Detecting Access Control Vulnerabilities in Web Applications

Zhendong Su

UC Davis

Joint with Fangqi Sun and Liang Xu

How we got here?

- Formalizing SQL injection attacks (POPL'06)
- Dynamic detection of SQL injections (POPL'06)
- Static detection of SQL injections (PLDI'07)
- Static detection of XSS vulnerabilities (ICSE'08)
- Concolic testing to detect SQL injections (ISSTA'08)

What's next?

- SQL injection & XSS ⇔ buffer overflow
  - Application agnostic
- Access control errors ⇔ wrong logic
  - Application specific

- Challenge: Can application-specific access control errors be detected fully automatically *without explicit access control policies*?

Access Control Vulnerabilities

- Failure to guard privileged resource
  - A chain is as strong as its weakest link

- > 14% web applications have it [WASC'07]
  - Difficult to design and implement perfect checks
- Culprit of privilege escalation attacks
  - Exposure of sensitive information or operations

Predictable URLs

□ Earnings of NetApp and Disney were leaked to Bloomberg in Nov., 2010

LEAKED

http://media.netapp.com/documents/financial-fy11-q2.pdf
http://media.netapp.com/documents/financial-q1-fy11.pdf
http://media.netapp.com/documents/financial-10-q4.pdf
http://media.netapp.com/documents/financial-q3-10.pdf

Bloomberg

File posted **without any required password**

NetApp™

File obtained from "**a restricted area of the company's website.**"

Exploited Access Control Errors

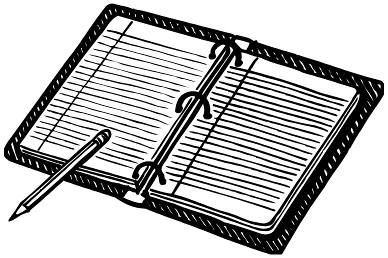
- Business Wire
  - Important files were uploaded but not linked
  - Link format

    http://website/press_release/**08/29/2007/00001**.html

- Lohmus Haavel & Viisemann
  - Obtained trading information
  - Profited over $8 million

Key Challenge

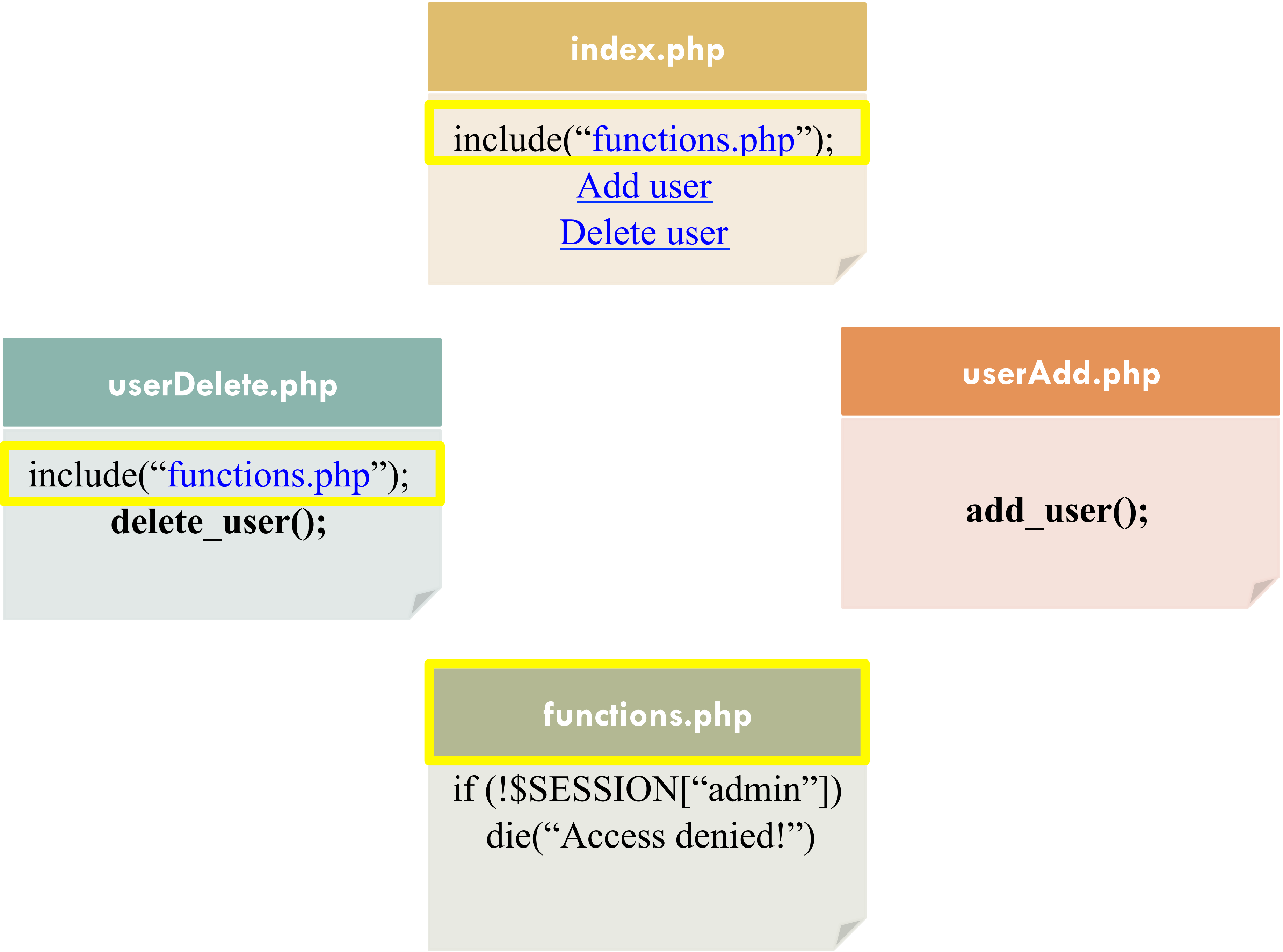
- Specifications for automated detection
  - Manual specification
    - Time-consuming, and often absent
  - Probabilistic-based inference
    - Imprecise and computationally expensive



