

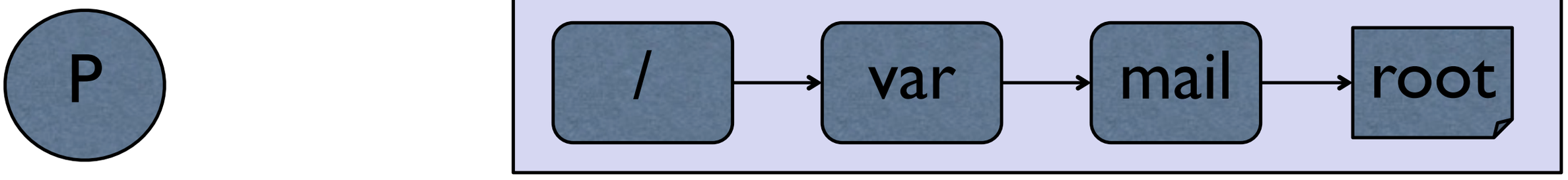
STING: Finding Name Resolution Vulnerabilities in Programs

Hayawardh Vijayakumar, Joshua Schiffman, Trent Jaeger

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

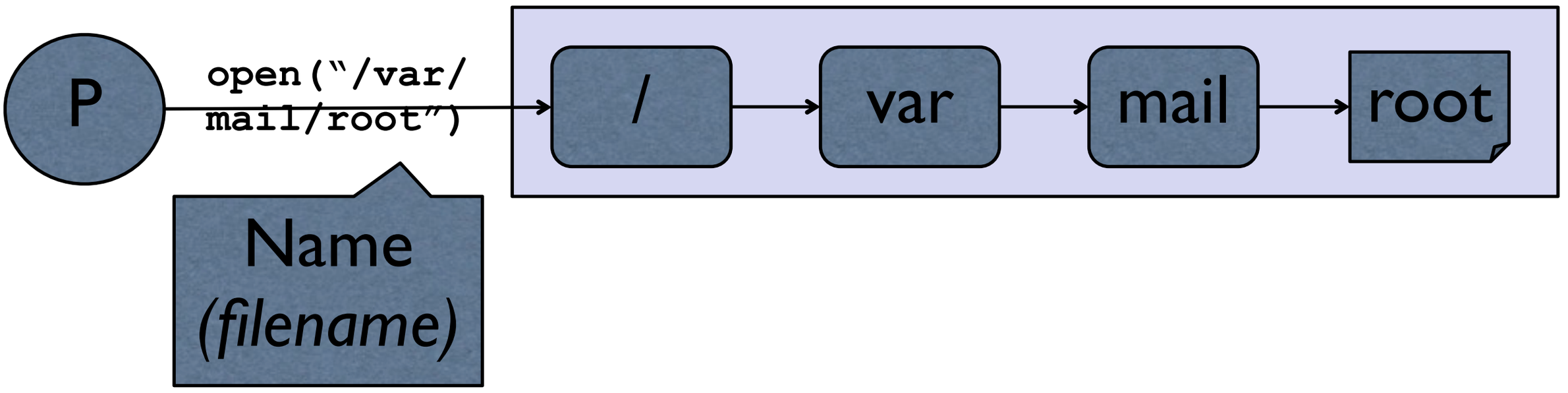
Name Resolution

- 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, …



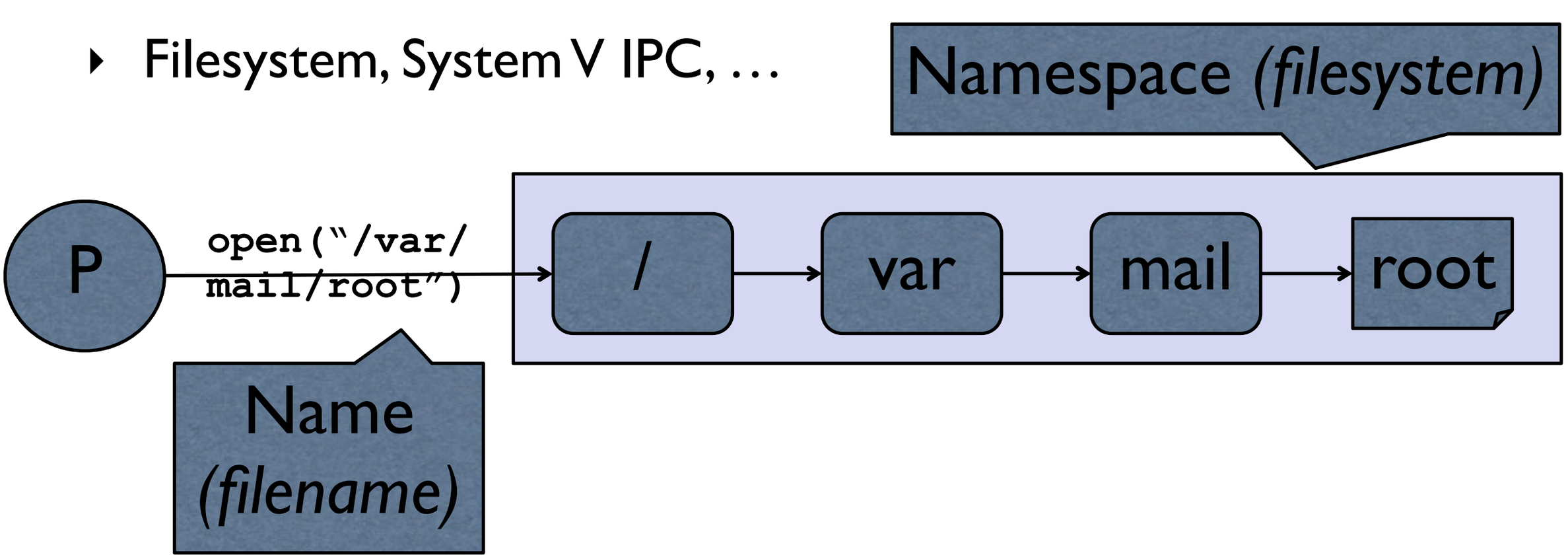
Name Resolution

- 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, …



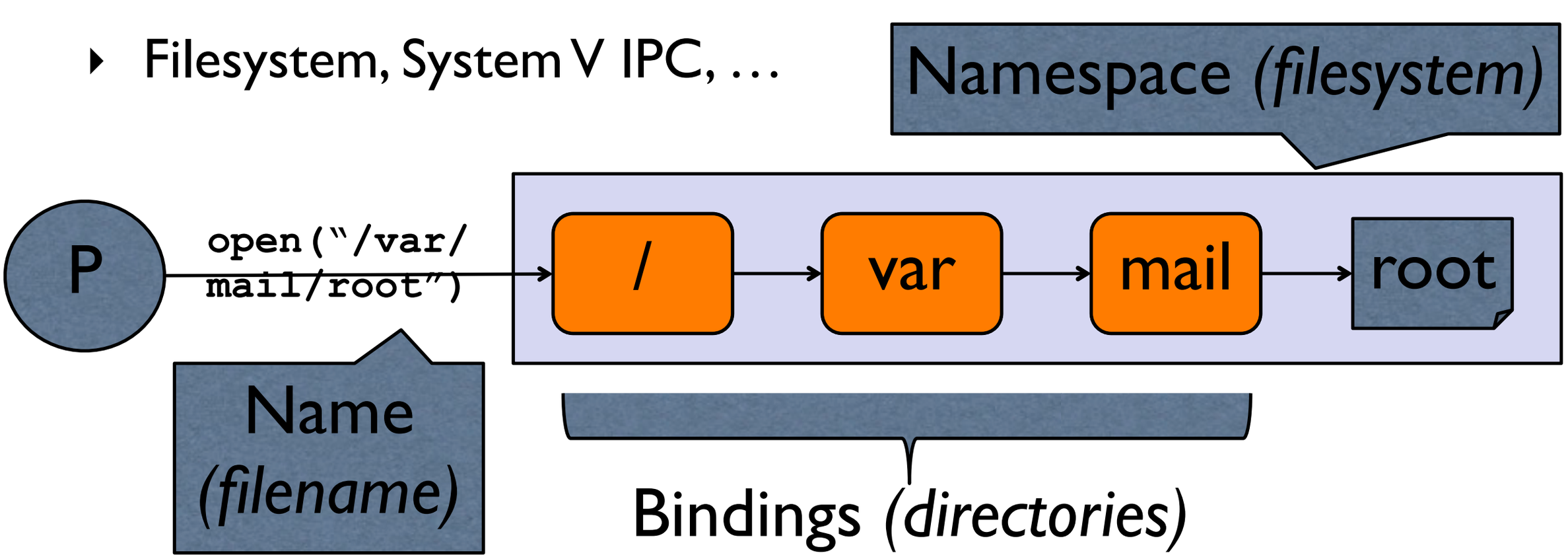
Name Resolution

- 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, …



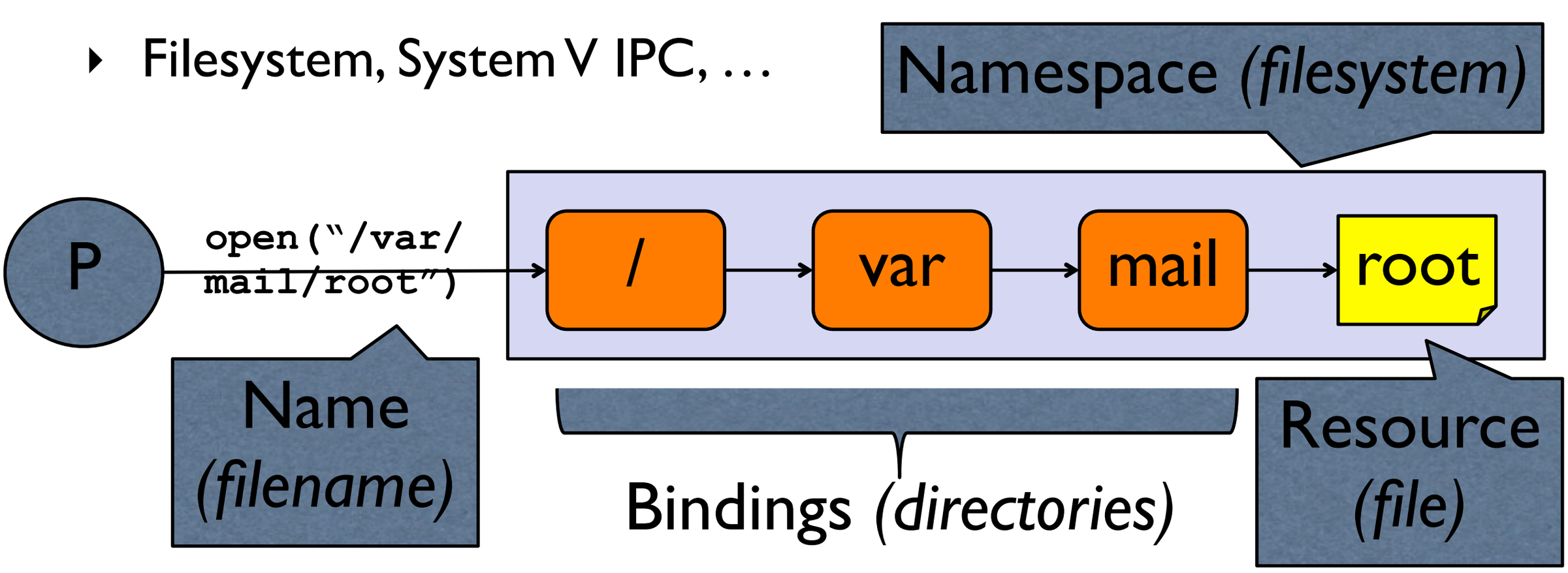
Name Resolution

- 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, …



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, …



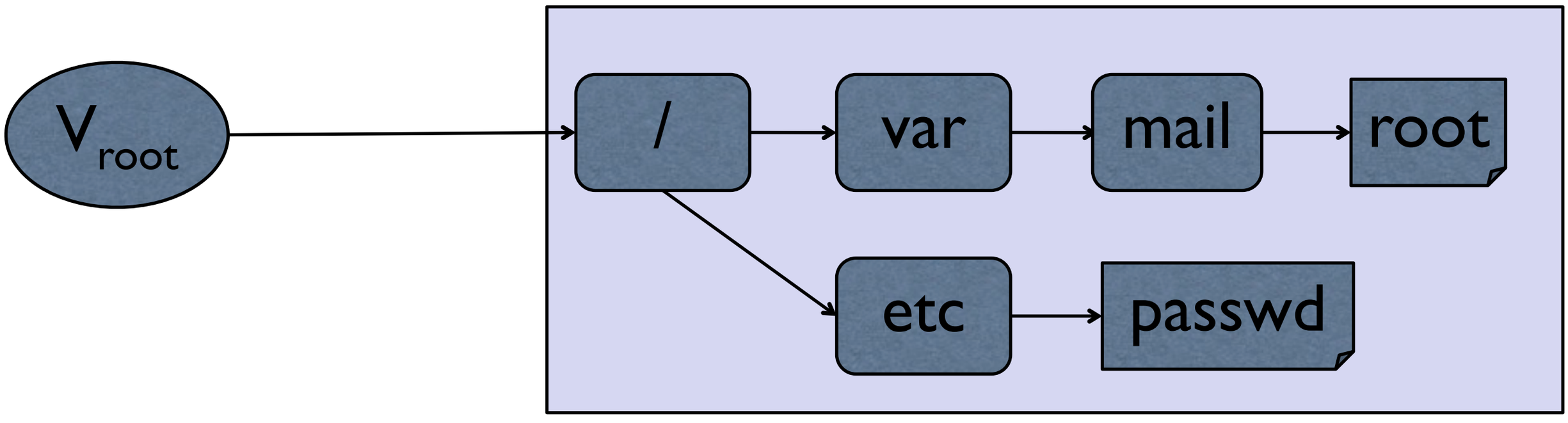
Namespace Sharing Problems

- Security problems occur because low-integrity adversary processes share the same OS namespaces as high-integrity victim processes
  - Adversary processes attempt to affect name resolution of victim processes
- Permissions for `/var/mail`
  - Group `mail` can create and delete files

```
drwxrwsr-x  2 root mail  4096 2011-01-13 10:06 mail
```

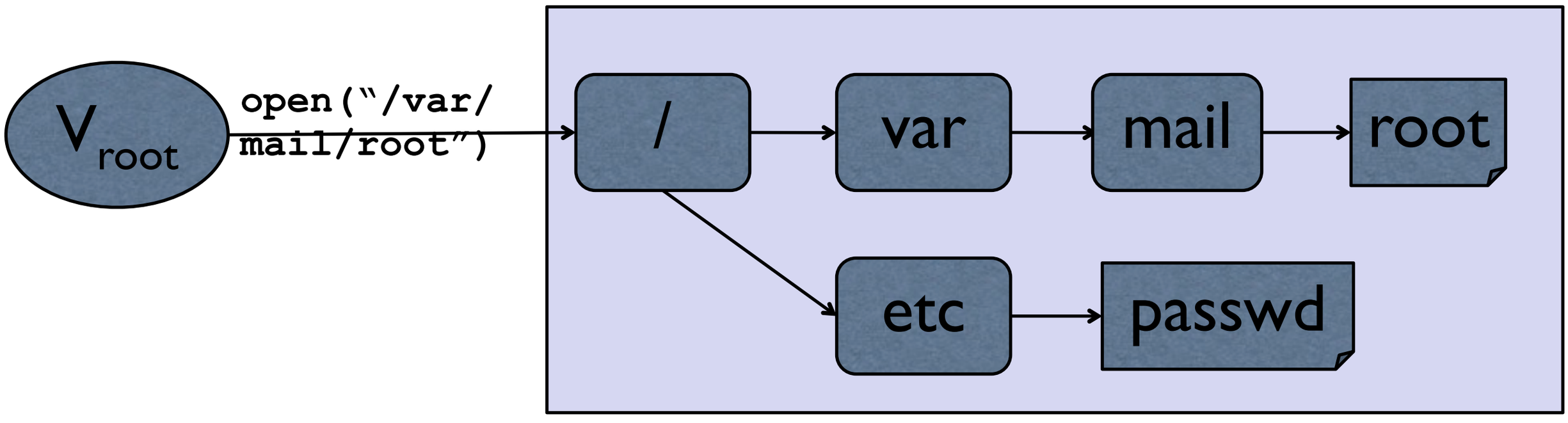
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)
  - Symbolic link, hard link attacks
  - Victim expects low integrity/secrecy, gets high instead



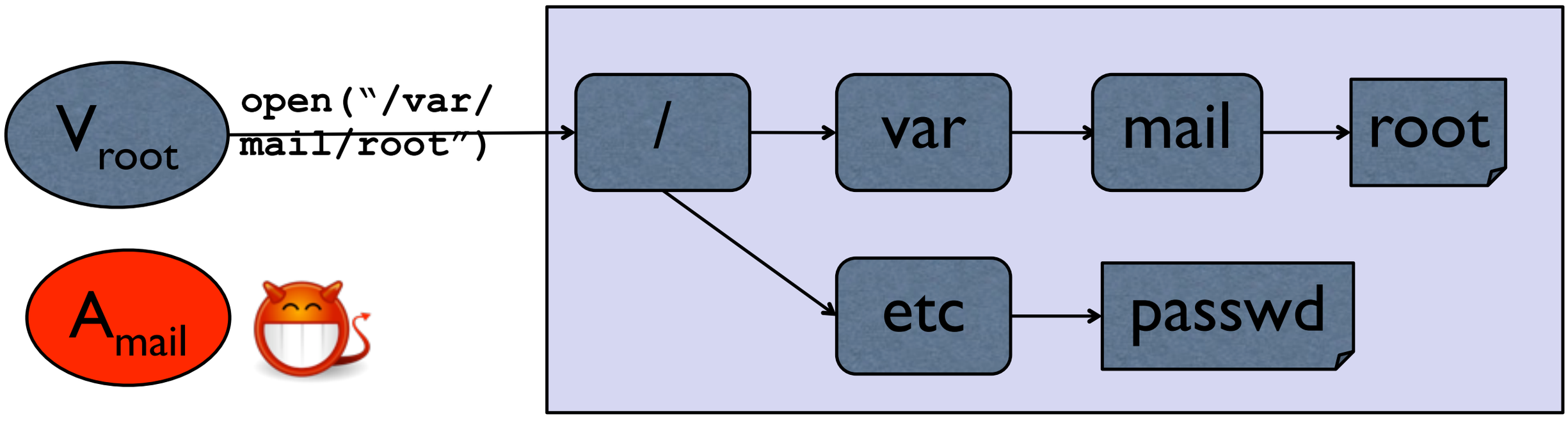
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)
  - Symbolic link, hard link attacks
  - Victim expects low integrity/secrecy, gets high instead



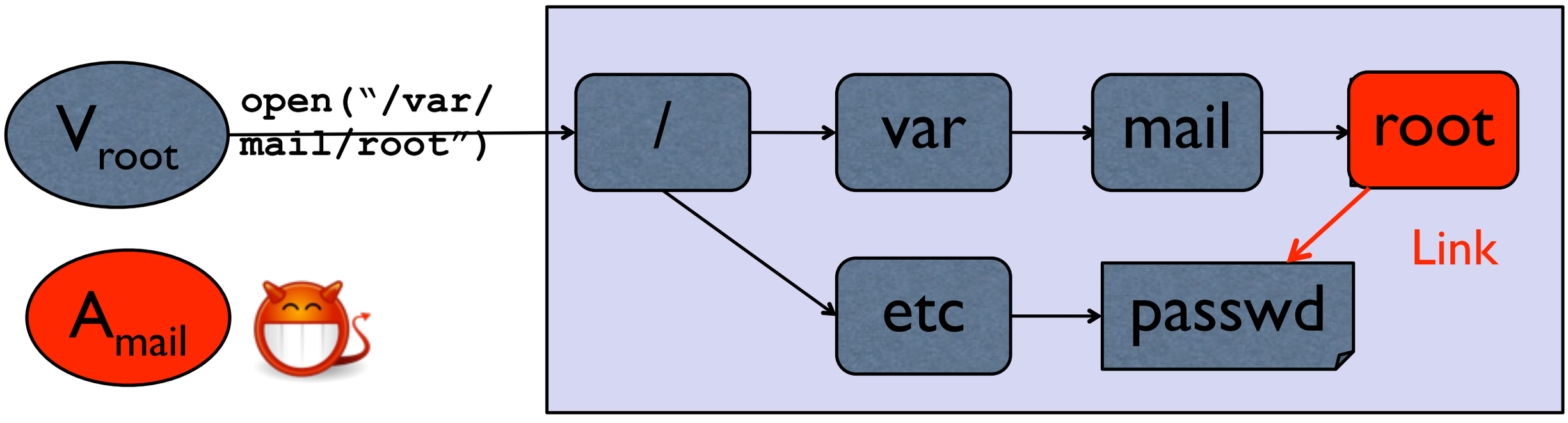
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)
  - Symbolic link, hard link attacks
  - Victim expects low integrity/secrecy, gets high instead



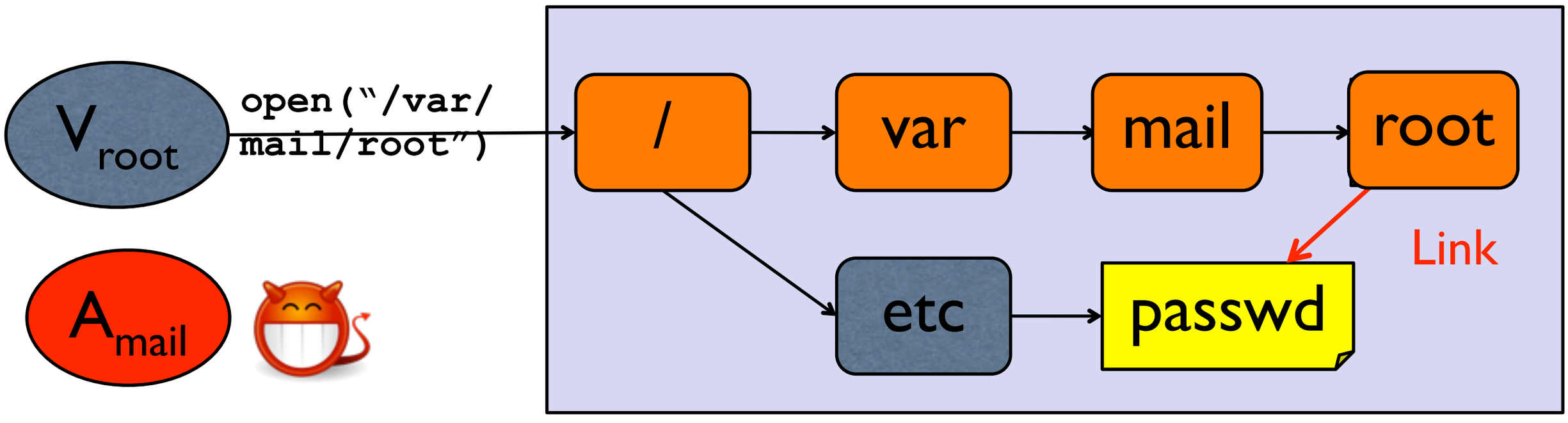
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)
  - Symbolic link, hard link attacks
  - Victim expects low integrity/secrecy, gets high instead



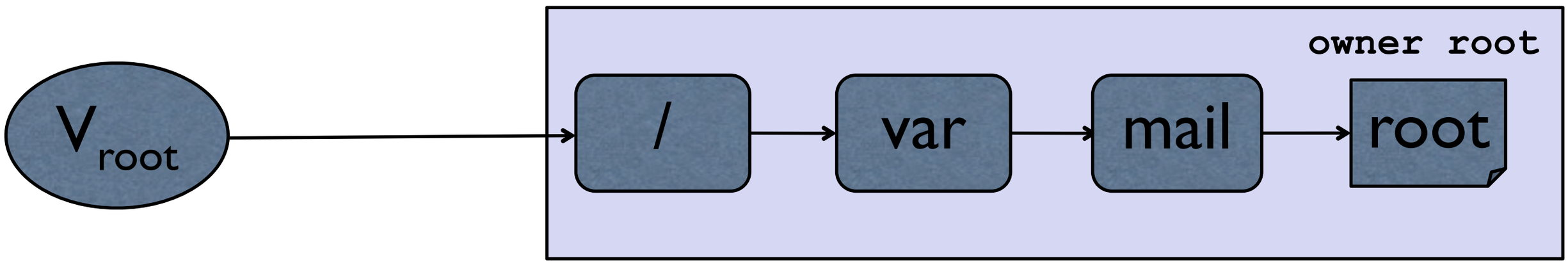
Attacks on Name Resolution

- Improper Binding Attack
  - Adversary controls bindings to redirect victim to a resource not under adversary's control (confused deputy)
  - Symbolic link, hard link attacks
  - Victim expects low integrity/secrecy, gets high instead



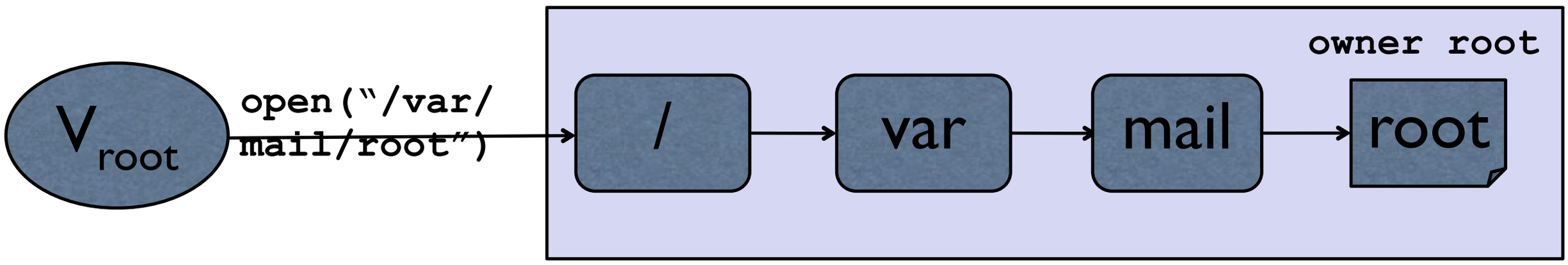
Attacks on Name Resolution

- Improper Resource Attack
  - Adversary controls final resource in unexpected ways
  - Untrusted search paths (e.g., Trojan library), file squatting
  - Victim expects high integrity, gets low integrity instead



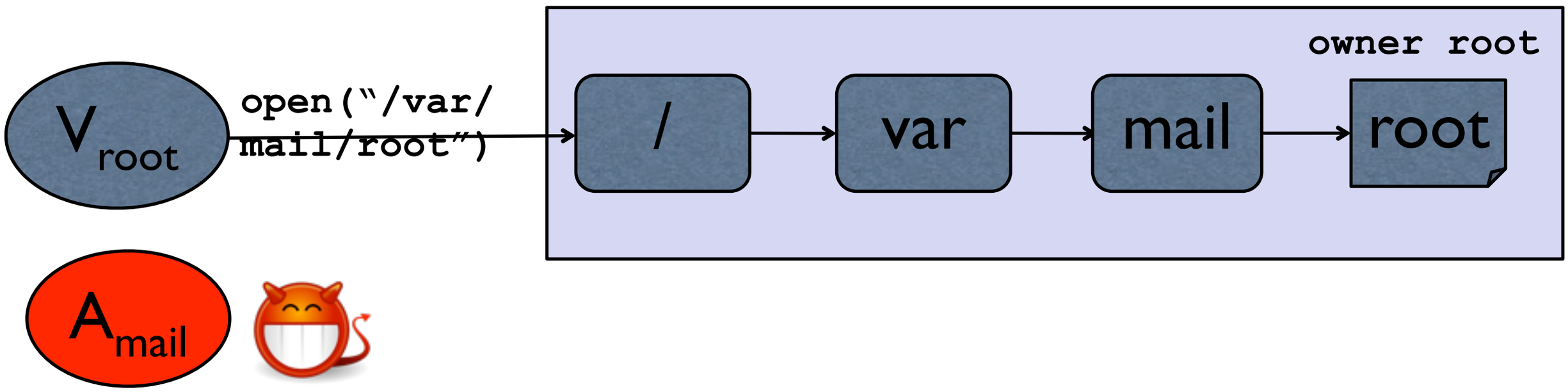
Attacks on Name Resolution

- Improper Resource Attack
  - Adversary controls final resource in unexpected ways
  - Untrusted search paths (e.g., Trojan library), file squatting
  - Victim expects high integrity, gets low integrity instead



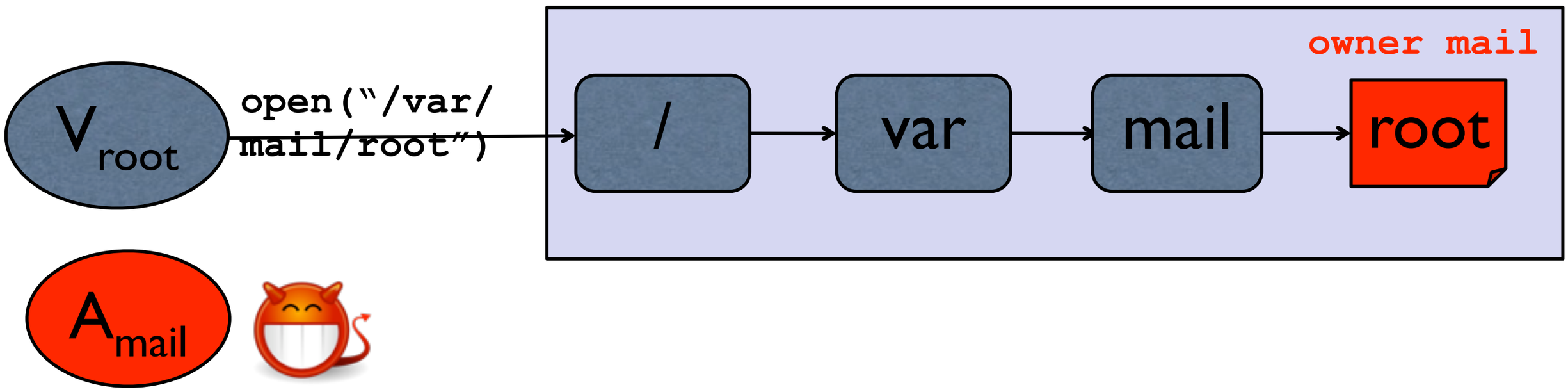
Attacks on Name Resolution

- Improper Resource Attack
  - Adversary controls final resource in unexpected ways
  - Untrusted search paths (e.g., Trojan library), file squatting
  - Victim expects high integrity, gets low integrity instead



