A Case for Live, Adversary-Aware Program Testing

Trent Jaeger
Systems and Internet Infrastructure Security Lab
Penn State University

Testing Goal

PENNSTATE 1855

- Find potential vulnerabilities…
  - Key question: What is a vulnerability?

Vulnerabilities

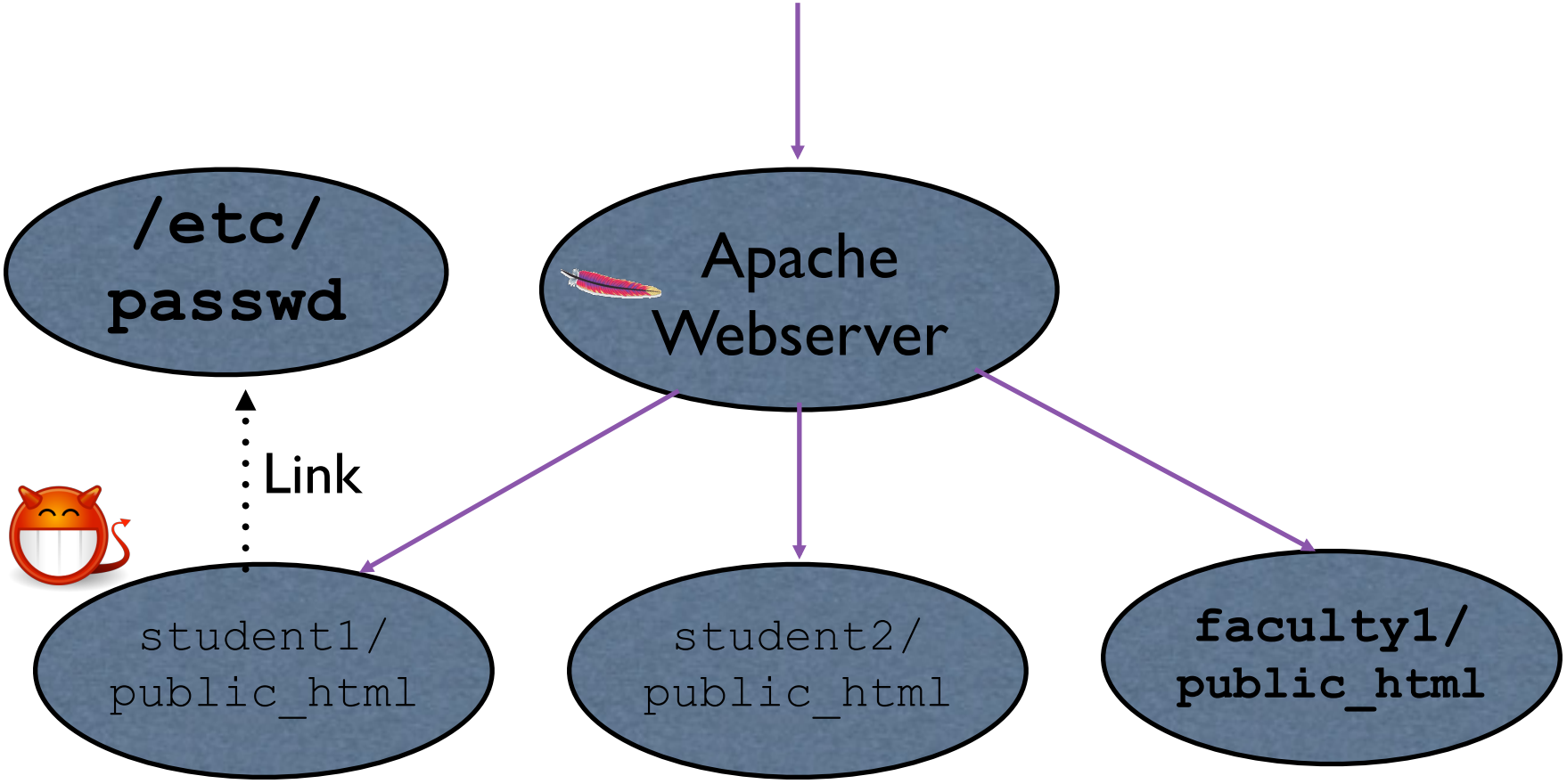
PENNSTATE 1855

- A program vulnerability consists of three elements:
  - A flaw
  - Accessible to an adversary
  - Adversary has the capability to exploit the flaw
- Claim: Testing techniques focus on subset of these elements
  - But all conditions must be present for a true vulnerability

