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

JIGSAW: Protecting Resource Access by Inferring Programmer Expectations

Samsung Research
Hayawardh Vijayakumar, Xinyang Ge,
Purdue University
Mathias Payer, Trent Jaeger
SIIS Lab, Penn State University
University of California Berkeley

Resource Access

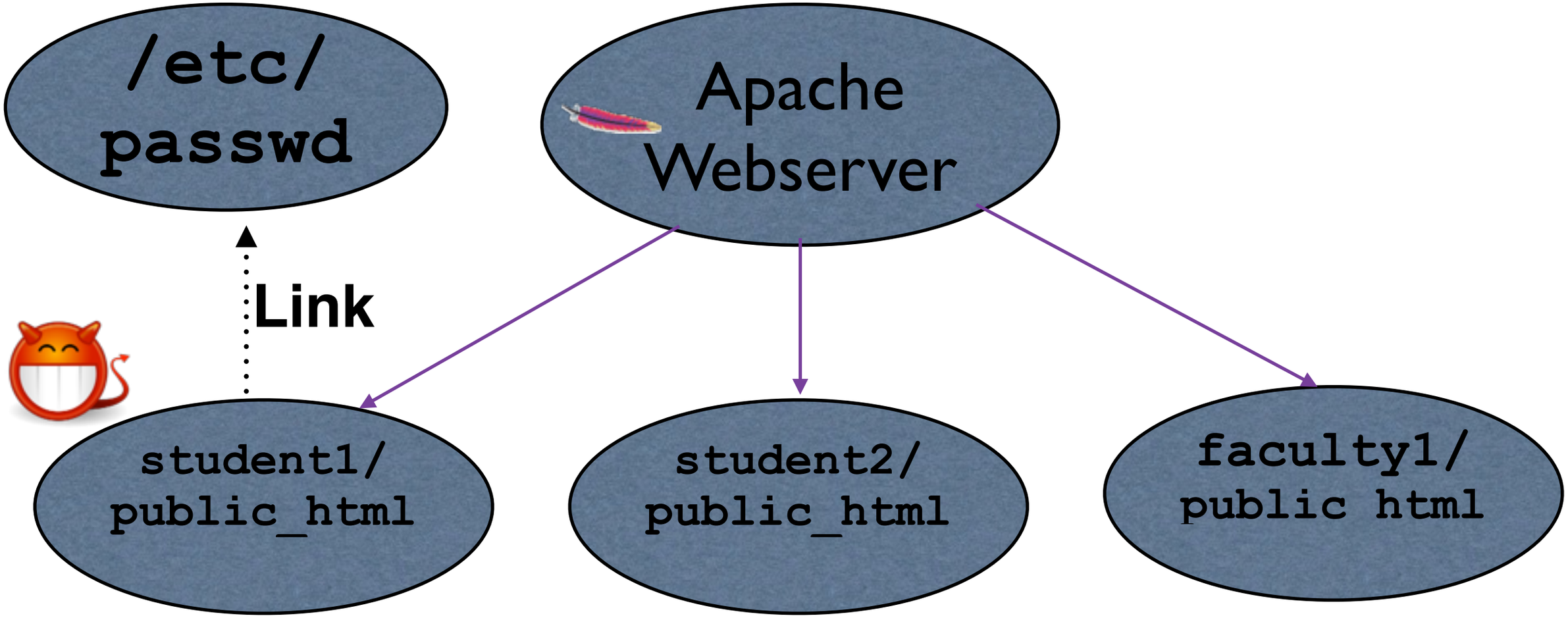
- Programs require a variety of resources to function
  - *Libraries*
  - *Configuration Files*
  - *Log Files*
  - *Application Data*
- How hard can accessing resources securely be?
  - Just a simple `open(filename)`, right?



