# Dive deep into solving web challenges

Presented by Mohammed Alharbi

# Agenda

Important tips

Common web vulnerabilities

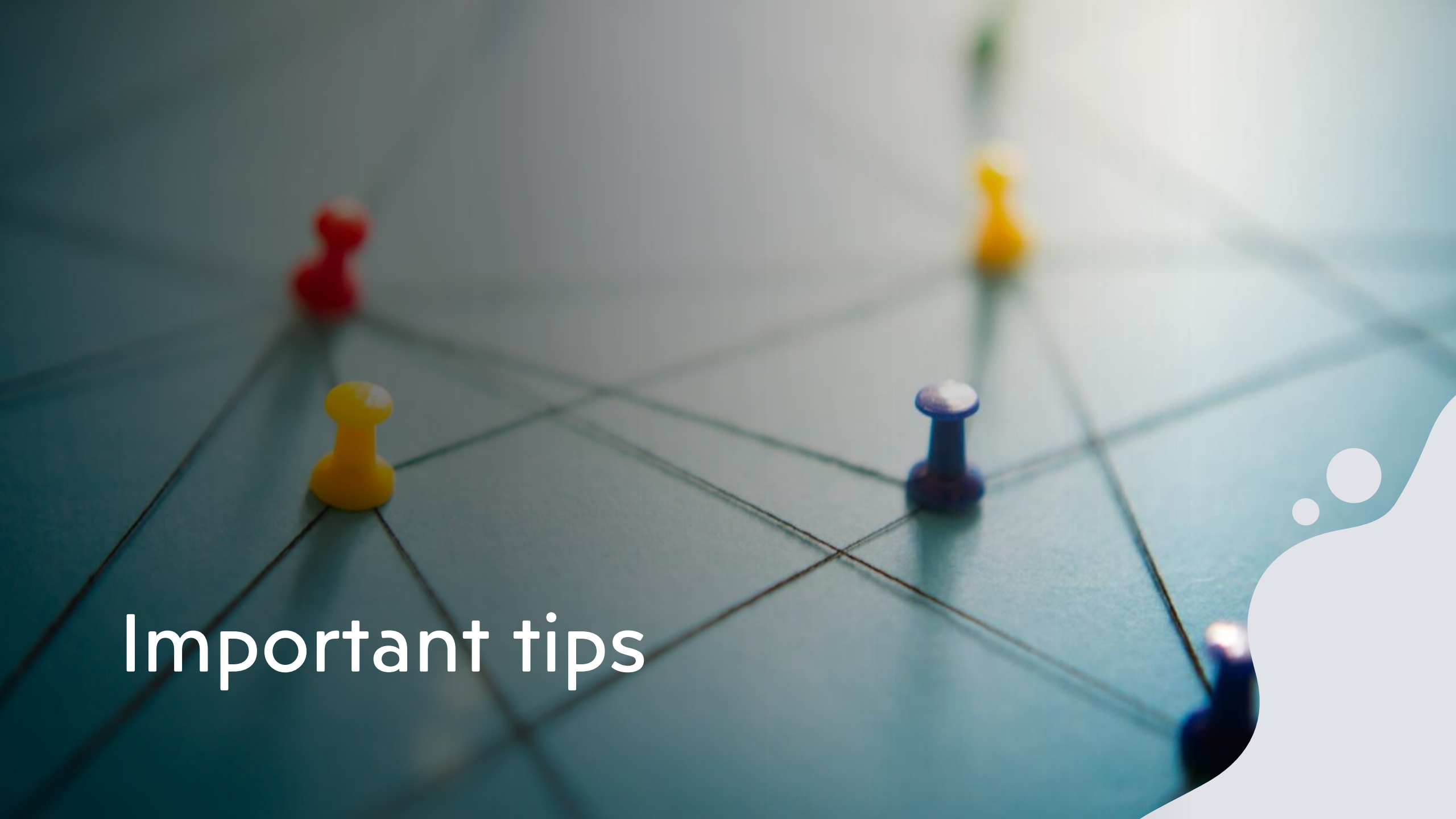
Common pitfalls in web languages

Prepare the workspace

Ready for warmup?