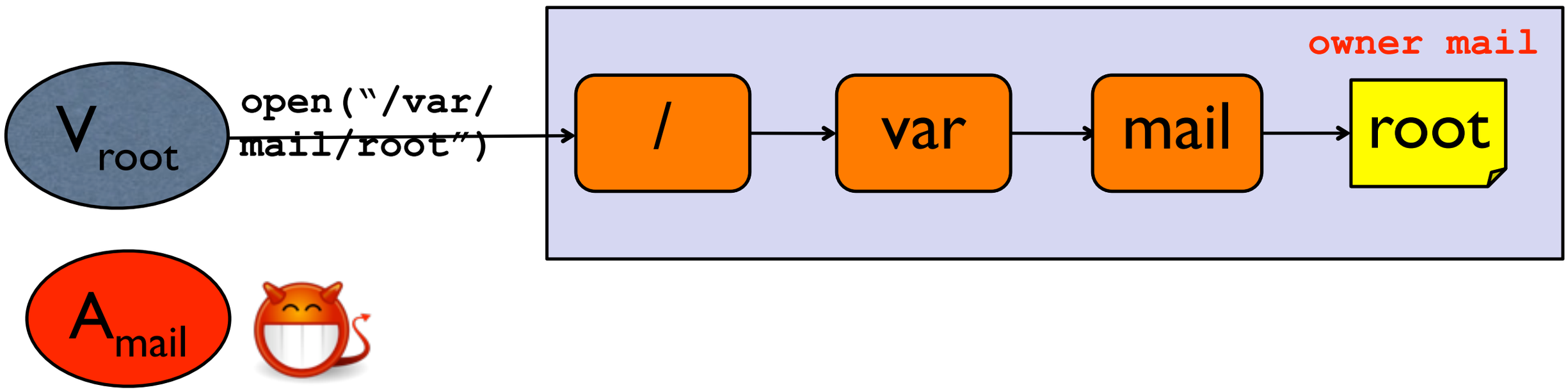
Attacks on Name Resolution

- Improper Resource Attack
  - Adversary controls final resource in unexpected ways
  - Untrusted search paths (e.g., Trojan library), file squatting
  - Victim expects high integrity, gets low integrity instead



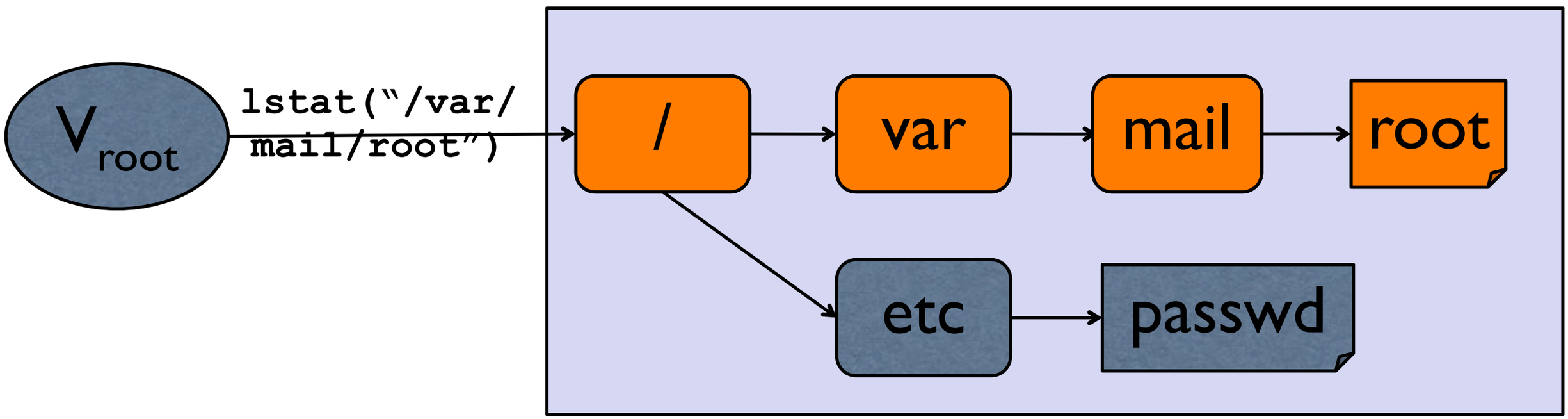
Attacks on Name Resolution

- Improper Resource Attack
  - Adversary controls final resource in unexpected ways
  - Untrusted search paths (e.g., Trojan library), file squatting
  - Victim expects high integrity, gets low integrity instead



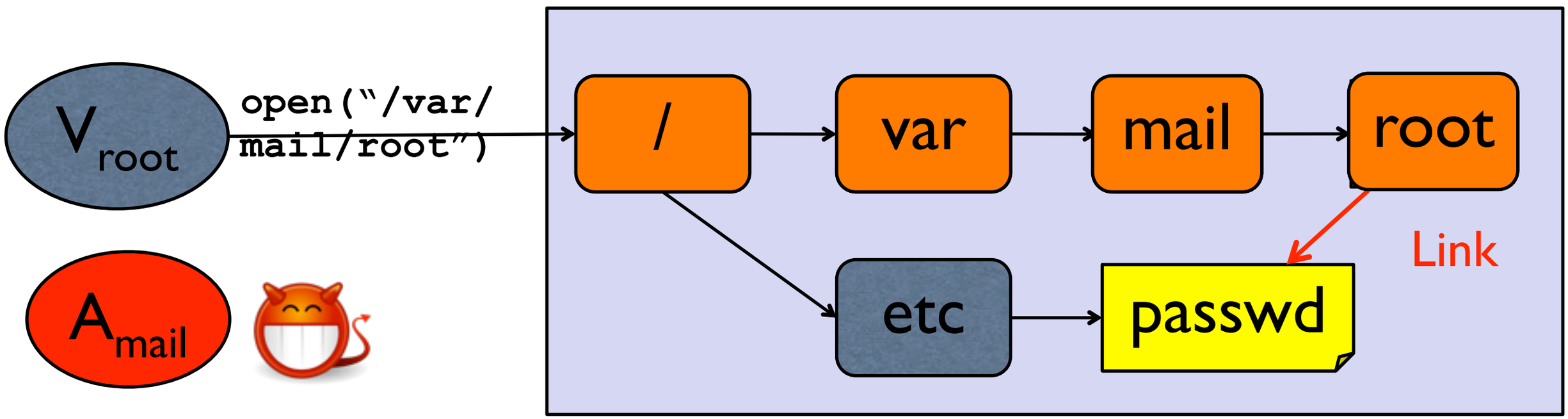
Attacks on Name Resolution

- Race Conditions
  - Adversary exploits non-atomicity in "check" and "use" of resource to conduct improper resource and improper binding attacks
  - Well-known "TOCTTOU" attacks



Attacks on Name Resolution

- Race Conditions
  - Adversary exploits non-atomicity in "check" and "use" of resource to conduct improper resource and improper binding attacks
  - Well-known "TOCTTOU" attacks



How Serious a Problem?

- Who can launch local exploits?
  - Untrusted local users in a multi-user environment (e.g., university)
  - Remote attackers who have broken into networked programs through bugs or misconfigurations and want to further escalate privileges
    - Downloaded malware, compromised server programs, …



How Serious a Problem?

- Name resolution vulnerabilities accounts for 5-10% CVE entries each year
- These are particularly hard to eradicate as they involve multiple parties
  - Programmers who write code

```
if ((stat("/var/mail/root", st)) == 0 && !S_ISLNK(st->st_mode))
```

  - OS distributors who define access control policies

```
drwxrwsr-x  2 root mail  4096 2011-01-13 10:06 mail
```

  - Administrators who configure end system

Existing Program Defenses

- Name resolution attacks have been with us
  - TOCTTOU attacks first published by McPhee in 1974
  - Like buffer overflows – known for decades
- Program API to convey intended context to OS
  - E.g.,
    - *O_EXCL* flag in open(): if a resource already exists, fail
      - mkstemp creates an unpredictable name
    - *O_NOFOLLOW* don't follow a link on this name resolution
    - *openat* and related allow use of same directory for access
- Programmers do not always use APIs properly
  - Lots of exceptions
  - Impractical to determine whether defenses should be on

Program Defenses

- Often don't work…

```
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, ...)
```

Proposed System Defenses

- Many defenses have been proposed by researchers
  - And broken…
  - Mainly for TOCTTOU
- Cai *et al.* [Oakland 2009] showed
  - All system defenses fundamentally limited because they do not have program knowledge
- Chari *et al.* [NDSS 2010] propose a system defense for improper binding attacks
  - Have false positives

This Work's Goal

- Given the difficulty of proper defenses, we propose actively finding name resolution vulnerabilities in programs
  - So programs can be fixed to perform correct checks

```
if ((stat("/var/mail/root", st)) == 0 && !S_ISLNK(st->st_mode))
```

  - Or access control policies can be tightened

```
root@mantra:/var# ls -l | grep mail
drwxr-sr-x    2 root mail    4096 2012-03-29 02:05 mail
```

Prior – Static Analysis

Prior – Static Analysis

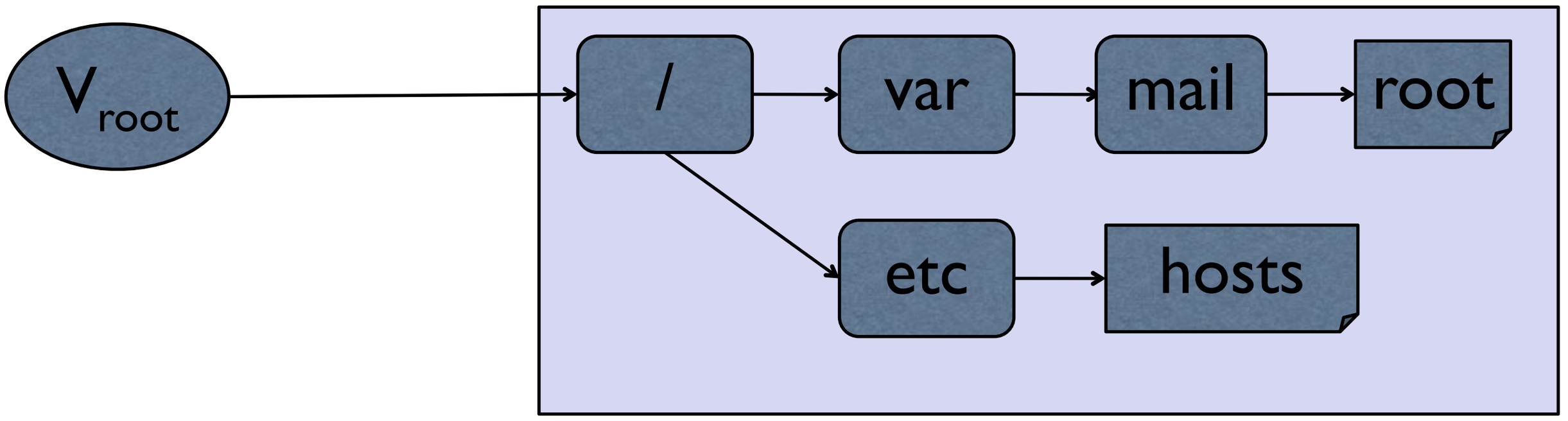
- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU

Prior – Static Analysis

- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
  - Our runtime study found only around 5% of name resolutions were accessible to adversaries

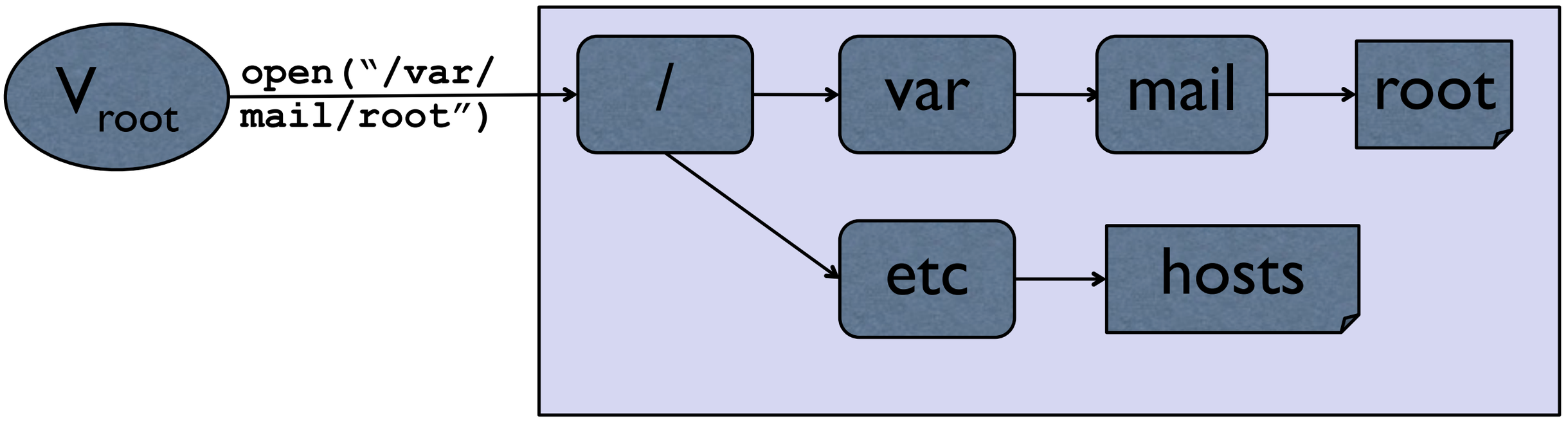
Prior – Static Analysis

- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
    - Our runtime study found only around 5% of name resolutions were accessible to adversaries



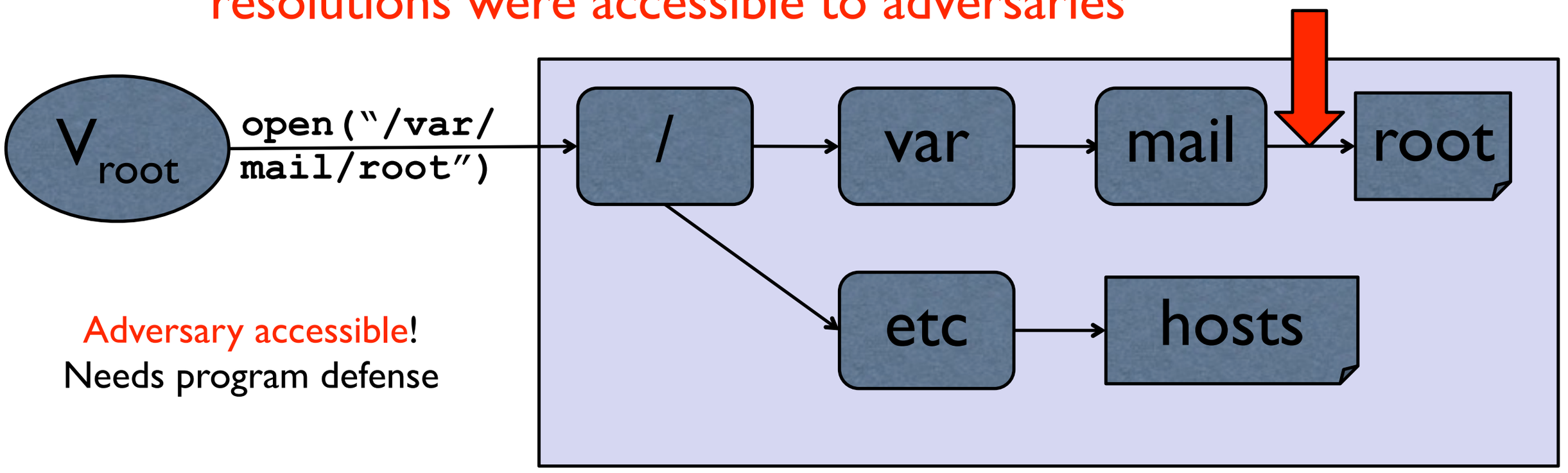
Prior – Static Analysis

- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
    - Our runtime study found only around 5% of name resolutions were accessible to adversaries



Prior – Static Analysis

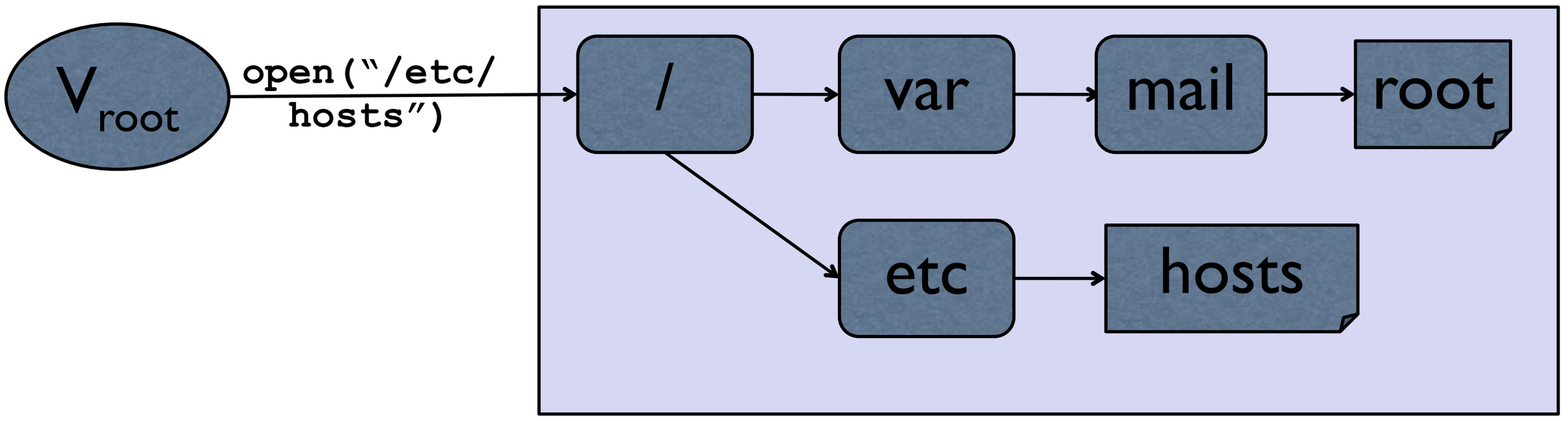
- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
    - Our runtime study found only around 5% of name resolutions were accessible to adversaries



Adversary accessible!
Needs program defense

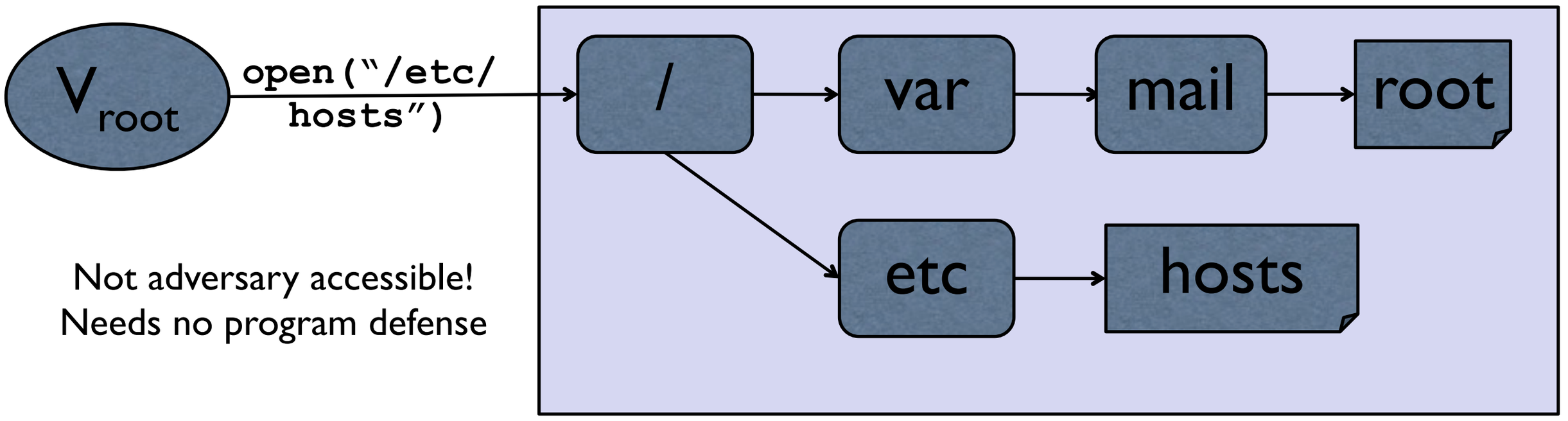
Prior – Static Analysis

- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
    - Our runtime study found only around 5% of name resolutions were accessible to adversaries



Prior – Static Analysis

- Analyze program to find potentially vulnerable name resolution calls
  - Due to complexity of checks, mainly limited to TOCTTOU
- Deficiencies
  - False positives due to adversary inaccessibility
    - Our runtime study found only around 5% of name resolutions were accessible to adversaries



Prior – Runtime Analysis

Prior – Runtime Analysis

- Have both access control policy and program system calls

Prior – Runtime Analysis

- Have both access control policy and program system calls
- Still, many false positives
  - Program code might defend itself
    - Manual audits impractical
  - In our study, only 13% of adversary-accessible name resolutions are actually vulnerable

Prior – Runtime Analysis

- Have both access control policy and program system calls
- Still, many false positives
  - Program code might defend itself
    - Manual audits impractical
  - In our study, only 13% of adversary-accessible name resolutions are actually vulnerable

```
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, ...)
```

Prior – Runtime Analysis

- Have both access control policy and program system calls
- Still, many false positives
  - Program code might defend itself
    - Manual audits impractical
  - In our study, only 13% of adversary-accessible name resolutions are actually vulnerable

```
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, ...)
```

???

Prior – Runtime Analysis

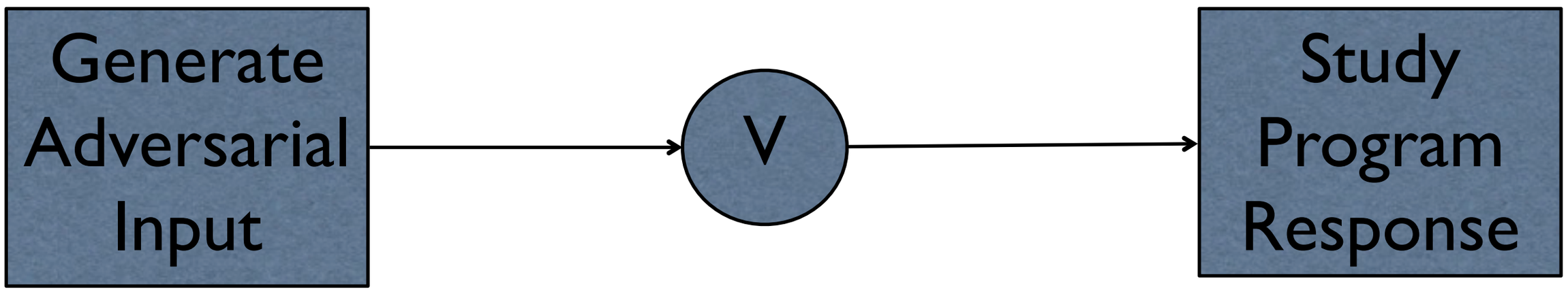
- False negatives during normal runtime
  - Attacks require very specific conditions that do not occur in normal runtime
    - Example: *mountall untrusted search path* vulnerability required:
      - Launching that program in an untrusted directory, and
      - Symbolic links named `none` and `fusectl`

Our Solution

- Thus, we have to actively change the namespace to create adversarial scenarios
  - And evaluate process response to scenario
- We take inspiration from "grey-box" testing
  - Feed known adversarial inputs to programs and examine process response (e.g., detect SQL injection vulnerability)

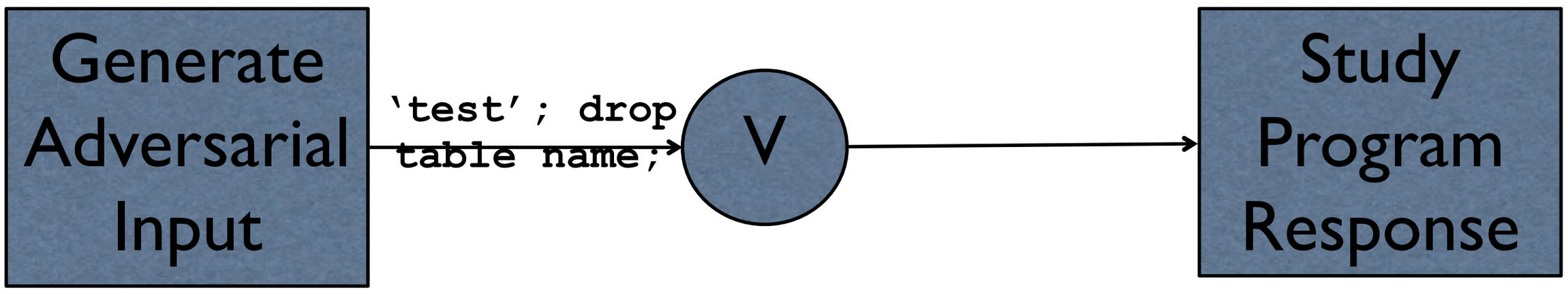
Our Solution

- Thus, we have to actively change the namespace to create adversarial scenarios
  - And evaluate process response to scenario
- We take inspiration from "grey-box" testing
  - Feed known adversarial inputs to programs and examine process response (e.g., detect SQL injection vulnerability)



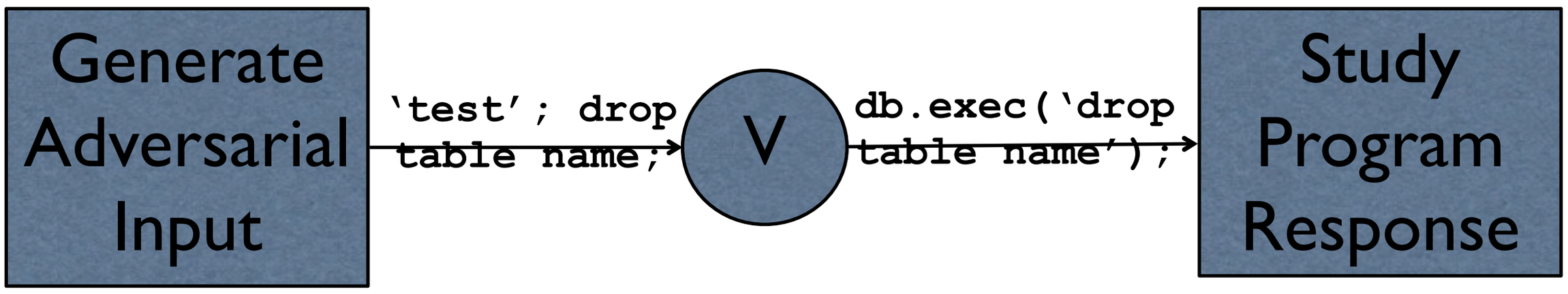
Our Solution

- Thus, we have to actively change the namespace to create adversarial scenarios
  - And evaluate process response to scenario
- We take inspiration from "grey-box" testing
  - Feed known adversarial inputs to programs and examine process response (e.g., detect SQL injection vulnerability)



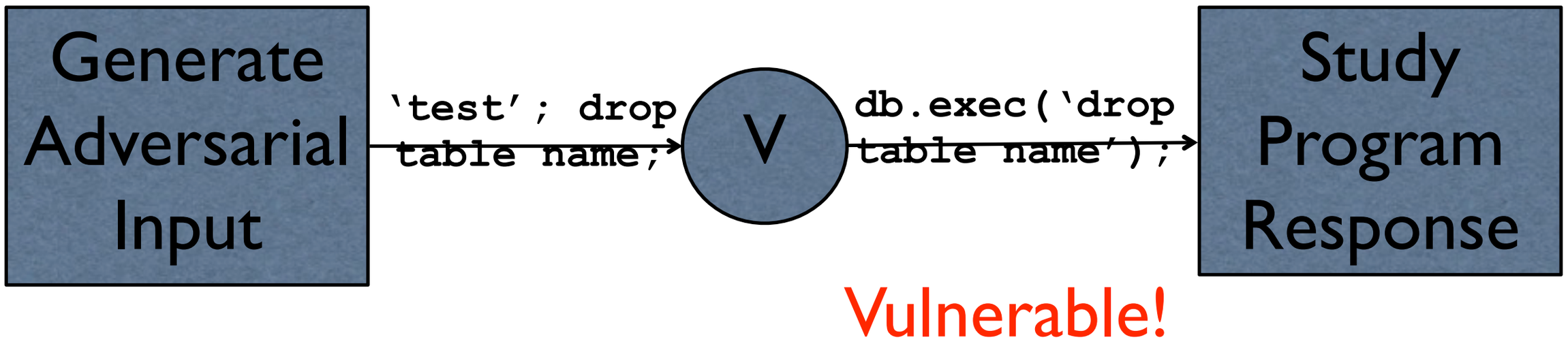
Our Solution

- Thus, we have to actively change the namespace to create adversarial scenarios
  - And evaluate process response to scenario
- We take inspiration from "grey-box" testing
  - Feed known adversarial inputs to programs and examine process response (e.g., detect SQL injection vulnerability)



Our Solution

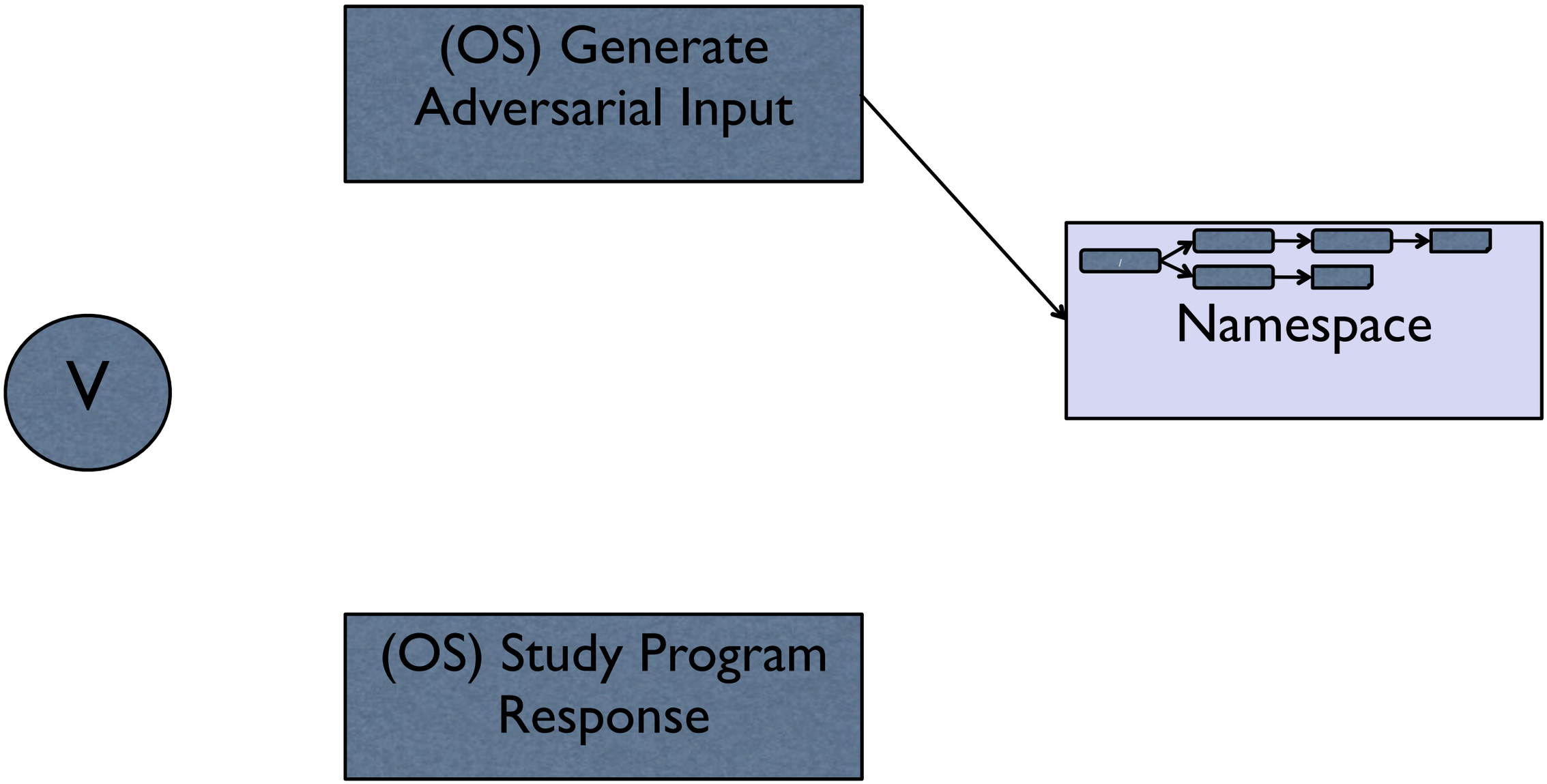
- Thus, we have to actively change the namespace to create adversarial scenarios
  - And evaluate process response to scenario
- We take inspiration from "grey-box" testing
  - Feed known adversarial inputs to programs and examine process response (e.g., detect SQL injection vulnerability)