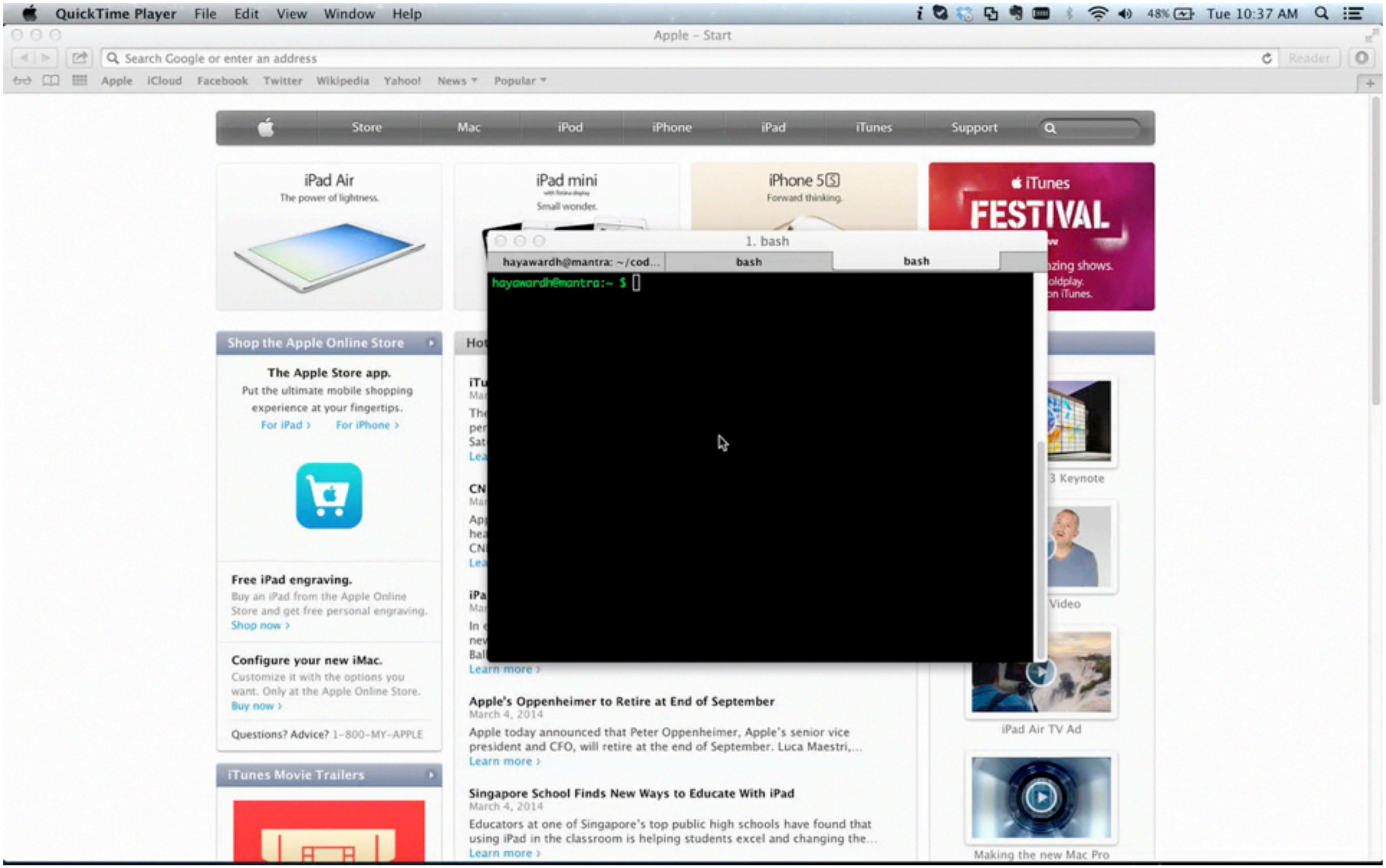
A Webserver's Story …

- Consider a university department webserver …

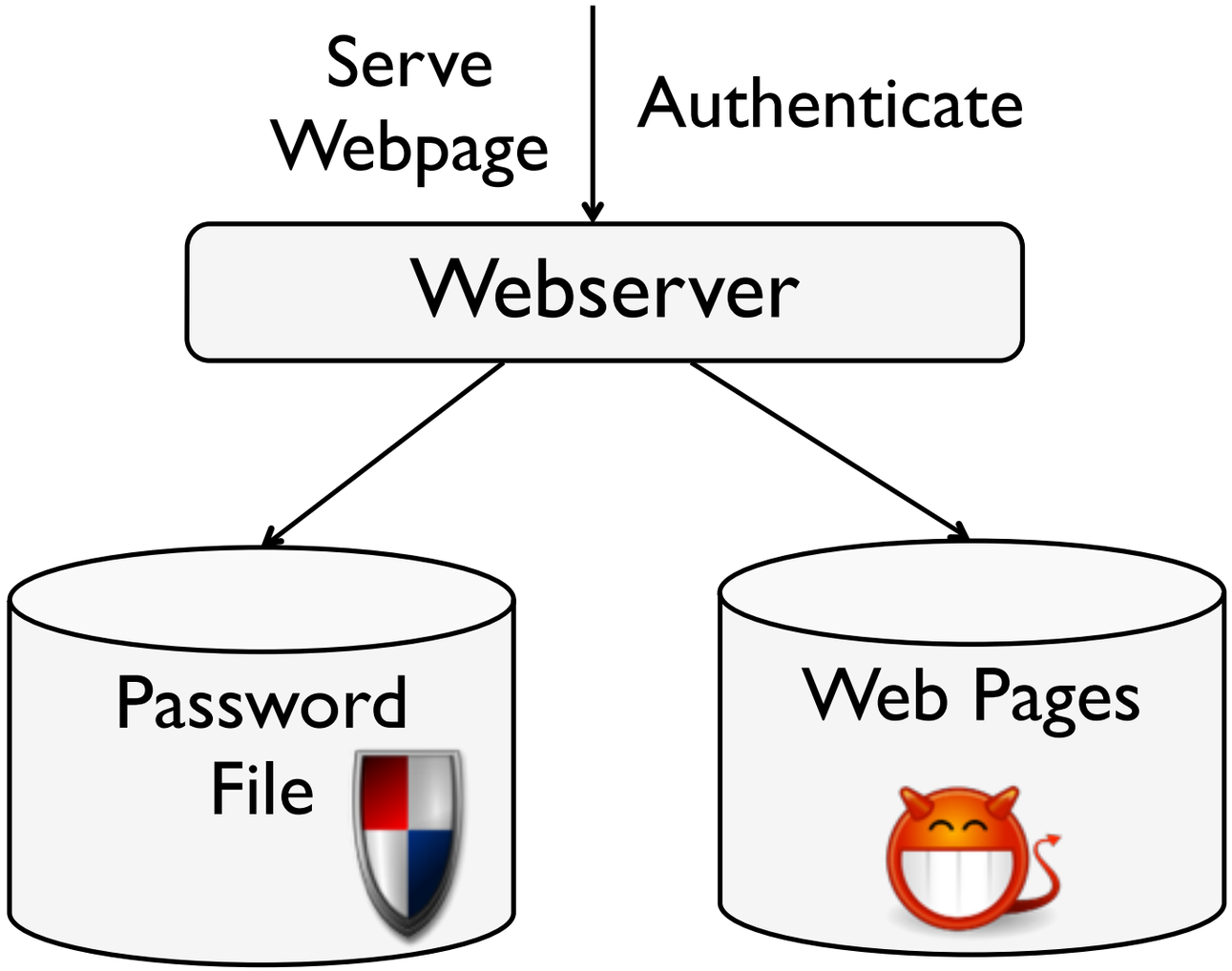
GET /~student1/index.html HTTP/1.1

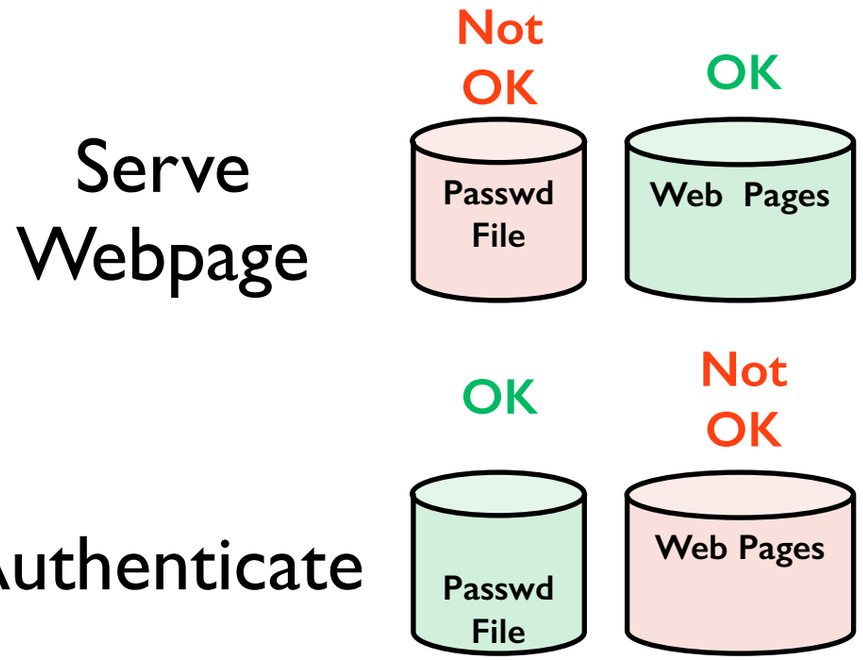


Attack Video



What Just Happened?