Let’s dive deeper

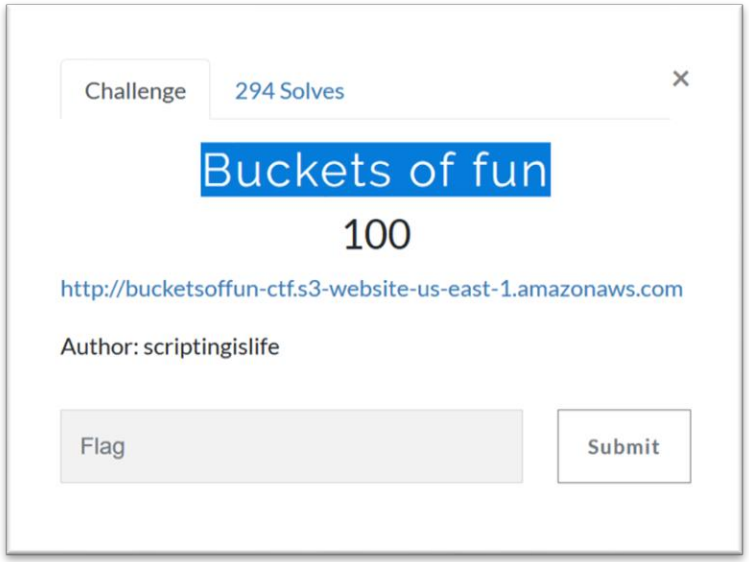
More vulnerabilities

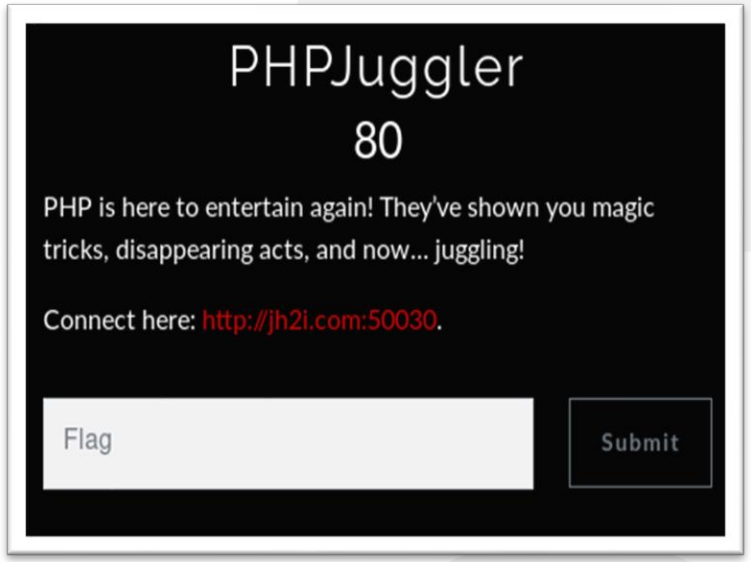


# Important tips

# Read first

Read the challenge's name and description



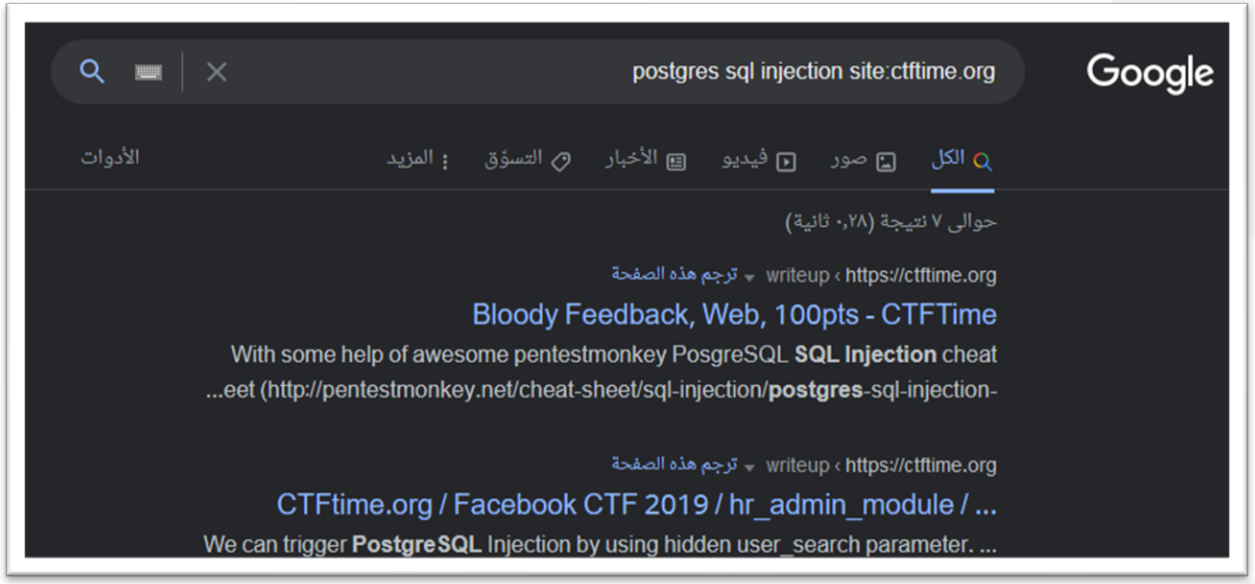



# Issues?

Check if the web application/framework exists in GitHub or somewhere, this will help you to find CVEs/issues quickly

# Google is your friend

Search about similar challenges in Google or Ctftime.org



# Be unique

Be unique and think like there is no box

# Information gathering

Collect information before solving the challenge

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory | | - | |
| heroes.xml | 02-Nov-2014 23:52 | 1.2K | |
| web.config.bak | 02-Nov-2014 23:52 | 7.4K | |
| wp-config.bak | 02-Nov-2014 23:52 | 1.5K | |

*Apache/2.2.8 (Ubuntu) DAV/2 mod_fastcgi/2.4.6 PHP/5.2.4-2ubuntu5 with Suhosin-Patch mod_ssl/2.2.8 OpenSSL/0.9.8g Server at 192.168.0.131 Port 80*

# Common web vulnerabilities

# SQL injection (SQLi)

Happened when untrusted user input passed to a SQL query, so the attacker can manipulate the query and execute malicious commands