Key Insights

- Source code of an application implicitly documents intended accesses of each *role*

- Access control policy can be extracted from differences in *per-role sitemaps*



index.php

```
include("functions.php");
```

Add user

Delete user

userDelete.php

```
include("functions.php");
delete_user();
```

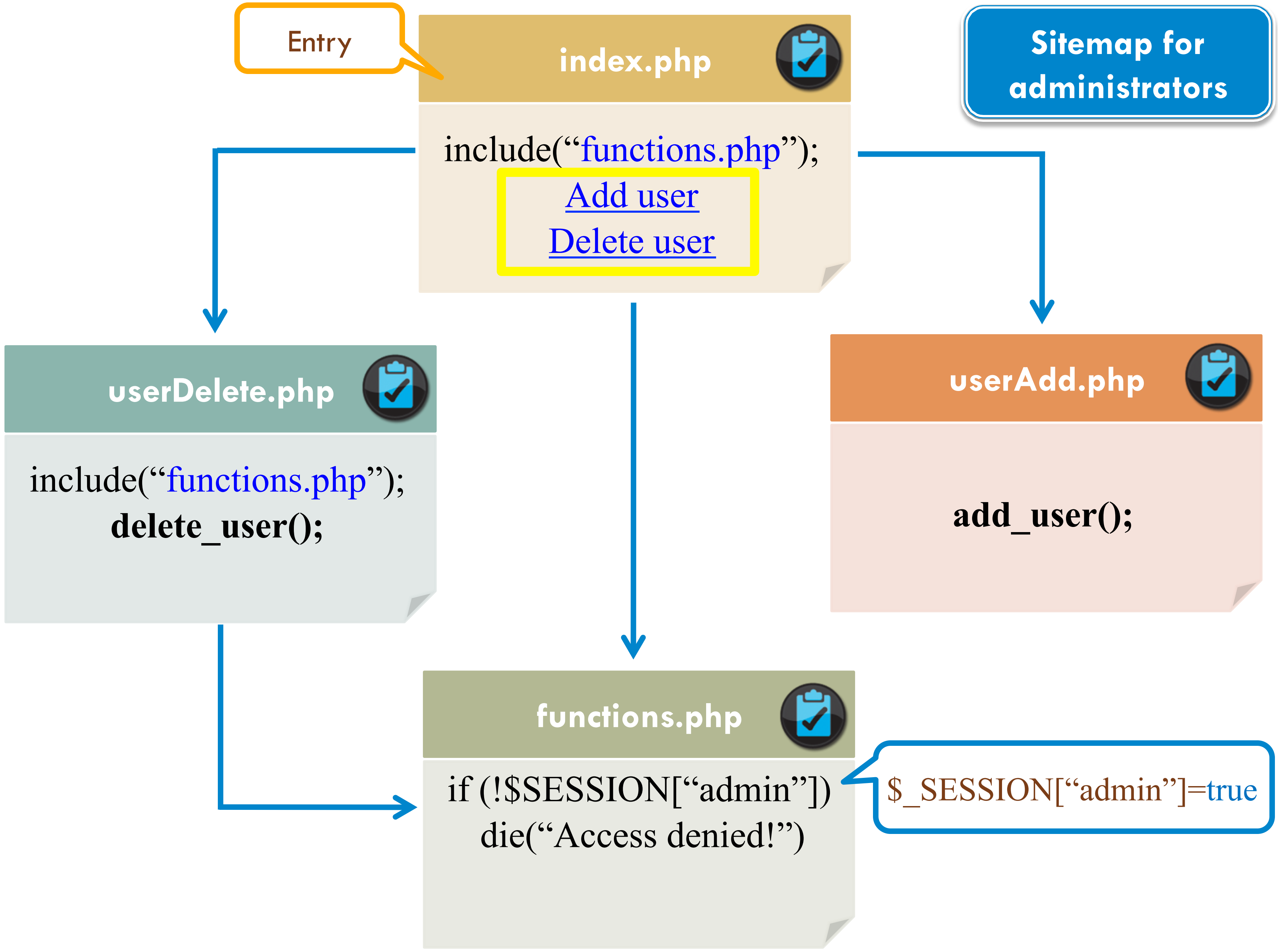
