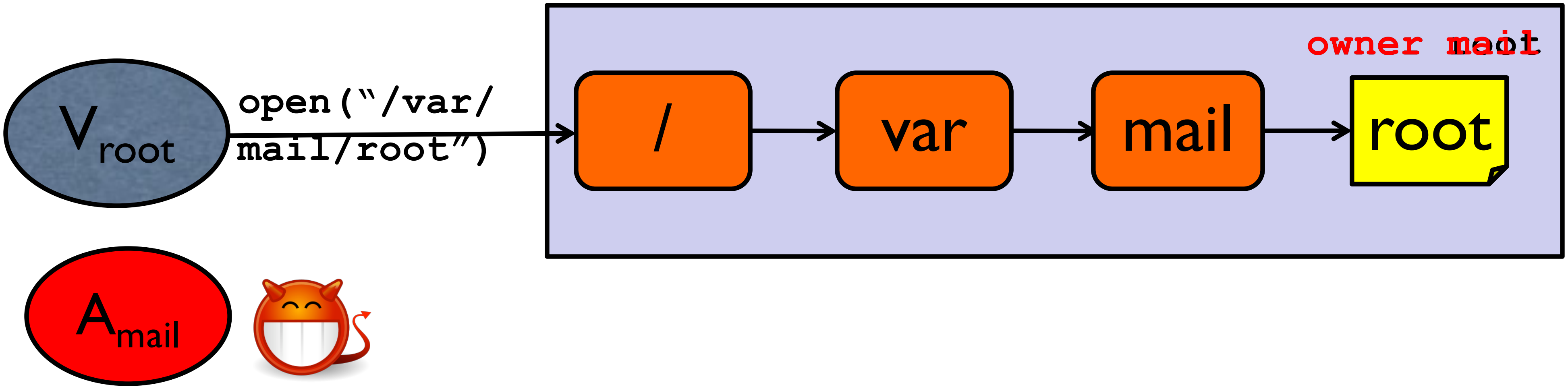
etc

passwd

Link

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



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

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 asks to open the file path "/tmp/just_a_normal_file_here"
  - What file will you 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

Link Traversal

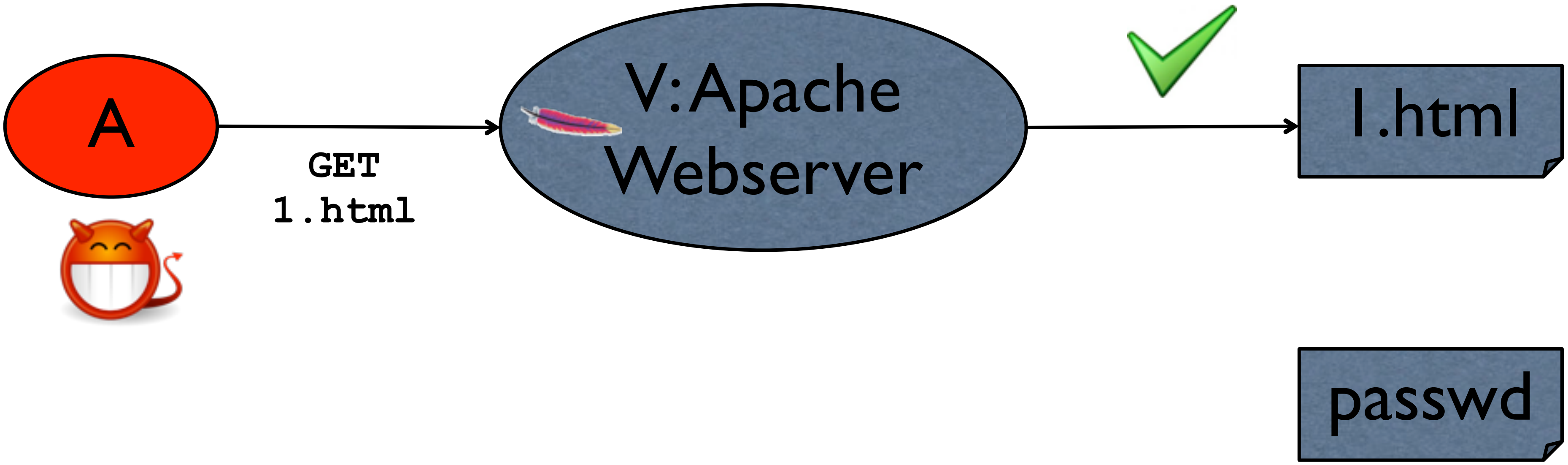
- Suppose your program is asked to open the file path “/tmp/just_a_normal_file_here”
  - What file will you open?
    - An adversary could have created this as a symbolic link to any file in the system
    - And it is difficult/expensive to verify that this is not a symbolic link
      - lstat – provides file system information (like “stat”) for the file referenced by a link if the path name refers to a link
      - RACES: But, adversary could place a file at the time of the lstat check and replace with a link before the open
  - Causes your program to access an adversary-chosen file

TOCTTOU Races

- Time-of-check-to-time-of-use Race Attacks
- Check System Calls
  - Does the requesting party have access to the file? (stat, access)
  - Is the file accessed via a symbolic link? (lstat)
- Use System Calls
  - Convert the file name to a file descriptor (open)
  - Modify the file metadata (chown, chmod)
- Can an adversary modify the filesystem in between?