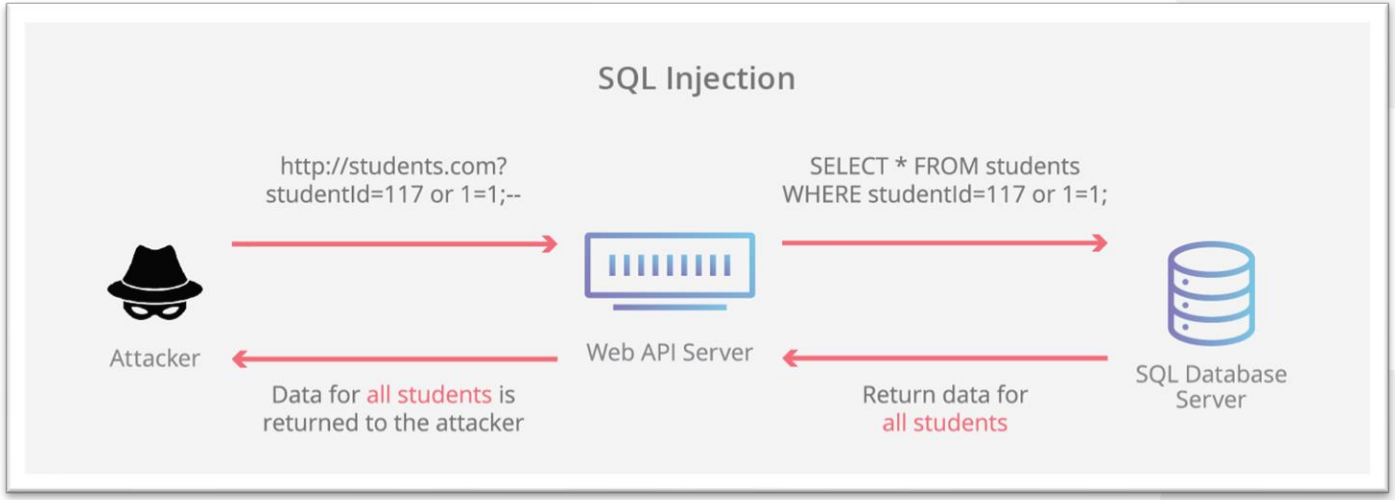



Photo by https://www.cloudflare.com/fr-fr/learning/security/threats/sql-injection/

# Cross site scripting (XSS)

Happened when untrusted user input passed to a any function can print or output text, so it give the chance to the attacker to embed malicious client-side scripts in the page

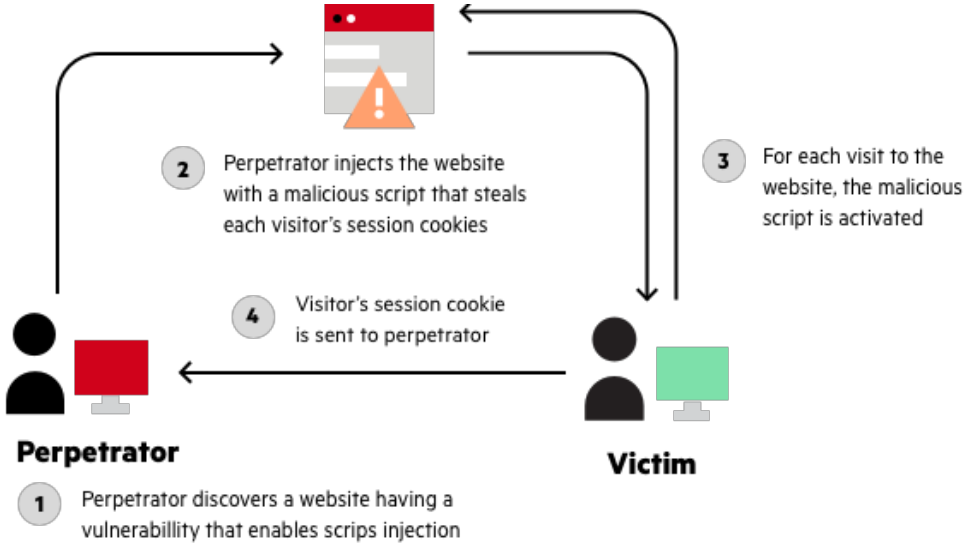



Photo by https://www.imperva.com/learn/application-security/reflected-xss-attacks/

# File inclusion (LFI or RFI)

Happened when untrusted user input passed to a any function can read file and evaluate it like include(), so the attacker can include malicious code and execute it or read a sensitive file

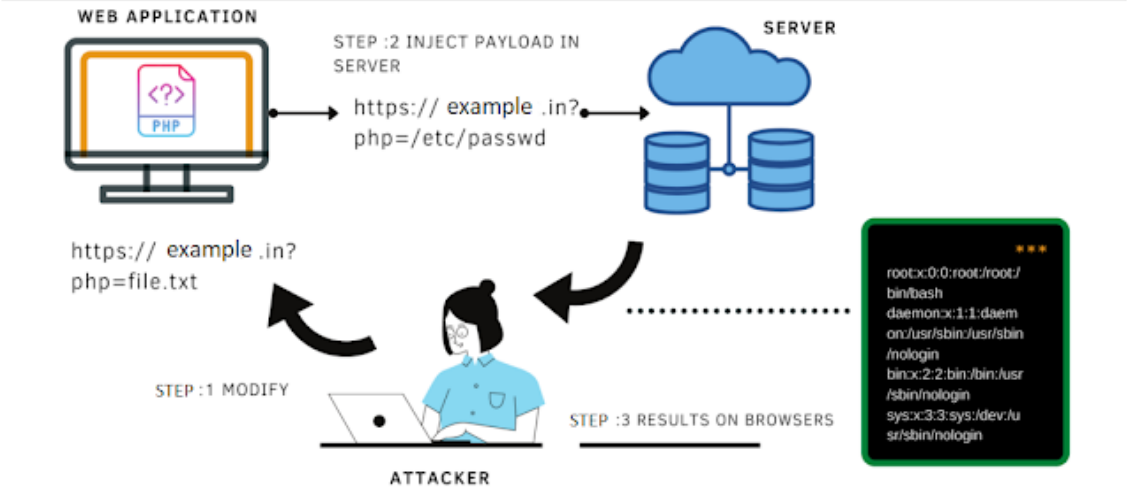



Photo by https://medium.com/@tanmay_deshpande/local-file-inclusion-lfi-attack-46485f294aef

# Path traversal

Happened when untrusted user input passed to any function can read or access file like file_get_contents(), so the attacker can read some sensitive files