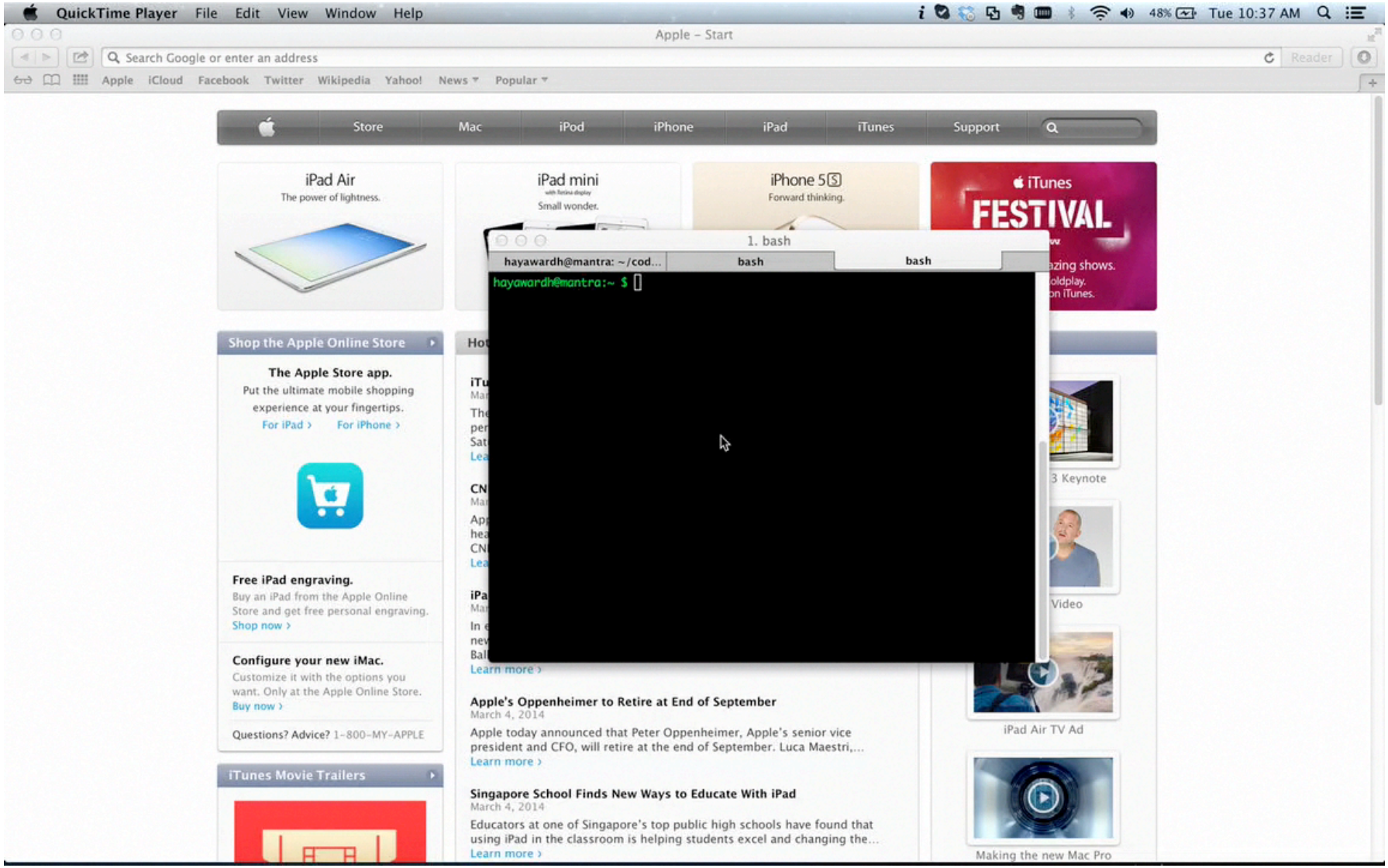
A Webserver's Story …

- Consider a university department webserver …

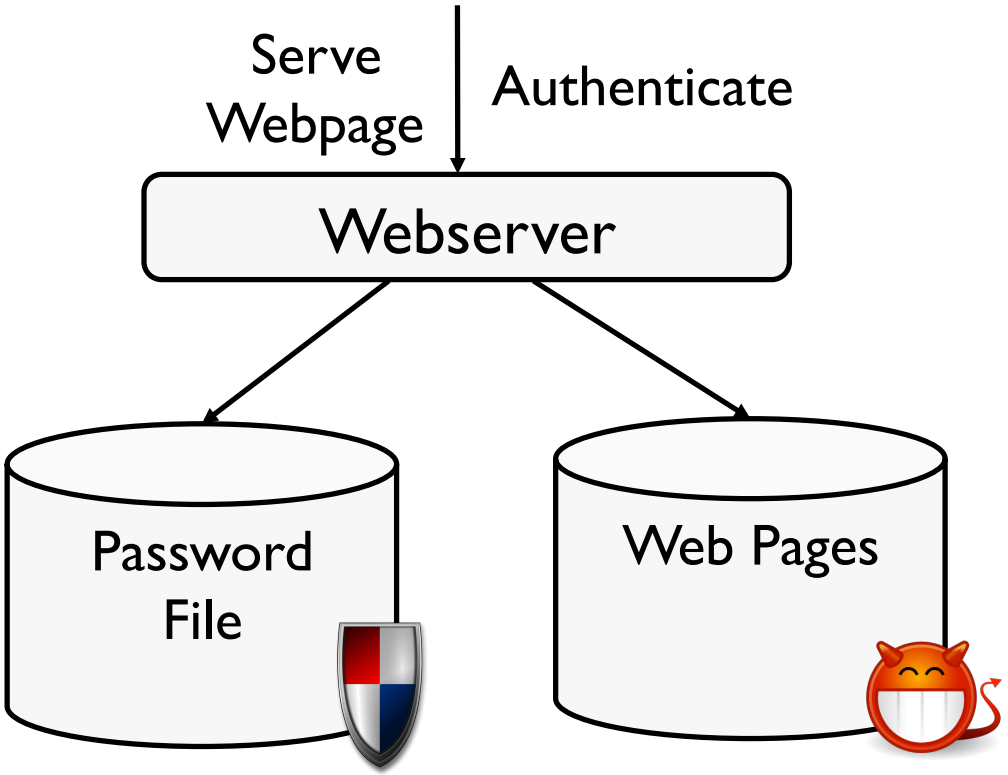
GET /~student1/index.html HTTP/1.1

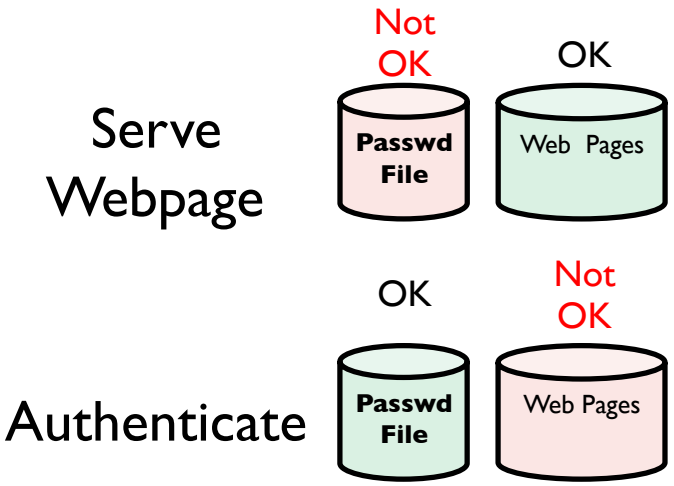


Attack Video



What Just Happened?