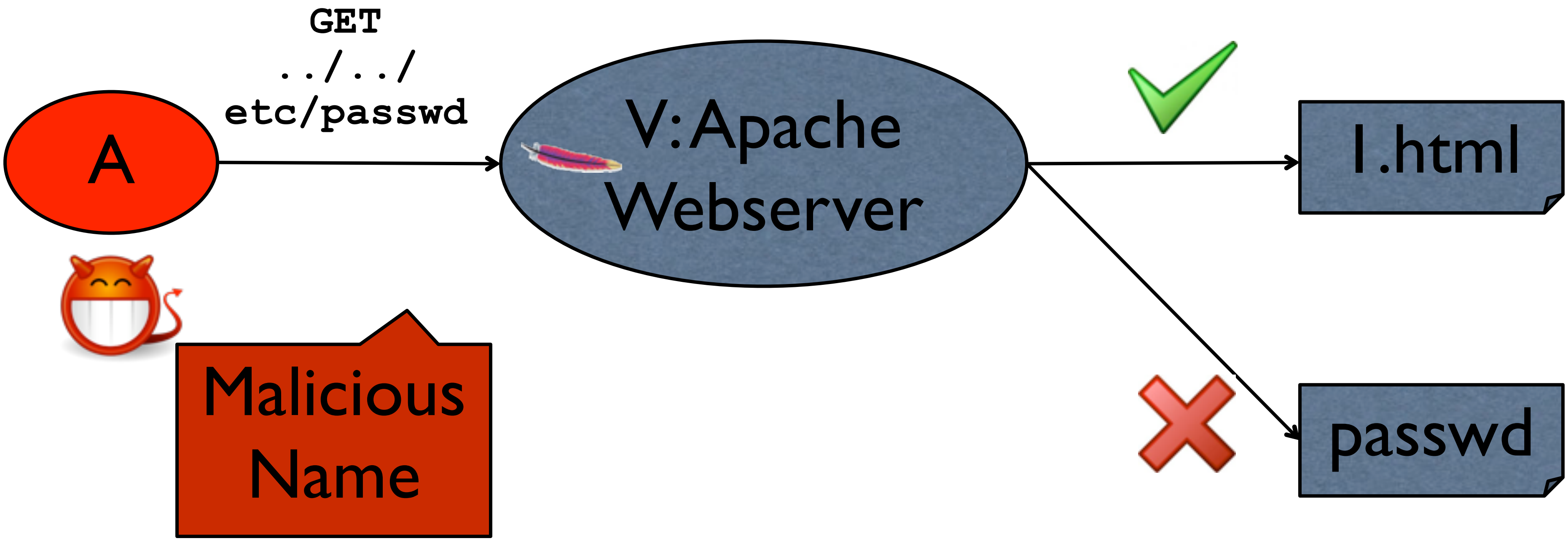
Directory Traversal

- Adversary controls the name to direct victim to an adversary inaccessible (high integrity) resource



Directory Traversal

- Adversary controls the name to direct victim to an adversary inaccessible (high integrity) resource
- Victim expects adversary accessible (low integrity) resource



Common Threat

- What is the threat that enables directory traversal attacks?

Common Threat

- What is the threat that enables directory traversal attacks?
- In this case, the victim uses adversary input to construct file names
  - Any parts of file names

File Name Input

- Suppose your program uses network input to construct a file name
  - What can go wrong?

File Name Input

- Suppose your program uses network input to construct a file name
  - What can go wrong?
- Suppose your program appends network input to the path "/tmp/" to open the file /tmp/<input>
  - Safe?

Common Threat

- What is the threat that enables directory traversal attacks?
- Suppose your program appends input to the path "/tmp/" to open the file /tmp/<input>
  - Safe?
  - No. An adversary could input: "../etc/shadow"
    - What file will be opened?

Overall Lesson

- What is the takeaway lesson from all these vulnerabilities?

Overall Lesson

- What is the takeaway lesson from all these vulnerabilities?
  - Any time you use adversary-controlled inputs in your programs you must be careful to vet that input
    - The same for using program input and filesystem resources as input
  - Does this correspond to any security principle you learned in CMPSC 443?