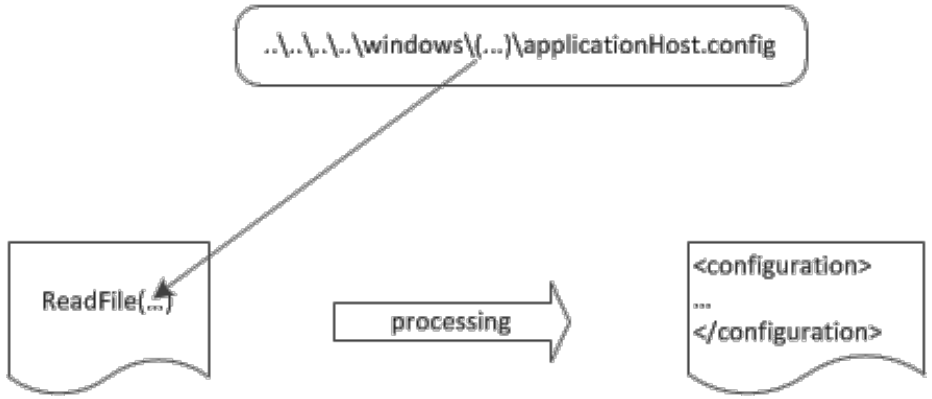



Photo by https://www.howtoasp.net/how-to-control-path-composition-to-protect-asp-net-web-application-from-directory-traversal-vulnerability-in-c/

# Code injection

Happened when untrusted user input passed to a any function evaluate a command like eval(), so the attacker to include malicious code and execute it

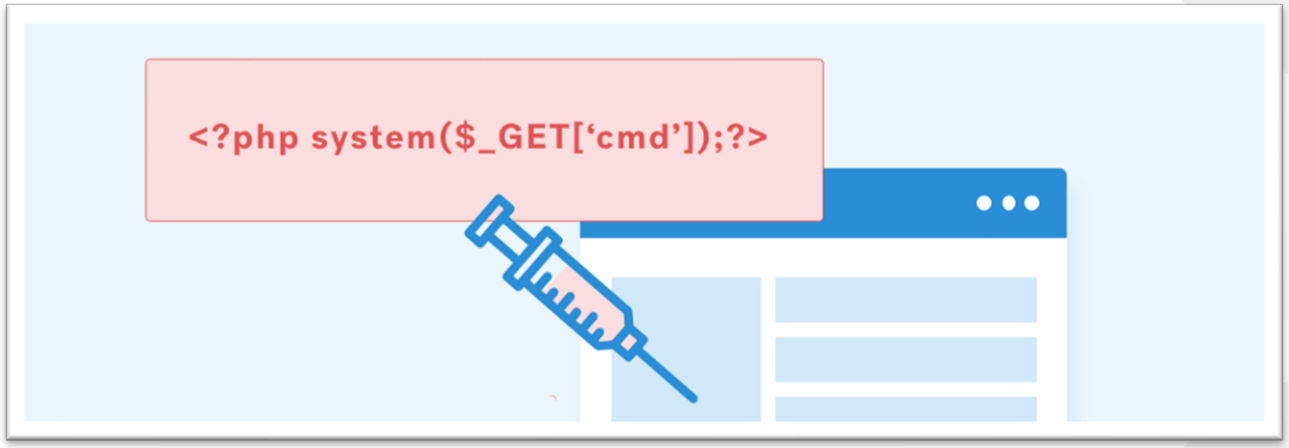



Photo by https://www.acunetix.com/websitesecurity/php-security-2/

# Command injection

Happened when untrusted user input passed to a any function execute command on the OS like exec(), so the attacker to include malicious commands and execute them

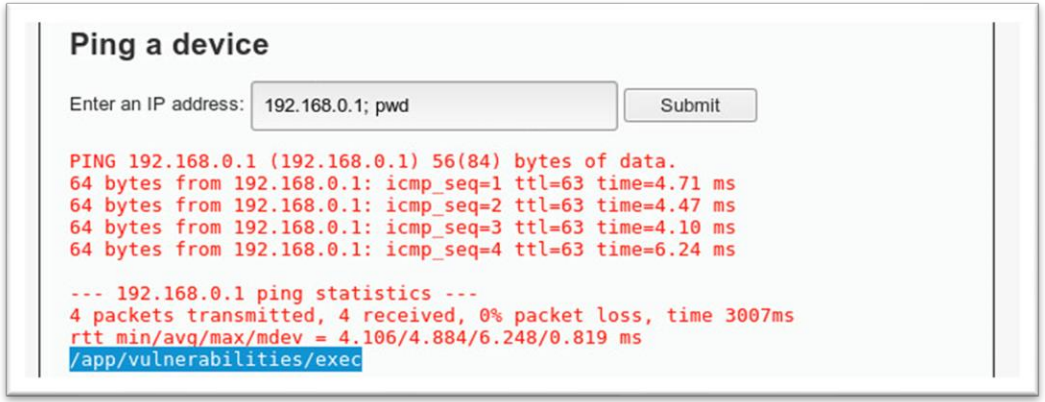



Photo by https://chris-young.net/2018/03/28/dvwa-command-injection/

# Unsafe deserialization

Happened when untrusted user input used to abuse the logic of an application like unserialize(), so the attacker can manipulate serialized objects to change the program's flow

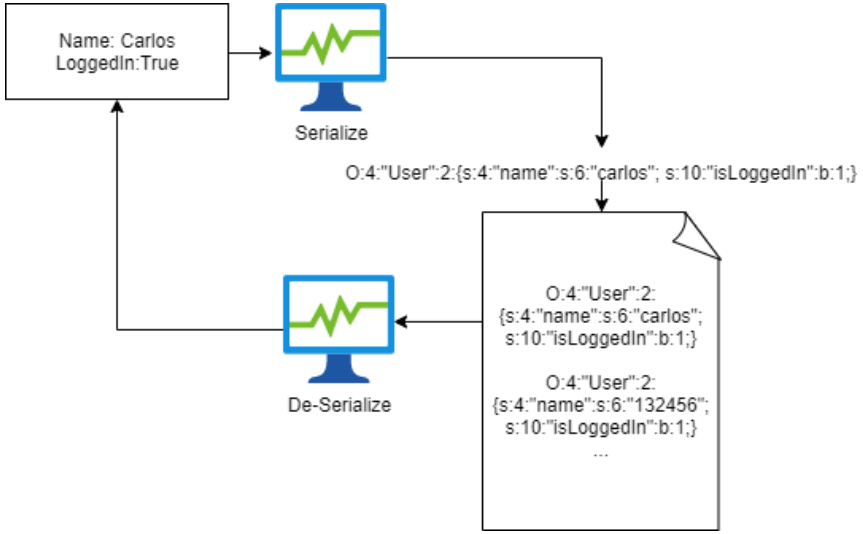



Photo by https://infosecwriteups.com/insecure-deserialization-its-super-hard-or-is-it-94d89e2847