userAdd.php

```
add_user();
```

functions.php

```
if (!$SESSION["admin"])
  die("Access denied!")
```

Sitemap for administrators



Entry

index.php

```
include("functions.php");
```

Add user

Delete user

userDelete.php

```
include("functions.php");
delete_user();
```

userAdd.php

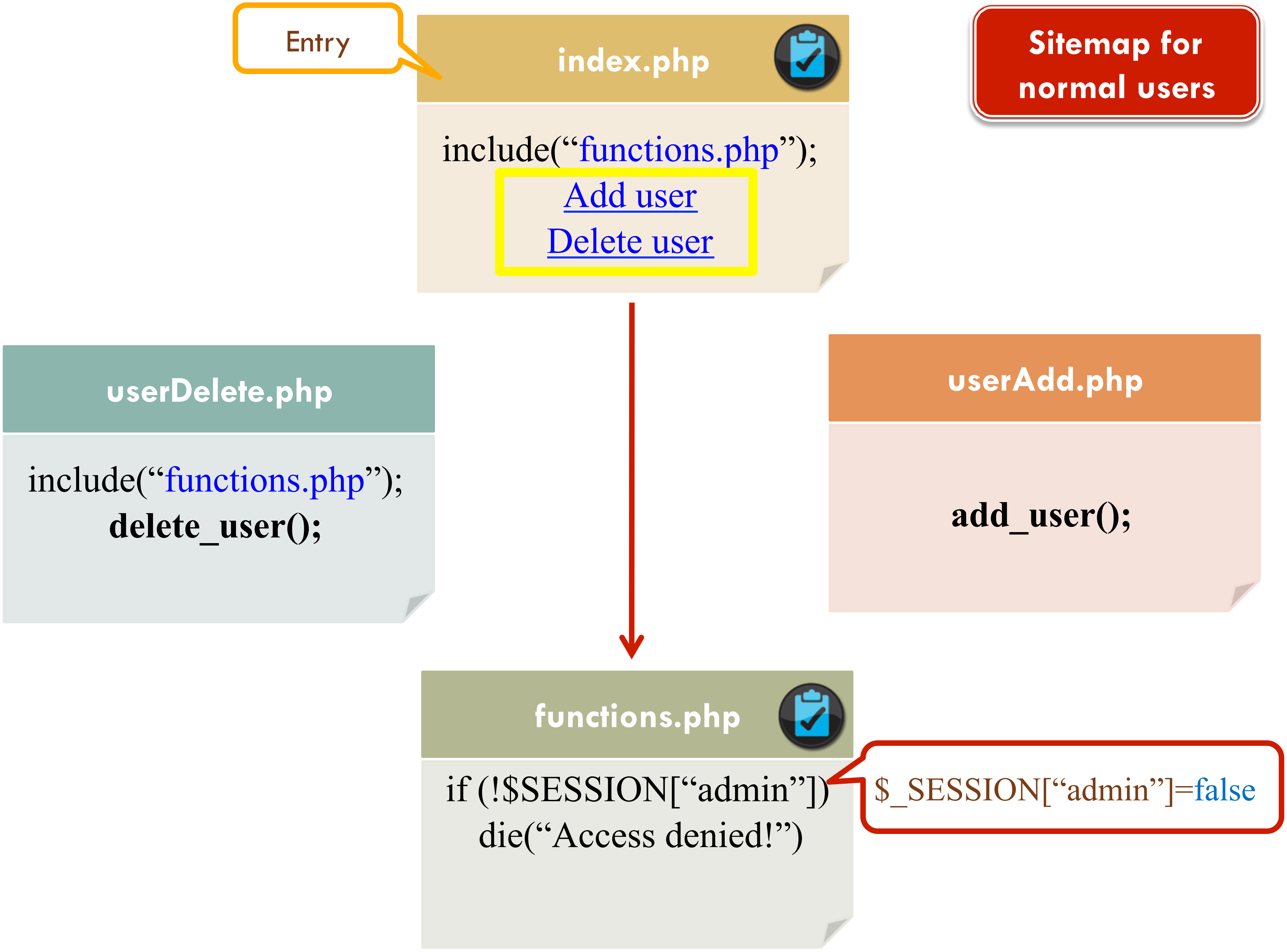
```