Overall Lesson

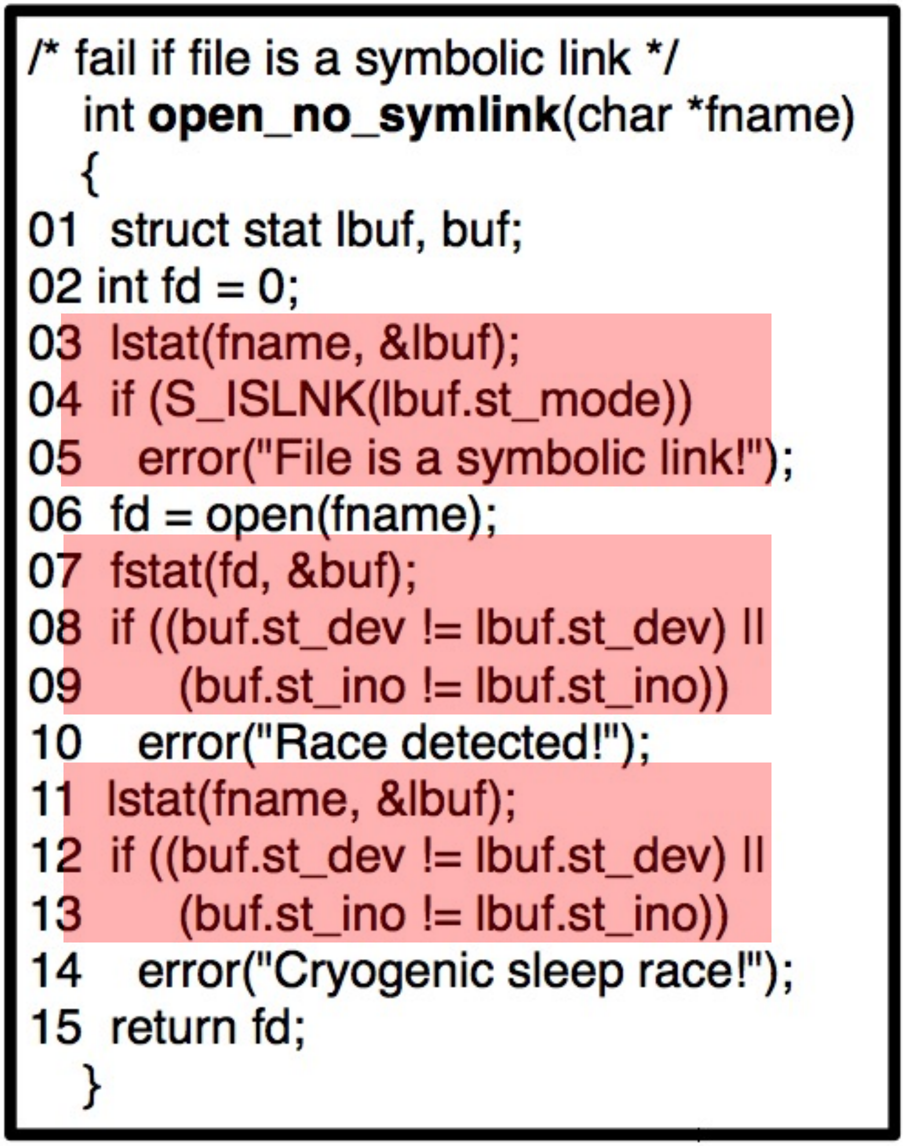
- What is the takeaway lesson from all lthese vulnerabilities?
  - Any time you use adversary-controlled inputs in processing you must be careful to vet that input
  - Does this correspond to any security principle you learned in CMPSC 443?
    - How about Biba integrity?
    - Low-water mark integrity?
    - Clark-Wilson integrity?

Current Defenses

- Are there defenses to prevent such attacks?
- For filesystem inputs (file squat and link traversal)
  - Yes, but the defenses are not comprehensive
- For using inputs to construct filenames (directory traversal)
  - No, you are on your own
  - Some research defenses have been proposed, but need to know about the program
    - May need programmers to do more in the future

Open_No_Symlink Defense

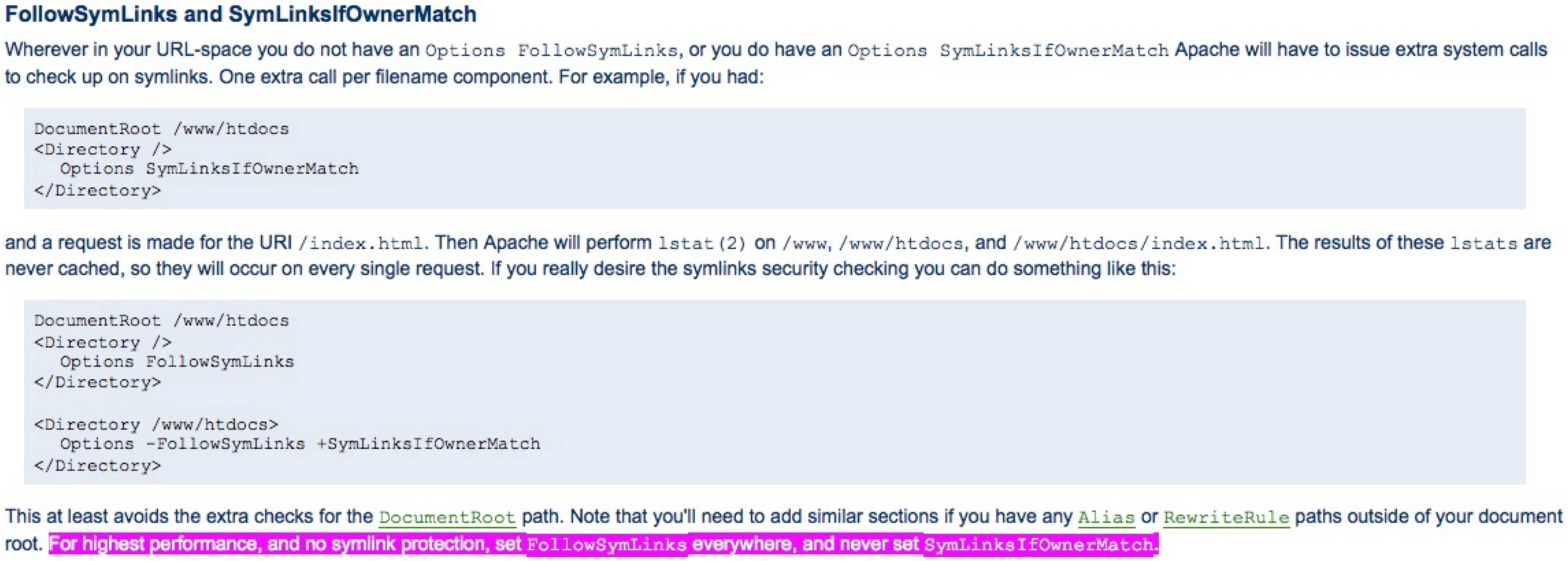
- Check for symbolic link (lstat)
- Check for lstat-open race
- Check for inode recycling
- Do checks for each path component (`safe_open`)
    - /, var, mail, …
- What if you want to use symlinks – just safely?



```
/* fail if file is a symbolic link */
   int open_no_symlink(char *fname)
   {
   struct stat lbuf, buf;
  int fd = 0;
   lstat(fname, &lbuf);
   if (S_ISLNK(lbuf.st_mode))
     error("File is a symbolic link!");
   fd = open(fname);
   fstat(fd, &buf);
   if ((buf.st_dev != lbuf.st_dev) II
        (buf.st_ino != lbuf.st_ino))
     error("Race detected!");
   lstat(fname, &lbuf);
   if ((buf.st_dev != lbuf.st_dev) II
        (buf.st_ino != lbuf.st_ino))
     error("Cryogenic sleep race!");
   return fd;
   }
```