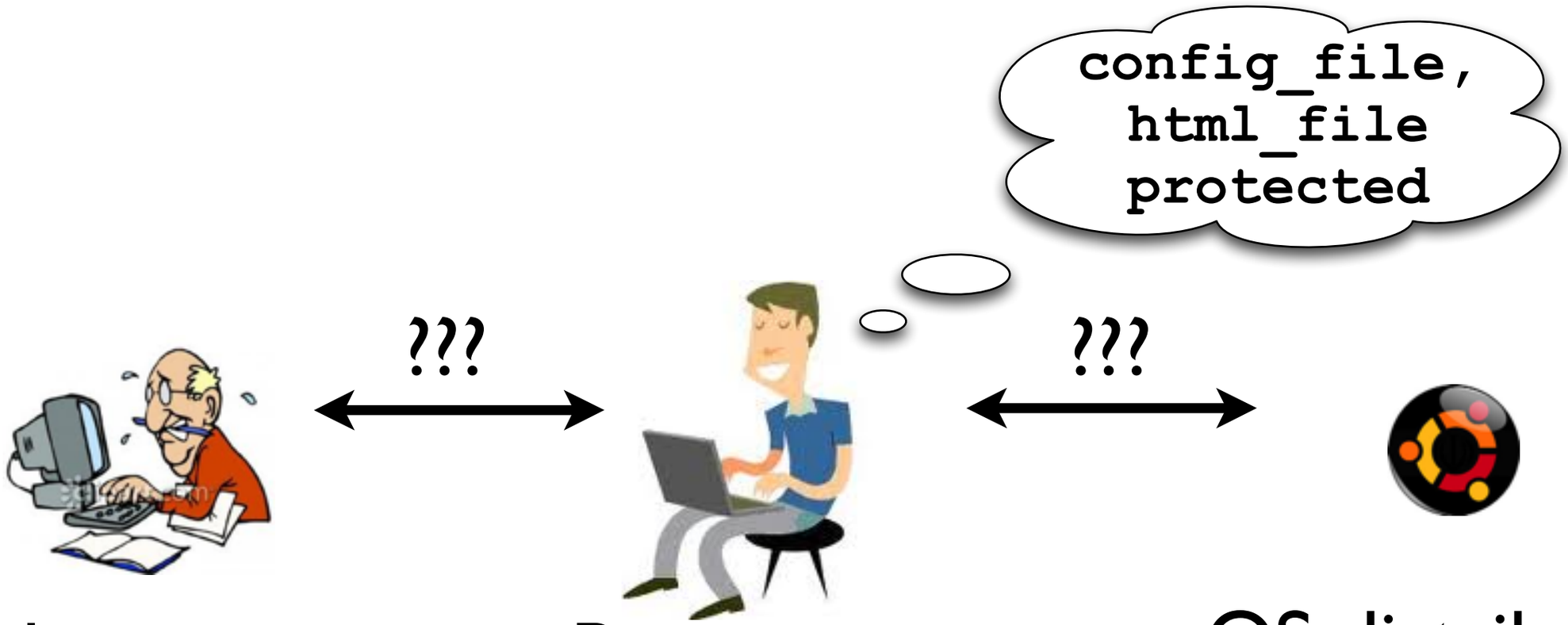


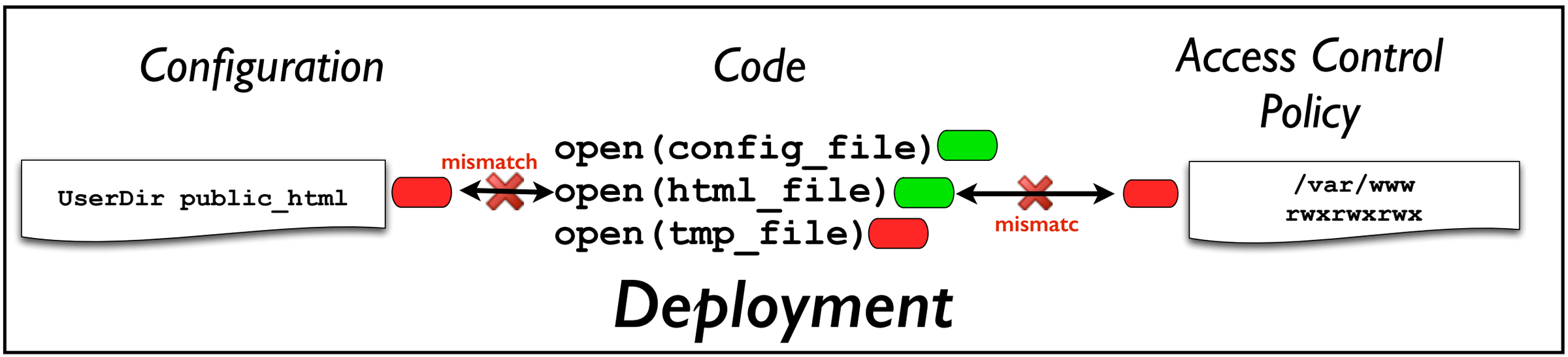


- Program received an *unexpected* resource
  -  when expecting 
  -  when expecting 