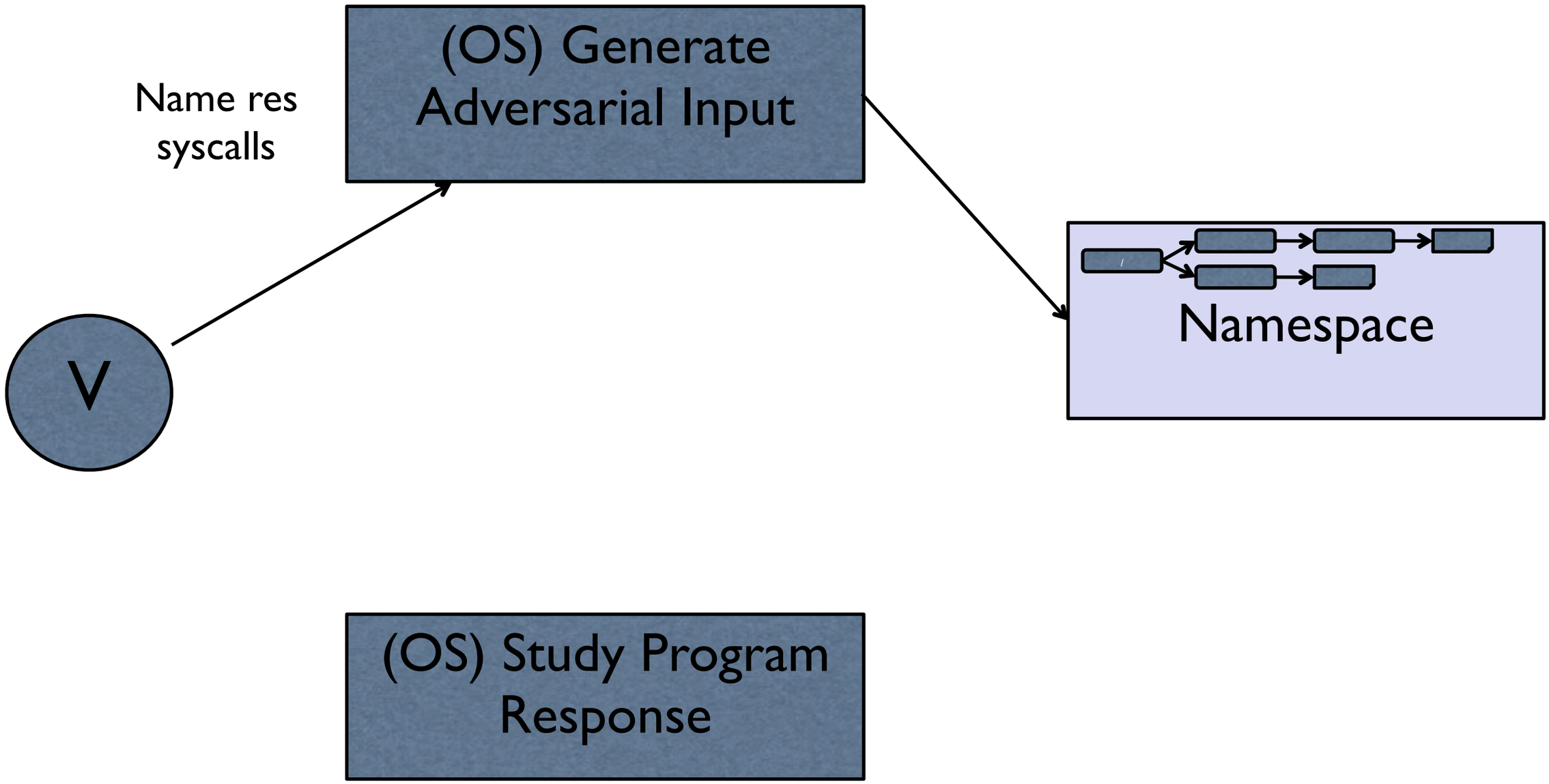
Grey-Box Test Using OS

- OS is in charge of namespace
  - Use OS to feed adversarial input in response to program name resolution requests, and study program response
  - System-wide testing
- Generate Adversarial Input
- Examine Program Response

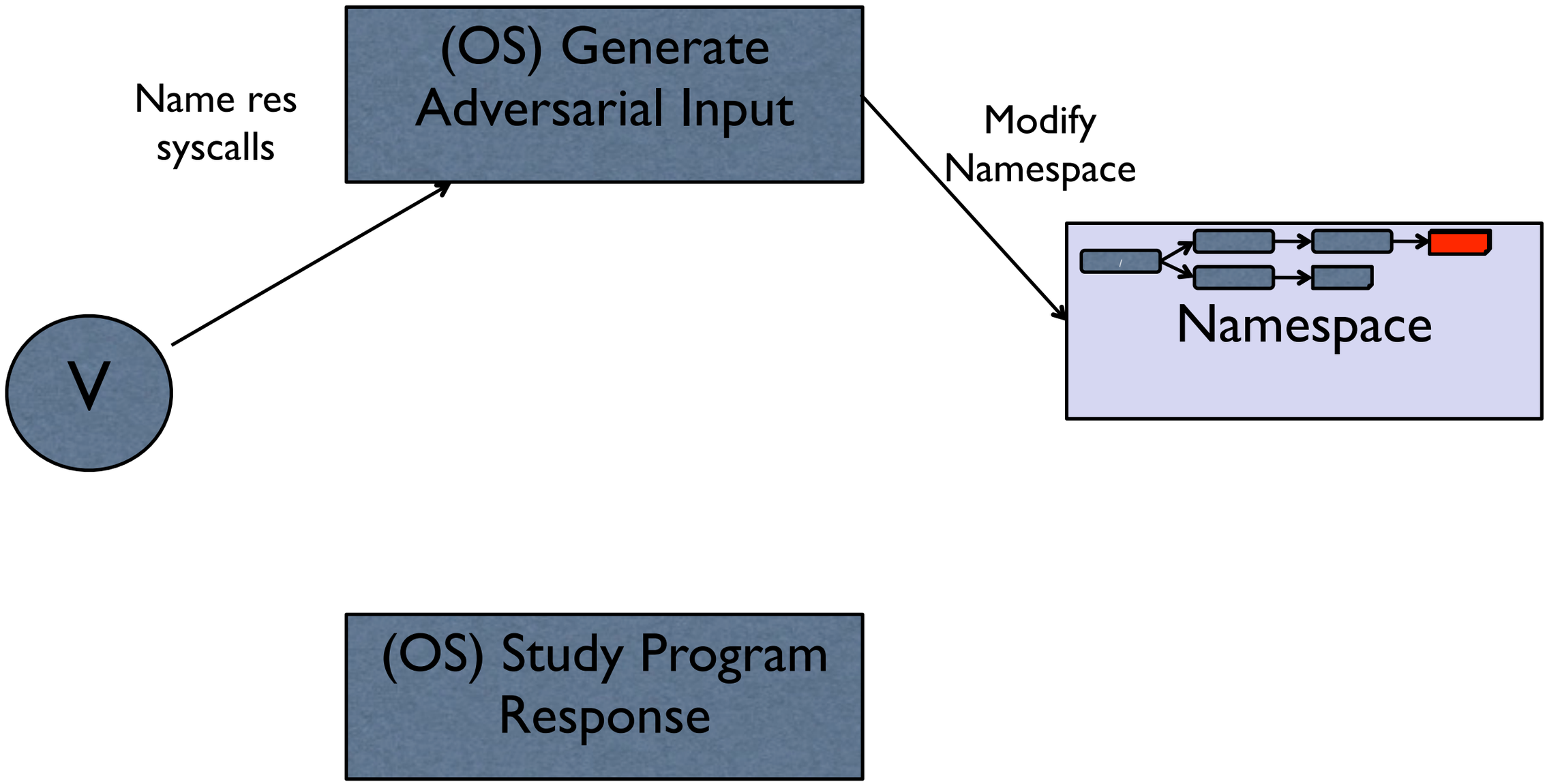
Solution Overview



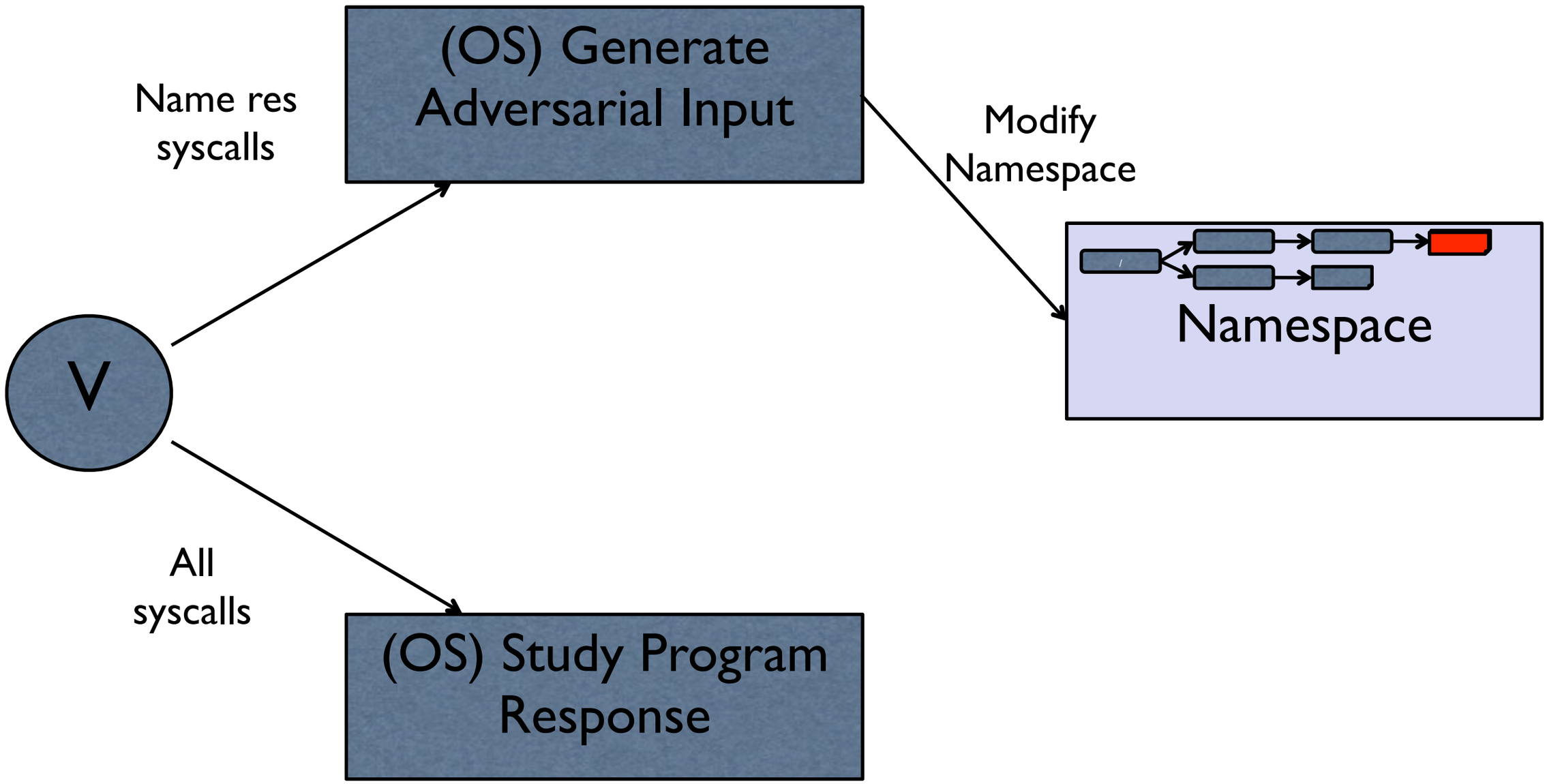
Solution Overview



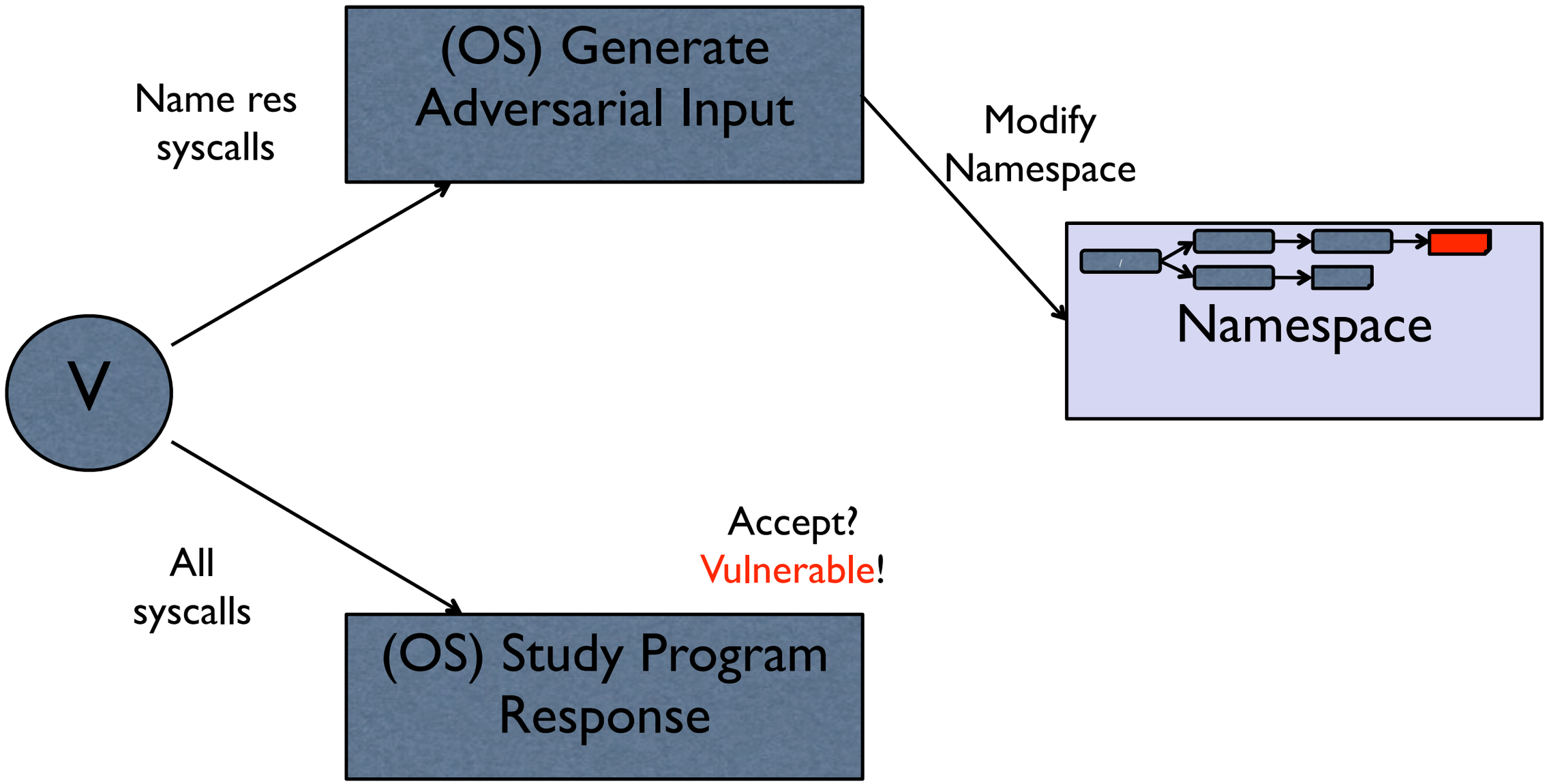
Solution Overview



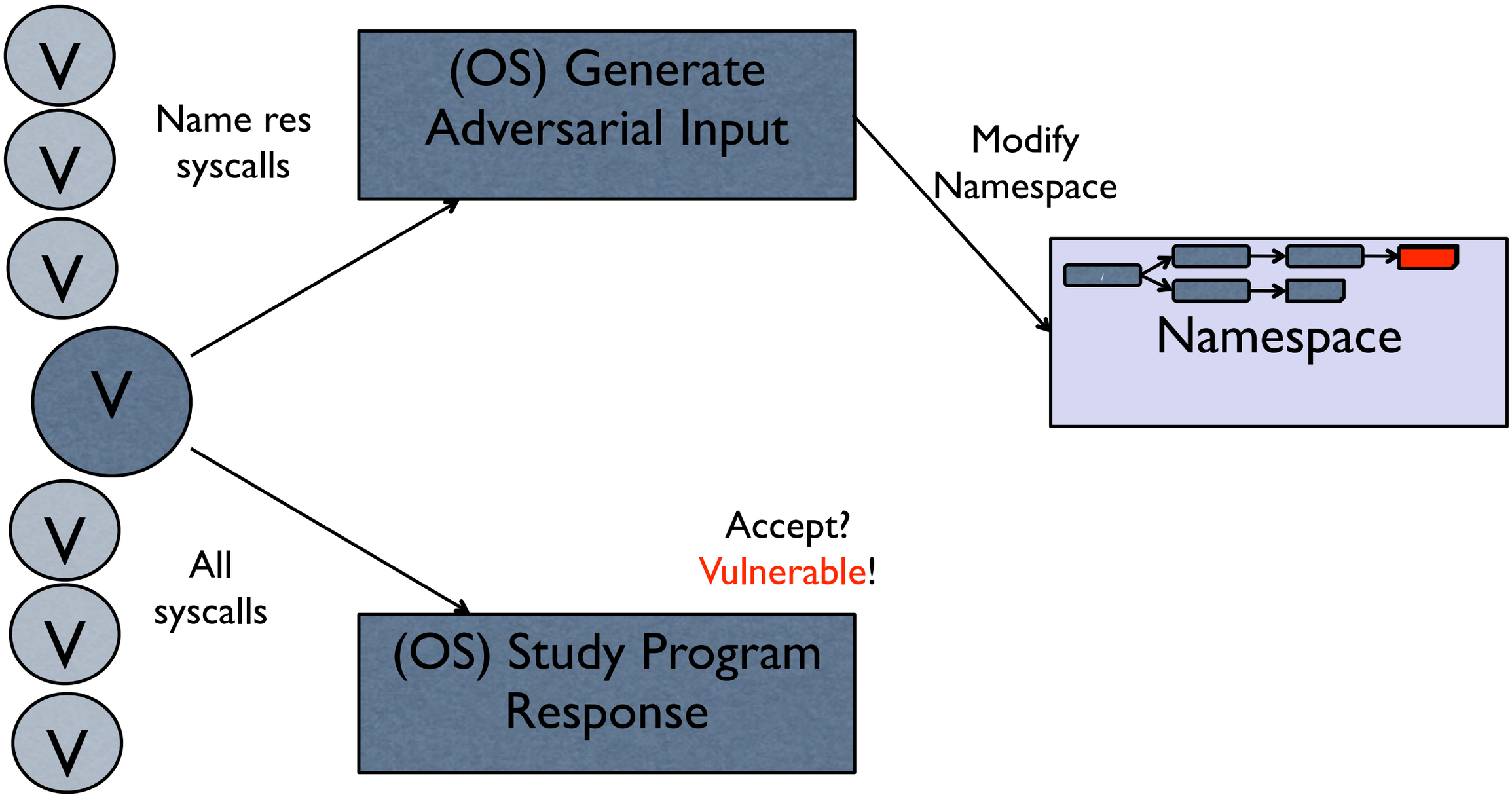
Solution Overview



Solution Overview

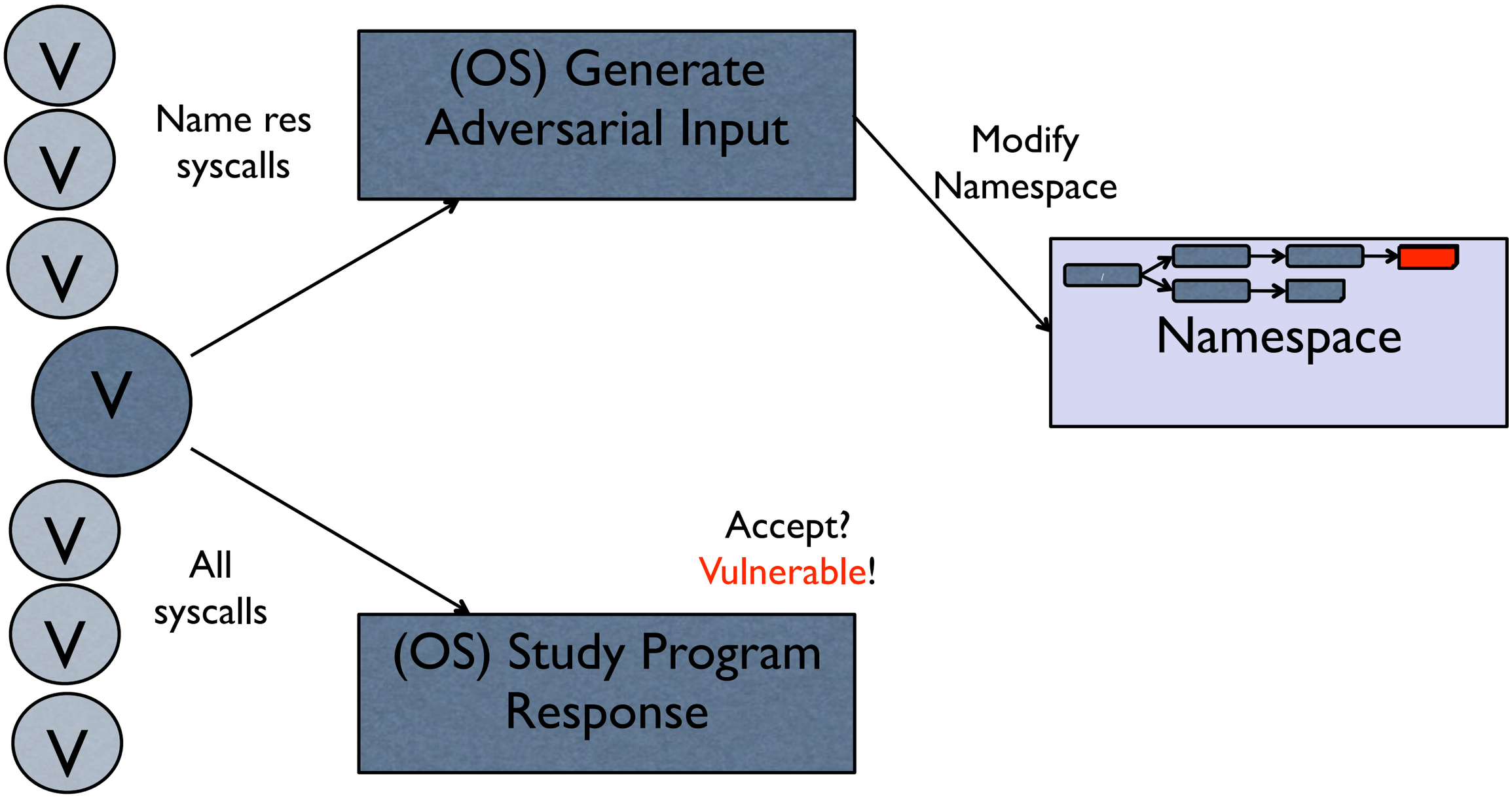


Solution Overview



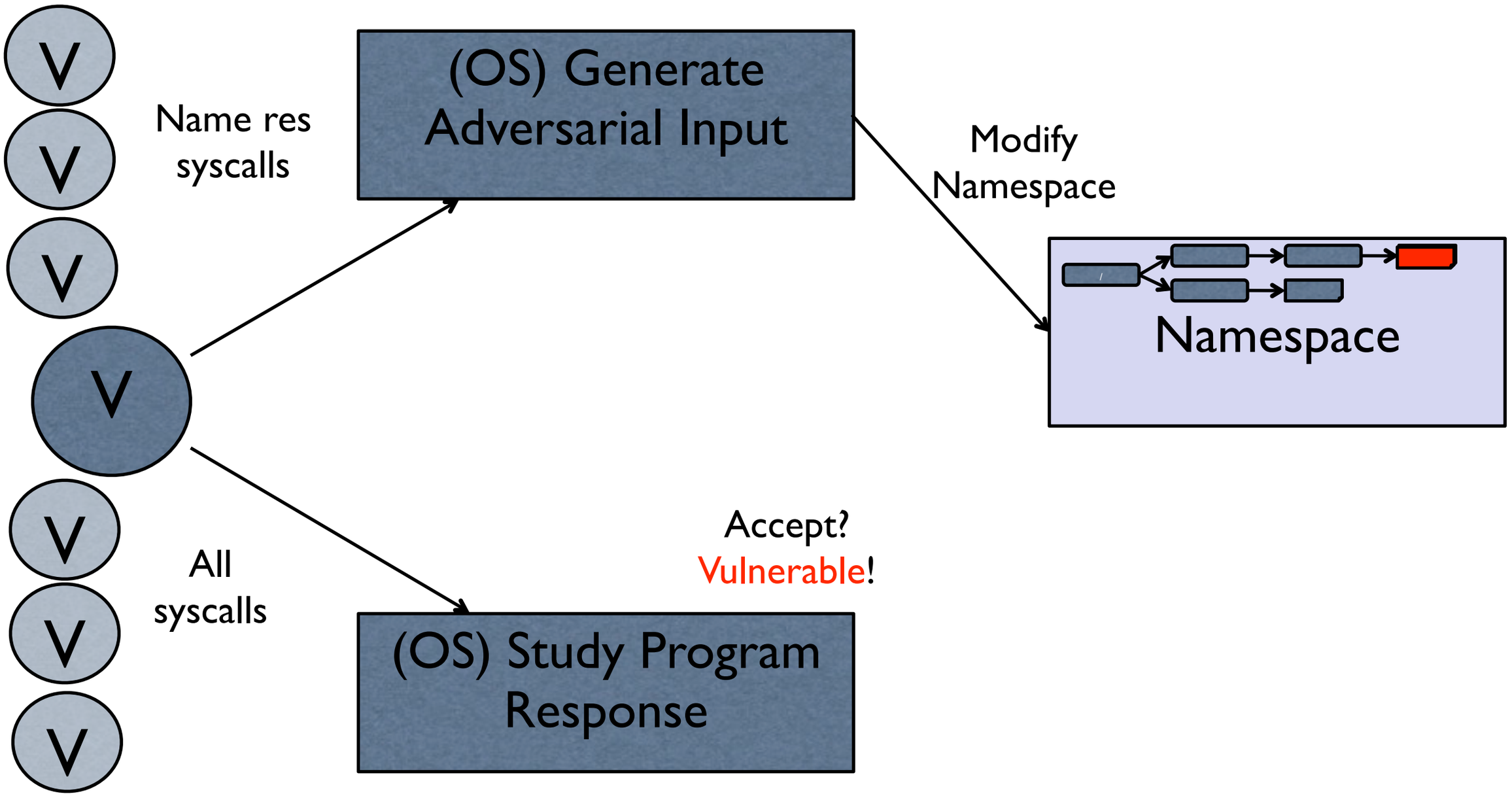
Solution Overview

System-wide?



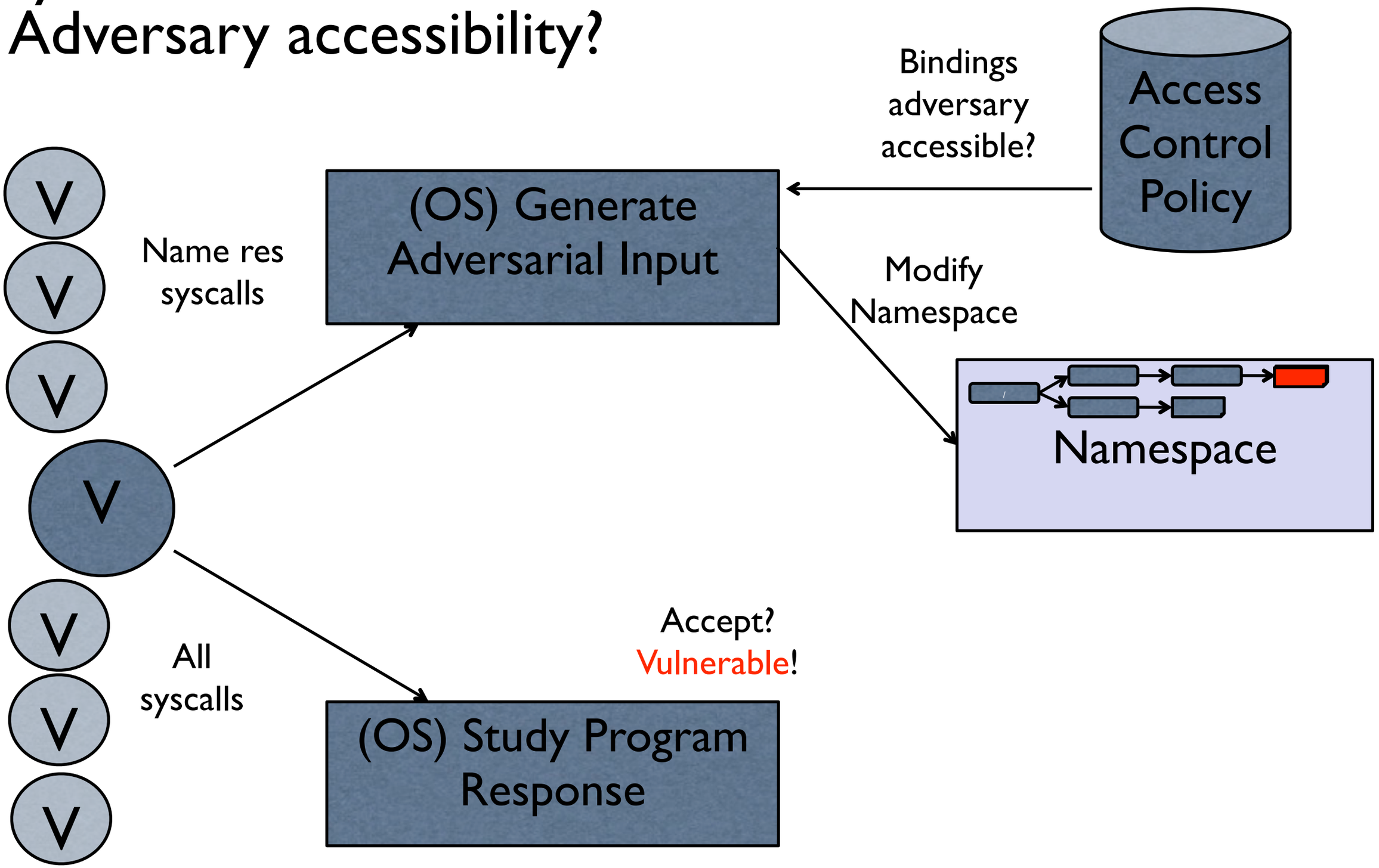
Solution Overview

System-wide?
Adversary accessibility?