Problem - Inefficient

- Checking retrieved resources is expensive
  - Single open() requires *4 * path length* additional syscalls
  - Programmers omit checks to improve performance
    - Example: Apache documentation recommended switching off resource access checks



FollowSymLinks and SymLinksIfOwnerMatch

Wherever in your URL-space you do not have an `Options FollowSymLinks`, or you do have an `Options SymLinksIfOwnerMatch` Apache will have to issue extra system calls to check up on symlinks. One extra call per filename component. For example, if you had:

```
DocumentRoot /www/htdocs
<Directory />
  Options SymLinksIfOwnerMatch
</Directory>
```

and a request is made for the URI `/index.html`. Then Apache will perform `lstat(2)` on `/www`, `/www/htdocs`, and `/www/htdocs/index.html`. The results of these `lstats` are never cached, so they will occur on every single request. If you really desire the symlinks security checking you can do something like this:

```
DocumentRoot /www/htdocs
<Directory />
  Options FollowSymLinks
</Directory>

<Directory /www/htdocs>
  Options -FollowSymLinks +SymLinksIfOwnerMatch
</Directory>
```

This at least avoids the extra checks for the `DocumentRoot` path. Note that you'll need to add similar sections if you have any `Alias` or `RewriteRule` paths outside of your document root. For highest performance, and no symlink protection, set `FollowSymLinks` everywhere, and never set `SymLinksIfOwnerMatch`.

Defenses

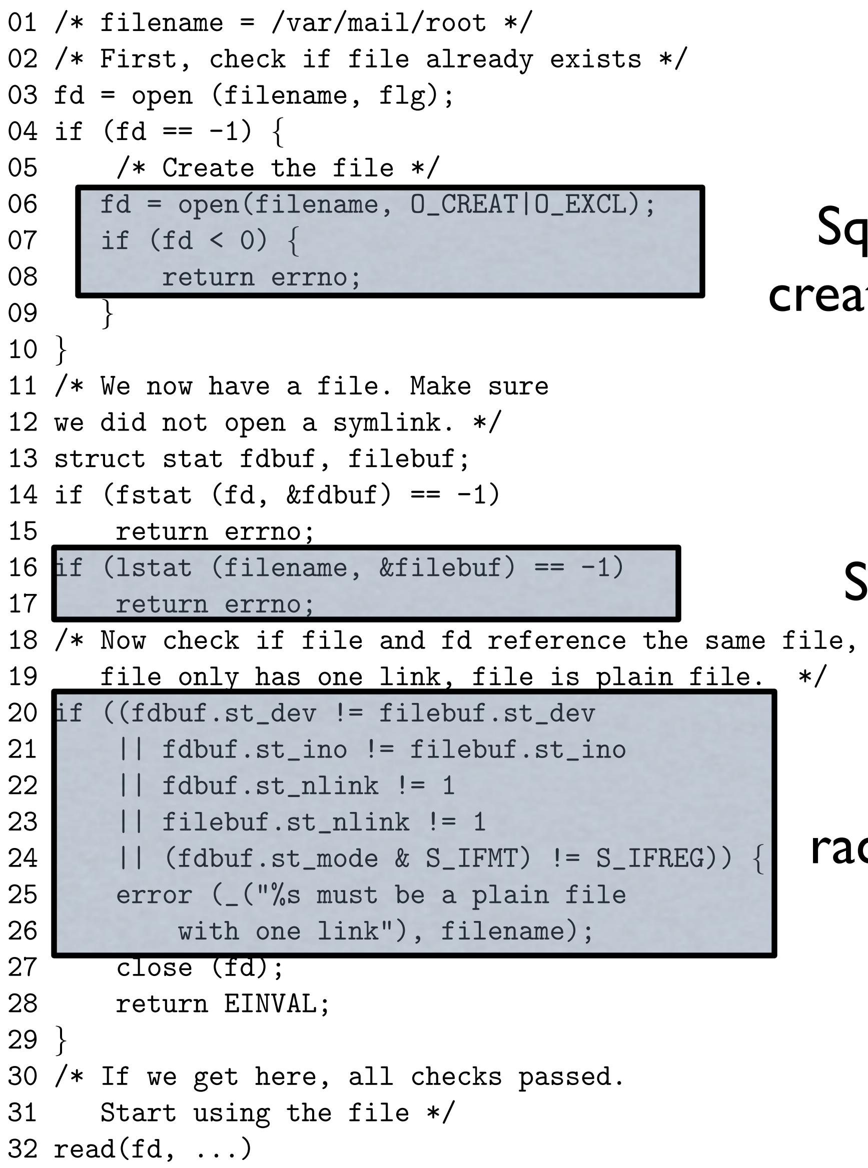
- Variants of the “open” system call
  - Flag “O_NOFOLLOW” – do not follow any symbolic links (prevent link traversal)
    - Does not help if you need to follow symbolic links
    - May not be available on your system
  - Flag “O_EXCL” and “O_CREAT” – do not open unless the new file is created (prevent file squatting)
    - Does not help if the file may or may not be created already
- 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 part of the name resolution (for dirfd) and prevent further use of links
  - Supports O_NOFOLLOW
    - *int openat(int dirfd, const char *path, int oflag, ...);*
  - Helps if resolution of directory “dirfd” is unsafe, but is limited if resolution of the “path” is unsafe
    - Check “dirfd” with the “fstat” syscall – “stat” for descriptors
- The “openat2” system call
  - More flags limiting “how” name resolution is done for “path”