# Template injection (SSTI)

Happened when untrusted user input passed to template syntax in application uses template engine, so the attacker execute some malicious code depends on the engine

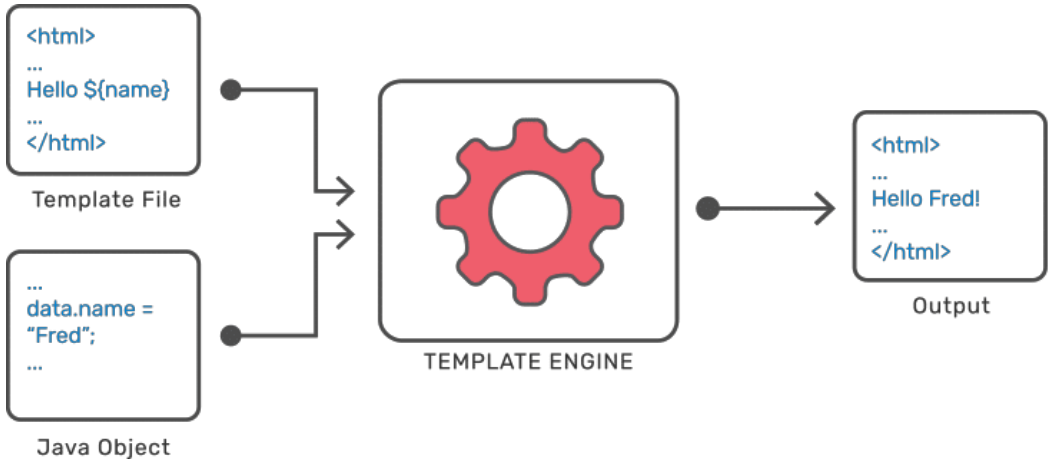



Photo by https://www.indusface.com/learning/server-side-template-injectionssti/

# Server-side request foreign (SSRF)

Happened when untrusted user input passed to any function can open stream resource like file_get_contents(), so the attacker access some internal resources

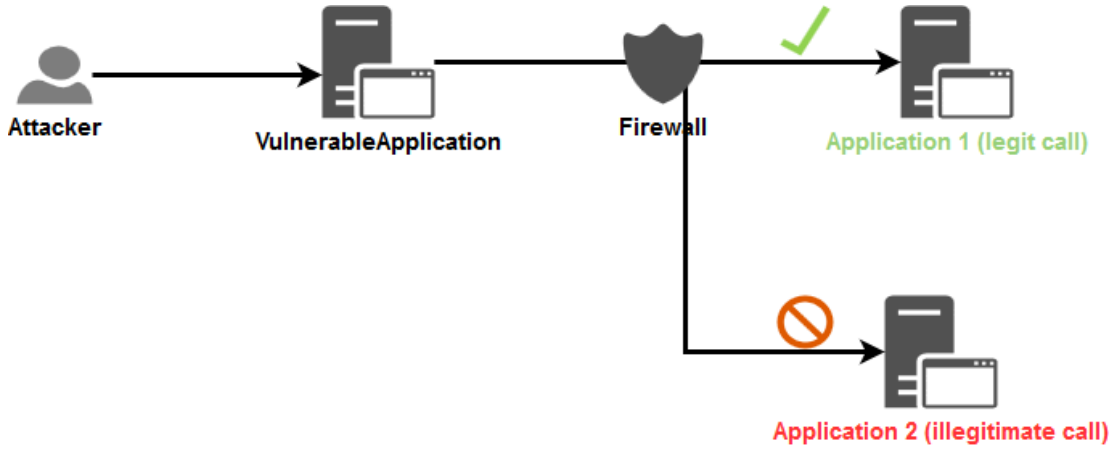
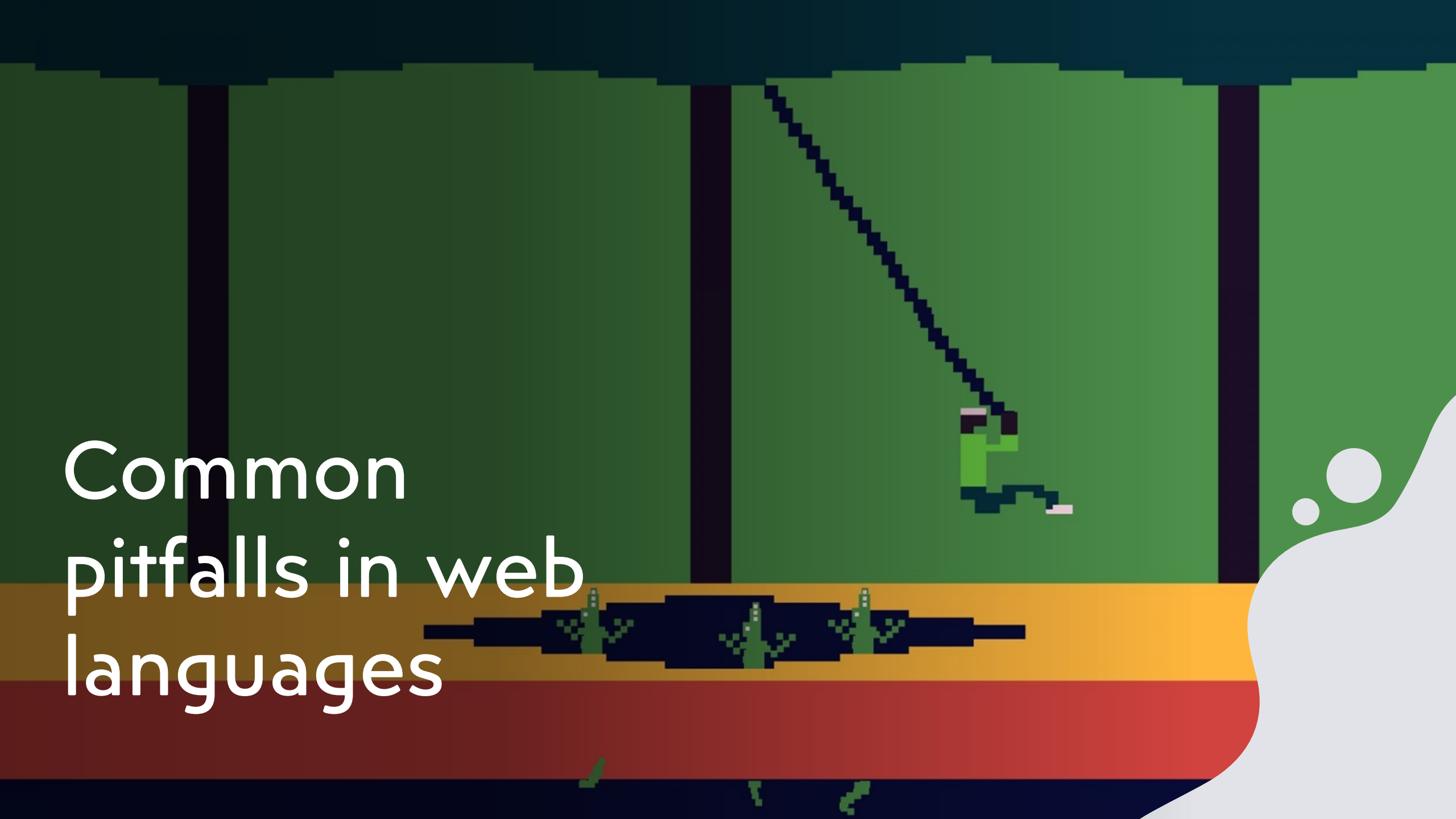