Cause 1 - Multiple Parties

Expectations mismatch, blame each other





Cause 2 - Code Complexity

- Even when expected, filters complicated to write
- Check for symlink (lstat)
- Check for lstat-open race
- Check for inode recycling
- For each binding
  - /, home, public_html, ...
- Cannot expect programmers to get this right!

```
/* fail if file is a symbolic link */
   int open_no_symlink(char *fname)
   {
01  struct stat lbuf, buf;
02 int fd = 0;
03  lstat(fname, &lbuf);
04  if (S_ISLNK(lbuf.st_mode))
05    error("File is a symbolic link!");
06  fd = open(fname);
07  fstat(fd, &buf);
08  if ((buf.st_dev != lbuf.st_dev) ||
09      (buf.st_ino != lbuf.st_ino))
10    error("Race detected!");
11  lstat(fname, &lbuf);
12  if ((buf.st_dev != lbuf.st_dev) ||
13      (buf.st_ino != lbuf.st_ino))
14    error("Cryogenic sleep race!");
15  return fd;
   }
```

Solution Overview

- 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 
    - Well-known "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 access adversary-modifiable resources

$$S \subseteq P$$

Invariant: Unexpected Adversary Control(r) **:**

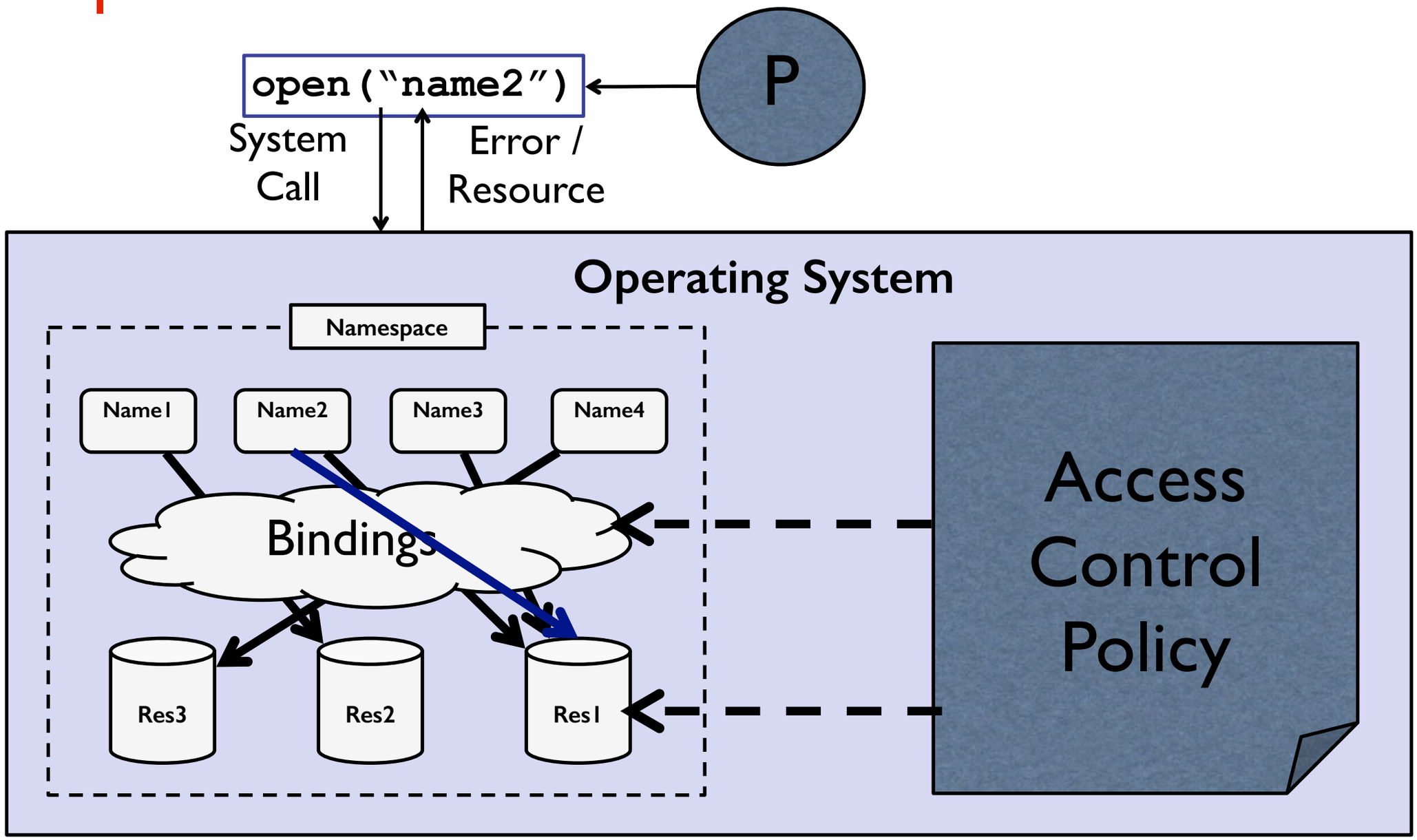
$$(r \in S) \rightarrow (r \in P)$$

Invariant: Confused Deputy(r) **:**

$$(r \in S) \rightarrow (r \in R)$$

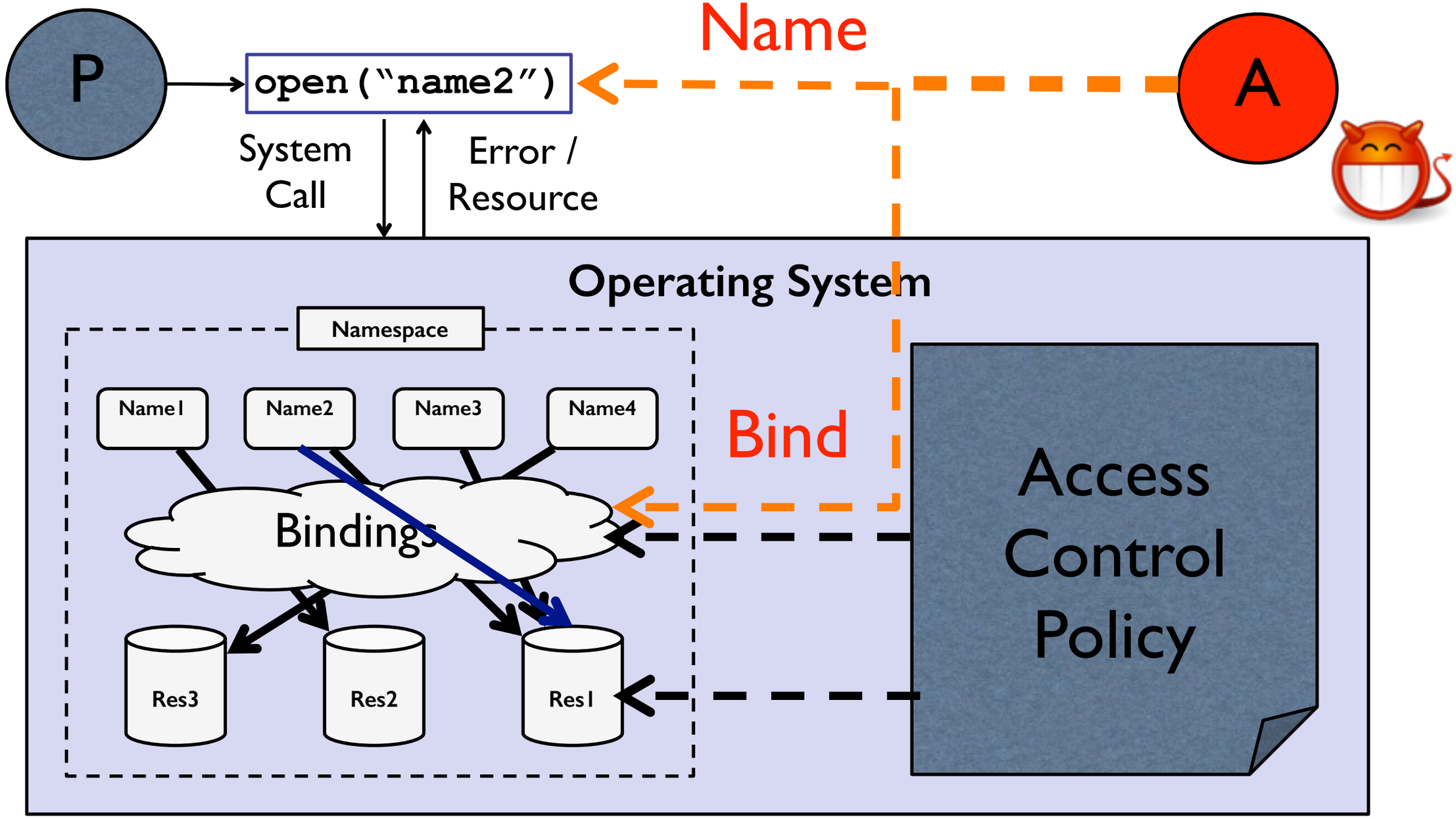
Resource Access

- Inputs: name, bindings
- Output: resource



Resource Access Attacks

- Adversaries control inputs -- name, bindings to direct victim programs to unexpected resources