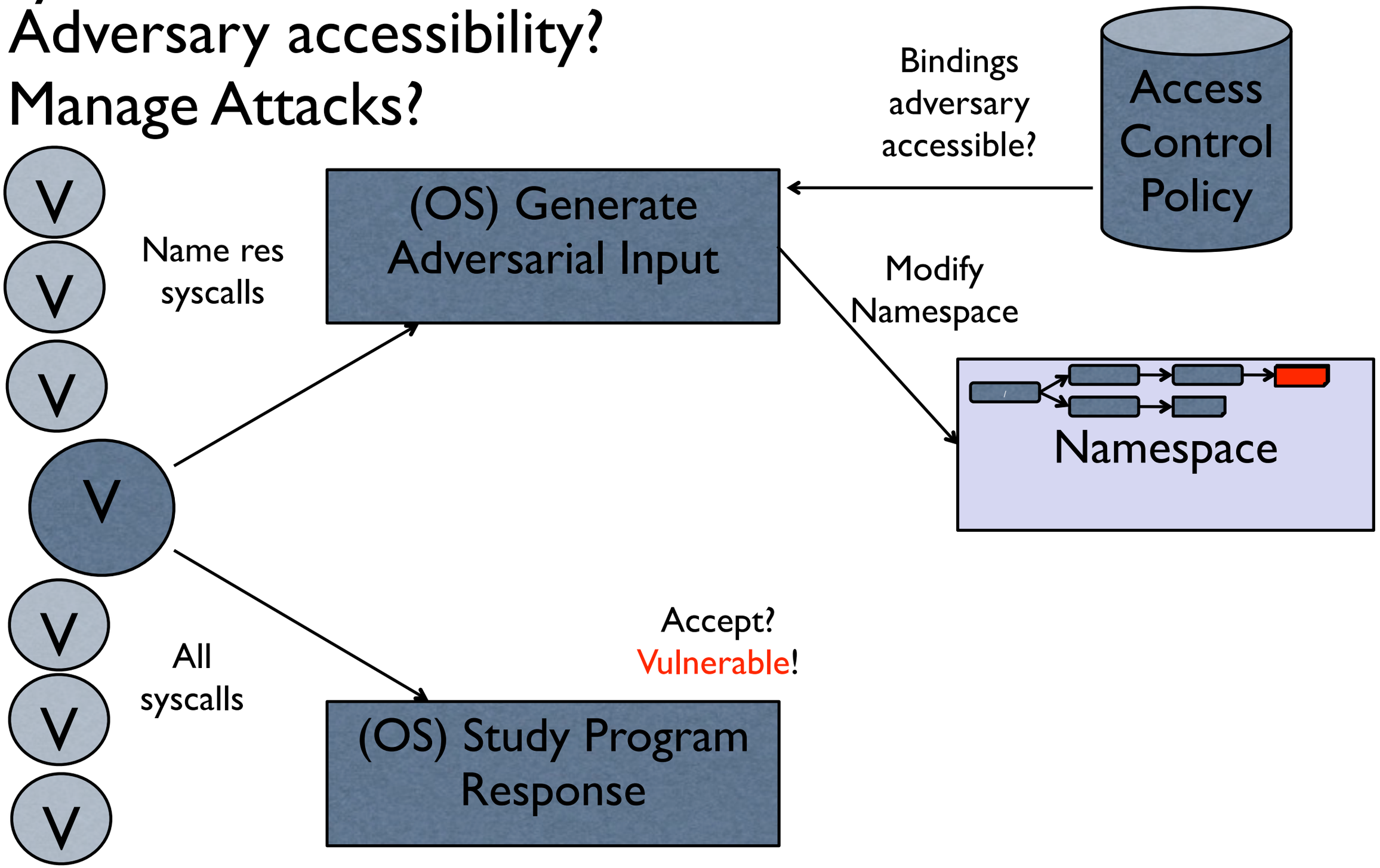
Solution Overview

System-wide?
Adversary accessibility?



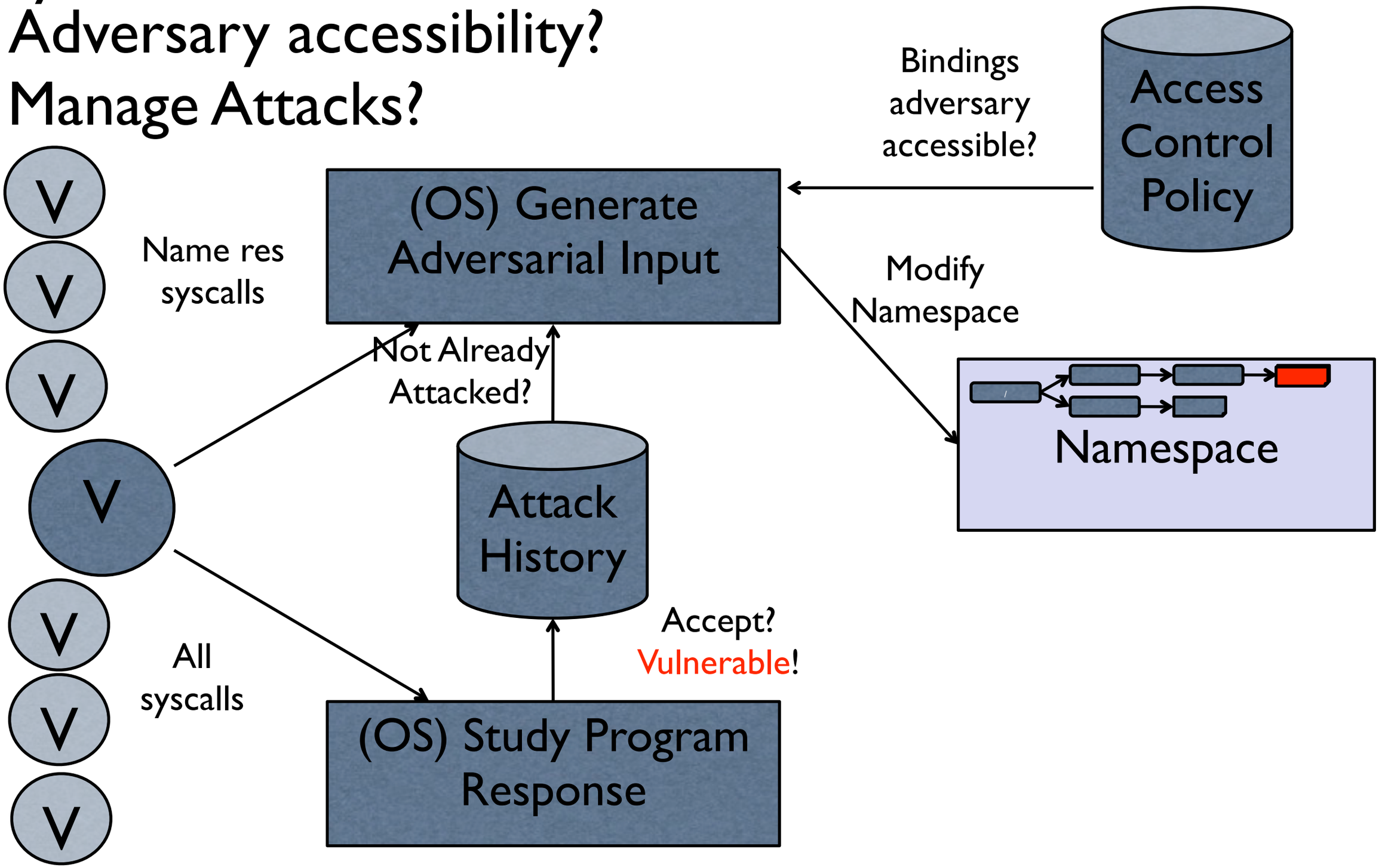
Solution Overview

System-wide?
Adversary accessibility?
Manage Attacks?



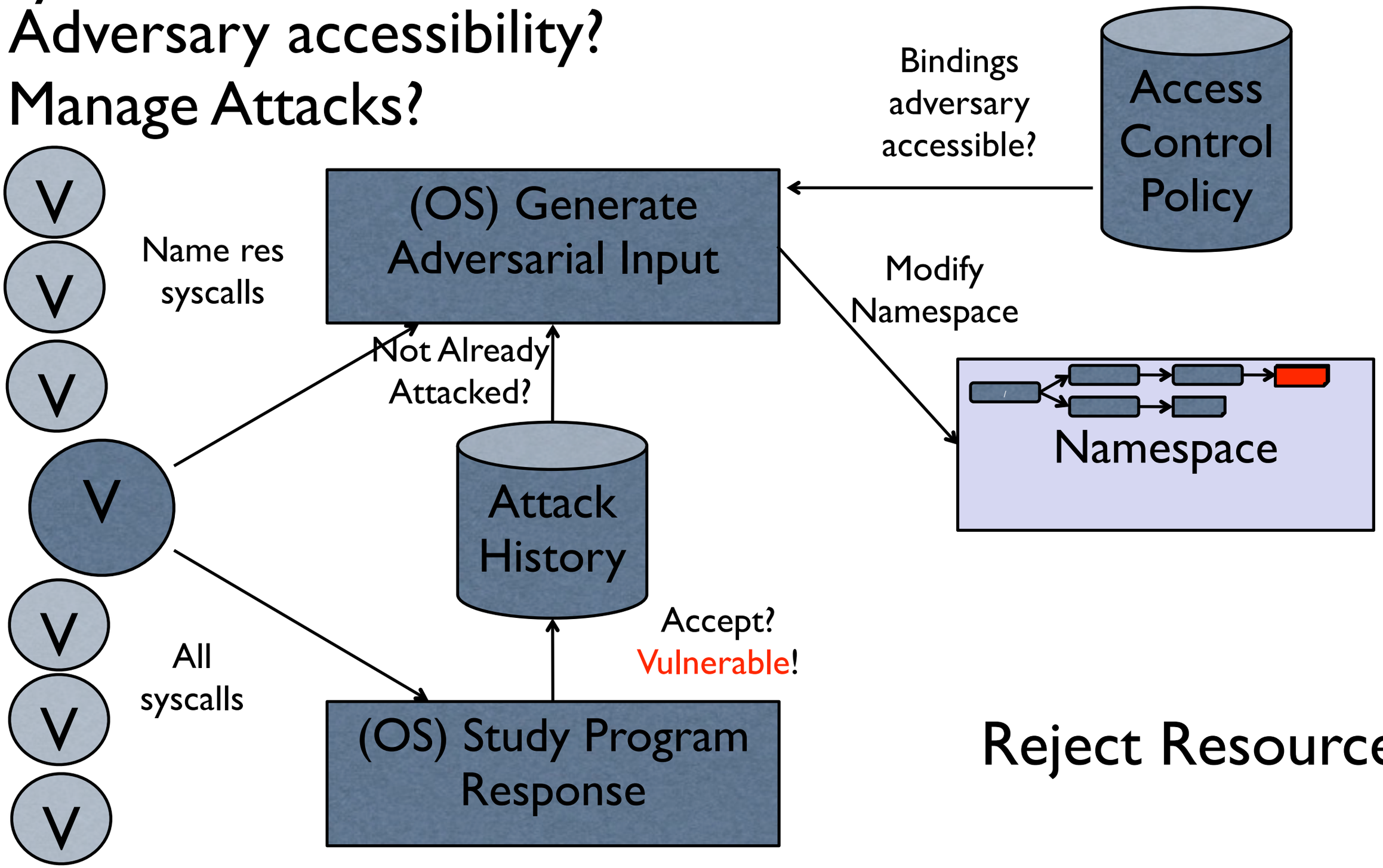
Solution Overview

System-wide?
Adversary accessibility?
Manage Attacks?



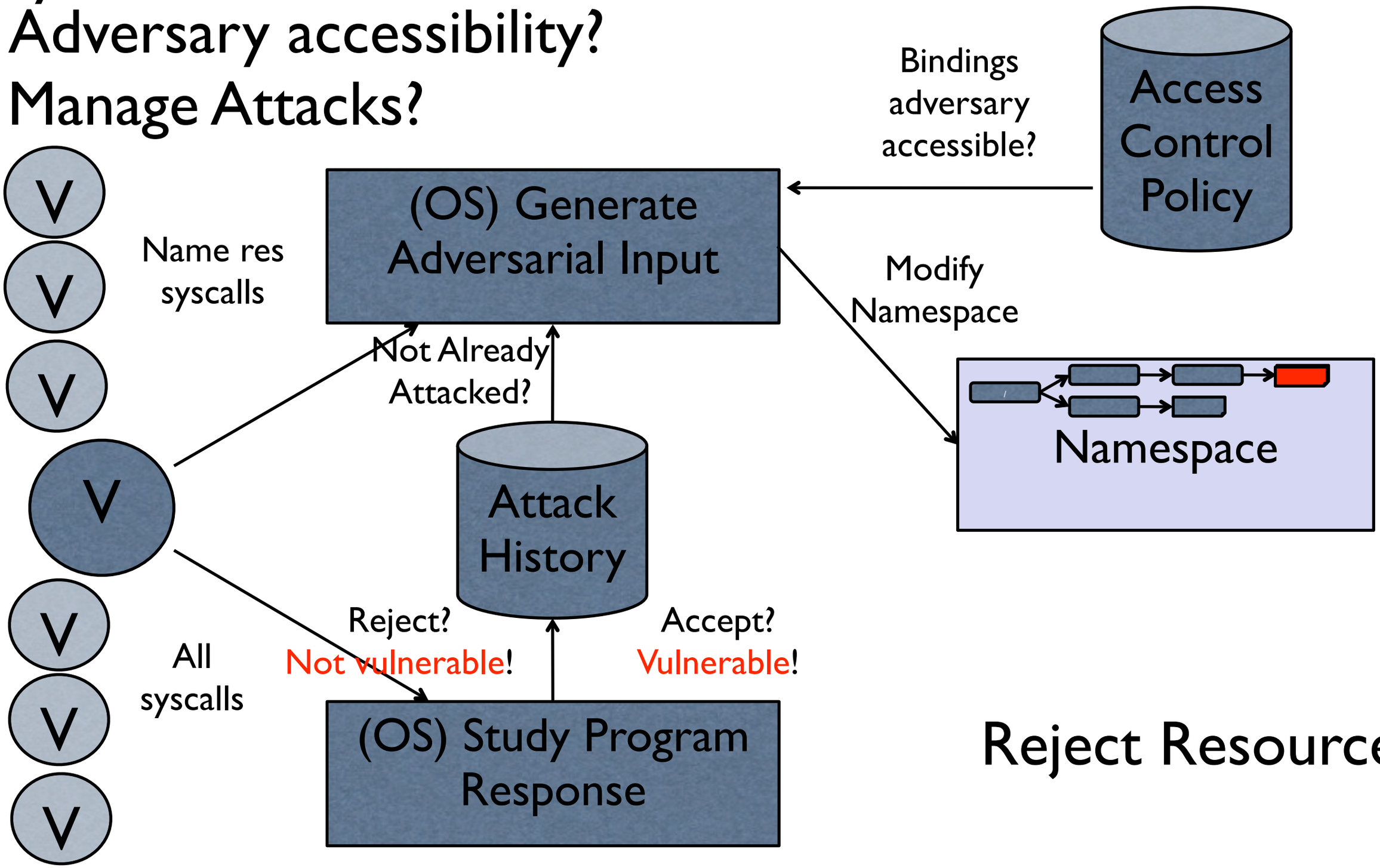
Solution Overview

System-wide?
Adversary accessibility?
Manage Attacks?



Solution Overview

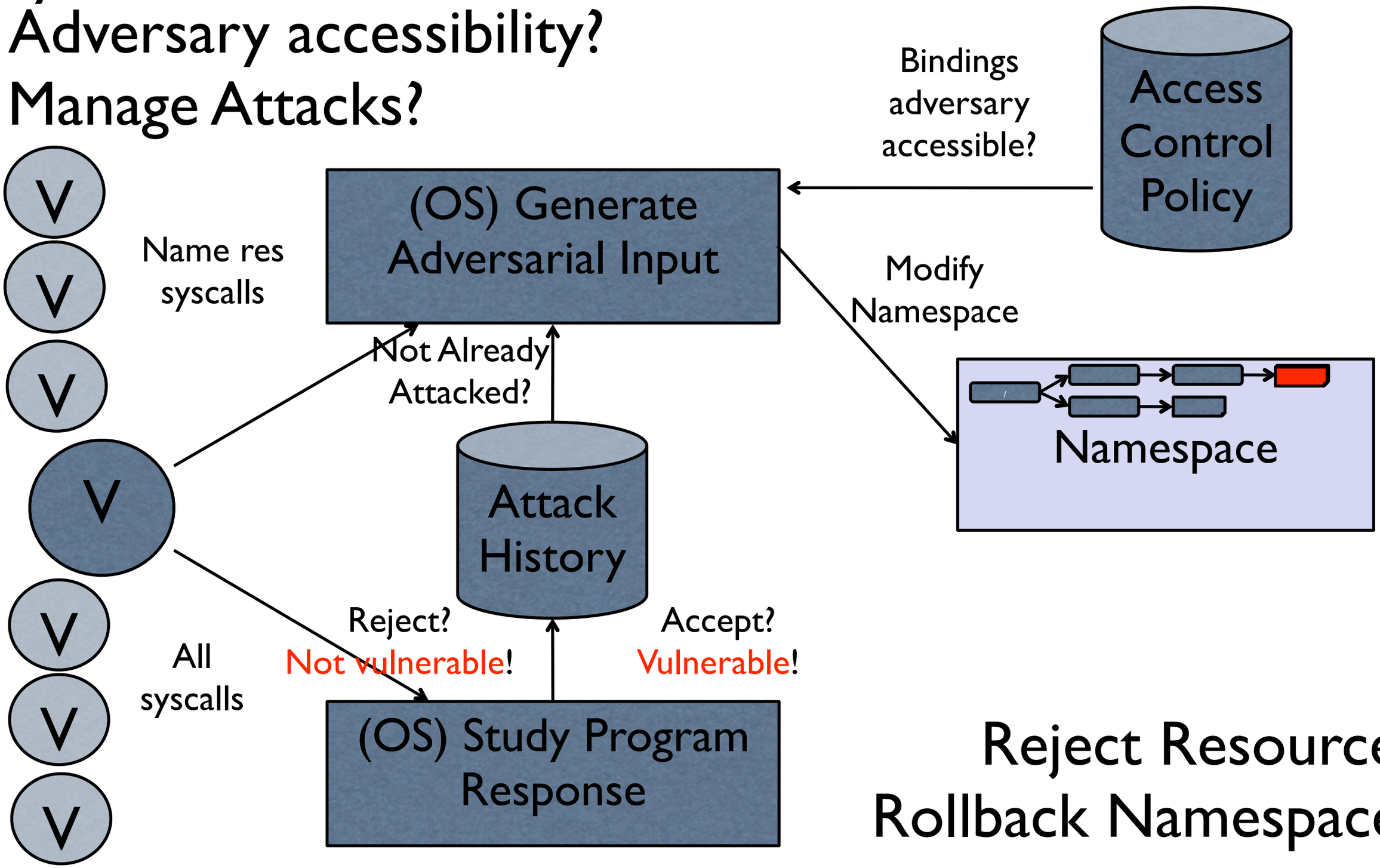
System-wide?
Adversary accessibility?
Manage Attacks?



Reject Resource?

Solution Overview

System-wide?
Adversary accessibility?
Manage Attacks?

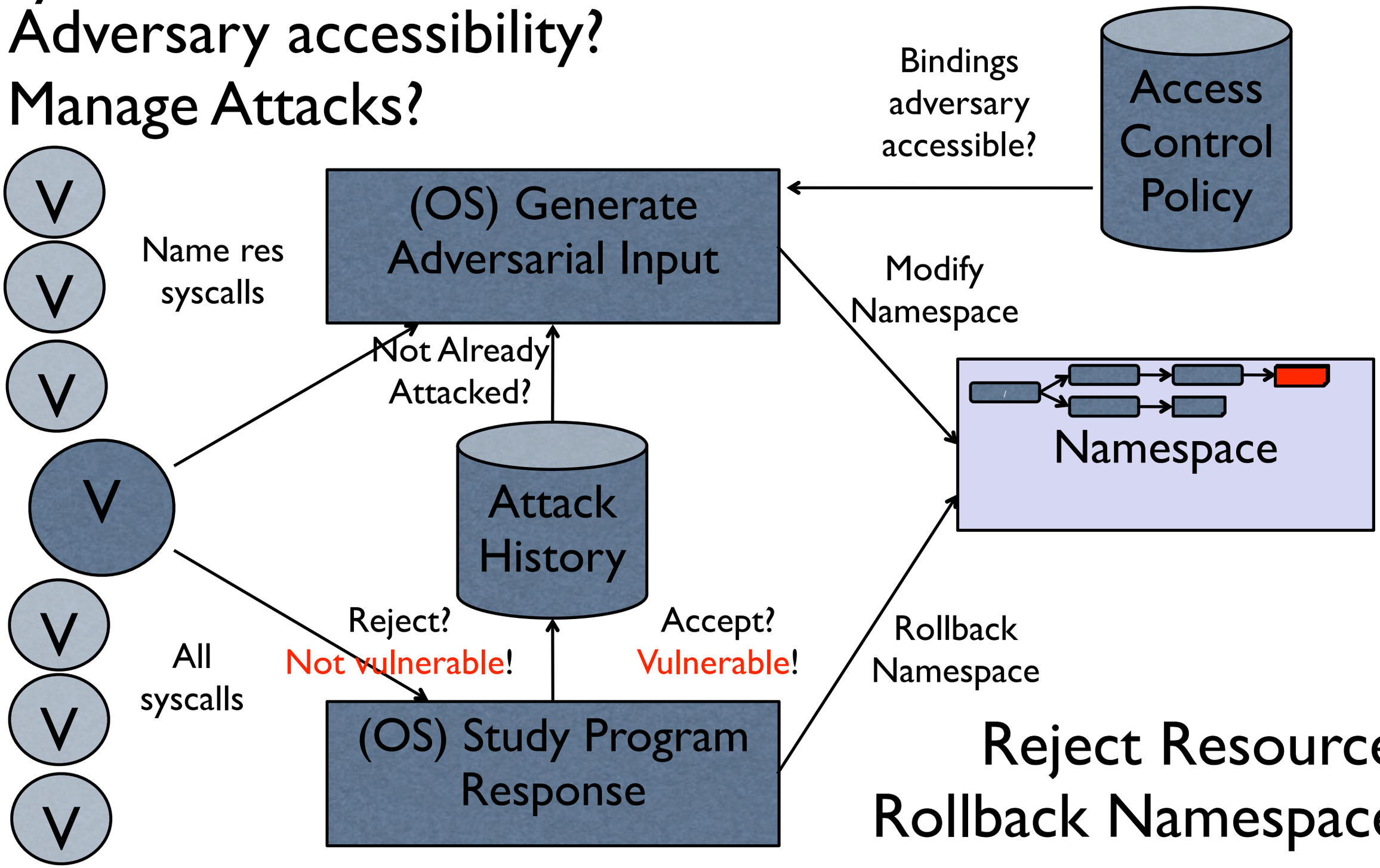


Reject Resource?
Rollback Namespace?

Solution Overview

PENNSTATE 1855

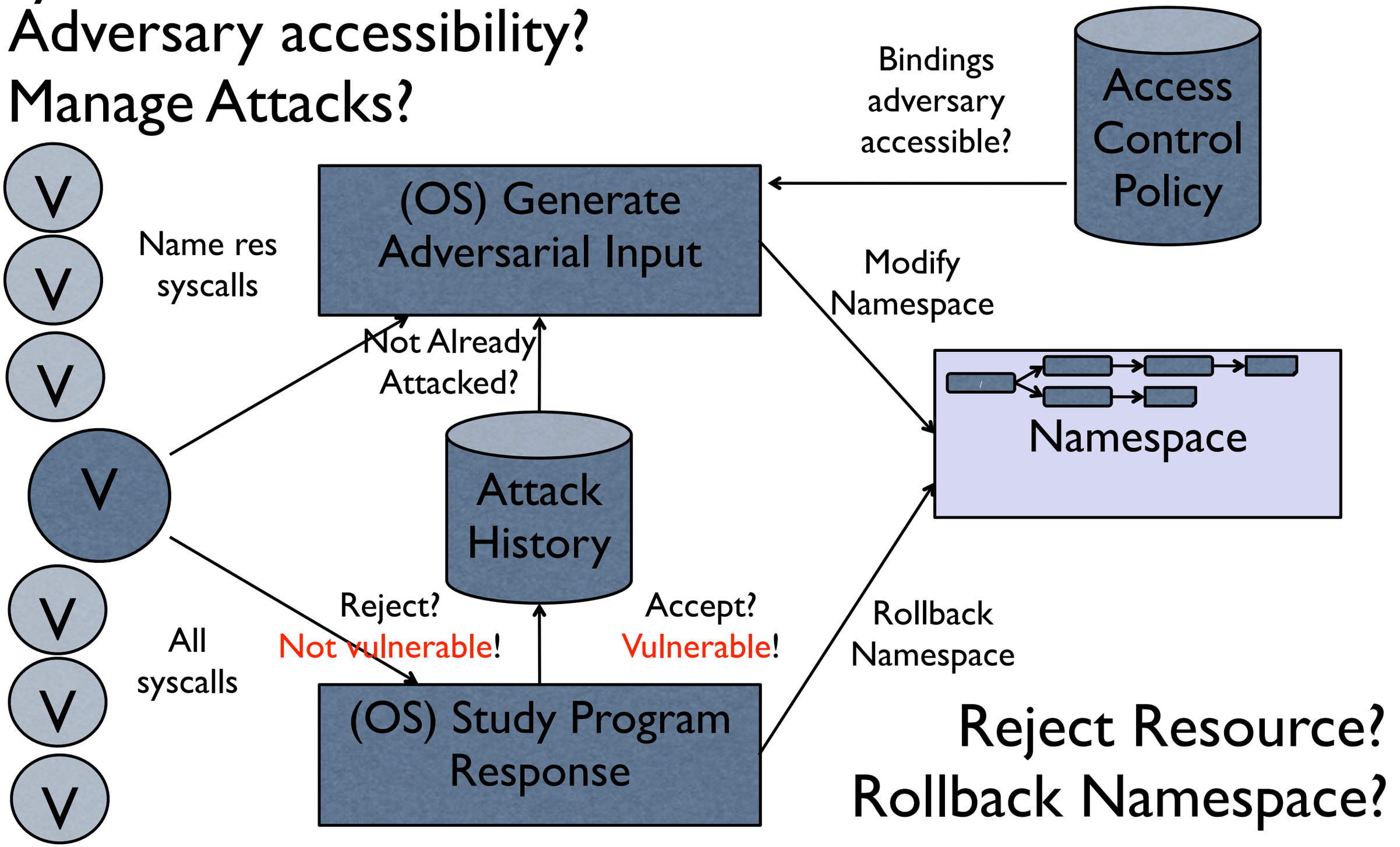
System-wide?
Adversary accessibility?
Manage Attacks?



Reject Resource?
Rollback Namespace?

Solution Overview

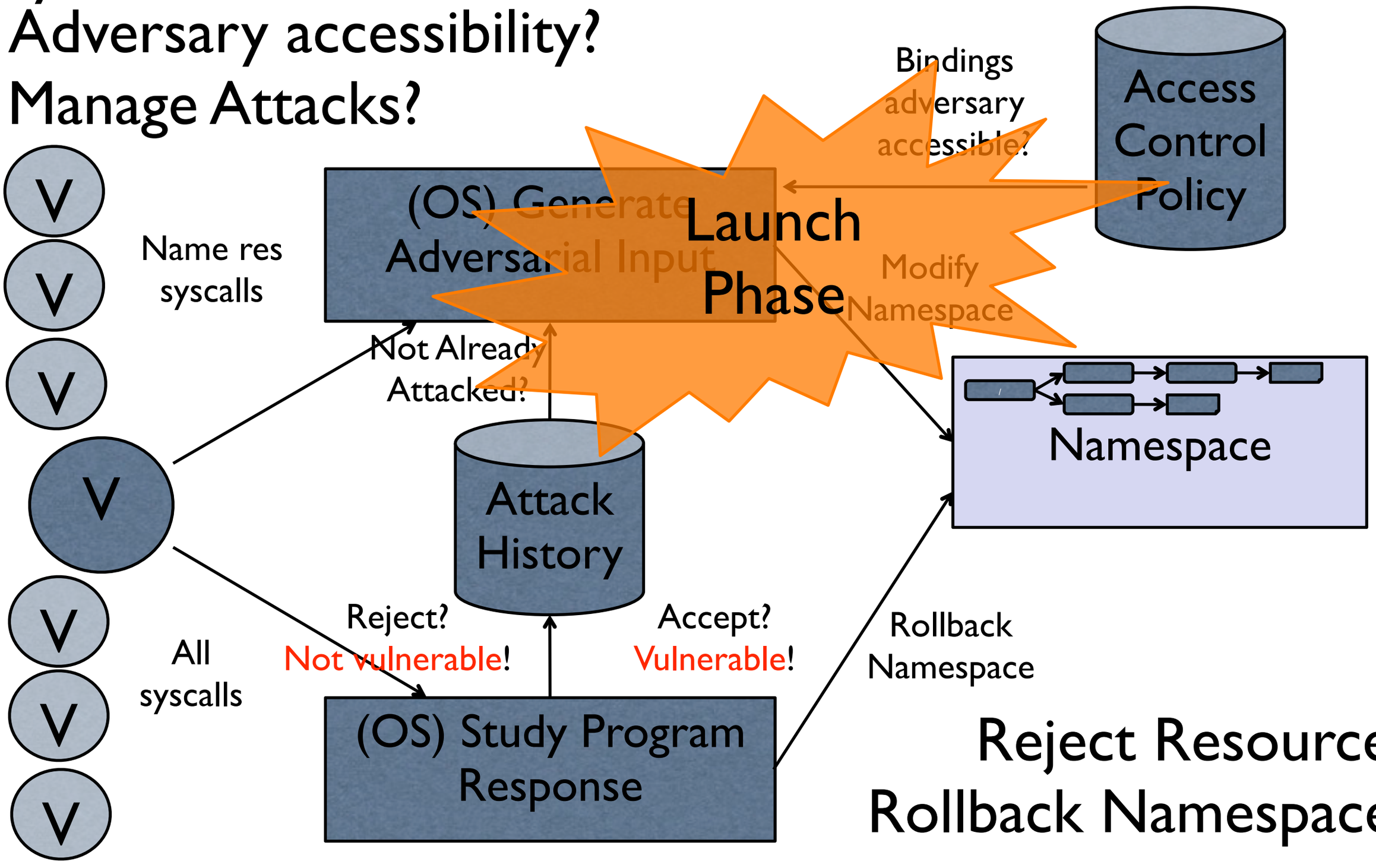
System-wide?
Adversary accessibility?
Manage Attacks?



Reject Resource?
Rollback Namespace?