add_user();
```

functions.php

```
if (!$SESSION["admin"])
die("Access denied!")
```

$_SESSION["admin"]=true

Sitemap for normal users



Entry

index.php

```
include("functions.php");
Add user
Delete user
```

userDelete.php

```
include("functions.php");
delete_user();
```

userAdd.php

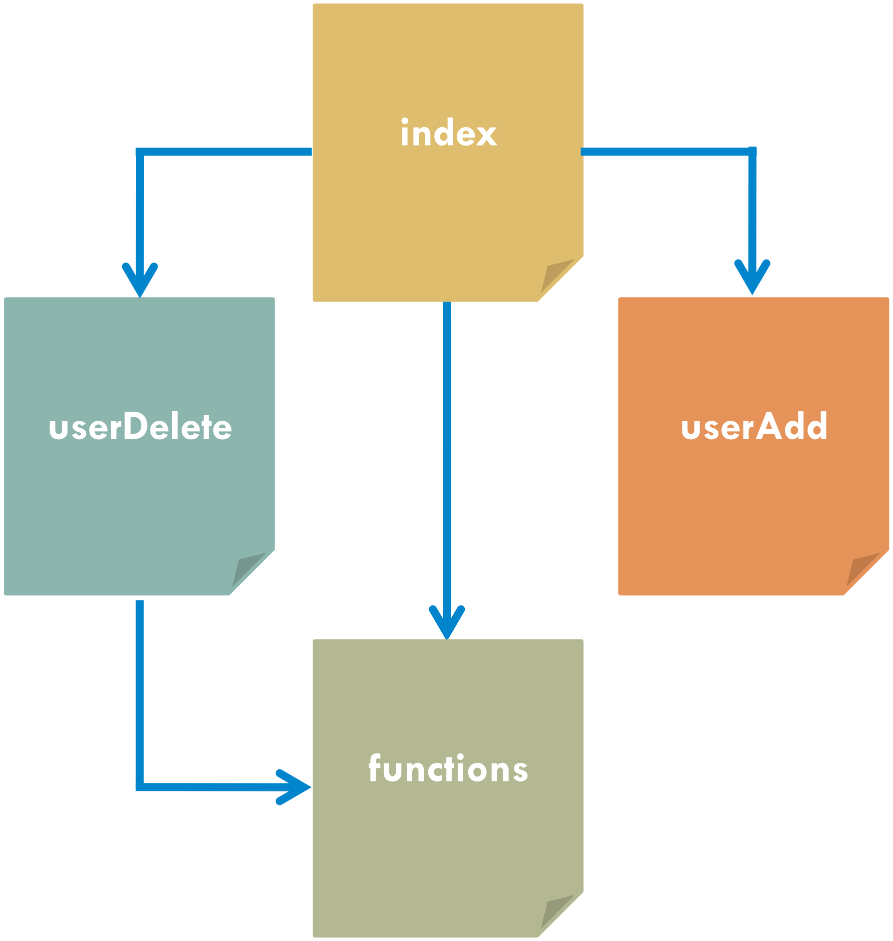
```
add_user();