Photo by https://cheatsheetseries.owasp.org/



# Common pitfalls in web languages

# PHP pitfalls

Type juggling

Phar deserialization

# JavaScript pitfalls

Type juggling

Prototype pollution

# Java pitfalls

JNDI injection

EL injection

# Python pitfalls

String format

Pickle deserialization

# Ready for warmup?

# Challenge 1

var > www > html > challenges > 1 > index.php

```php
<?php

$file = $_GET['file'] ?? null;

if(!empty($file)){
 @include($file);
} else {
    die("No paramater called file!");
}
```

# RECALL File inclusion (LFI or RFI)

Happened when untrusted user input passed to a any function can read file and evaluate it like include(), so the attacker can include malicious code and execute it or read a sensitive file

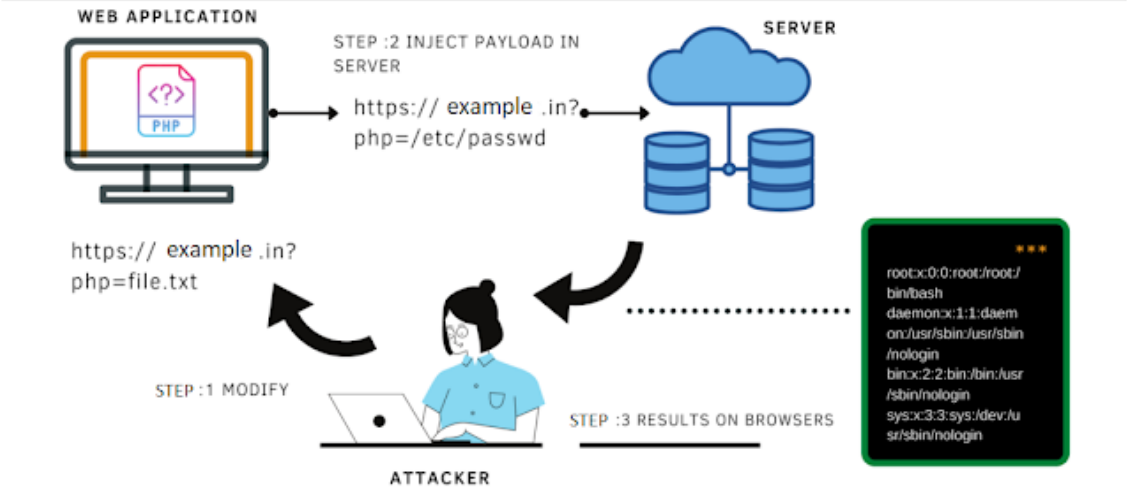



Photo by https://medium.com/@tanmay_deshpande/local-file-inclusion-lfi-attack-46485f294aef

# Challenge 2

var > www > html > challenges > 2 > index.php

```php
<?php

$url = $_GET['url'] ?? null;

if(!empty($url)){
 echo @file_get_contents($url);
} else {
 die("No paramater called url!");
}
```

# RECALL Server-side request foreign (SSRF)

Happened when untrusted user input passed to any function can open stream resource like file_get_contents(), so the attacker access some internal resources