Solution Overview

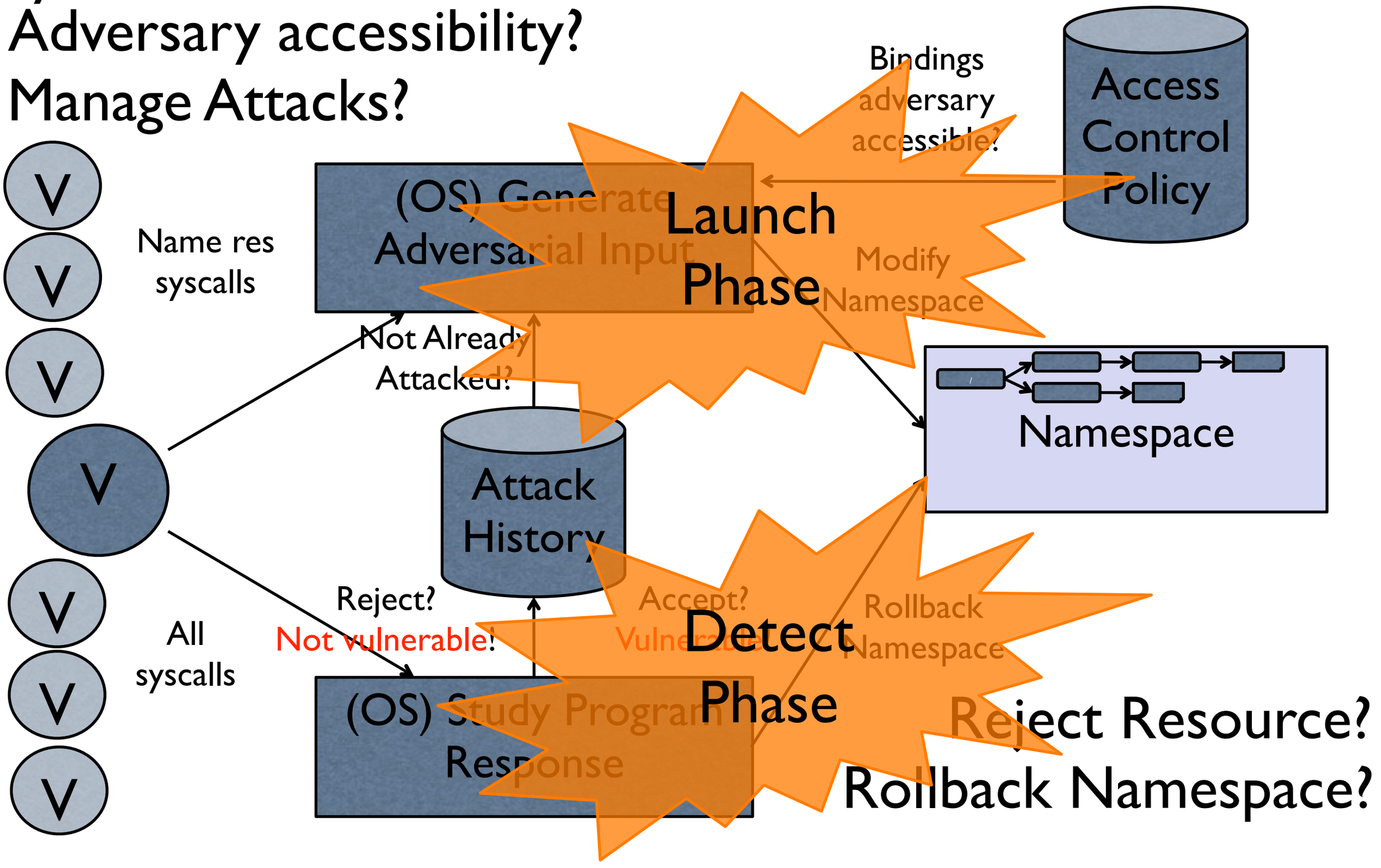
System-wide?
Adversary accessibility?
Manage Attacks?



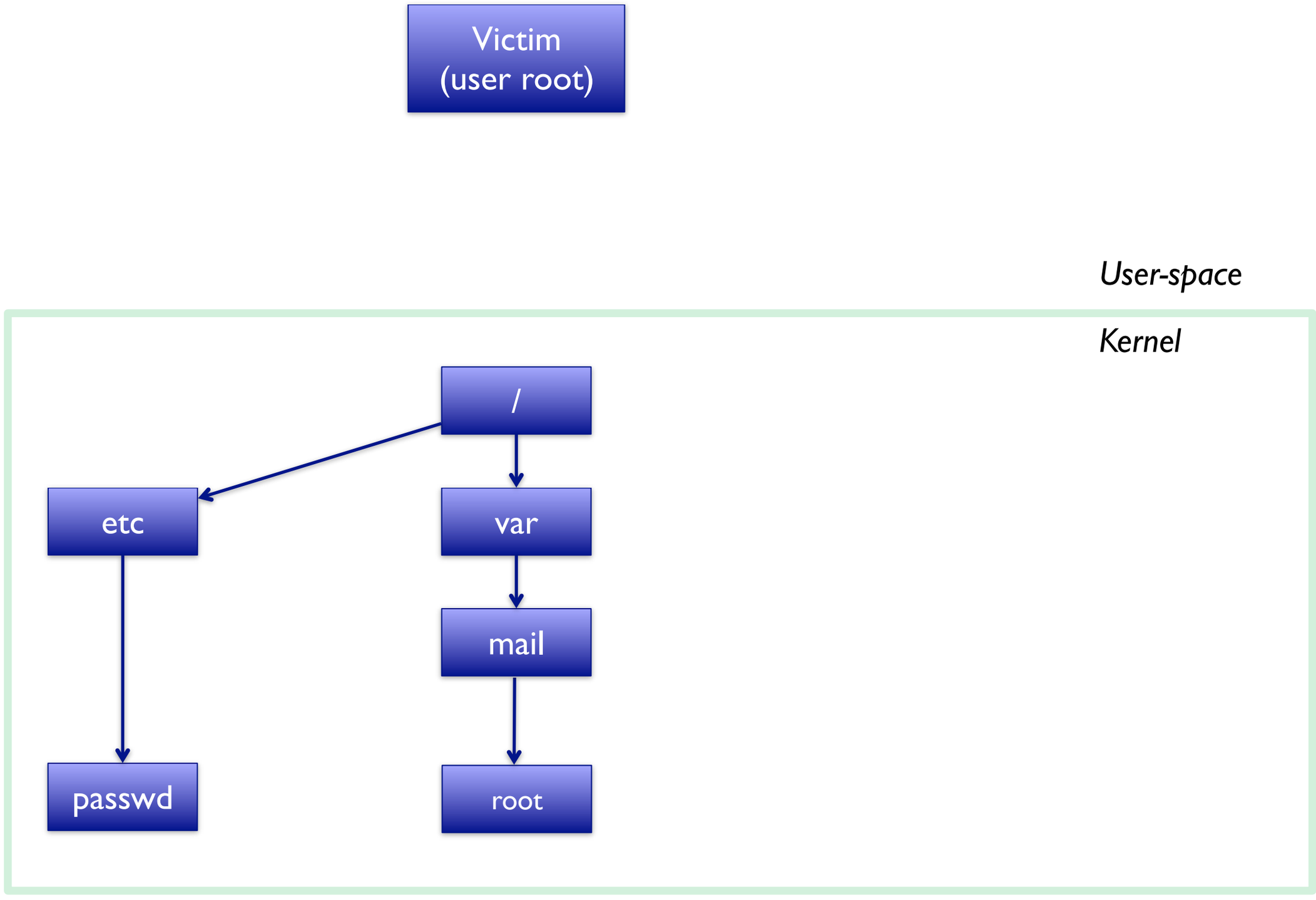
Reject Resource?
Rollback Namespace?

Solution Overview

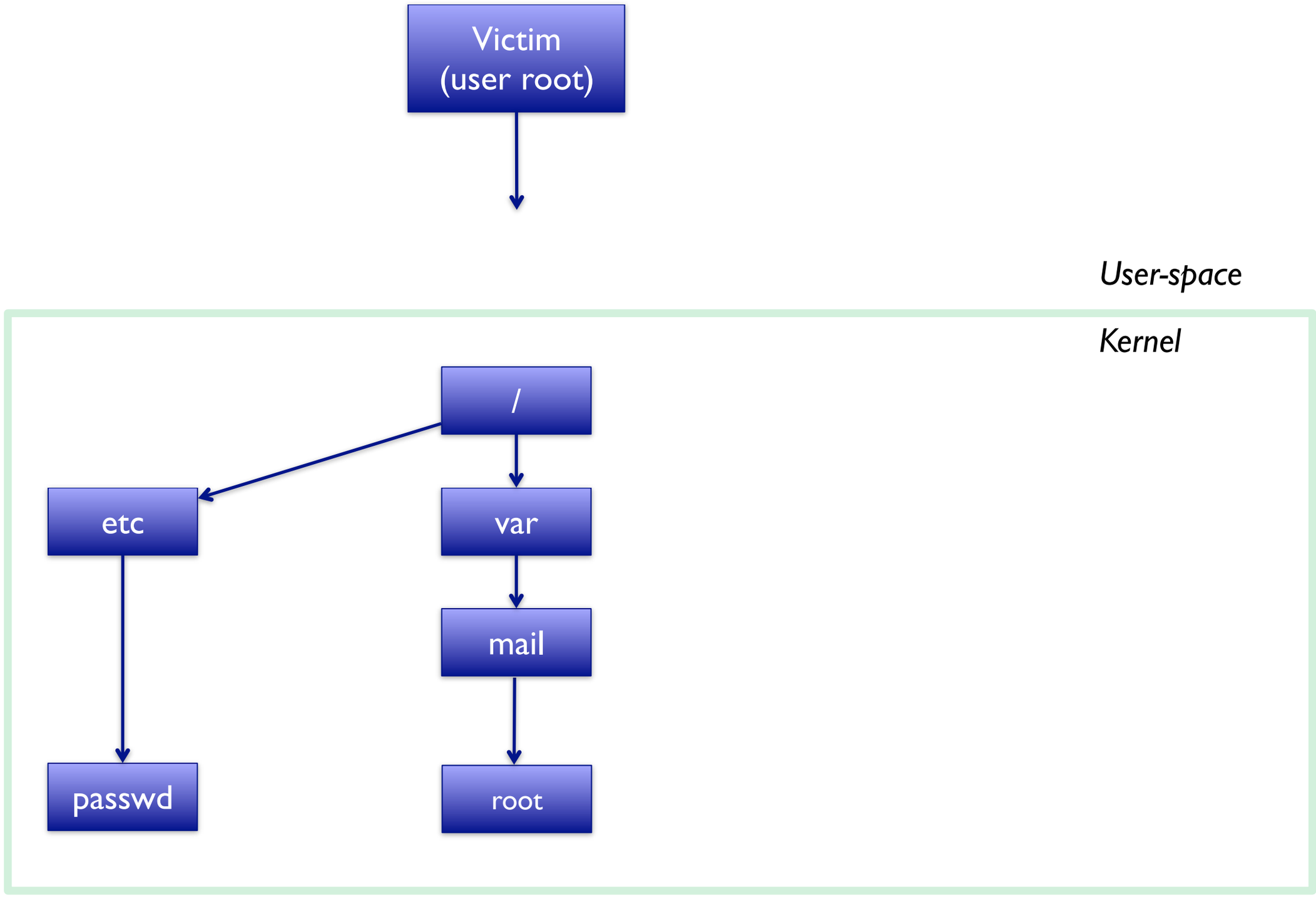
System-wide?
Adversary accessibility?
Manage Attacks?



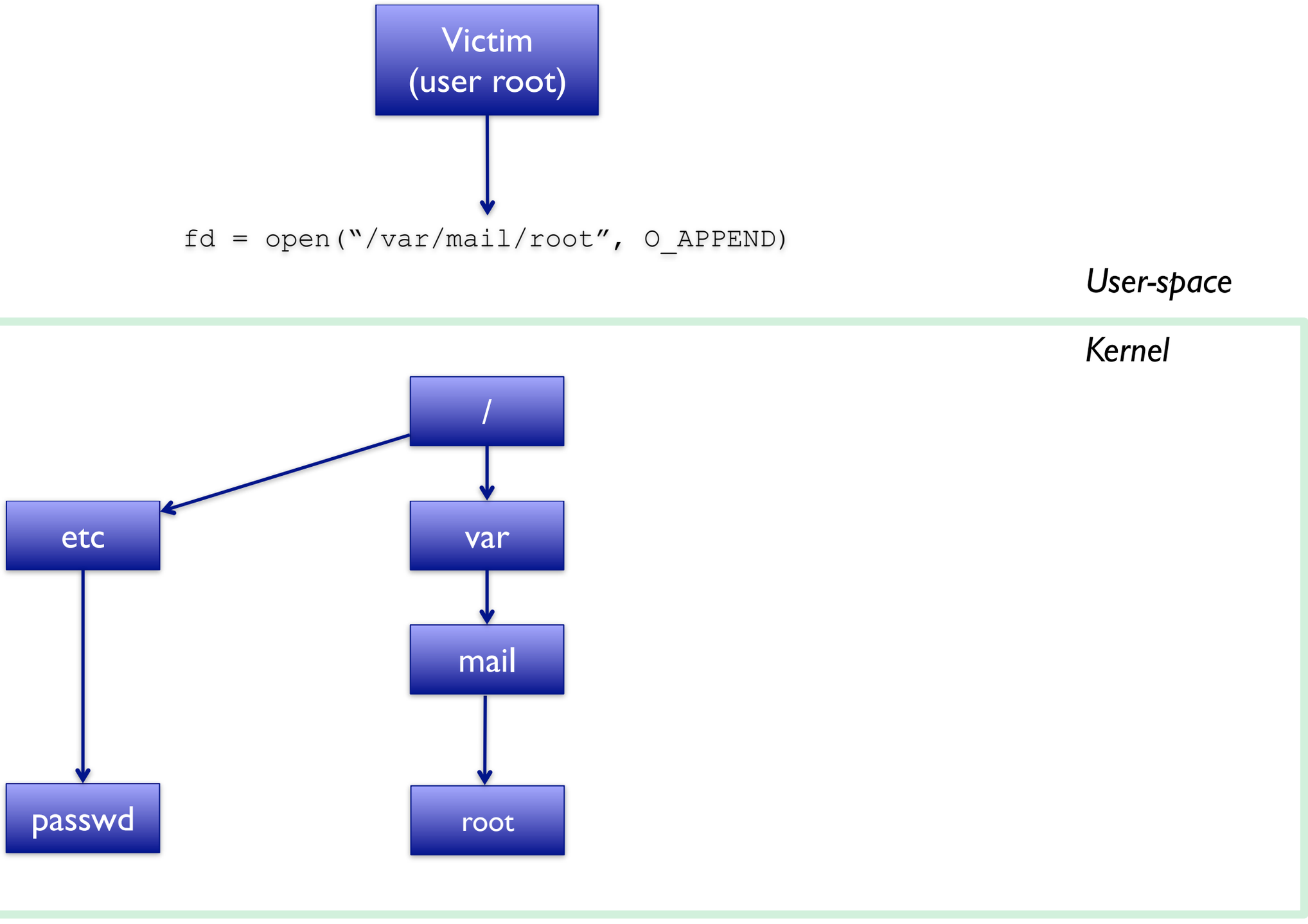
Launch Phase



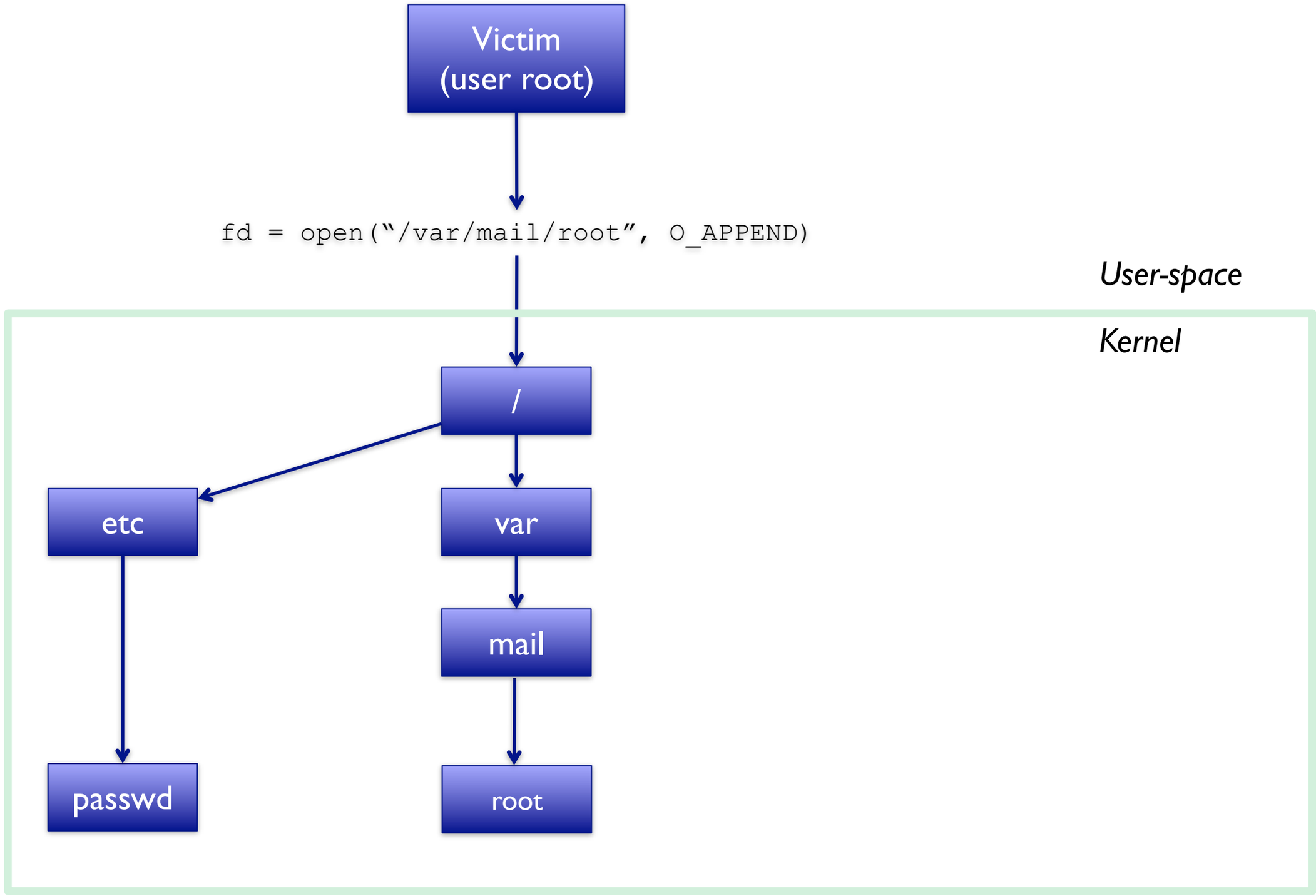
Launch Phase



Launch Phase

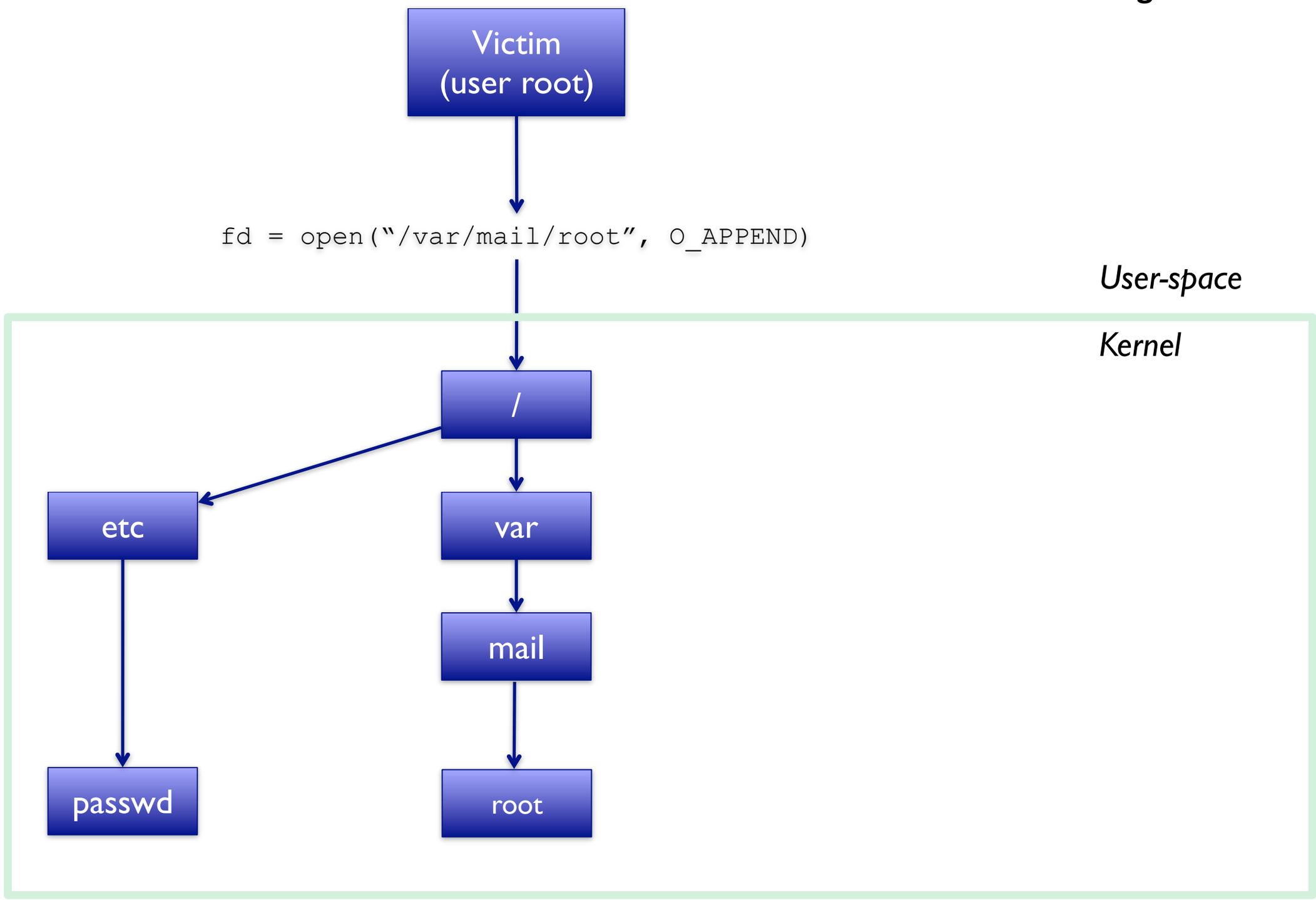


Launch Phase



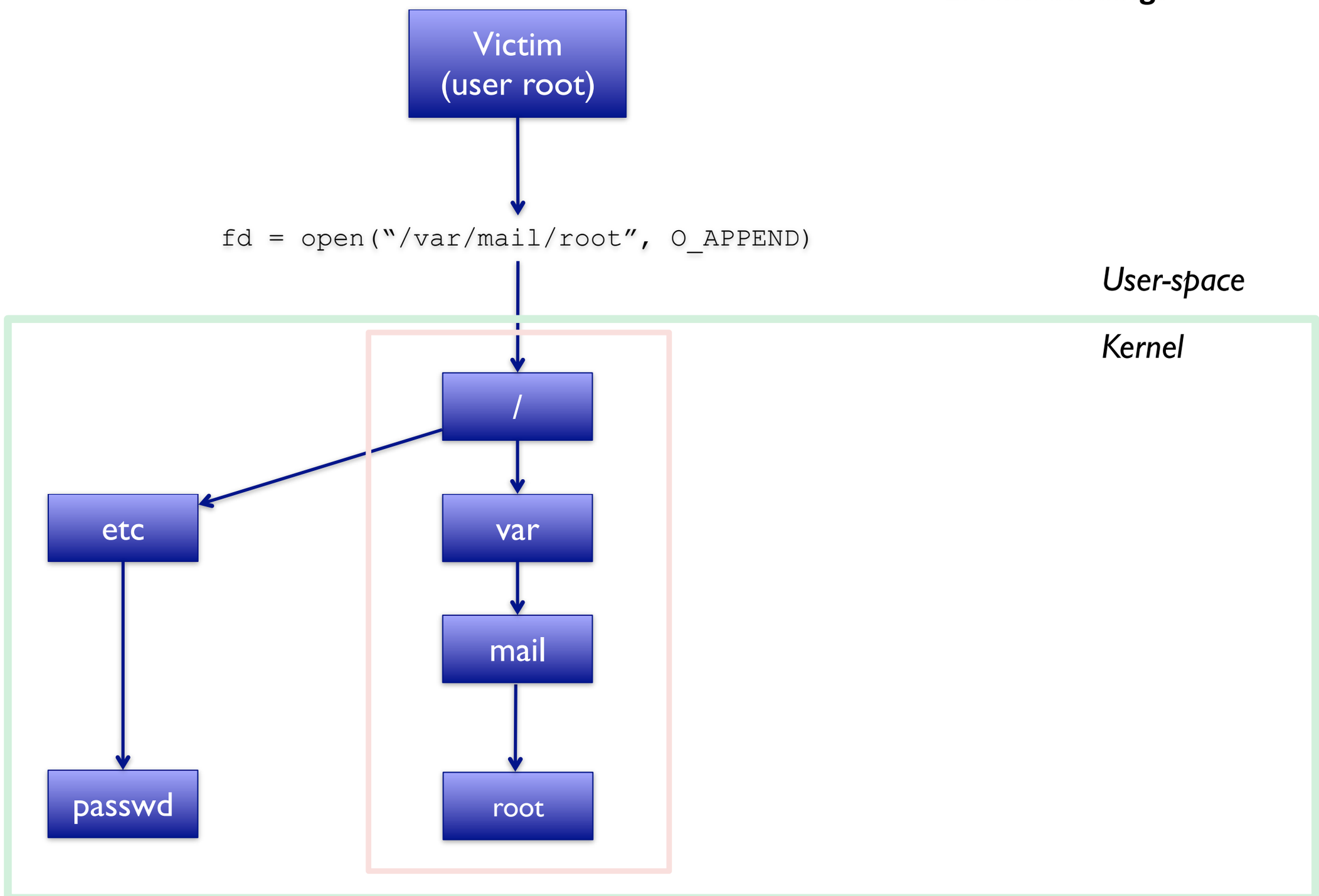
Launch Phase

1. Find bindings



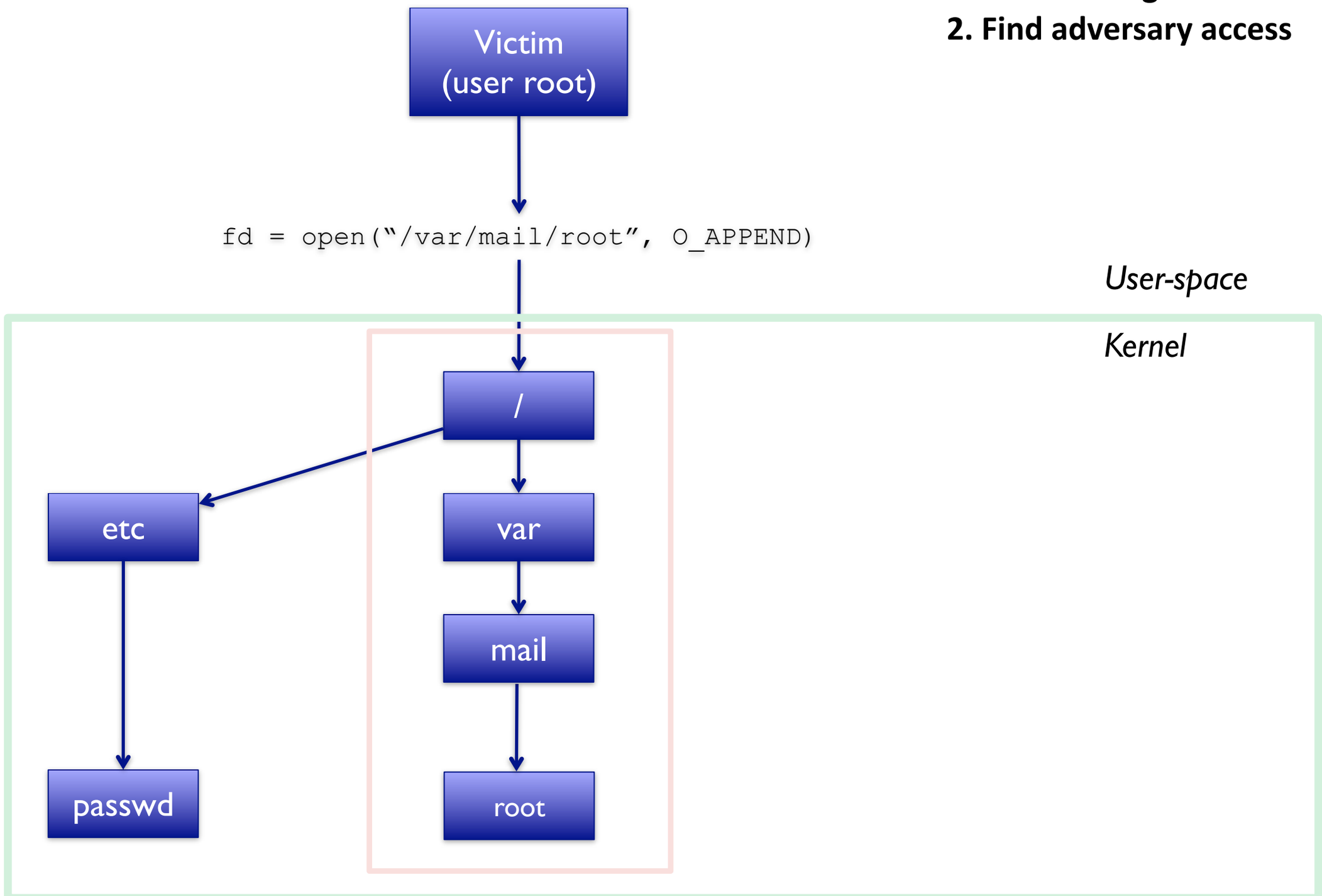
Launch Phase

1. Find bindings



Launch Phase

1. **Find bindings**
2. **Find adversary access**



Launch Phase

1. **Find bindings**
2. **Find adversary access**

Victim
(user root)