```

functions.php

```
if (!$SESSION["admin"])
die("Access denied!")
```

$_SESSION["admin"]=false

Sitemap for administrators



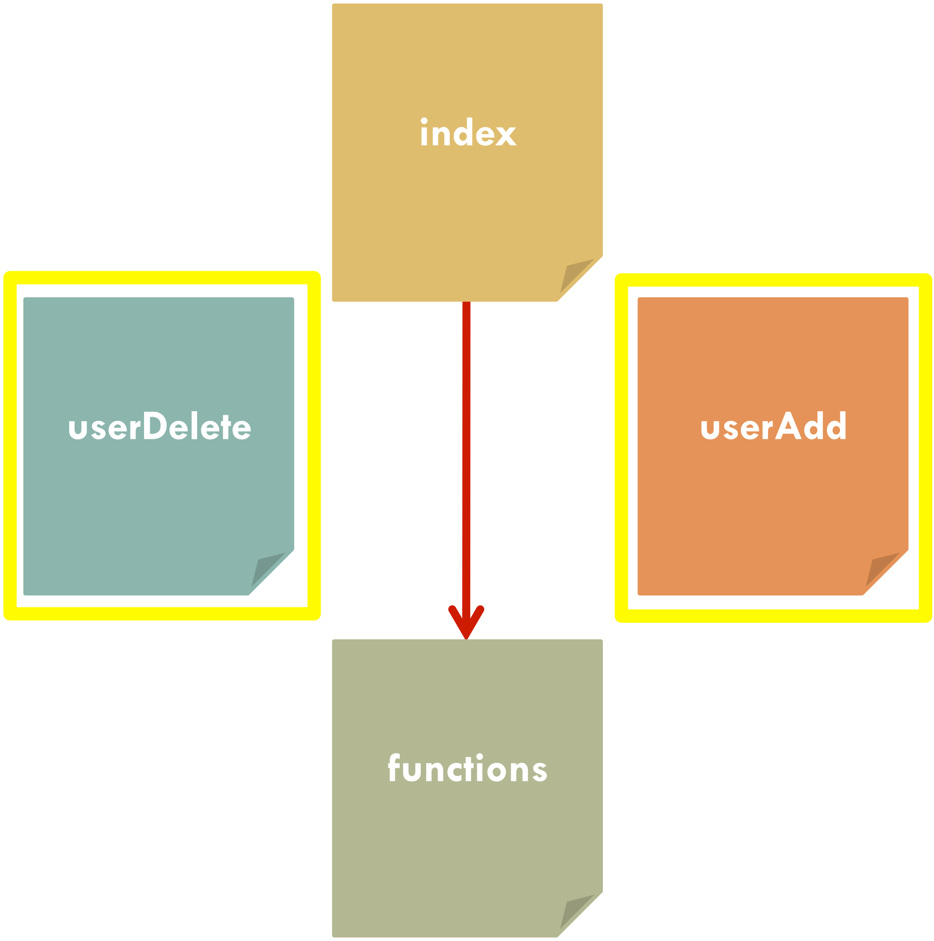
index

userDelete

userAdd

functions

Sitemap for normal users



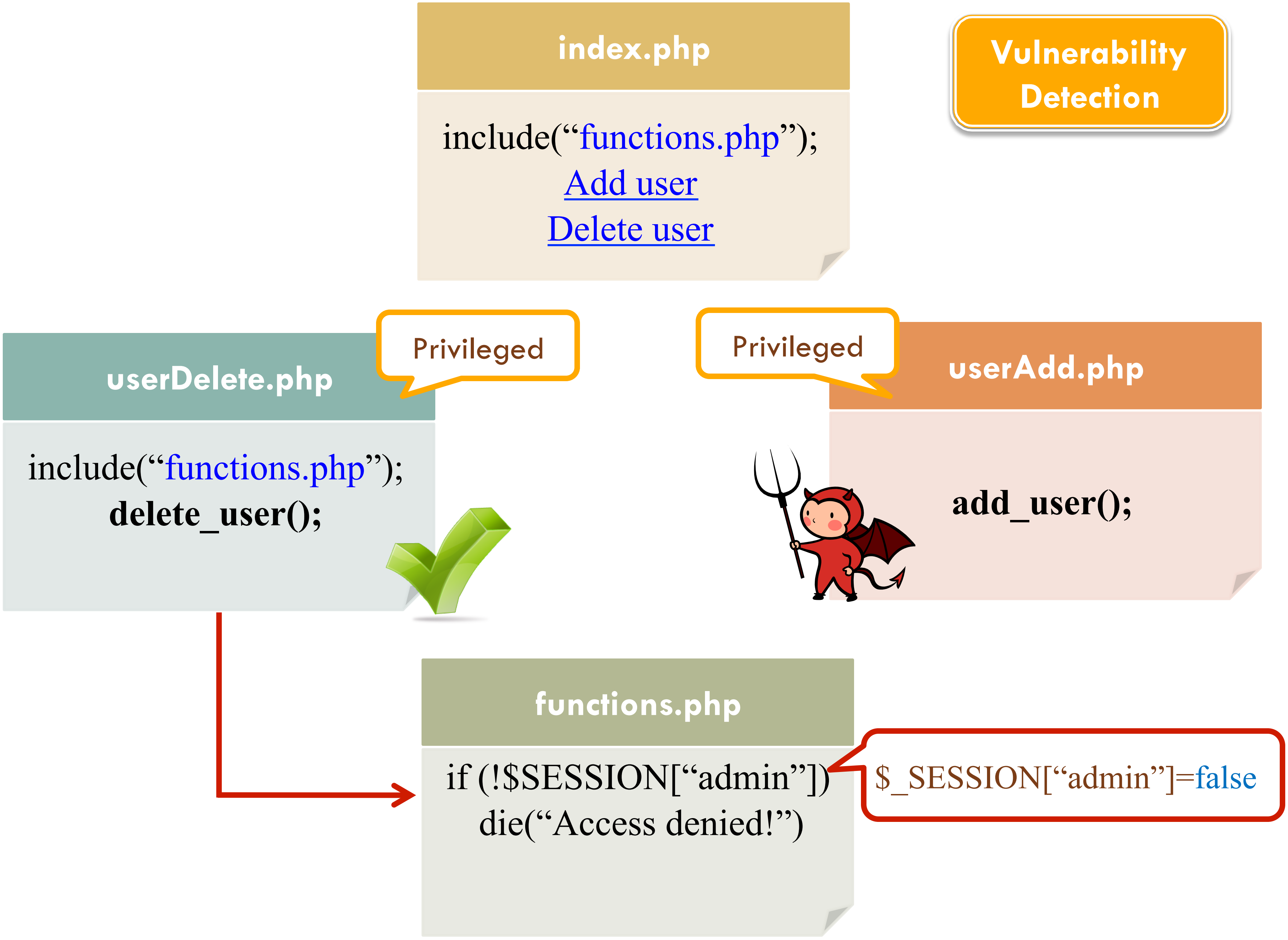
index

userDelete

userAdd

functions

Vulnerability Detection