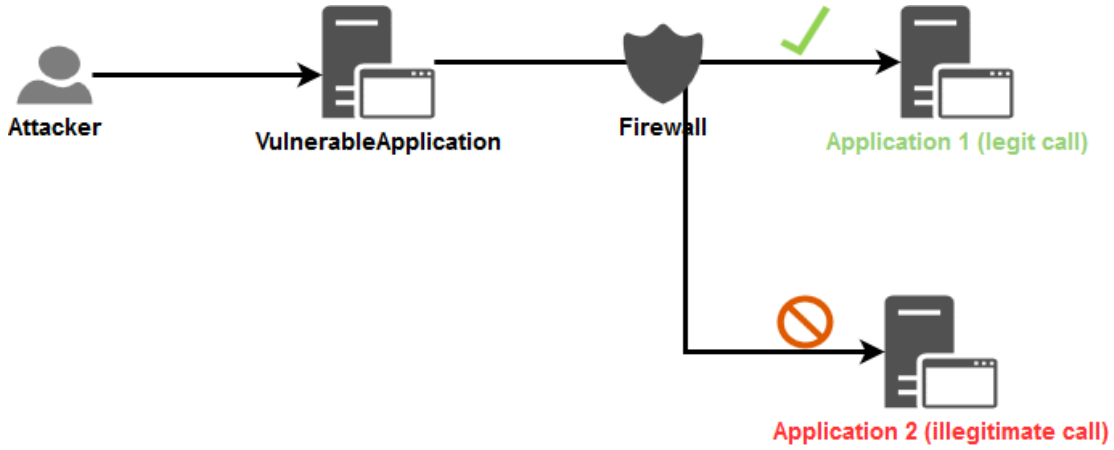



Photo by https://cheatsheetseries.owasp.org/

# Challenge 3

var > www > html > challenges > 3 > index.php

```php
<?php

$dir = $_GET['dir'] ?? null;

if(!empty($dir)){
 system("ls -la $dir");
} else {
 die("No paramater called dir!");
}

```

# RECALL Command injection

Happened when untrusted user input passed to a any function execute command on the OS like exec(), so the attacker to include malicious commands and execute them

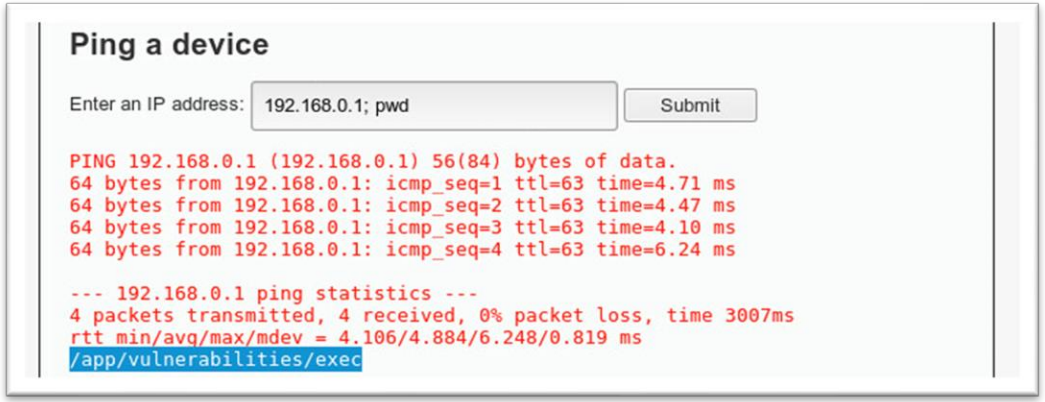



Photo by https://chris-young.net/2018/03/28/dvwa-command-injection/

# Let’s dive deeper

# Challenge 4

```php
<?php

include_once("flag.php");

$pass = 0x1337;
$input = $_GET[1337] ?? null;

if(empty($input)){
    highlight_file(__FILE__);
    die();
}

if($input == $pass){
    echo $flag;
} else {
    echo "Wrong!";
}
```

# Run the code

If you find some difficulties in reading and understanding a part of code, then run it in a sandbox environment, this will help you to get the output quickly

```php
<?php
$pass = 0x1337;
echo $pass;
```

# Challenge 5

```php
<?php

include_once("flag.php");

$pass = "password_is_secret";
$input = $_GET["password"] ?? null;

if(empty($input)){
    highlight_file(__FILE__);
    die();
}

$input = str_replace("password", "", $input);
$input = str_replace("secret", "", $input);


if($input === $pass){
    echo $flag;
} else {
    echo "Wrong!";
}
```

# Bypass replace function

Replace function will replace the search string with the replacement string, so it will search first then replace, so you need to try these solutions:

- Change the case of letters -> iAu
- Write the string inside string -> iaIAUu
- Is it replacing another string? -> iaANOTHERu

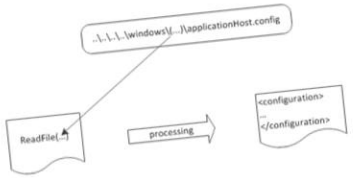
# Challenge 6

```php
<?php

$input = $_GET["file"] ?? null;

if(empty($input)){
    highlight_file(__FILE__);
    die();
}

$input = str_replace("../", "", $input);

$content = file_get_contents("files/" . $input) ?? null;

if(empty($content)){
    die("File not found!");
}

echo $content;
```

# RECALL

## Path traversal

Happened when untrusted user input passed to any function can read or access file like file_get_contents(), so the attacker can read some sensitive files