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

User-space

Kernel

/

etc

var

mail

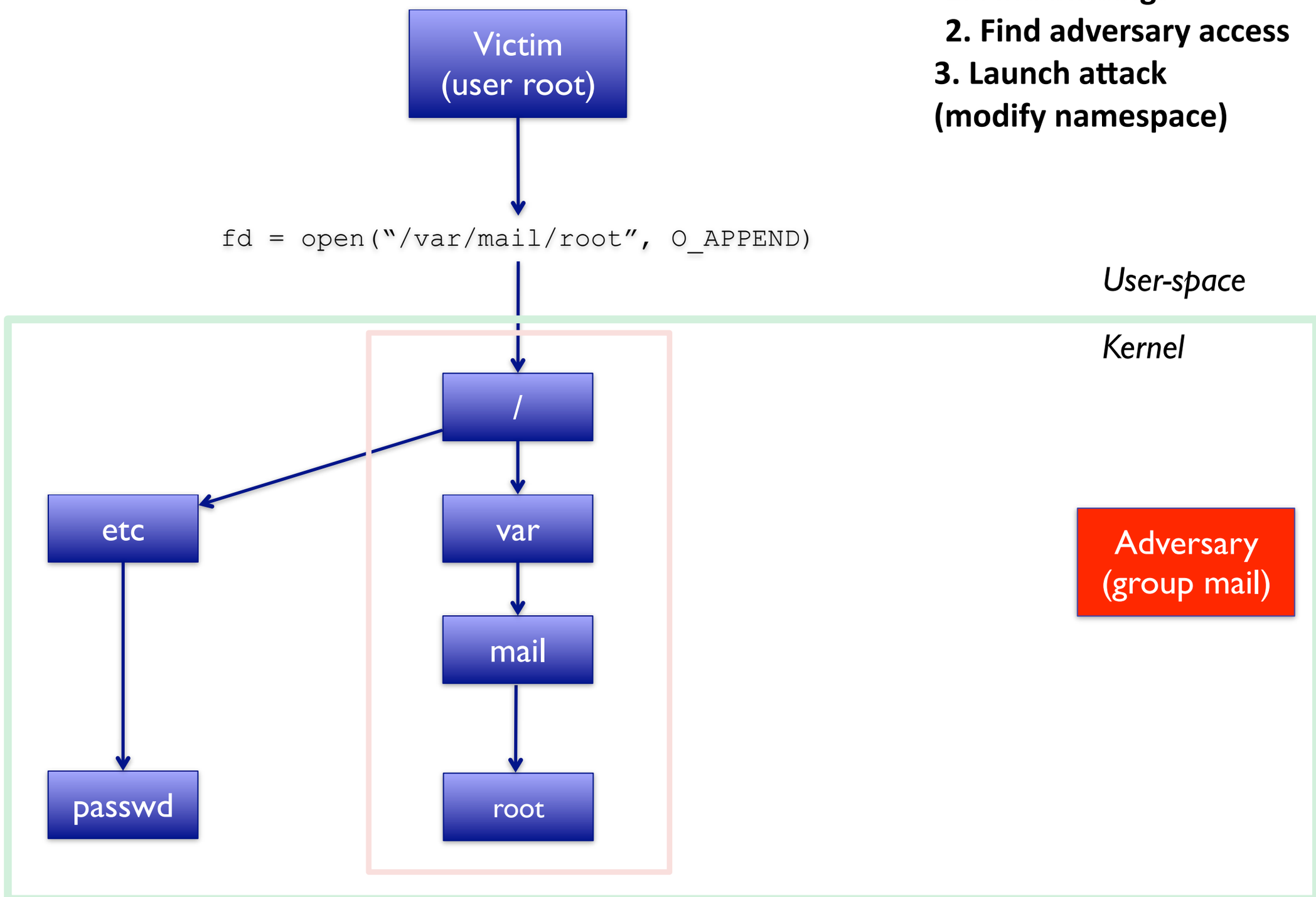
passwd

root

Adversary
(group mail)

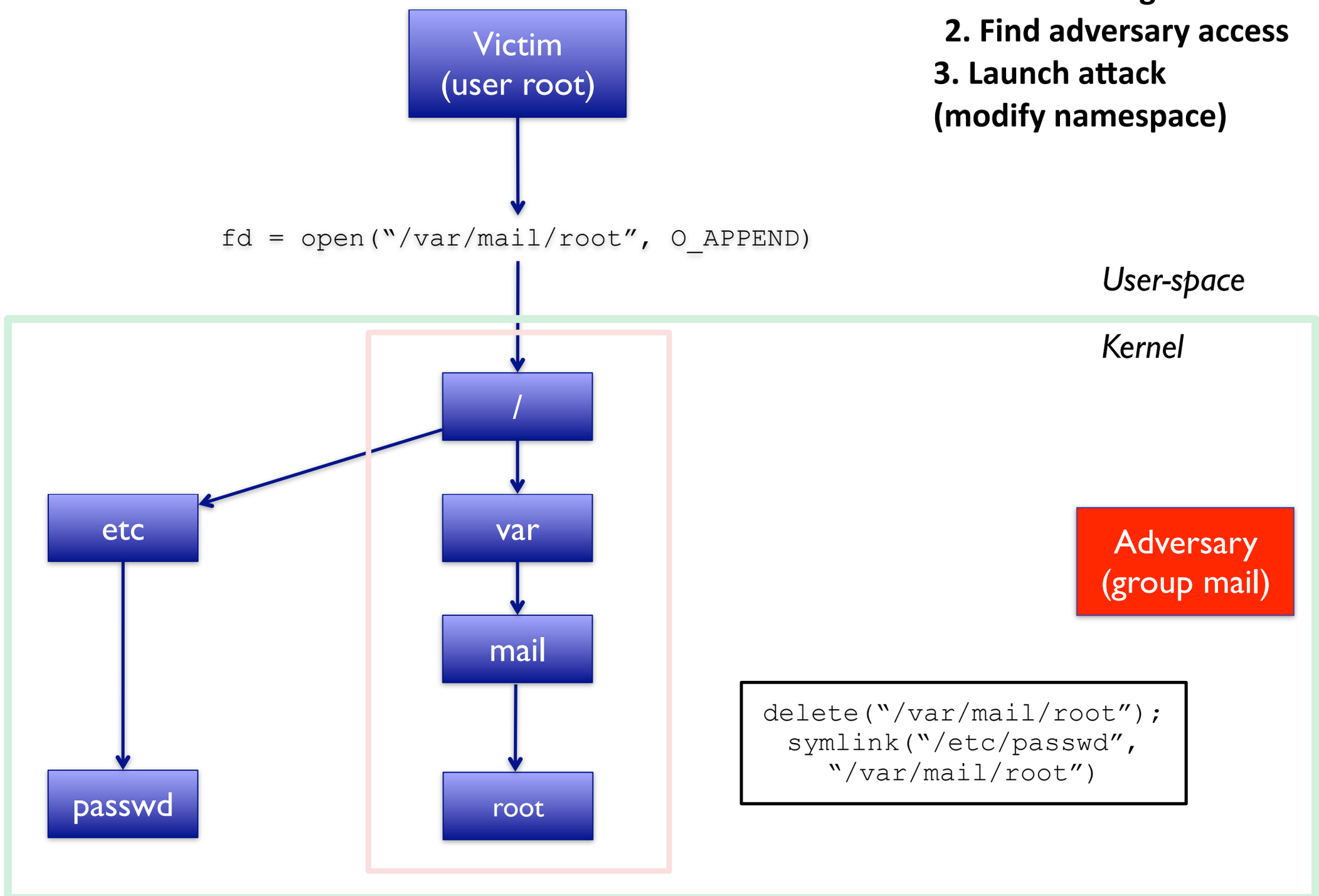
Launch Phase

1. Find bindings
2. Find adversary access
3. Launch attack (modify namespace)



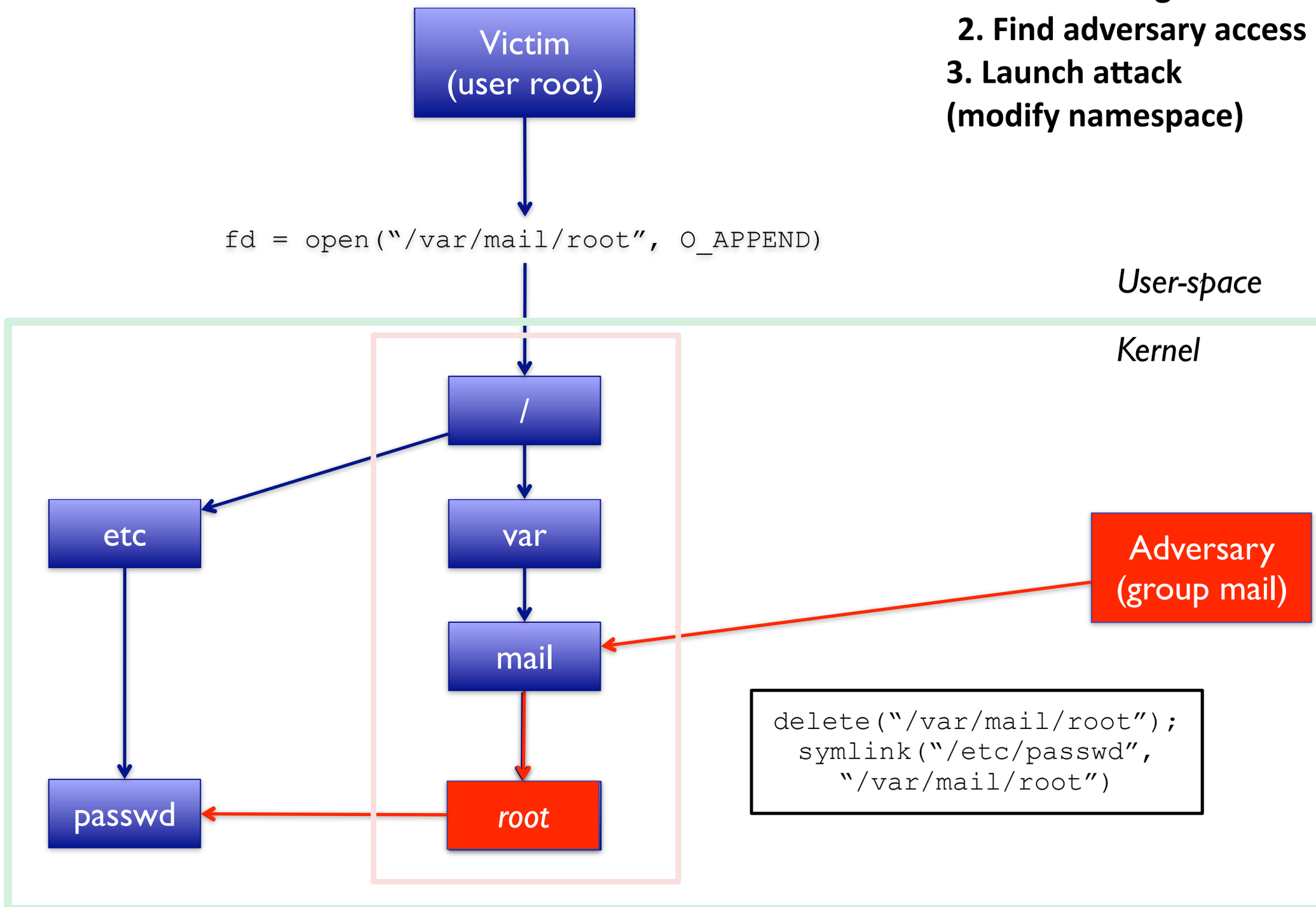
Launch Phase

1. Find bindings
2. Find adversary access
3. Launch attack (modify namespace)



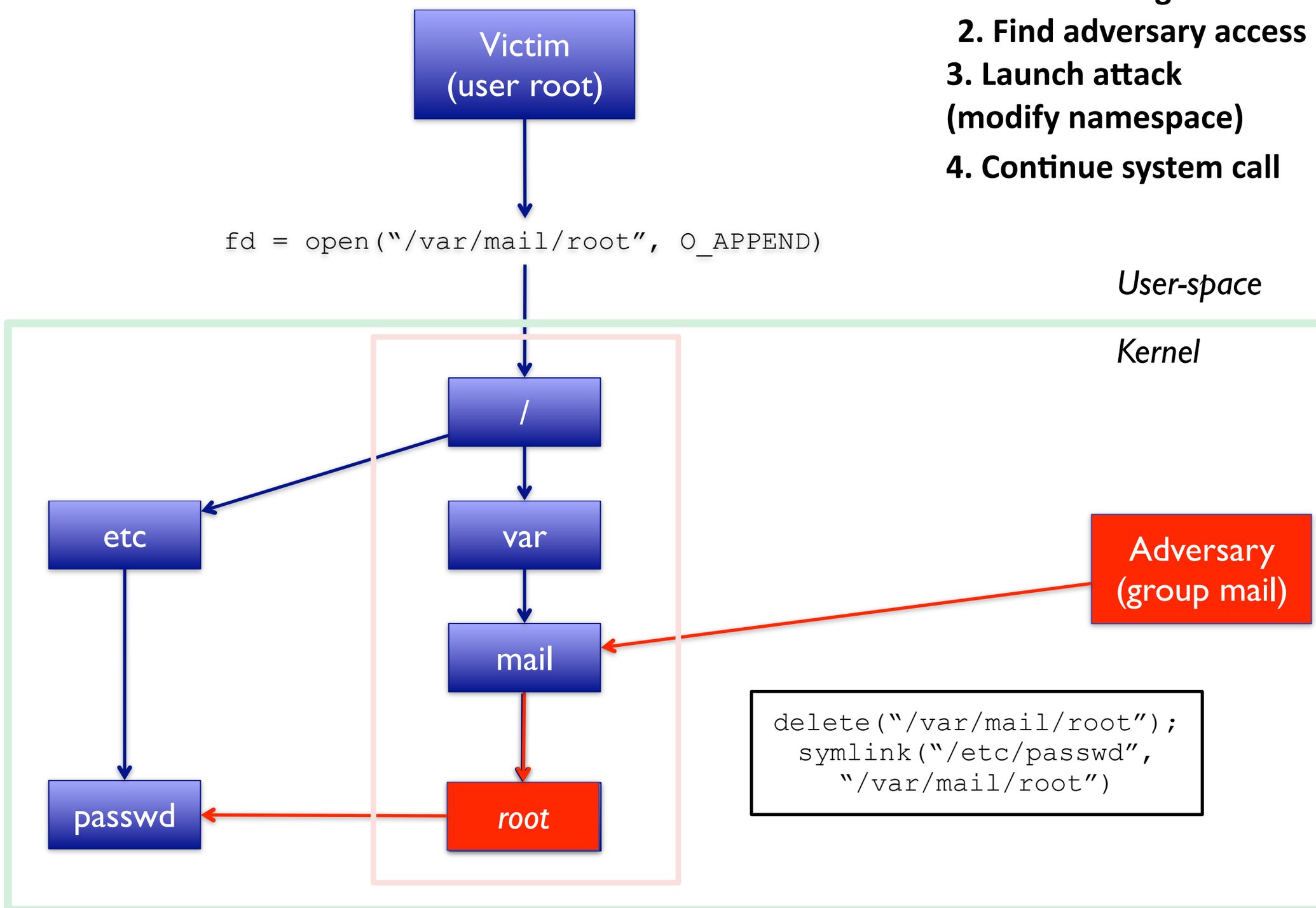
Launch Phase

1. Find bindings
2. Find adversary access
3. Launch attack (modify namespace)

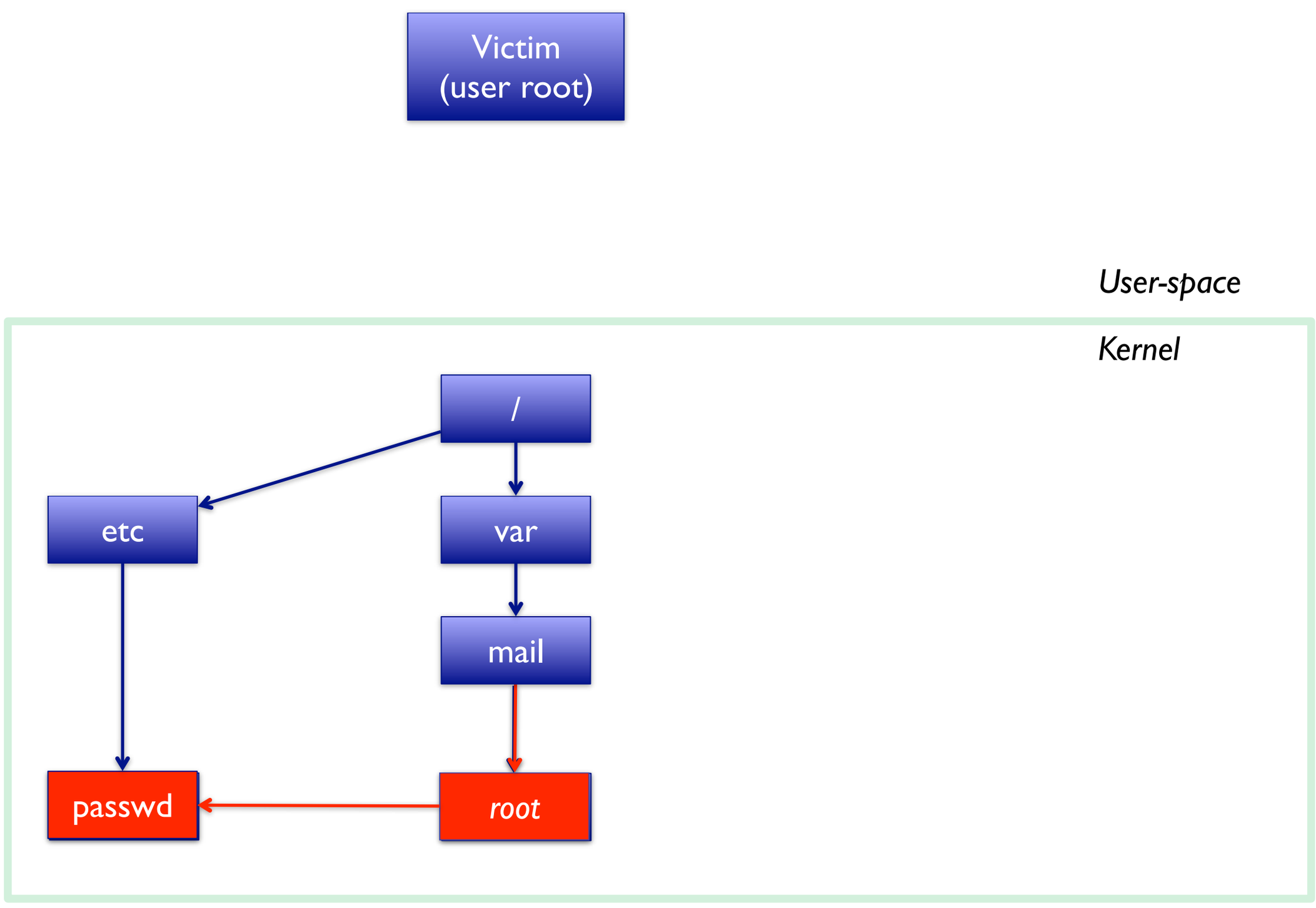


Launch Phase

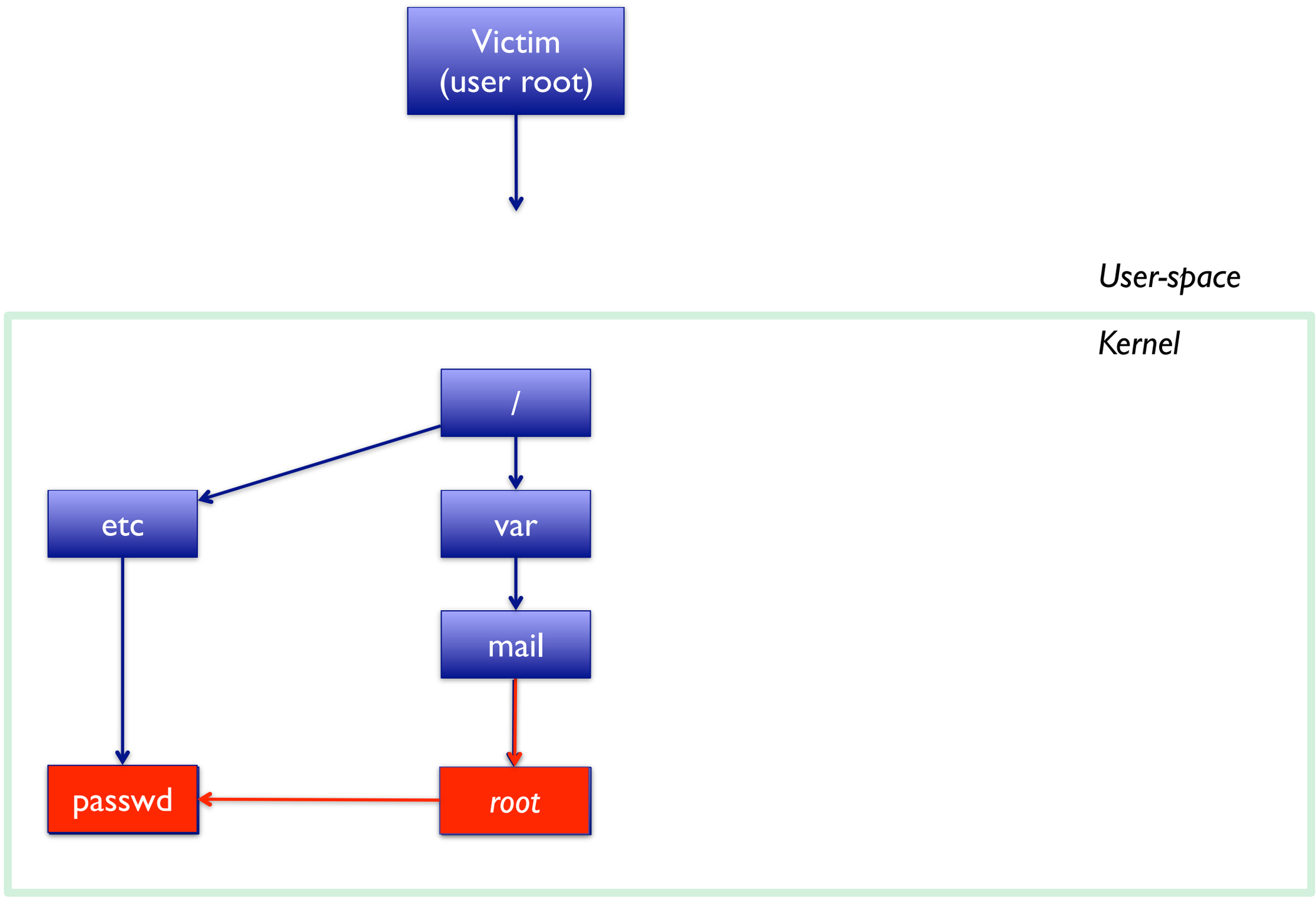
1. Find bindings
2. Find adversary access
3. Launch attack (modify namespace)
4. Continue system call