index.php

```
include("functions.php");
Add user
Delete user
```

userDelete.php

```
include("functions.php");
delete_user();
```

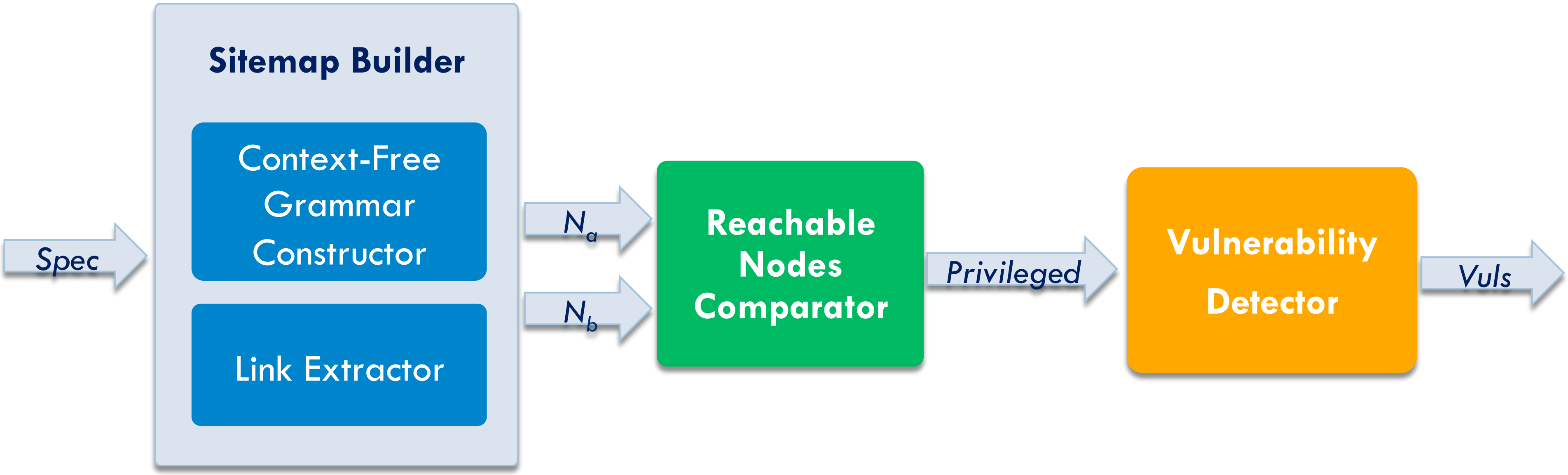
userAdd.php

```
add_user();
```

functions.php

```
if (!$SESSION["admin"])
die("Access denied!")
```

Technical Approach



Spec inputs : (entry points, role-based states)