- Program received an *unexpected* resource
  - when expecting (unexpected attack surface)
  - when expecting (confused deputy)

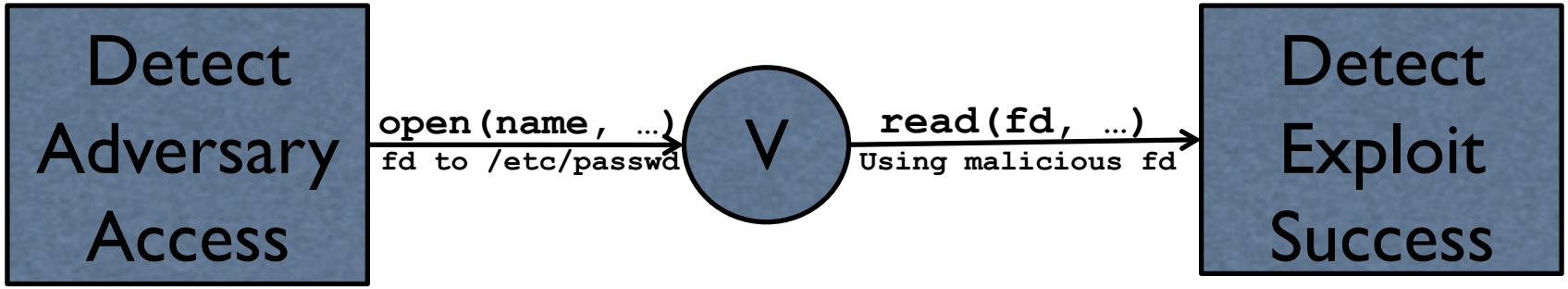
Vulnerabilities

PENNSTATE 1855

- Flaw finding
  - Opening a file is not necessarily a flaw
  - Checks for correct name filtering and/or binding is expensive, so not done unless a reason
- Restricting exploits
  - Mandatory access control is insufficient
  - Victim needs to communicate with potential adversaries and access sensitive resources
- Adversary access
  - Not tracked systematically