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, ...)
```

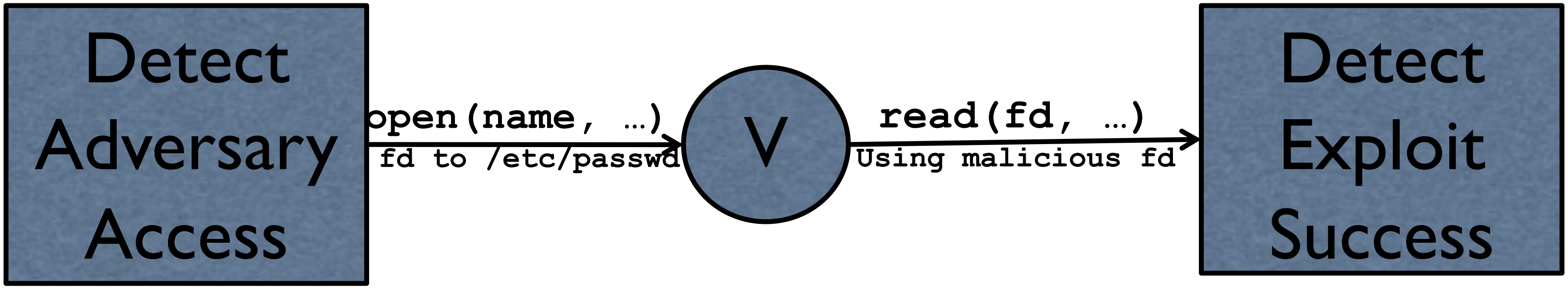
Squat during create (resource)

Symbolic link

Hard link, race conditions

Runtime Testing [STING]

- We actively change the namespace whenever an adversary can write to a directory in resolution
  - Fundamental problem: adversaries may be able to write directories used in name resolution