Resource Access

- Account for 10-15% of total vulnerabilities yearly

Attack	Adversary Control	CWE class	CVE Count	
			<2007	2007-2012
Untrusted Search Path	Name, Resource	CWE-426	109	329
Untrusted Library Load	Name, Resource	CWE-426	97	91
File/IPC squat	Resource	CWE-283	13	9
Directory Traversal	Name	CWE-22	1057	1514
PHP File Inclusion	Name, Resource	CWE-98	1112	1020
Symbolic/Hard Link Following	Binding	CWE-59	480	357
TOCTTOU Races	Binding, Resource	CWE-362	17	14
Signal Races	Resource	CWE-479	9	1
Total Resource Access CVEs	-	-	2894	3335
Total CVEs	-	-	23337	35409
% Total CVEs	-	-	**12.40**	**9.41%**

Previous Work

- Problems during resource access known for a very long time
  - TOCTTOU attacks known at least since 1973
- Solutions proposed, broken
  - Cai et al. [SP '10]
    - All system-only defences for TOCTTOU have false positives unless they know about system call intent
  - More generally,
    - Need to account for programmer expectations and system deployment

In this talk ...

Can we find vulnerabilities due to mismatches?

Can we protect programs from being exploited?

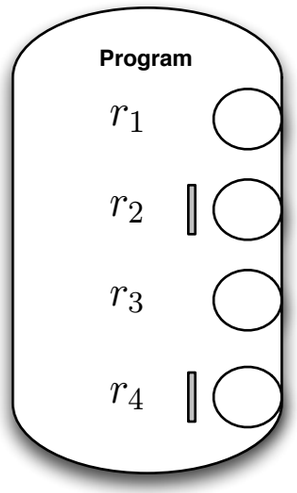
To Find Mismatches

- Need a model that describes
  - How a program performs resource access.
    - How do programs build names, bindings?
  - What are programmer expectations for resource access?
    - If they expect adversary access to names, bindings: *protect*
    - If not: *do nothing* :->