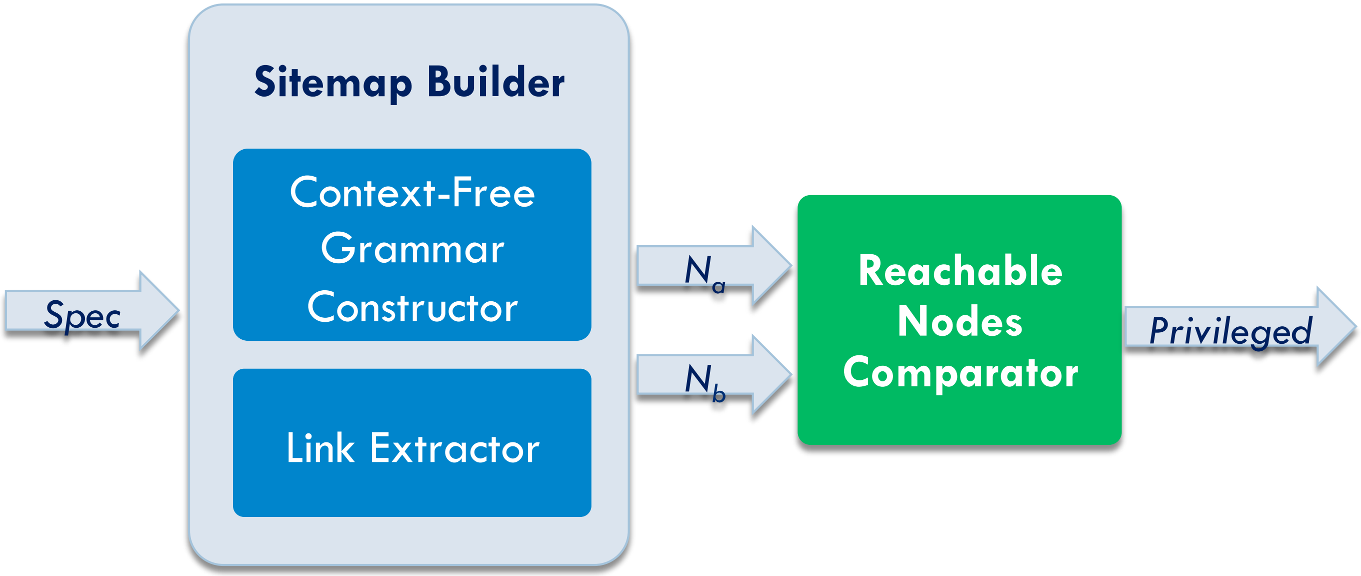
N_a explicitly reachable nodes of role *a* (adminstrators)

N_b explicitly reachable nodes of role *b* (normal users)

Privileged privileged nodes

Vuls vulnerabilities

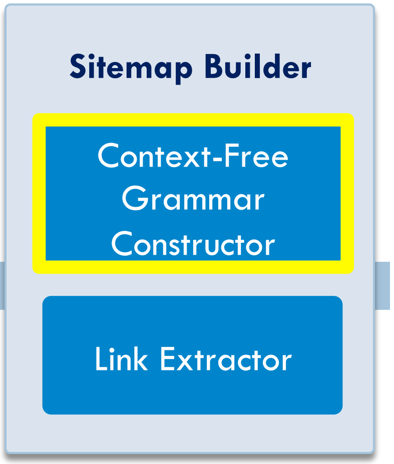
Sitemap Builder



Context-Free Grammar Constructor: Given role-based specifications , it statically generates CFGs to approximate dynamic outputs

Link Extractor: With a Deterministic Finite Automaton (DFA) that matches links, it extracts explicit links from a CFG

Context-Free Grammar Constructor



- CFG approximates dynamic output
  - PHP page → AST → IR → grammar rules → CFG

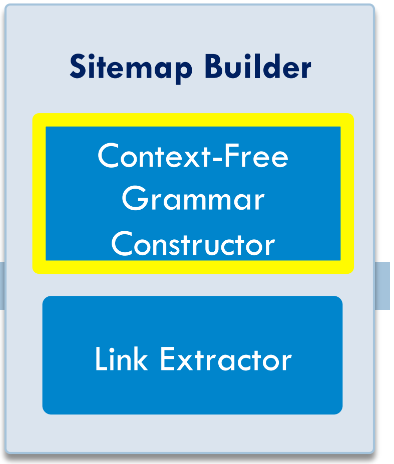
- Path exploration based on branch feasibility
  - Z3 for arithmetic/Boolean constraints
  - Our own string solver for string constraints

```
function checkUser() {
 if (!$_SESSION["validUser"])
   header("Location: login.php");
}
checkUser();
sensitiveOperation();
```