- Use adversary model to identify program adversaries and vulnerable directories



Detect Adversary Access

open(name, …)

fd to /etc/passwd

V

read(fd, …)

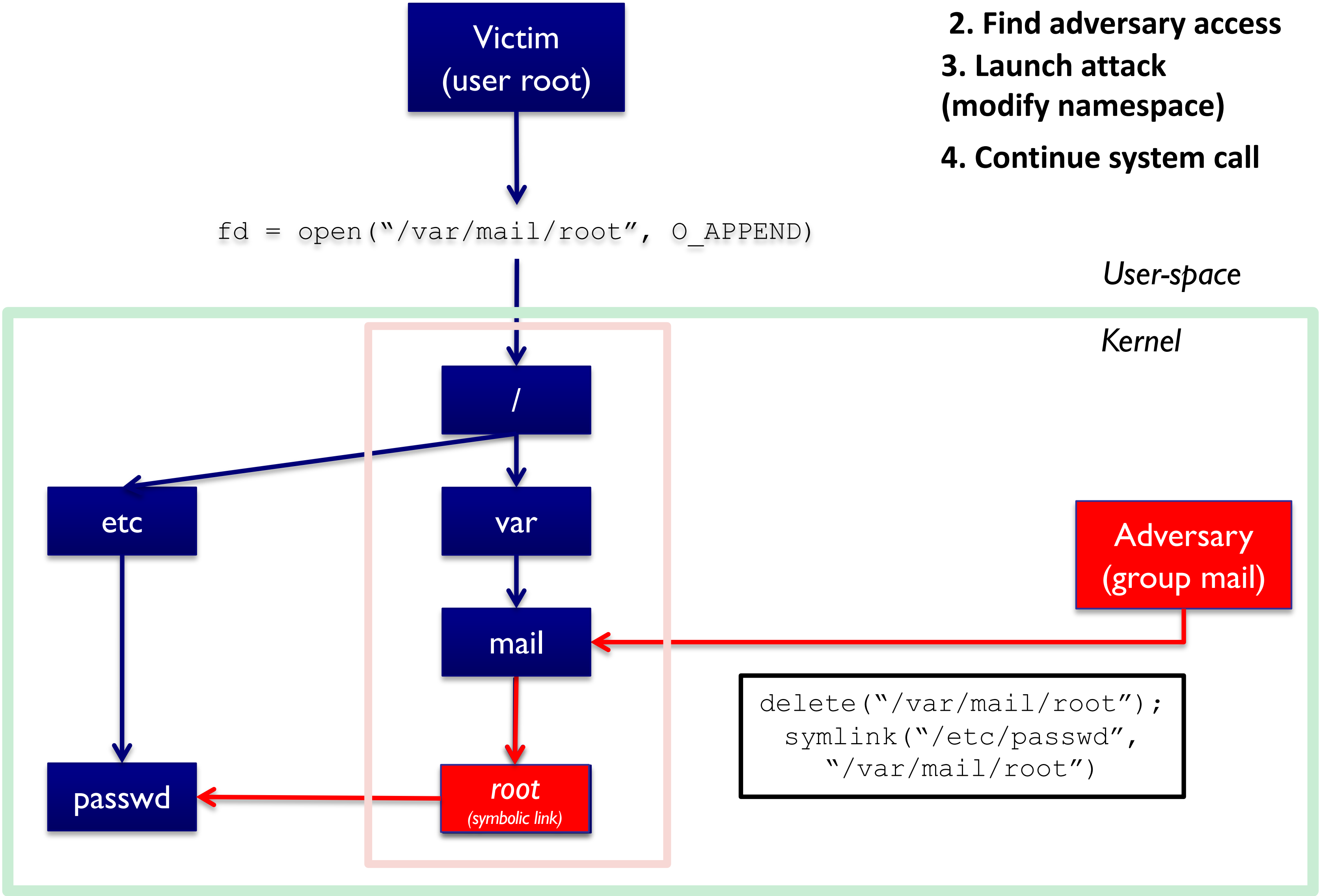
Using malicious fd

Detect Exploit Success

Vulnerable!

STING Launch Phase

1. **Find bindings**
2. **Find adversary access**
3. **Launch attack (modify namespace)**
4. **Continue system call**

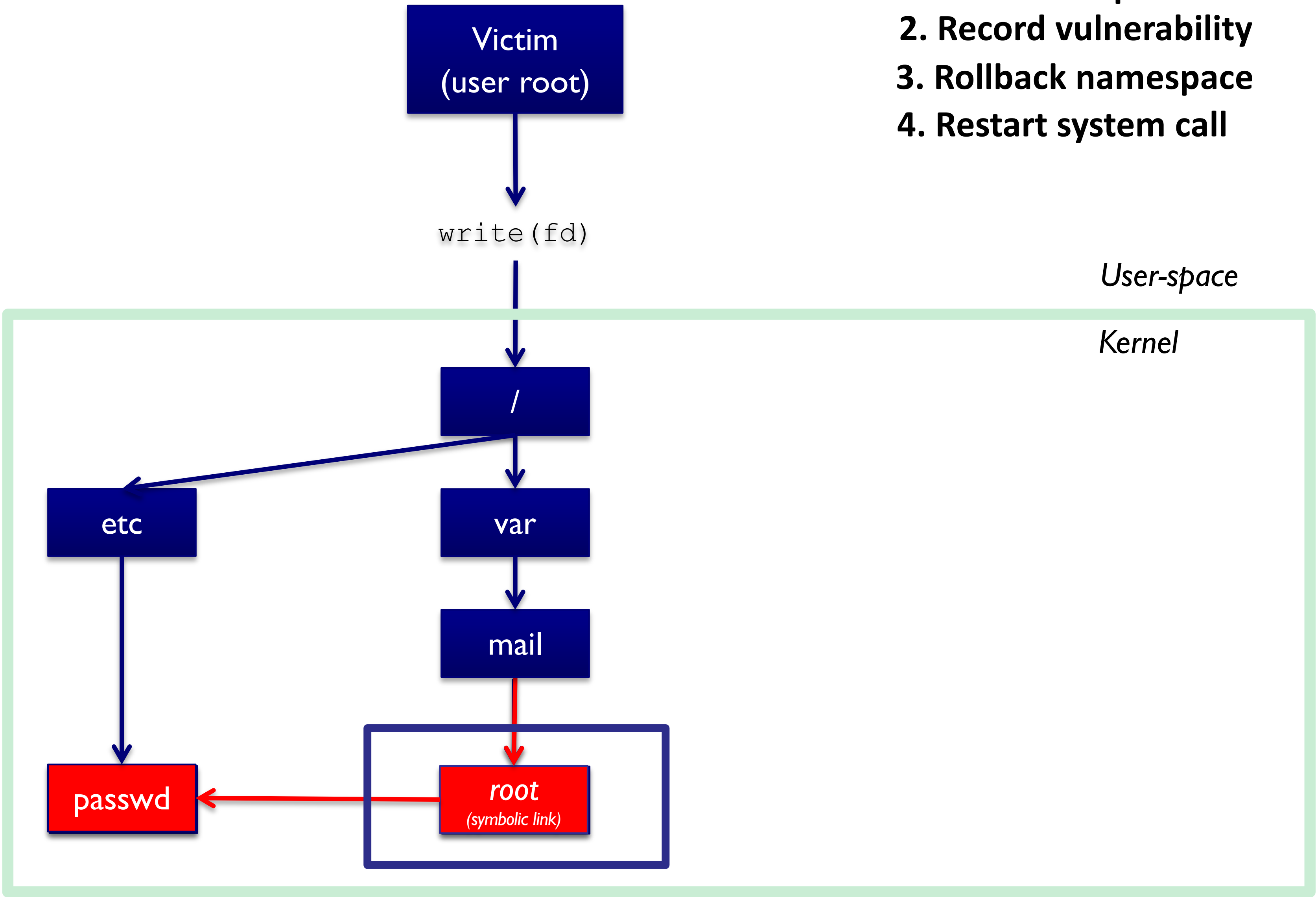