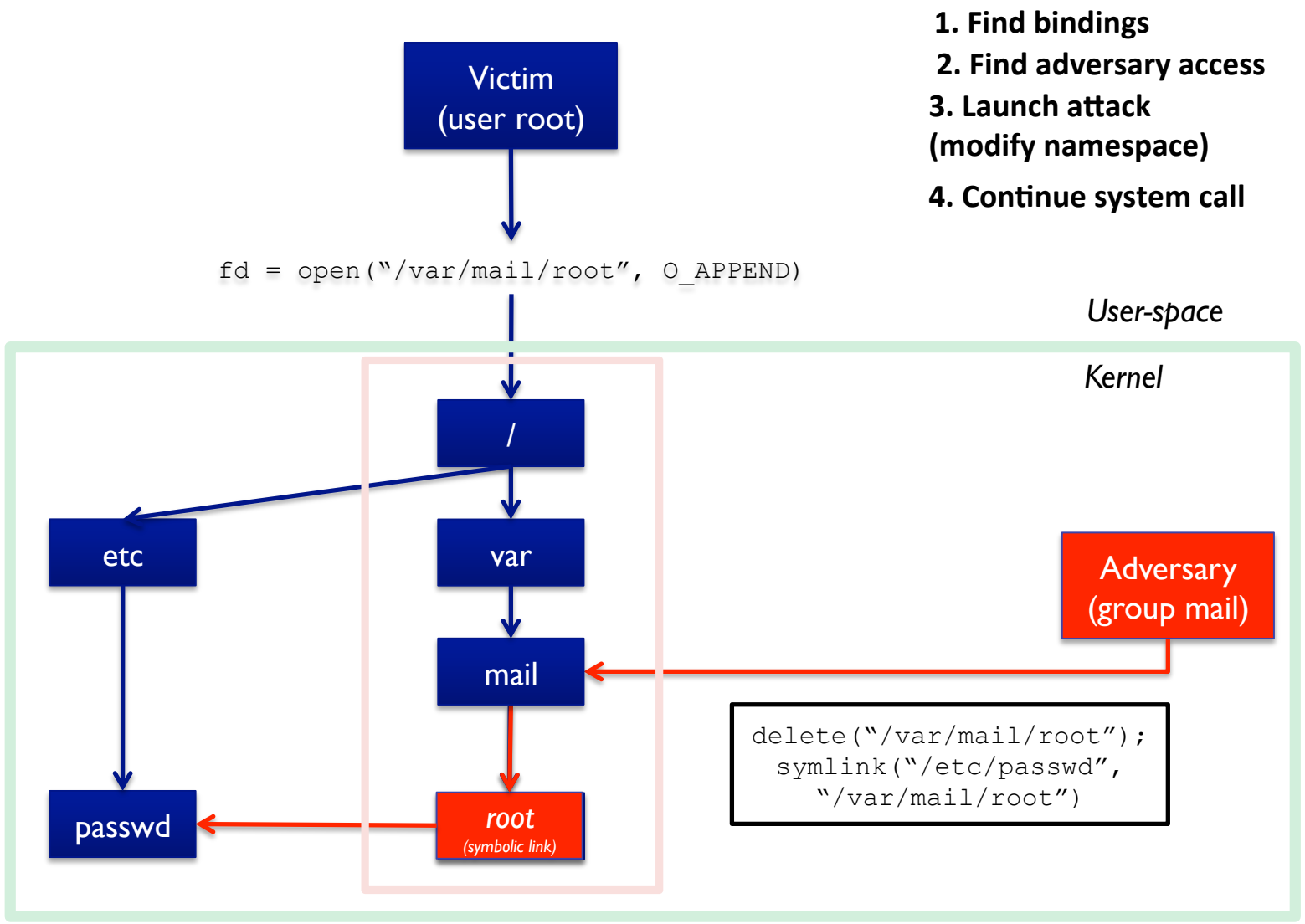
STING [USENIX 2012]

- 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
- Generate an adversarial test case and see how program reacts – live (unoptimized) for <8% overhead