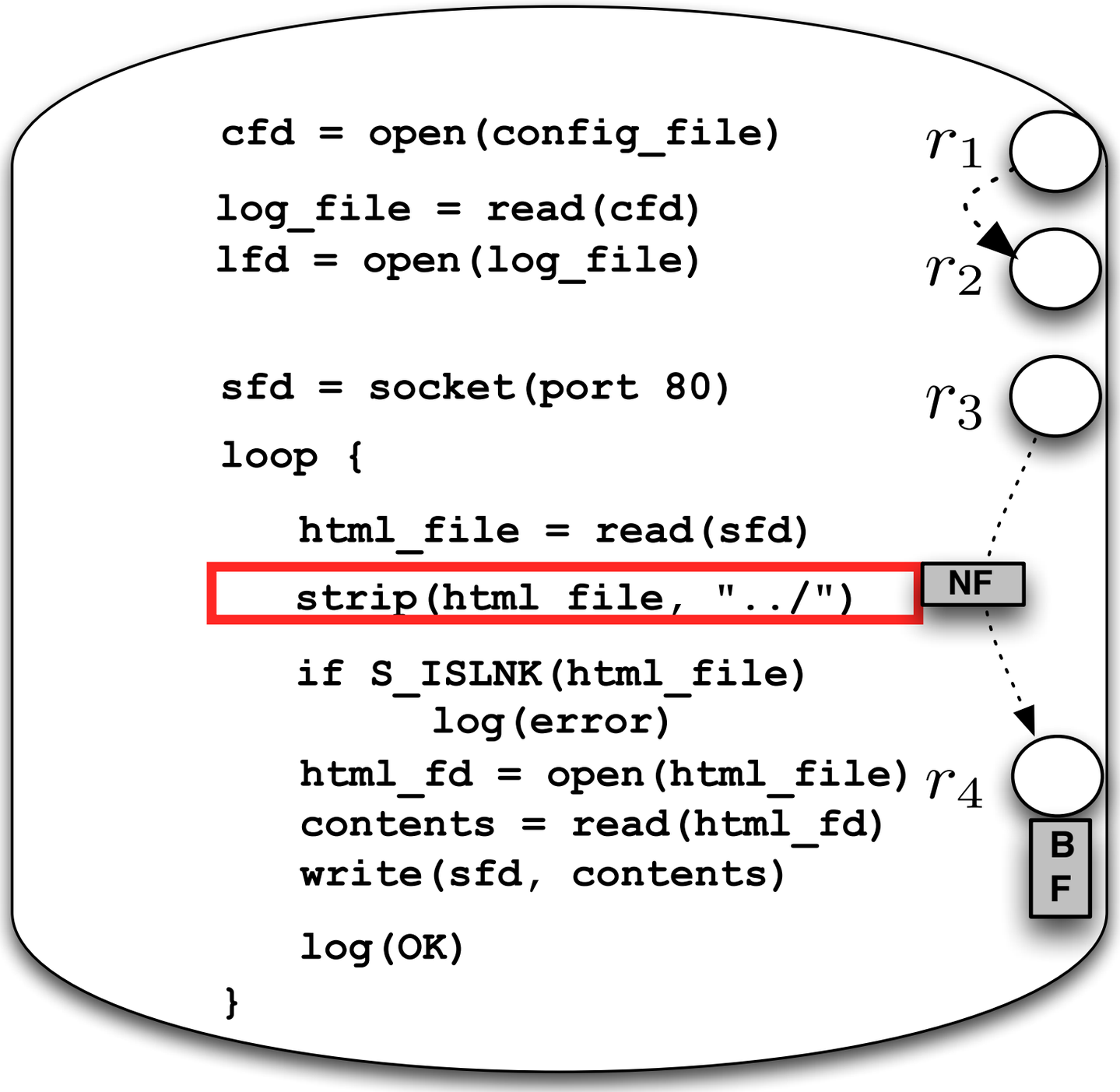
Programmer Expectations

- 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

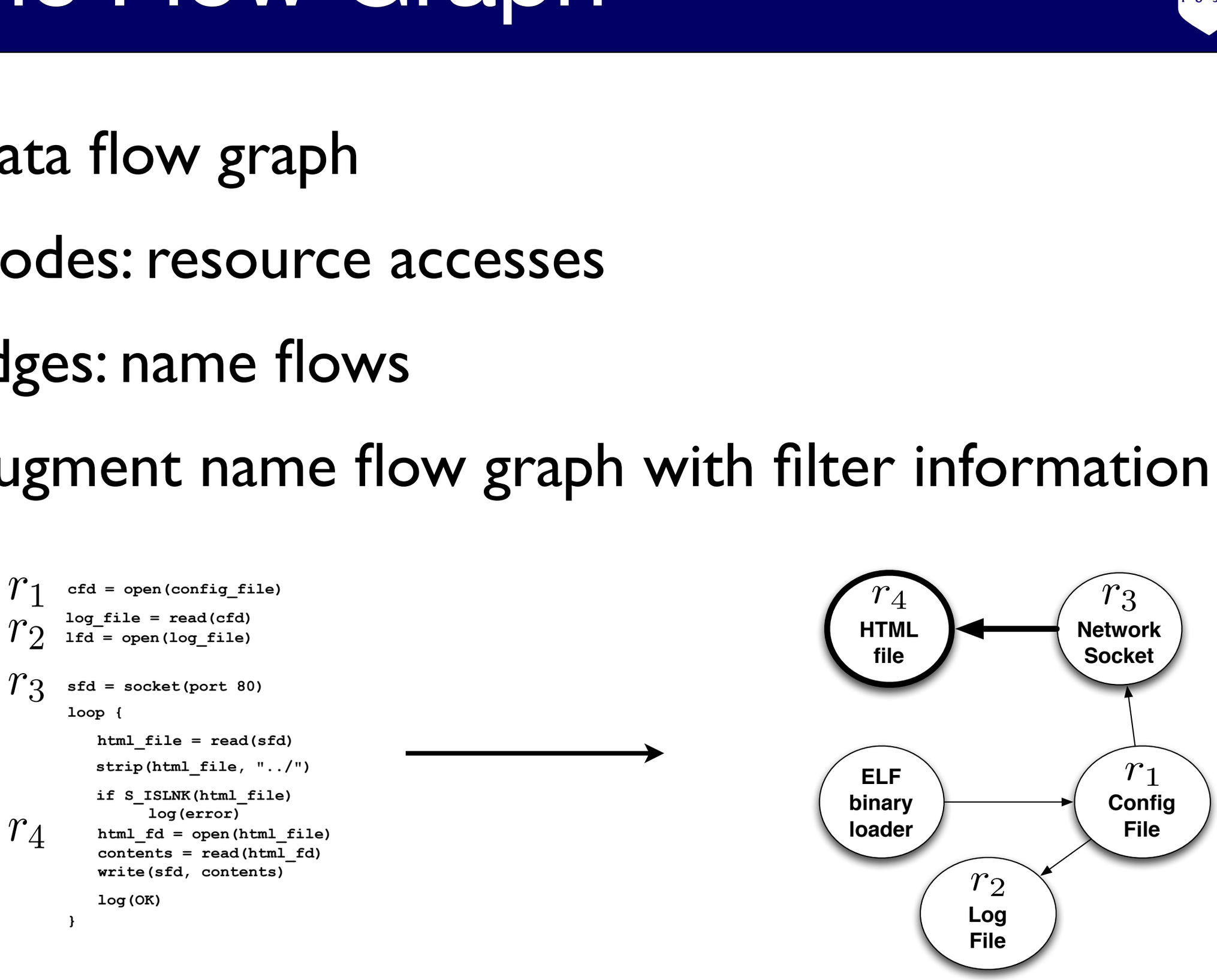


Filters as Annotations

- *Heuristic*: If programmer expects adversarial control of resource access, she will add name/binding filters
  - *Corollary*: No filter $\Longrightarrow$ access only 