```
fd = open("/var/mail/root", O_APPEND)
```

```
delete("/var/mail/root");
symlink("/etc/passwd",
"/var/mail/root")
```

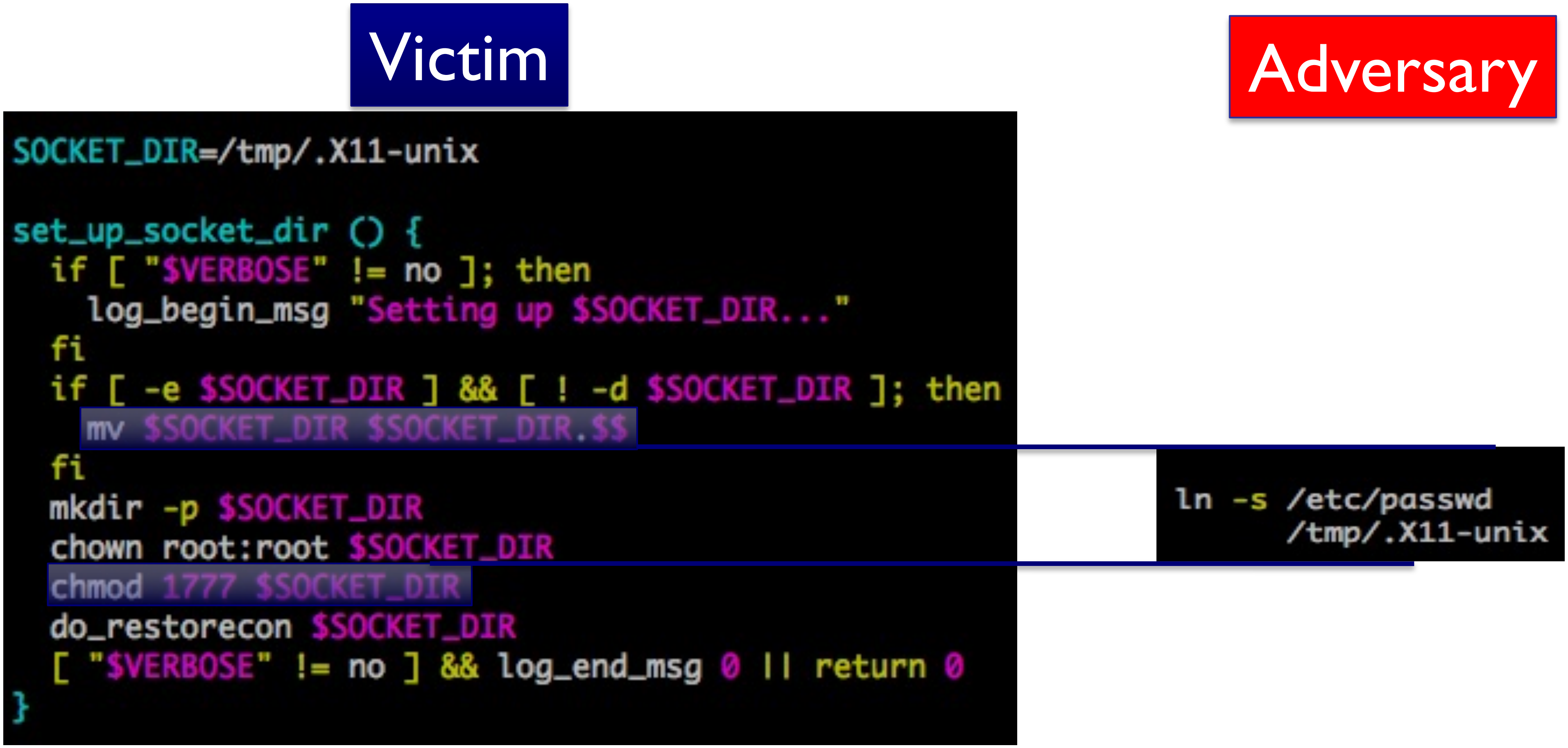
STING Detect Phase

1. Victim accepts resource
2. Record vulnerability
3. Rollback namespace
4. Restart system call



STING Detects TOCTTOU Races

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



Victim

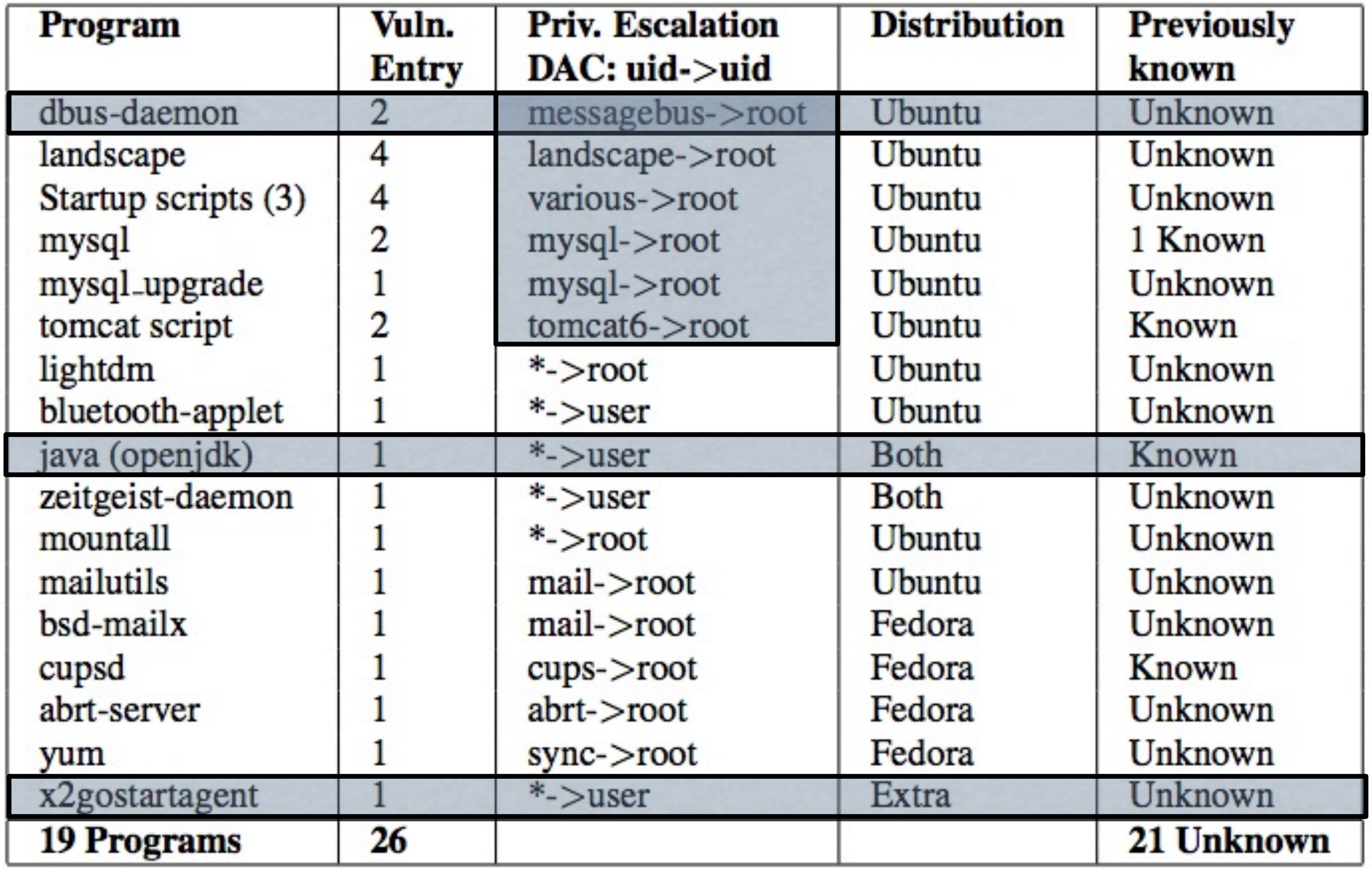
```
SOCKET_DIR=/tmp/.X11-unix