&

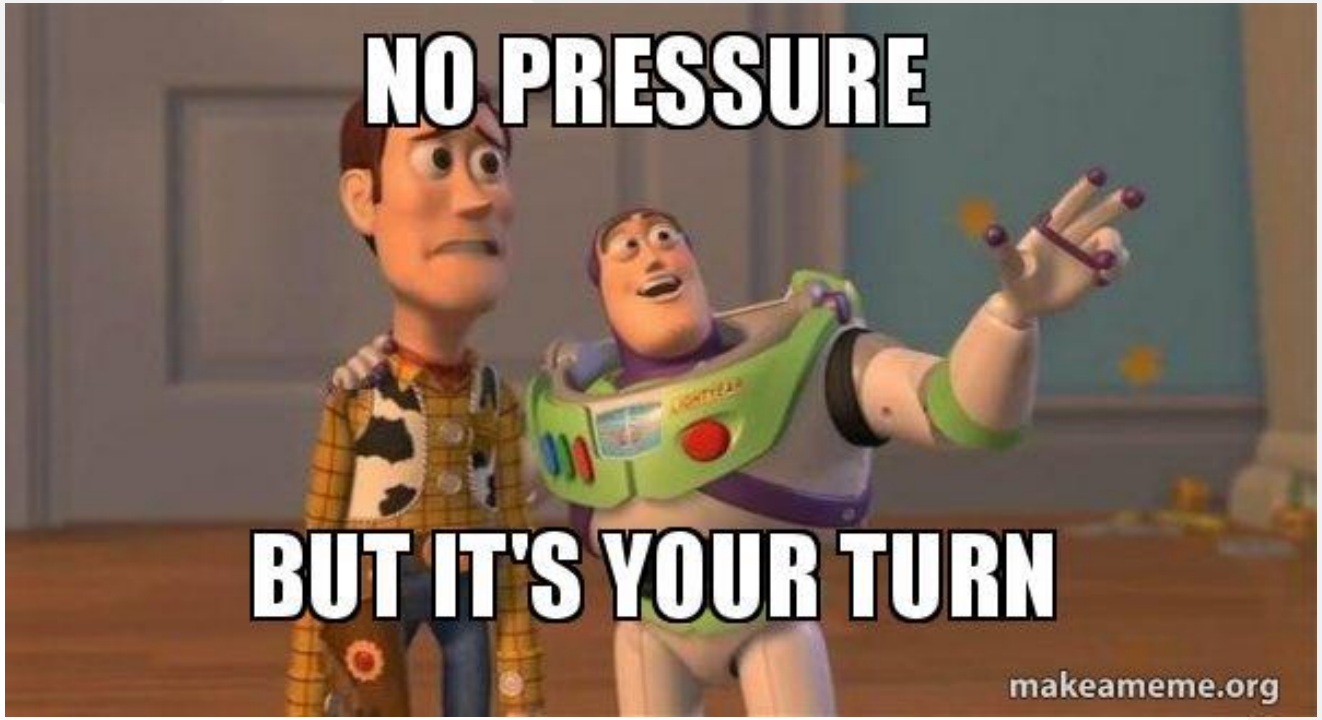
## Bypass replace function

Replace function will replace the search string with the replacement string, so it will search first then replace, so you need to try these solutions:

- Change the case of letters -> iAu
- Write the string inside string -> iaIAUu
- Is it replacing another string? -> iaANOTHERu

# Challenge 7

```php
if(empty($input)){
    highlight_file(__FILE__);
    die();
}

while(strpos($input, "../") !== false){
    $input = str_replace('../', "", $input);
}

while(strpos($input, "..\\") !== false){
    $input = str_replace('..\\', "", $input);
}

$content = file_get_contents("files/" . $input) ?? null;

if(empty($content)){
    die("File not found!");
```


NO PRESSURE
BUT IT'S YOUR TURN
makeameme.org

# More vulnerabilities

# Prototype pollution

Prototype pollution is a vulnerability that allows attackers to exploit the rules of the JavaScript language, by injecting properties into existing JavaScript construct prototypes

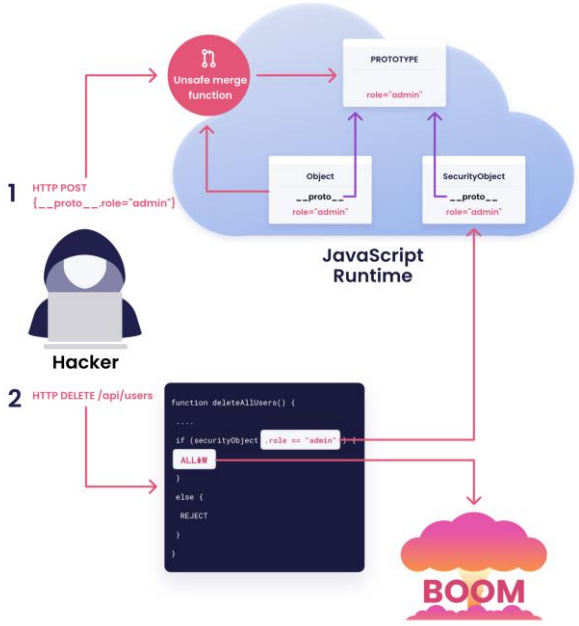



Photo by https://learn.snyk.io/lessons/prototype-pollution/javascript/

# Dependency confusion

Dependency confusion attack occurs when a software installer script is tricked into pulling a malicious code file from a public repository instead of the internal one

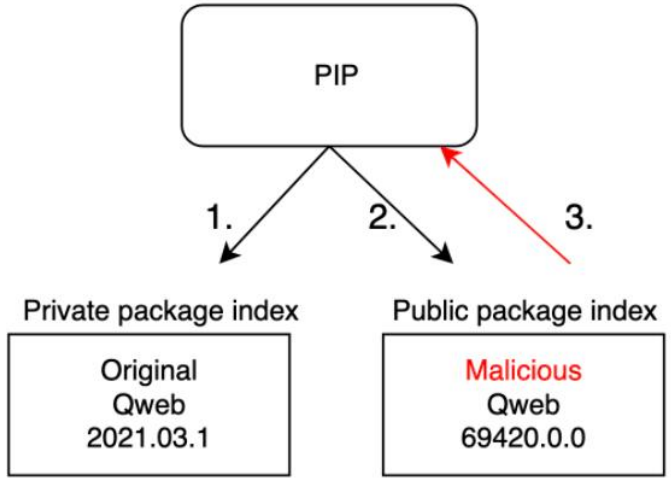



Photo by https://dhiyaneshgeek.github.io/web/security/2021/09/04/dependency-confusion/

# Format string in python

Happened when untrusted user input passed to format function, so the attacker can inject some malicious code and leak some sensitive data

```
def render(self, templateHTML, title, text):
    return (templateHTML.format(self=self, title=title, text=text))
```

# THANK YOU