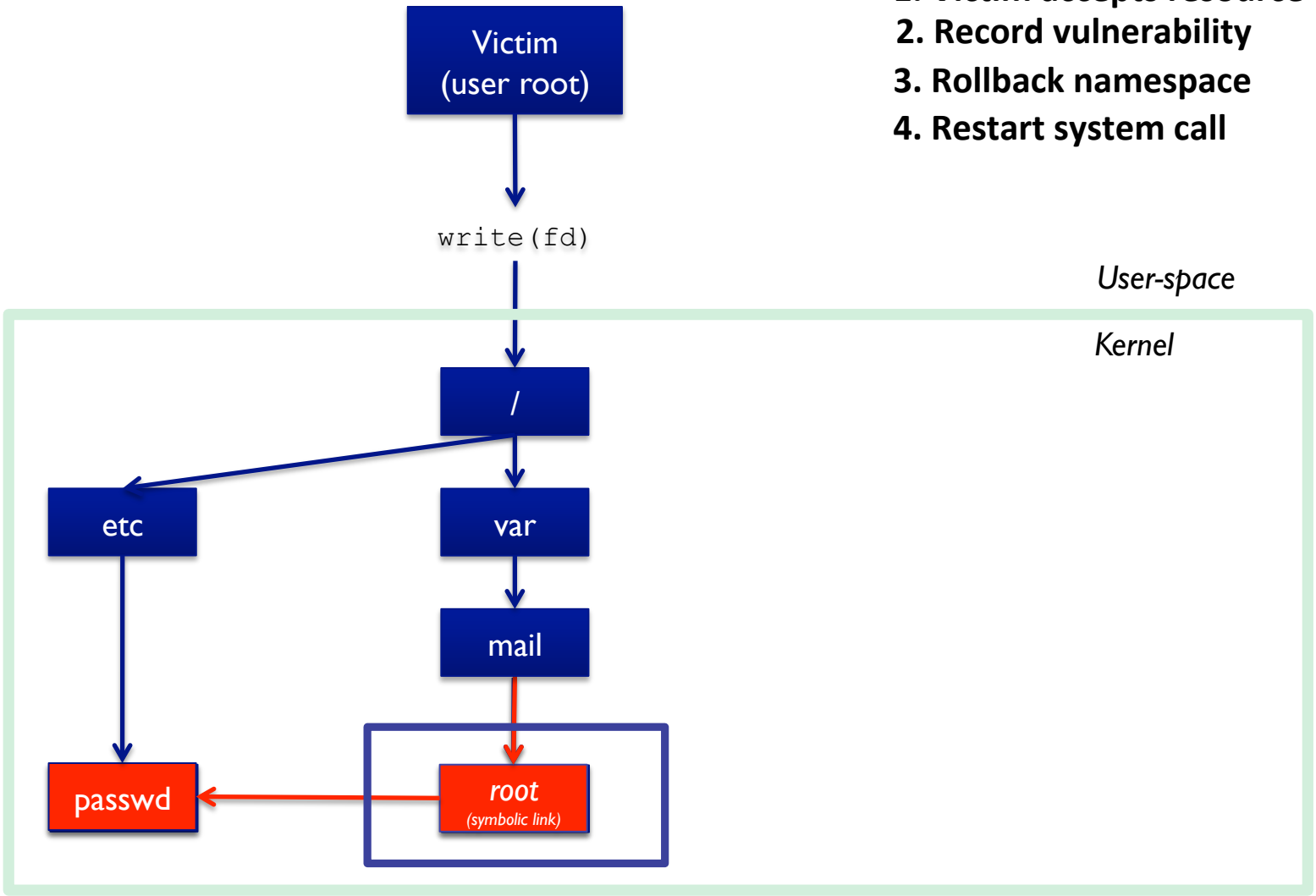


Vulnerable!

Launch Phase



Detect Phase



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

Results - Vulnerabilities

PENNSTATE 1855

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**

STING available at: https://github.com/TJAndHisStudents/sting-linux

Adversary Models

PENNSTATE 1855

- How should we identify adversaries of a program?

Adversary Models

PENNSTATE 1855

- How should we identify adversaries of a program?
- Researchers have evaluated research prototypes where adversaries are
  - Not a root process
  - Not a process with the same user id
- But, lots of root processes and processes with your user id – reasons not to trust them
  - Compromised root network daemons and user processes
  - ACL Policy may be modified by compromised processes

Adversary Models

- Instead we have explored using *available* mandatory access control (MAC) policies
  - Fine-grained: confine root processes
  - Immutable: mandatory system policy
  - Restrictive: least privilege MAC policies
- To define a conservative adversary models
  - What are the minimal set of subjects that a process must trust when it executes?
    - Those that already have the permissions sufficient to compromise the process

Subjects That Can Attack Already

PENNSTATE 1855

- Subjects a process must trust… [ASIACCS 2012]
  - Subjects that can modify the process's executable file
  - Subjects that can modify the kernel objects
  - Subjects that can modify executable files of these subjects
    - Applied transitively
- Practical?
  - Only 81 of over 2000 system call sites in Linux system service programs access resources modifiable by these adversaries

Testing Conclusions

- What are the lessons from STING that we can use?
- Adversary accessibility
  - Vulnerabilities require adversary accessibility
  - Can leverage conservative adversary models based on available MAC policies
- In-vivo
  - Test the program when it is in a state that accessible to adversaries
  - Launch attacks (in some way) to test program reaction, while keeping the program running (low overhead)
- Live, Adversary-Aware testing can be practical