Name Flow Graph

- Data flow graph
- Nodes: resource accesses
- Edges: name flows
- Augment name flow graph with filter information



Determine P from filters

- No filter $\Longrightarrow$ not in expected attack surface P
- If no binding filter

Base Case 1  $\Longrightarrow \quad r_1 \notin P$

- If no name filter on an outgoing name flow

Base Case 2  $\Longrightarrow \quad r_1 \notin P$

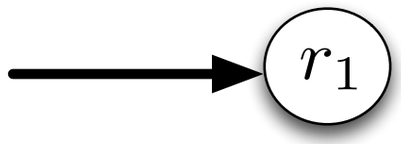
- If a resource access not in *P* is reachable

Transitive Closure  $\Longrightarrow \quad r_1 \notin P$

- Any remaining resource access is in *P*

Likely Missing Filters

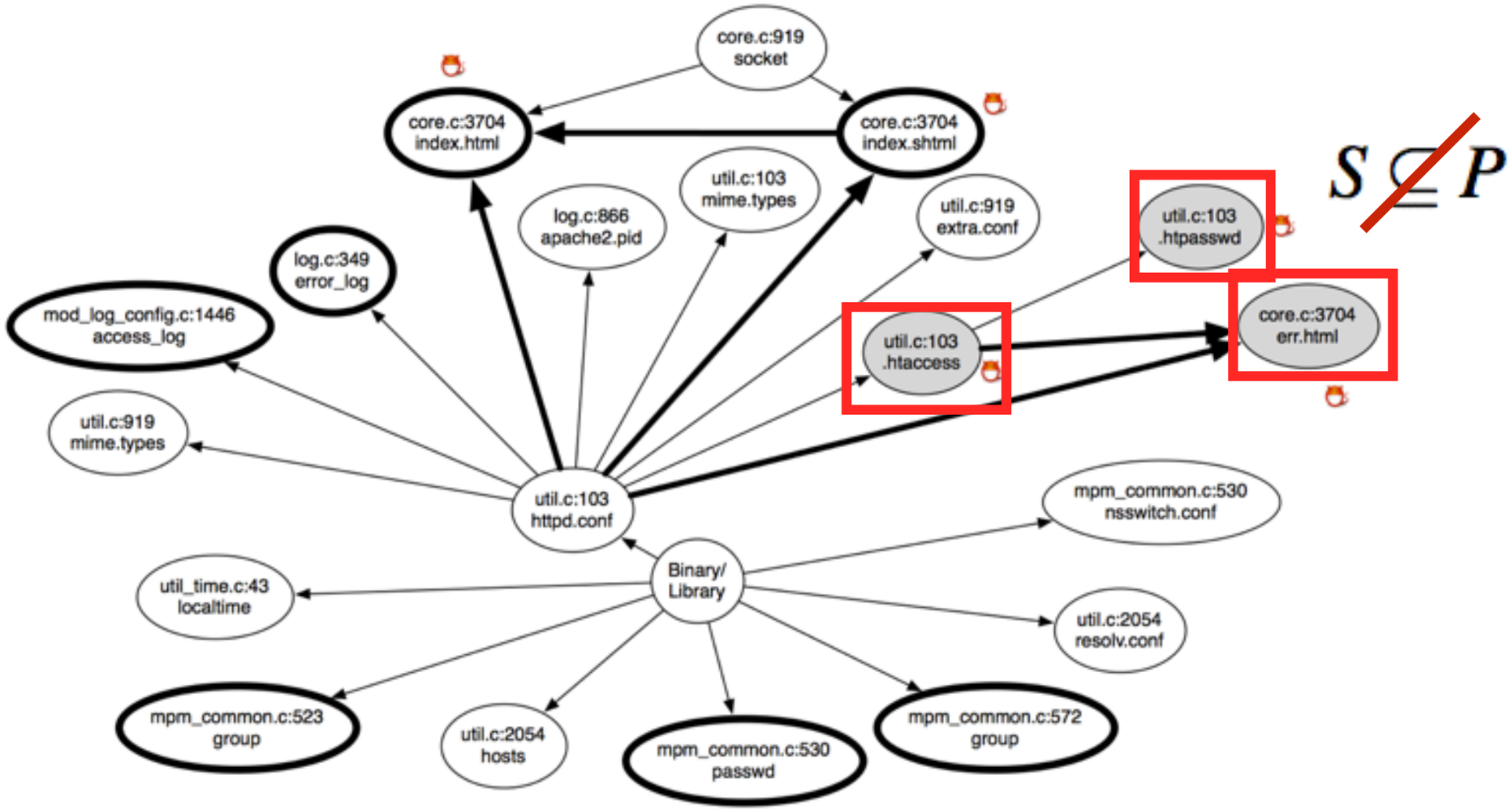
- From the name flow graph, we can locate cases where filters are likely missing
  - Filter incoming name flow but not binding