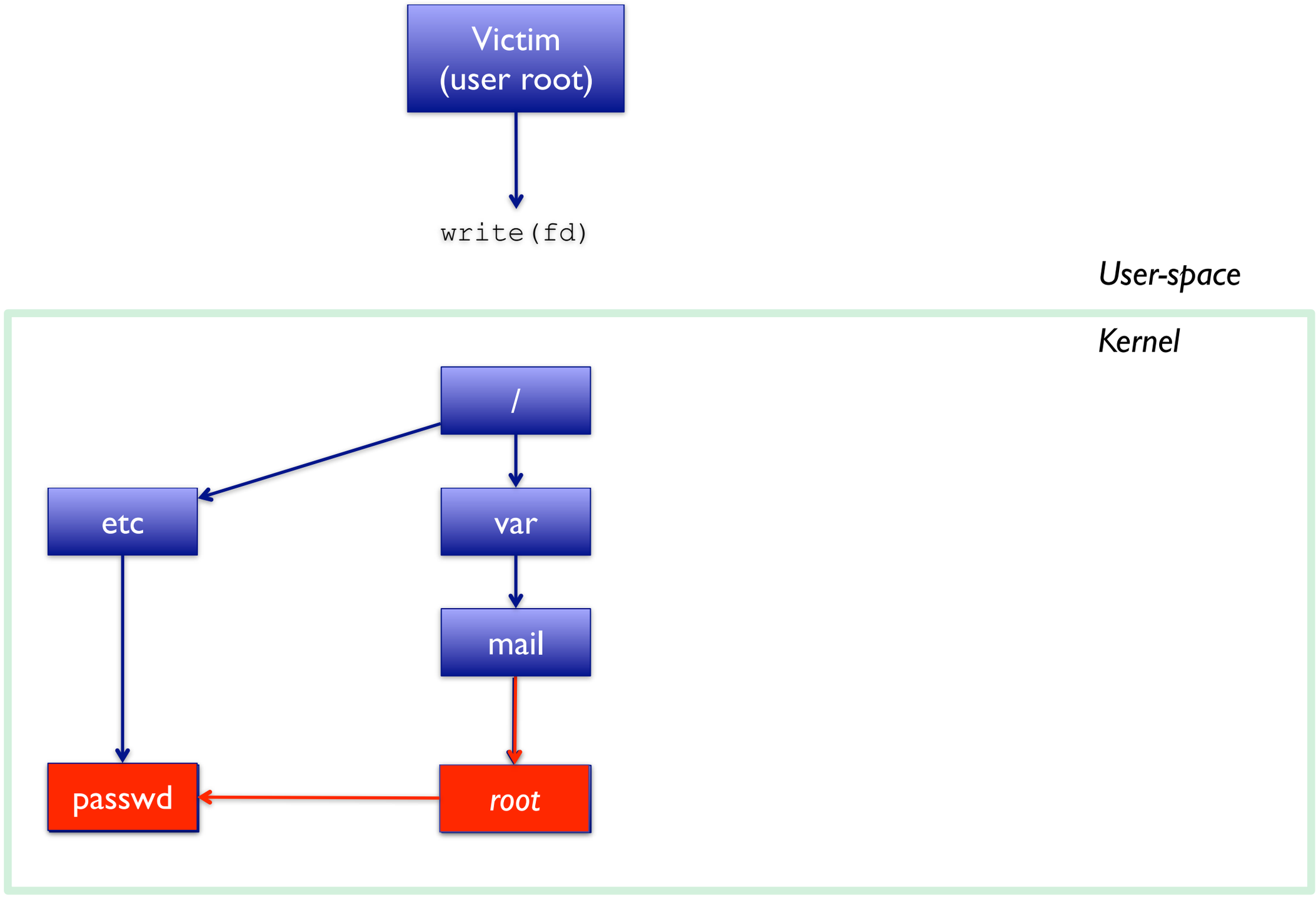
Detect Phase



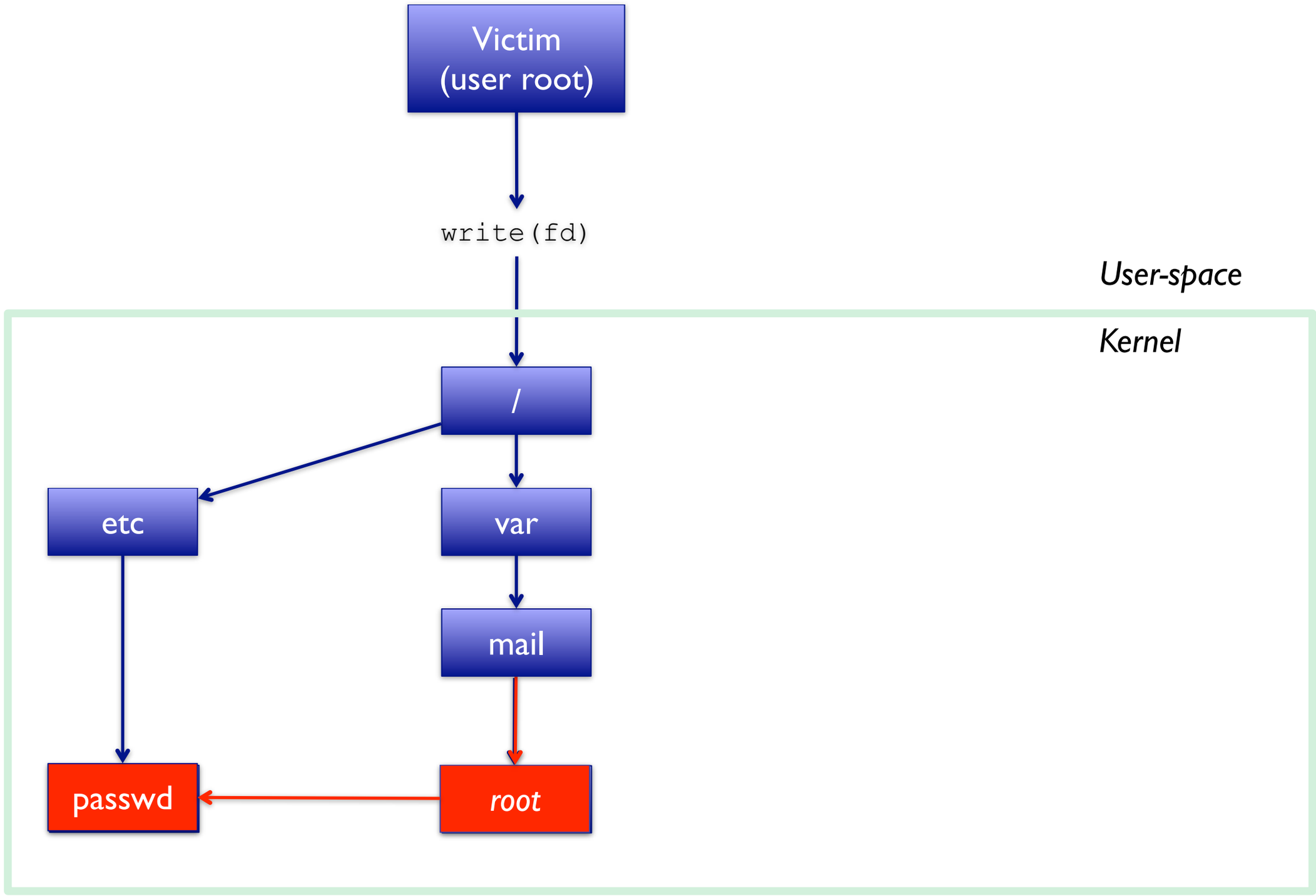
Detect Phase



Detect Phase

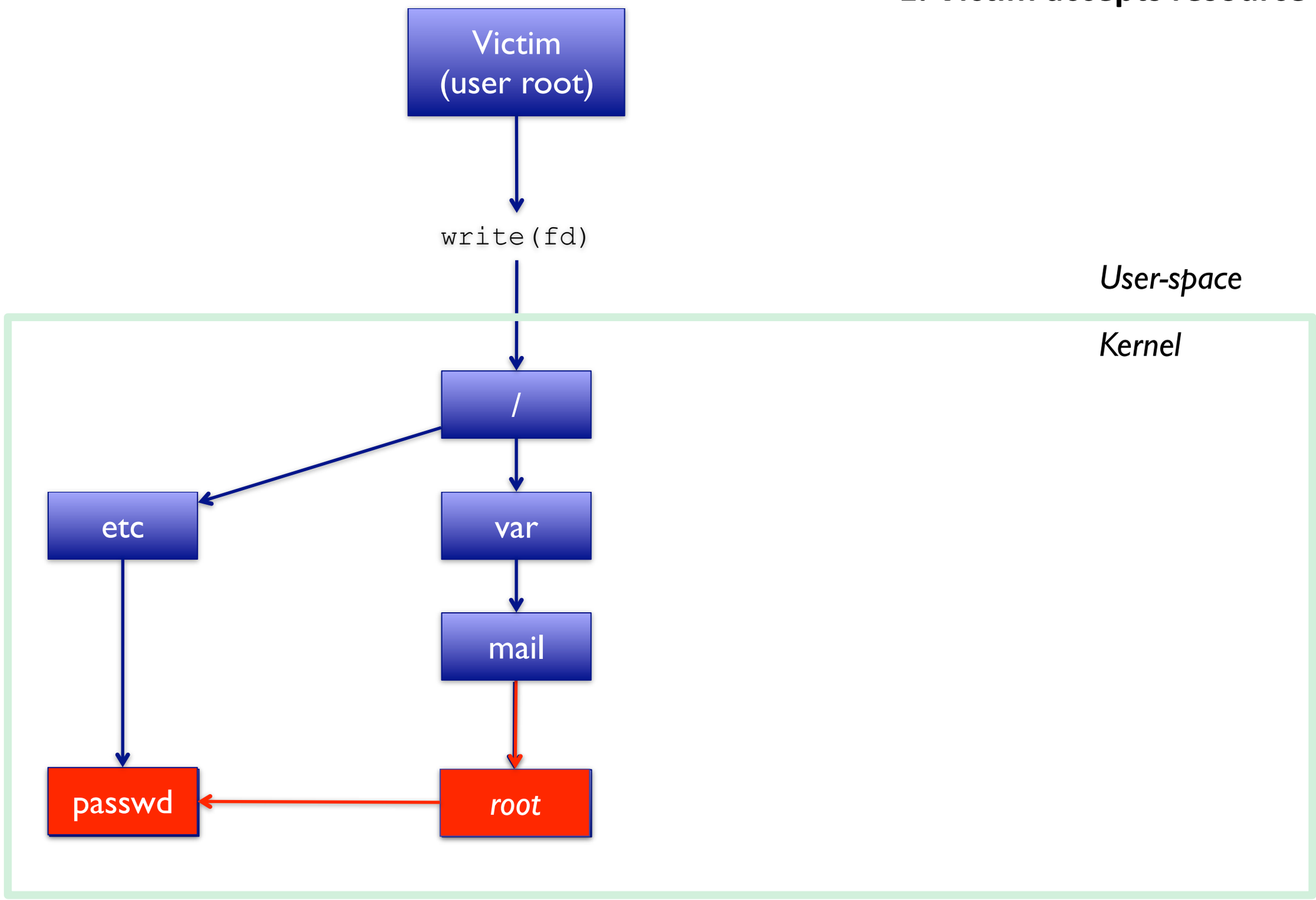


Detect Phase



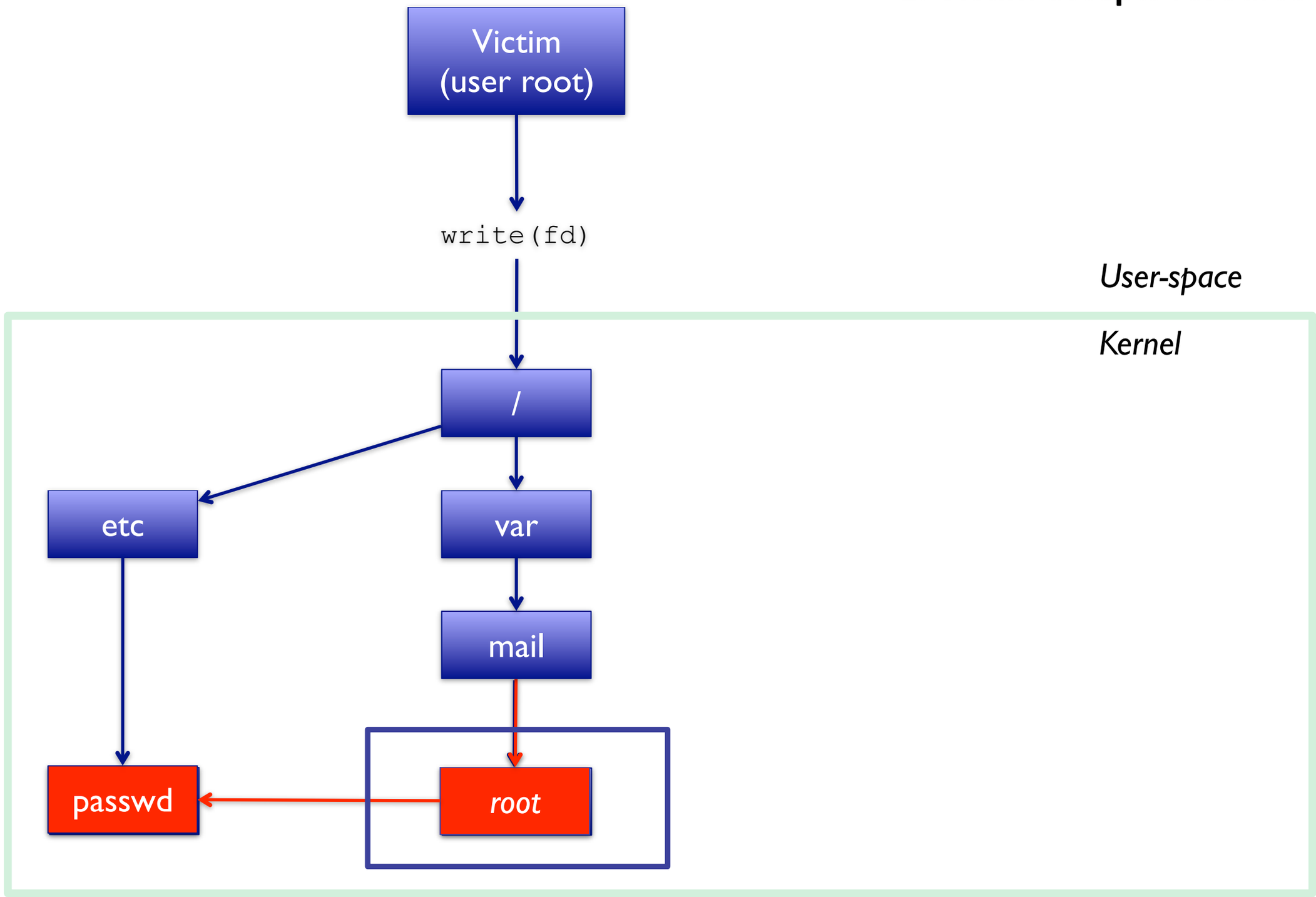
Detect Phase

1. Victim accepts resource



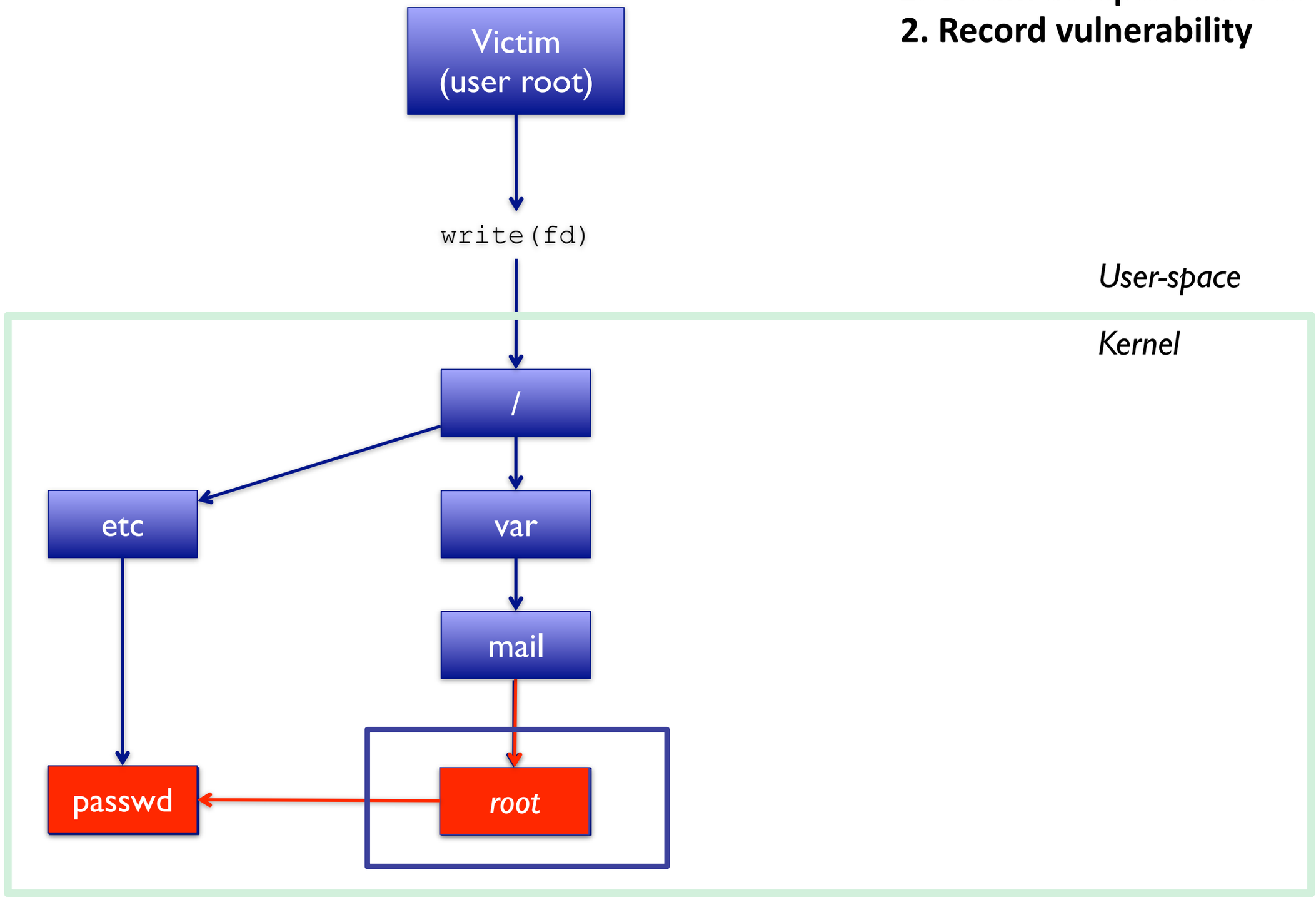
Detect Phase

1. Victim accepts resource



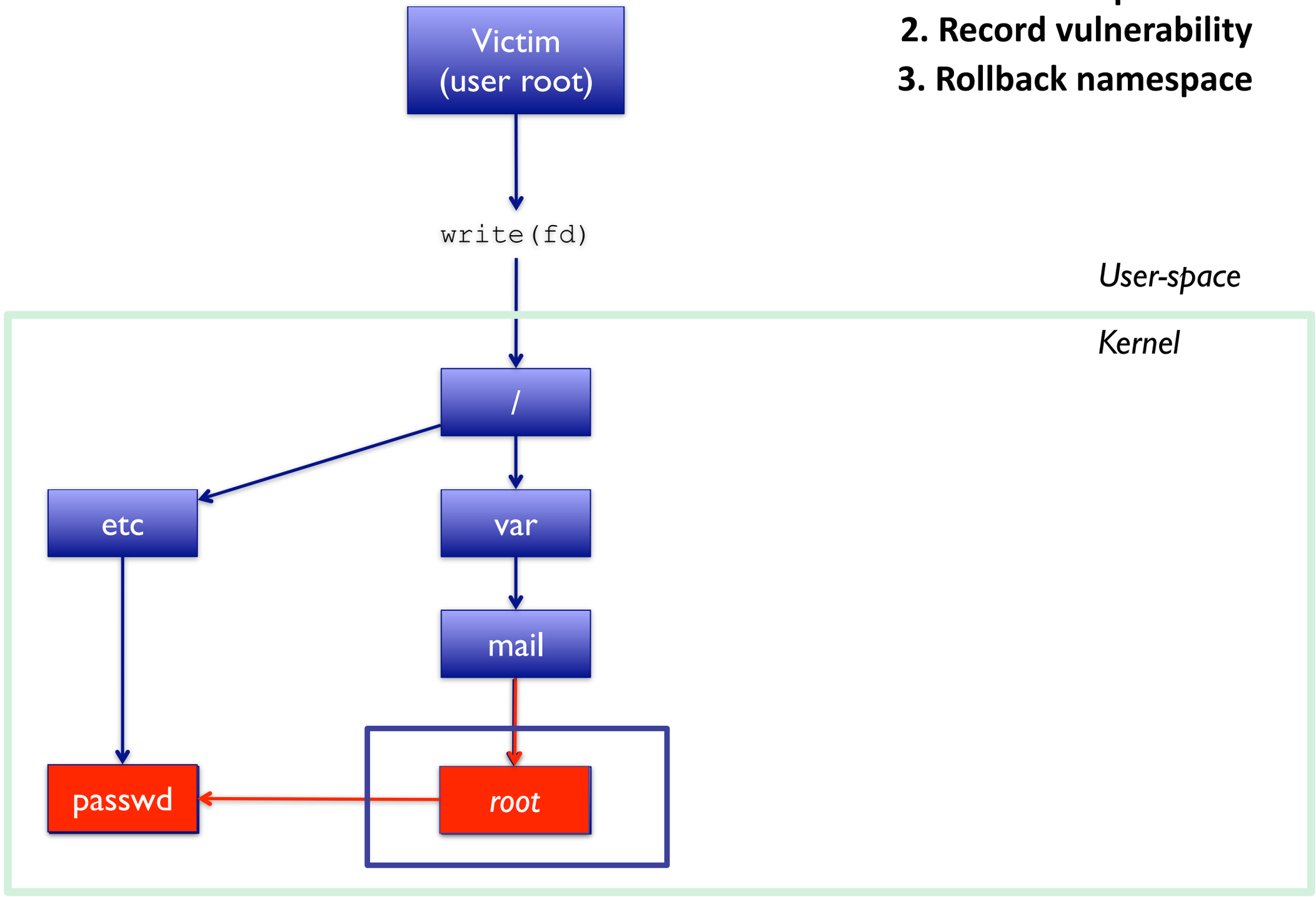
Detect Phase

1. Victim accepts resource
2. Record vulnerability



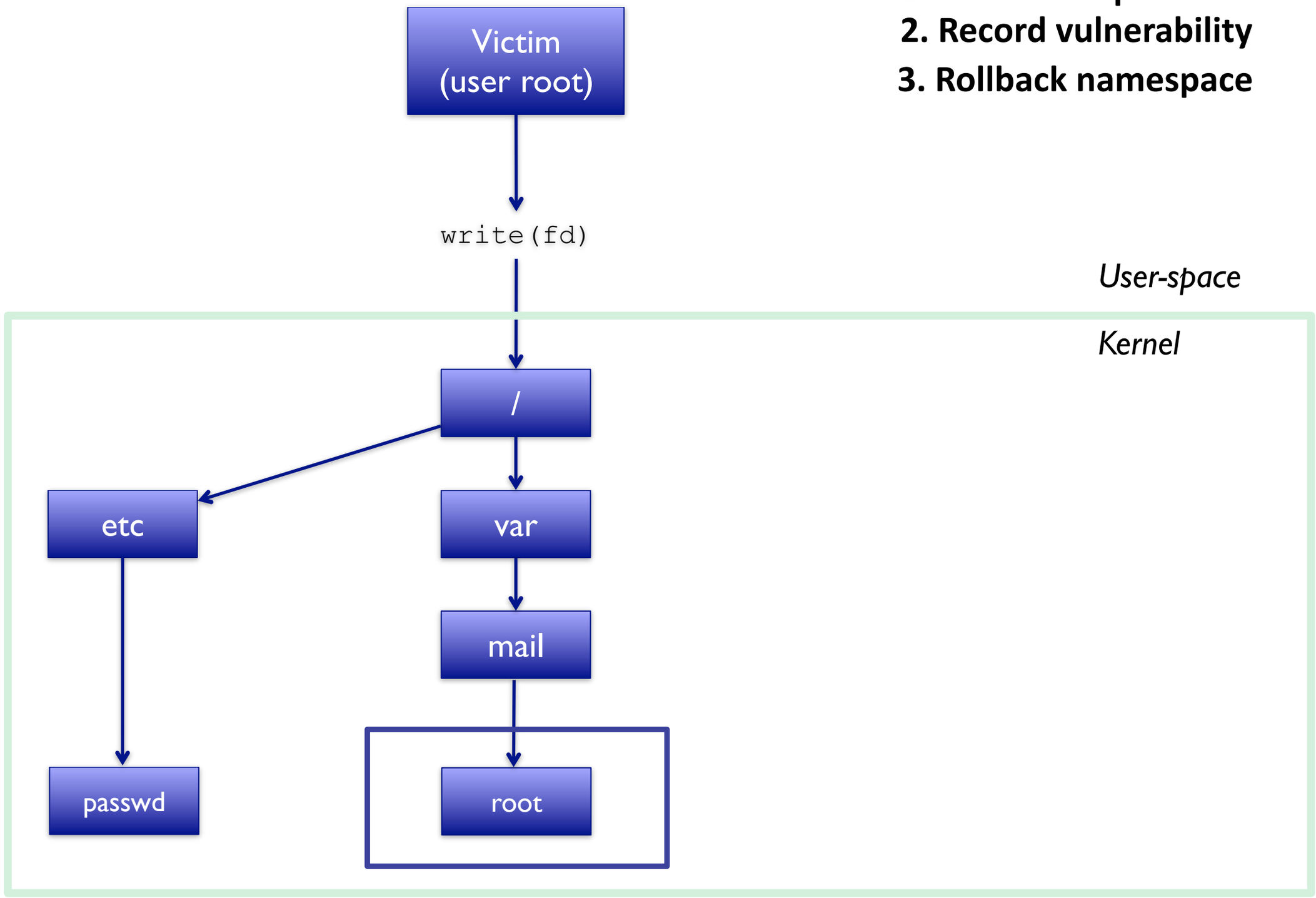
Detect Phase

1. Victim accepts resource
2. Record vulnerability
3. Rollback namespace



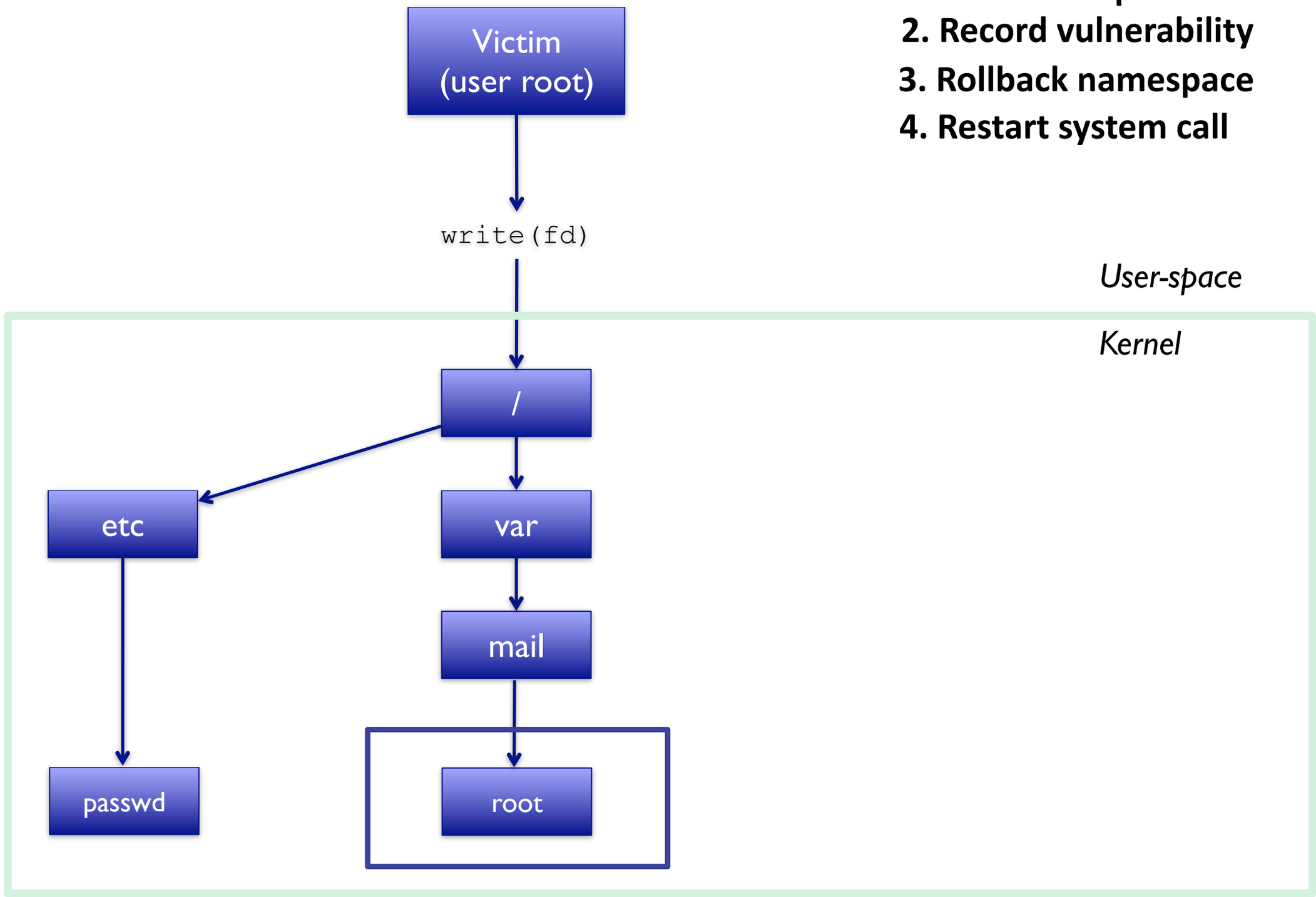
Detect Phase

1. Victim accepts resource
2. Record vulnerability
3. Rollback namespace

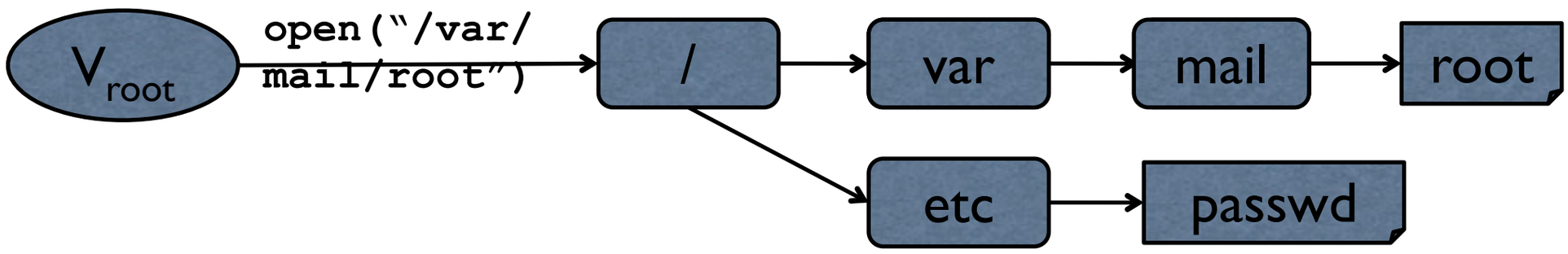


Detect Phase

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

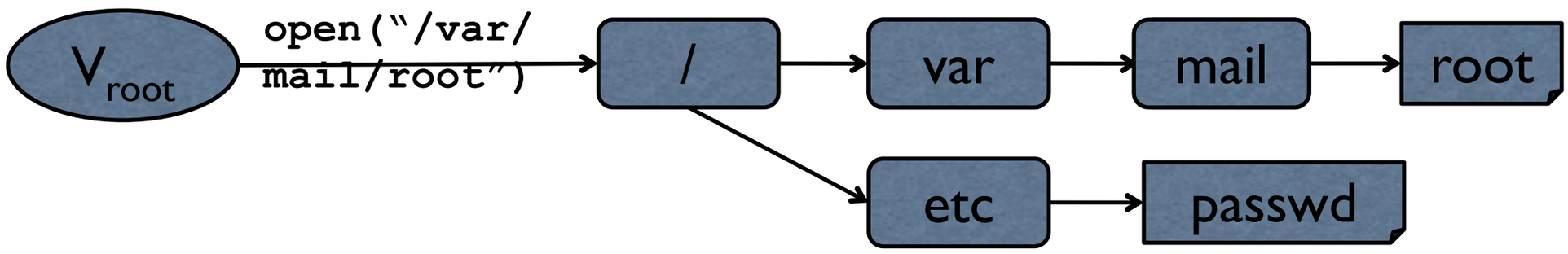


Find Accessible Bindings



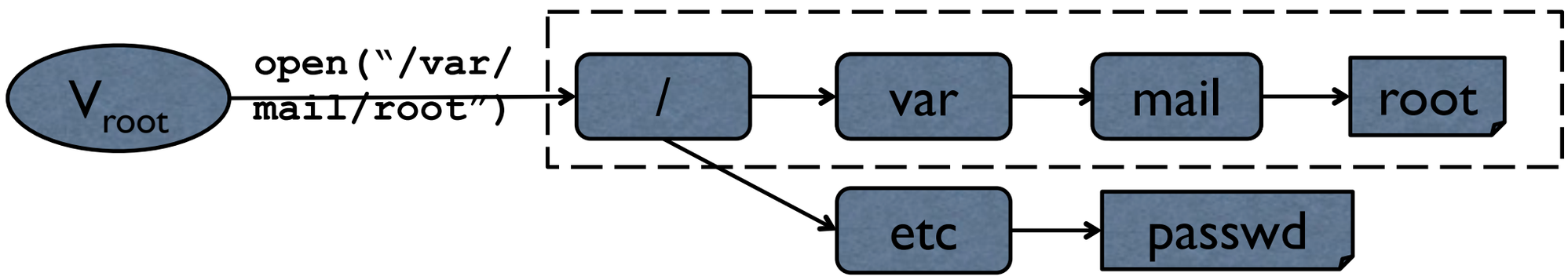
Find Accessible Bindings

- Find bindings - Shadow resolution
  - Extract name resolution code inside kernel and obtain bindings before system call starts



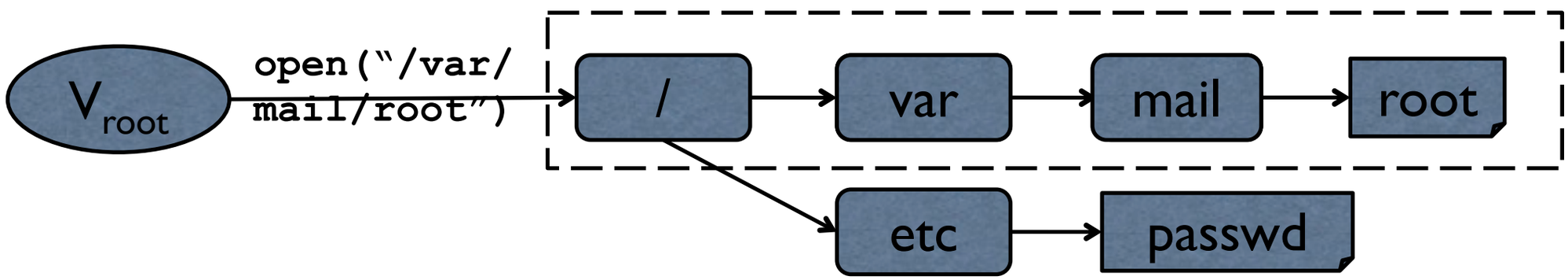
Find Accessible Bindings

- Find bindings - Shadow resolution
  - Extract name resolution code inside kernel and obtain bindings before system call starts



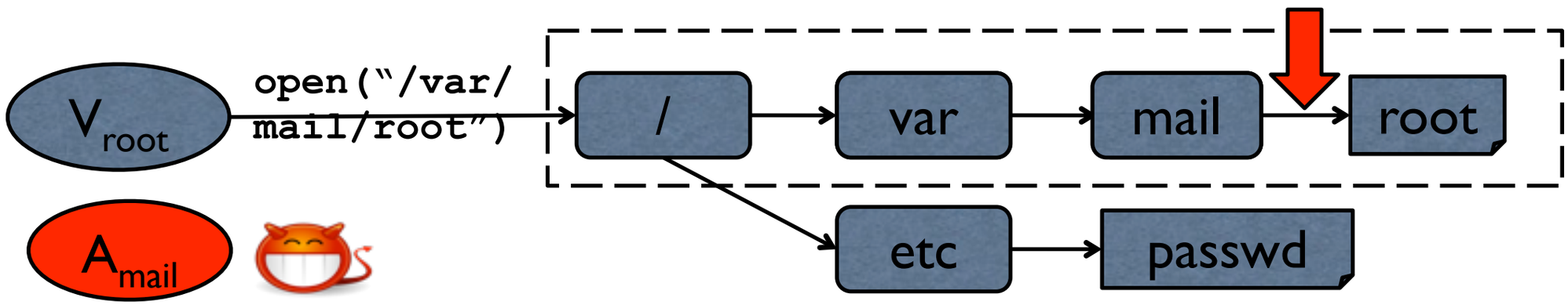
Find Accessible Bindings

- Find bindings - Shadow resolution
  - Extract name resolution code inside kernel and obtain bindings before system call starts
- Find adversary-accessible bindings - Adversary model
  - Use access control policy
    - DAC model: Any other user apart from *root*
    - MAC model: (SELinux)

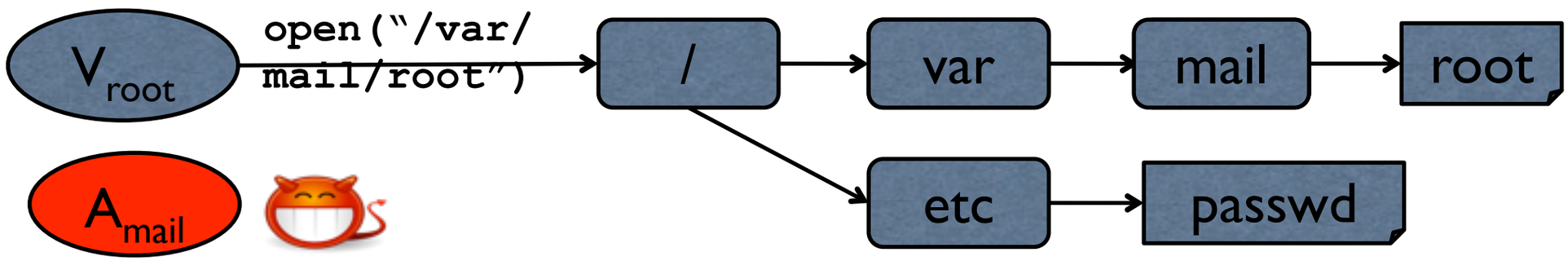


Find Accessible Bindings

- Find bindings - Shadow resolution
  - Extract name resolution code inside kernel and obtain bindings before system call starts
- Find adversary-accessible bindings - Adversary model
  - Use access control policy
    - DAC model: Any other user apart from *root*
    - MAC model: (SELinux)