Constraint: $_SESSION["validUser"] = false

- Only administrators can pass this check and reach sensitiveOperation()
- Normal users are redirected to "login.php"

Link Extractor


Sitemap Builder

Context-Free Grammar Constructor

Link Extractor

- Our link extraction algorithm
  - Does not directly intersect CFG with DFA
  - Efficiently extracts links from CFG based on DFA

```
echo "<div><a href=" . $lang . ".php>Anchor</a></div>";
```

CFG

```
S0 → S1 S2
S1 → "<div><a href="
S2 → S3 S4
S3 → "english" | "spanish" | "french"
S4 → ".php>Anchor</a></div>"
```

→

Links

- "english.php"
- "spanish.php"
- "french.php"

Vulnerability Detector

Vulnerability Detector

- Failed forced browsing
  - Redirects users to another location
  - Displays error messages
    - No sensitive information or operation

- When is a forced browsing successful?
  - CFG of administrator vs. CFG of normal users
    - No additional redirections
    - The CFG sizes are not significantly different

Implementation

- A static PHP analyzer
  - Based on work of Wassermann and Minamide
  - Adds support for roles
  - Connects nodes of a Web application
  - Explores paths based on branch feasibility

- Specification rules
  - Only require specifications of two sets
    - Set of entry nodes
    - Set of role-based critical application states
  - Support abstract and concrete value, and regular expressions

Evaluation

- Subjects
  - Seven applications
- Metrics
  - Effectiveness
    - Vulnerable nodes
    - False positives
  - Performance
    - Coverage
    - Analysis time

Subject	Files	LOC	
		PHP	HTML
SCARF	25	1,318	0
Events Lister	37	2,076	544
PHP Calendars	67	1,350	0
PHPoll	93	2,571	0
iCalendar	183	8,276	0
AWCM	668	12,942	5,106
YaPiG	134	4,801	1,271

Project	Privileged	Vulnerable	FP	Guarded	Admin		Normal	
					Node	Edge	Node	Edge
SCARF	4	1	0	3	19	149	15	69
SCARF (patched)	4	0	0	4	19	149	15	69
Events Lister v2.03	9	2	2	5	23	113	14	26
PHP Calendars	3	1	0	2	19	35	19	30
PHPoll v0.97 beta	3	3	0	0	21	63	19	58
iCalendar v1.1	1	0	0	1	51	292	50	292
AWCM v2.1	47	1	0	46	176	2,634	129	2,438
AWCM v2.2 final	47	0	0	47	180	2,851	133	2,612
YaPiG v0.95	11	0	0	11	54	260	44	154

Project	Privileged	Vulnerable	FP	Guarded	Admin		Normal	
					Node	Edge	Node	Edge
SCARF	4	1	0	3	19	149	15	69
SCARF (patched)	4	0	0	4	19	149	15	69
Events Lister v2.03	9	2	2	5	23	113	14	26
PHP Calendars	3	1	0	2	19	35	19	30
PHPoll v0.97 beta	3	3	0	0	21	63	19	58
iCalendar v1.1	1	0	0	1	51	292	50	292
AWCM v2.1	47	1	0	46	176	2,634	129	2,438
AWCM v2.2 final	47	0	0	47	180	2,851	133	2,612
YaPiG v0.95	11	0	0	11	54	260	44	154

Project	Nodes			Context-Free Grammar		Coverage	Time(s)
	Entry	Active	Orphan	Variables	Productions		
SCARF	1	19	0	158	719	100.0%	6.02
SCARF (patched)	1	19	0	159	719	100.0%	6.01
Events Lister v2.03	4	23	5	100	2,083	100.0%	3.84
PHP Calendars	3	15	0	48	255	80.0%	5.09
PHPoll v0.97 beta	5	21	6	115	224	100.0%	4.26
iCalendar v1.1	2	52	2	811	4,774	90.4%	760.62
AWCM v2.1	17	208	22	410	422	79.3%	89.48
AWCM v2.2 final	16	209	14	451	484	79.9%	108.51
YaPiG v0.95	7	59	3	332	532	91.5%	208.38

Conclusion

- First role-based static analysis
  - Detects access control vulnerabilities
  - Requires minimal manual effort
- Per-role sitemaps
  - Inference of implicit access assumption
  - Forced browsing to detect unprotected access
- Effective and scalable technique