  - Filter binding but not outgoing name flow



Runtime Mismatches



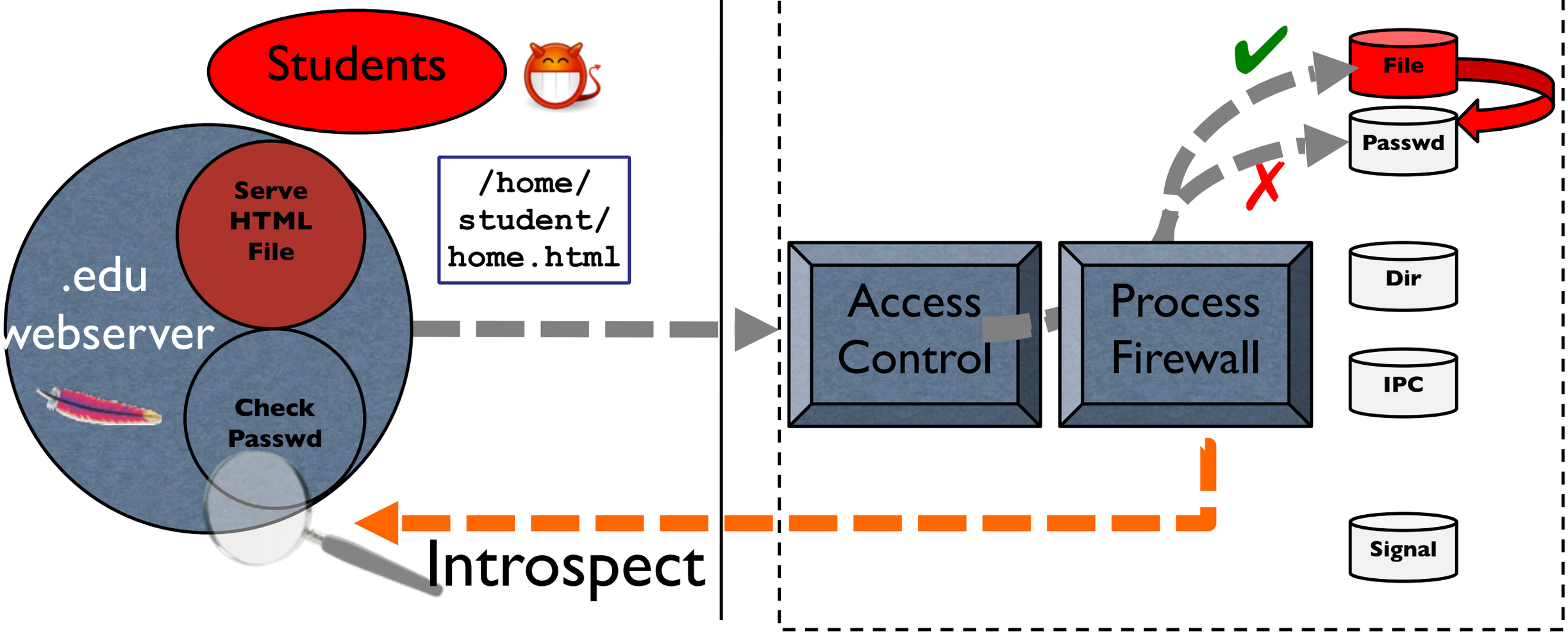
In this talk ...

Can we find vulnerabilities due to mismatches?

Can we protect vulnerabilities from being exploited?

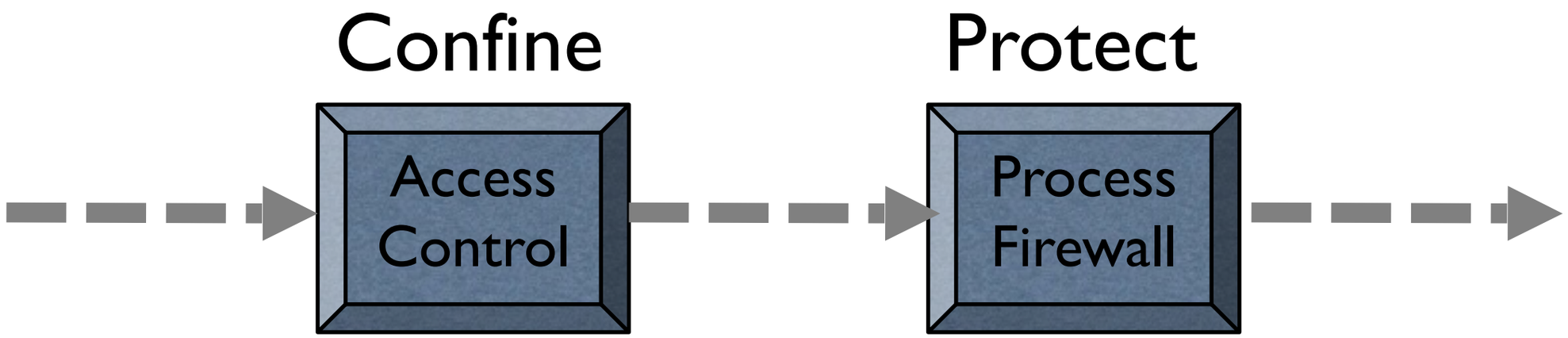
Process Firewall

Protection per resource access [Eurosys '13]



Process Introspection

- Why can we introspect into process?
  - Mimicry attacks (intrusion detection systems)?
- The Process Firewall *protects* *victim processes* instead of *confining* *adversary processes*
  - Mimicry only invalidates process's own protection



Process Firewall

- Enforce rules without requiring program code or system policy/configuration change
- Each invariant is instantiated using rule templates

Invariant: Unexpected Adversary Control(r) :

$$(r \in S) \rightarrow (r \in P)$$

For each $r \in \overline{P}$:

```
pftables -i r.ept -d LOW -o DIR_SEARCH -j DROP
```

Evaluation - Expectation

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

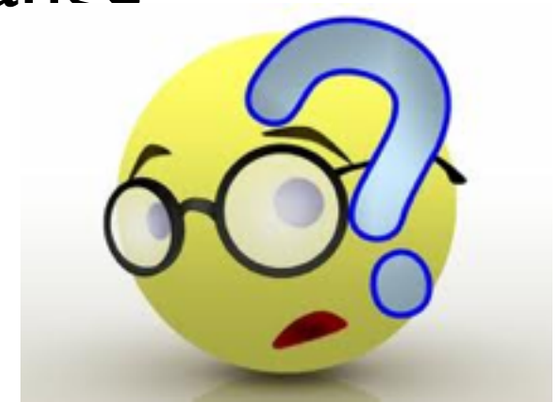
- Apache allows users to specify a password file to protect their pages in .htaccess

```
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

Vulnerability

- Typical example of resource access vulnerability
  - Who is to blame?
    - Admin for not recognizing adversaries and improper configuration?
    - OS distributor for default insecure configuration?
    - Programmer for providing the configuration option?
- Difficult to tell, but the Process Firewall can fix vulnerability without forcing code or access control policy change



Enforcement

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 total, 281 rules for Inv. 1, 97 for Inv. 2
- We verified the ability of our generated rules to block discovered vulnerabilities
  - In addition, we tested previously-known representative resource access



Conclusion

- We define resource access vulnerabilities and propose principled invariants to block them
- Evaluation on programs showed
  - Programmers have implicit expectations
    - In 4/5 programs we tested, > 55% of resource accesses were unprotected
  - Resource access currently reasoned in an ad-hoc way
    - Found vulnerabilities and a default misconfiguration in Apache
- Enforcement by the Process Firewall shows promise in blocking attacks without code or policy change

Process Firewall Availability

https://github.com/siis/pfwall

hayawardh@gmail.com