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
}
```

Adversary

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

Results - Vulnerabilities



Program	Vuln. Entry	Priv. Escalation DAC: uid->uid	Distribution	Previously known
dbus-daemon	2	messagebus->root	Ubuntu	Unknown
landscape	4	landscape->root	Ubuntu	Unknown
Startup scripts (3)	4	various->root	Ubuntu	Unknown
mysql	2	mysql->root	Ubuntu	1 Known
mysql_upgrade	1	mysql->root	Ubuntu	Unknown
tomcat script	2	tomcat6->root	Ubuntu	Known
lightdm	1	*->root	Ubuntu	Unknown
bluetooth-applet	1	*->user	Ubuntu	Unknown
java (openjdk)	1	*->user	Both	Known
zeitgeist-daemon	1	*->user	Both	Unknown
mountall	1	*->root	Ubuntu	Unknown
mailutils	1	mail->root	Ubuntu	Unknown
bsd-mailx	1	mail->root	Fedora	Unknown
cupsd	1	cups->root	Fedora	Known
abrt-server	1	abrt->root	Fedora	Unknown
yum	1	sync->root	Fedora	Unknown
x2gostartagent	1	*->user	Extra	Unknown
19 Programs	**26**			**21 Unknown**

Both old and new programs

Special users to root

Known but unfixed!

Take Away

- Programs can be exploited when retrieving system resources
    - Because adversaries may share access to resources and/or namespaces
    - Called Confused Deputy Attacks – trick a program into performing an operation of an adversary's choosing
- Adversaries may control two kinds of inputs
    - Filesystem configuration - where directories are shared
    - Program inputs – where could be from an untrusted party
- Can improve security through careful use of syscall APIs and through better runtime testing