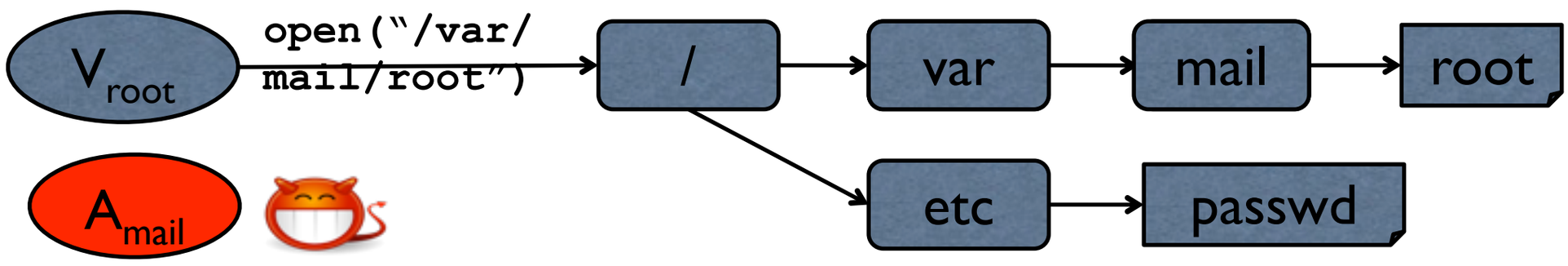


Launching an Attack



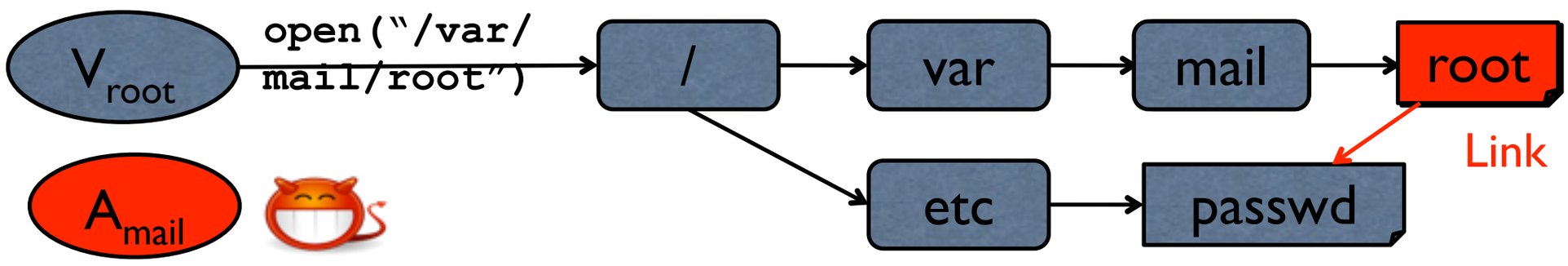
Launching an Attack

- Modify namespace to generate attack test case
  - Existing data should be backed up
  - Unix domain sockets, … cannot be recovered if deleted
  - Attack should be visible only to victims of the adversary
    - Not to all processes



Launching an Attack

- Modify namespace to generate attack test case
  - Existing data should be backed up
  - Unix domain sockets, … cannot be recovered if deleted
  - Attack should be visible only to victims of the adversary
    - Not to all processes



Launching an Attack

Launching an Attack

- Solution - Union filesystems
  - Combines “lower” read-only and “upper” read-write fs

Launching an Attack

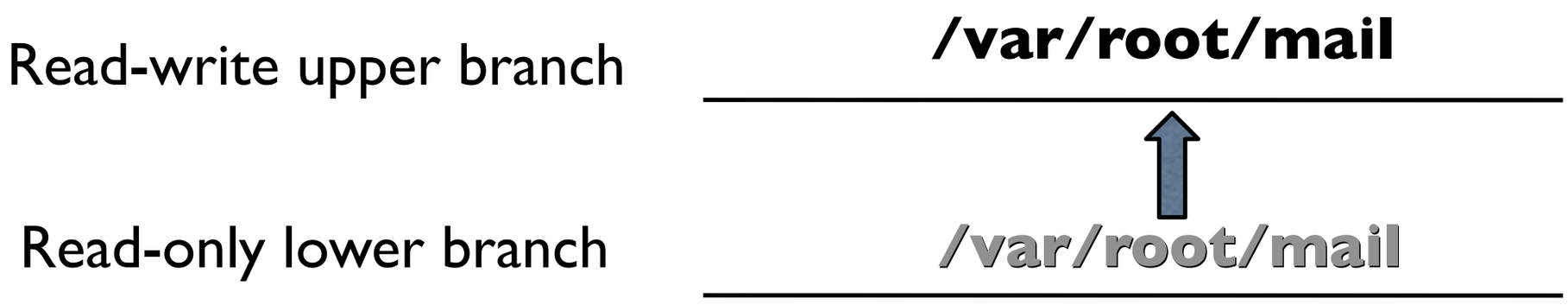
- Solution - Union filesystems
  - Combines “lower” read-only and “upper” read-write fs

Read-write upper branch

Read-only lower branch **/var/root/mail**

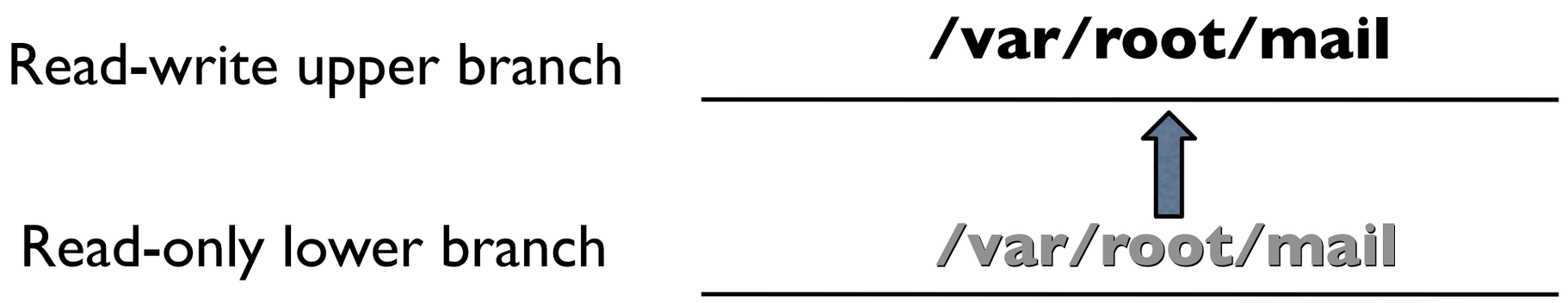
Launching an Attack

- Solution - Union filesystems
  - Combines “lower” read-only and “upper” read-write fs



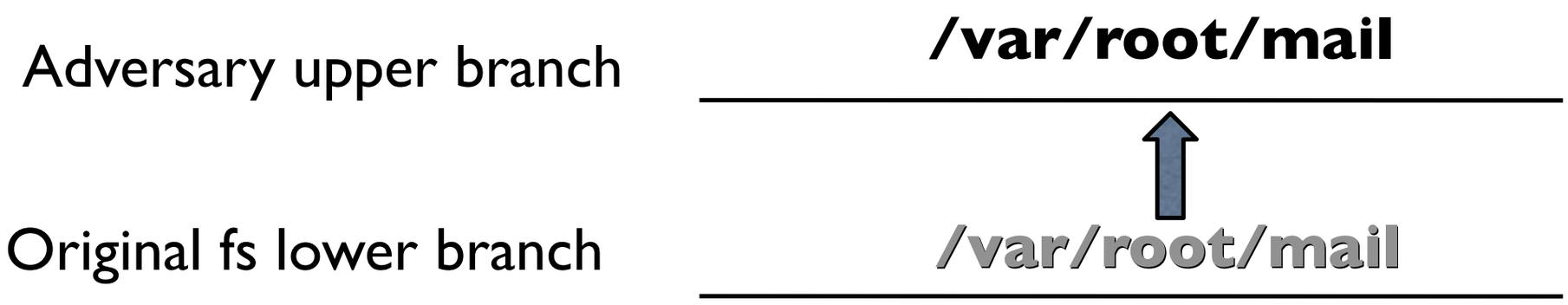
Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



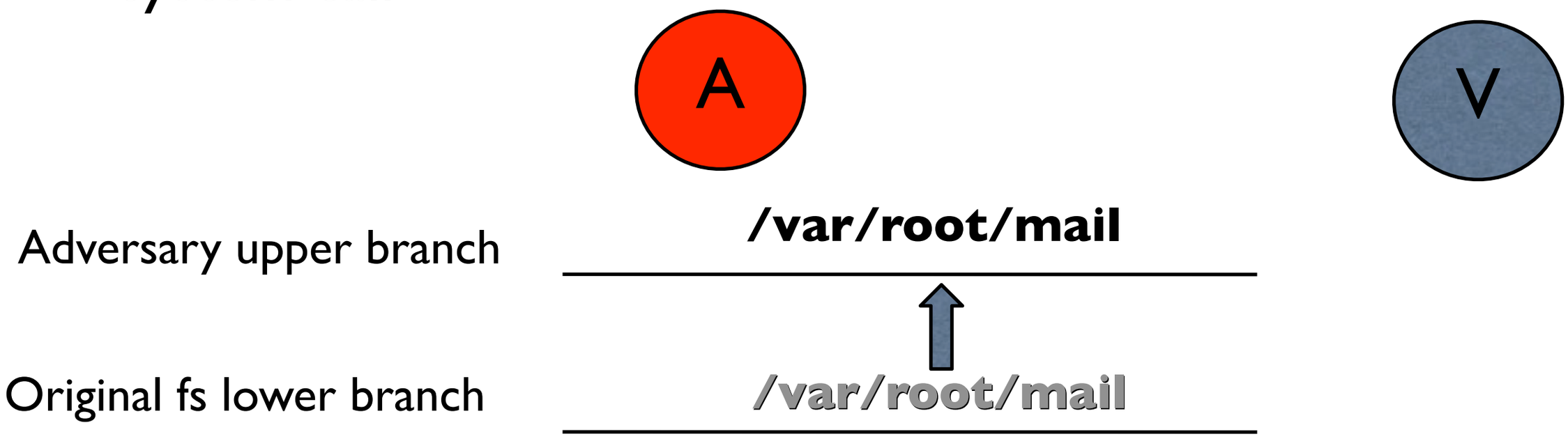
Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



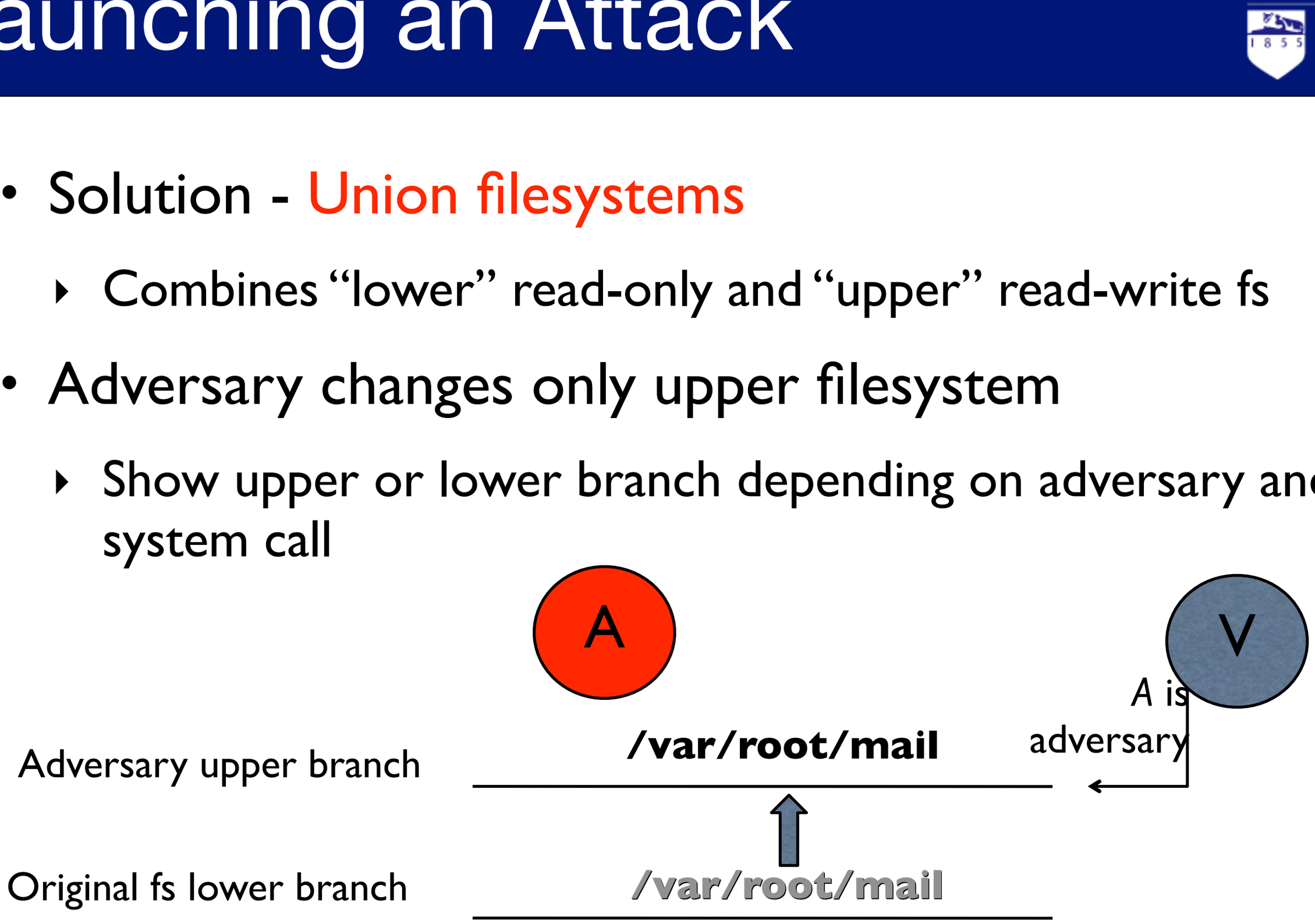
Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



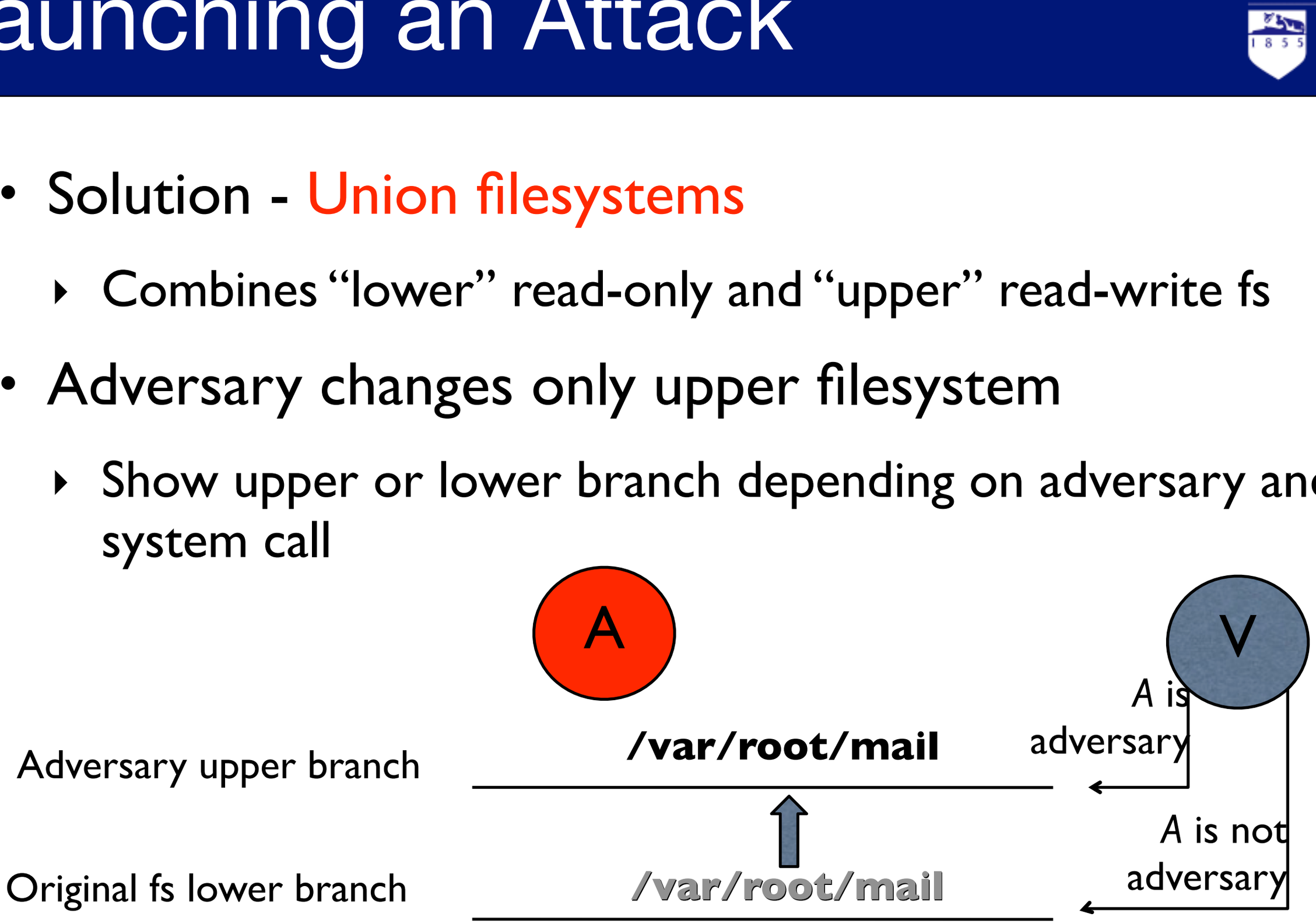
Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



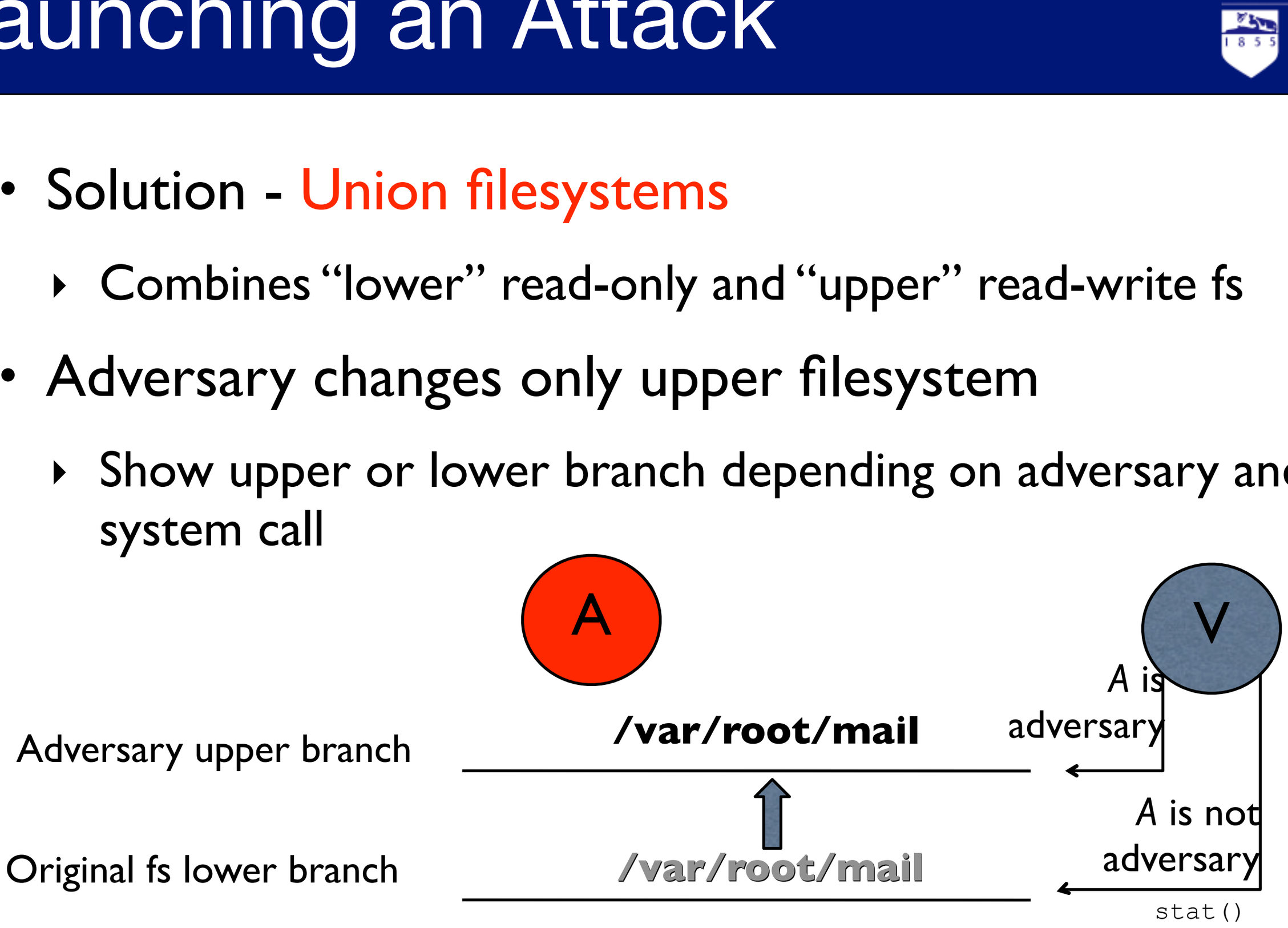
Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



Launching an Attack

- Solution - Union filesystems
  - Combines "lower" read-only and "upper" read-write fs
- Adversary changes only upper filesystem
  - Show upper or lower branch depending on adversary and system call



Manage Attacks

Manage Attacks

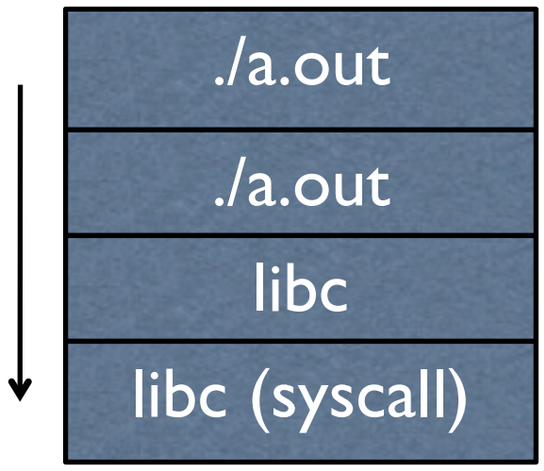
- Only run an attack test case once
  - How to identify current system call originates from code that has already been tested?

Manage Attacks

- Only run an attack test case once
  - How to identify current system call originates from code that has already been tested?
- Program entry points as unique identifiers
  - Program instruction calling library that performs system call
    - Obtained by user-stack backtrace within kernel
    - Extensions for interpreters (11-59 LOC per interpreter)

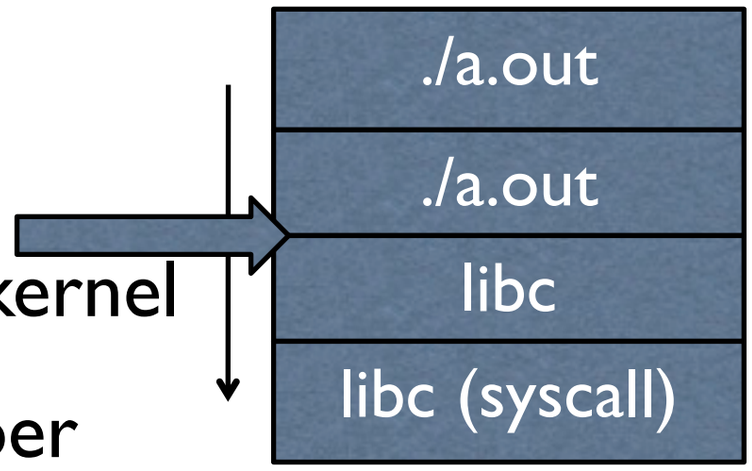
Manage Attacks

- Only run an attack test case once
  - How to identify current system call originates from code that has already been tested?
- Program entry points as unique identifiers
  - Program instruction calling library that performs system call
    - Obtained by user-stack backtrace within kernel
    - Extensions for interpreters (11-59 LOC per interpreter)



Manage Attacks

- Only run an attack test case once
  - How to identify current system call originates from code that has already been tested?
- Program entry points as unique identifiers
  - Program instruction calling library that performs system call
    - Obtained by user-stack backtrace within kernel
    - Extensions for interpreters (11-59 LOC per interpreter)



Detect Vulnerability

- How do we know victim process has accepted or rejected the resource?
- Accept resource
  - Program uses "accept" system calls on test case ("upper layer") resource
- Reject resource
  - Program retries system call at same entry point or exits without accepting

Detect Vulnerability

- Acceptance for attacks we consider
  - Not all system calls on tainted resources signify vulnerabilities.

Attack	Accept
Symbolic link	write-like, `read`, `readlink`
Hard link	write-like, `read`
File squat	write-like, `read`
Unix-domain socket squat	`connect`
System V IPC squat	`msgsnd`, `semop`, `shmat`

Recovery and Rollback

- Namespace rollback
  - Wipe adversarial resource from upper branch
    - Further name resolutions get resource from lower branch
  - Since we operate at VFS layer, we can redirect open file descriptors to lower layer
- Process recovery
  - Some processes retry – we don't do anything
  - For those that exit – we restart process
    - Linux has some rollback facilities we will examine, if necessary

Implementation

- STING as a kernel patch for Linux 2.6 and 3
  - ~2700 LOC
- User-space support
  - Init ramdisk scripts to mount stacked filesystem, load attack history log, load adversary model
- We have a package for Ubuntu 12.04
  - `apt-get install sting`
- Once installed, STING automatically starts testing the whole system
  - No special runtime environment or setup needed

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**

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

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**

Special users to root

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**

Known but unfixed!

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**

Vulnerabilities by Entrypoint

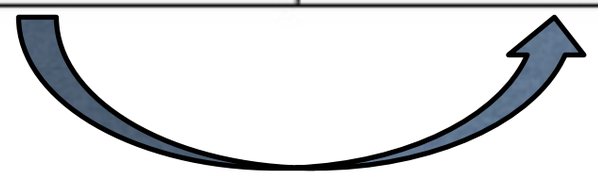
- Under DAC adversary model
  - Only 4% (Fedora) and 5.7% (Ubuntu) of total name resolution entrypoints were accessible to adversaries
  - Only 0.3% (Fedora) and 0.9% (Ubuntu) of total name resolutions were vulnerable

Adversary model	Total Resolutions	Adversary Access	Vulnerable
DAC - Ubuntu	2345	134 (5.7%)	21 (0.9%)
DAC - Fedora	1654	66 (4%)	5 (0.3%)

Vulnerabilities by Entrypoint

- Under DAC adversary model
  - Only 4% (Fedora) and 5.7% (Ubuntu) of total name resolution entrypoints were accessible to adversaries
  - Only 0.3% (Fedora) and 0.9% (Ubuntu) of total name resolutions were vulnerable

Adversary model	Total Resolutions	Adversary Access	Vulnerable
DAC - Ubuntu	2345	134 (5.7%)	21 (0.9%)
DAC - Fedora	1654	66 (4%)	5 (0.3%)



Static Analysis
False +

Vulnerabilities by Entrypoint

- Under DAC adversary model
  - Only 4% (Fedora) and 5.7% (Ubuntu) of total name resolution entrypoints were accessible to adversaries
  - Only 0.3% (Fedora) and 0.9% (Ubuntu) of total name resolutions were vulnerable

Adversary model	Total Resolutions	Adversary Access	Vulnerable
DAC - Ubuntu	2345	134 (5.7%)	21 (0.9%)
DAC - Fedora	1654	66 (4%)	5 (0.3%)

Static Analysis False +

Normal Runtime False +

STING detects TOCTTOU races

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

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
}
```

STING detects TOCTTOU races

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

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
}
```

STING detects TOCTTOU races

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

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
```

STING detects TOCTTOU races

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

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
```

STING creates scenarios

- That do not occur in normal runtime

STING creates scenarios

- That do not occur in normal runtime

Adversary

```
hayawardh@mantra:~$ touch my_file
hayawardh@mantra:~$ ln -s my_file none
hayawardh@mantra:~$ ln -s my_file fusectl
```

STING creates scenarios

- That do not occur in normal runtime

Adversary

```
hayawardh@mantra:~$ touch my_file
hayawardh@mantra:~$ ln -s my_file none
hayawardh@mantra:~$ ln -s my_file fusectl
```

Victim

```
root@mantra:/home/hayawardh# mountall
```

STING creates scenarios

- That do not occur in normal runtime

Adversary

```
hayawardh@mantra:~$ touch my_file
hayawardh@mantra:~$ ln -s my_file none
hayawardh@mantra:~$ ln -s my_file fusectl
```

Victim

```
root@mantra:/home/hayawardh# mountall
```

```
root@mantra:/home/hayawardh# cat /etc/mtab
/dev/sda1 / ext3 rw,relatime,errors=remount-ro,user_xattr 0 0
proc /proc proc rw 0 0
binfmt_misc /proc/sys/fs/binfmt_misc binfmt_misc rw,noexec,nosuid,nodev 0 0
sysfs /sys sysfs rw,noexec,nosuid,nodev 0 0
/home/hayawardh/my_file /sys/fs/fuse/connections fusectl rw 0 0
none /sys/kernel/debug debugfs rw 0 0
none /sys/kernel/security securityfs rw 0 0
udev /dev devtmpfs rw,mode=0755 0 0
devpts /dev/pts devpts rw,noexec,nosuid,gid=5,mode=0620 0 0
tmpfs /run tmpfs rw,noexec,nosuid,size=10%,mode=0755 0 0
none /run/lock tmpfs rw,noexec,nosuid,nodev,size=5242880 0 0
none /run/shm tmpfs rw,nosuid,nodev 0 0
/dev/sda6 /mnt/sda6 ext3 rw 0 0
/dev/sda2 /mnt/sda2 ext3 rw 0 0
/dev/sda5 /usr/src ext3 rw 0 0
```

Detects easily overlooked

```
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, ...)
```

Detects easily overlooked

- Manual checks can easily overlook vulnerabilities

```
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, ...)
```

Detects easily overlooked

- Manual checks can easily overlook vulnerabilities

```
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

Detects easily overlooked

- Manual checks can easily overlook vulnerabilities

```
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

Symbolic link

Detects easily overlooked

- Manual checks can easily overlook vulnerabilities

```
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

Symbolic link

Hard link, race conditions

Detects easily overlooked

- Manual checks can easily overlook vulnerabilities
- But, misses already existing file squat!

```
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

Symbolic link

Hard link, race conditions

Shows OS distributor challenge

- STING also found vulnerabilities where the problem seemed to be the system's access control policy
  - When contacted, a developer refused to fix bug claiming fault in system's access control policy
  - We found other vulnerabilities that seemed better fixed by the access control than code
    - E.g., postgres init script

Performance

- STING causes around 8% overhead on macrobenchmarks
  - Noticeable overhead, but we were able to use system
  - We are looking for further avenues to improve performance

Benchmark	Base	STING	Overhead
Apache 2.2.20 compile	151.65s	163.79s	8%
Apachebench: Throughput	231.77Kbps	221.89Kbps	4.33%
Latency	1.943ms	2.088ms	7.46%

Conclusions

- Name resolution is a fundamental process
  - But, has long been vulnerable to various attacks
- It is both difficult to prevent name resolution attacks and find program vulnerabilities
  - We use runtime grey-box testing
- STING is a system-wide, online tool that finds name resolution vulnerabilities in programs
  - By producing malicious test case when a program's adversary can modify bindings used in resolution
- Found 21 previously-unknown vulnerabilities
  - Highlights various issues

Availability

- STING webpage : http://siis.cse.psu.edu/sting
  - Please contact hvijay@cse.psu.edu for access to repository
- We envision STING be used on distributions during testing (e.g., alpha, beta) or by administrators on test systems before deployment to fix vulnerabilities before adversaries exploit them
- We have a package for Ubuntu 12.04
  - `apt-get install sting`

Thank You !

- Questions?
- E-mail for contact : hvijay@cse.psu.edu



Results – Retry vs Restart

- Around 32% of programs retried, whereas the rest had to be restarted
  - Programs that retry integrate well with STING
  - Restarted programs may lose state
  - We are investigating integrating process checkpointing for graceful recovery of process state

Guarantees

- If a process accepts an adversarial resource
  - There *is* a vulnerable name resolution
  - Reads may not be exploitable
    - Depends on program internals