



# WEB SECURITY

Cristian Spata  
Ignazio Leonardo Calogero Sperandeo

# DI COSA PARLEREMO ??

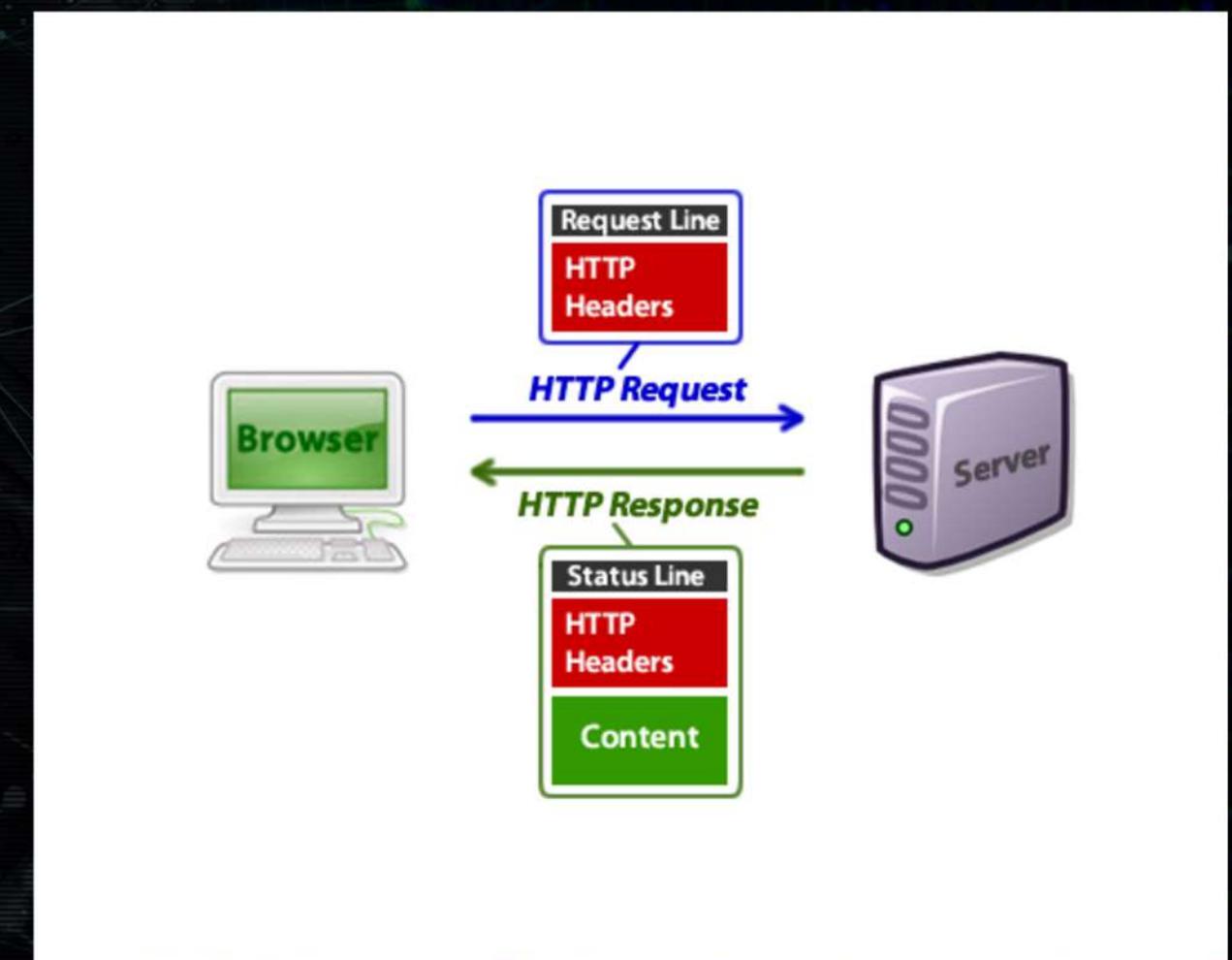


- HTTP (cenno sul funzionamento)
- Cookie e Sessioni - Come le gestisce PHP
- UNION in SQL
- FULL OUTER JOIN in MySQL
- Vulnerabilità SQL Injection (SQLi)
- Salvataggio sicuro delle password nei DB
- Vulnerabilità Cross-site scripting (XSS)
- Casi reali di attacchi

# HTTP PROTOCOL

HTTP (HyperText Transfer Protocol) è il protocollo che regola la comunicazione tra client (come i browser) e server web.

- È un protocollo stateless, cioè non mantiene lo stato tra una richiesta e l'altra: ogni richiesta è trattata in modo indipendente. Tuttavia, è possibile mantenere lo stato usando i Cookie.
- HTTP si basa su un meccanismo request/response.
- I codici di stato (es. 200, 404, 500) indicano se la richiesta è andata a buon fine o se si è verificato un errore.
- HTTP può essere usato con HTTPS (HTTP OVER TLS), la versione sicura che aggiunge crittografia tramite TLS.



## HTTP Request

```
Method      URL      Protocol Version  
GET /index.html HTTP/1.1  
Host: www.example.com  
User-Agent: Mozilla/5.0  
Accept: text/html, */*  
Accept-Language: en-us  
Accept-Charset: ISO-8859-1,utf-8  
Connection: keep-alive  
blank line
```

Headers {

Body (optional) {

# HTTP REQUEST

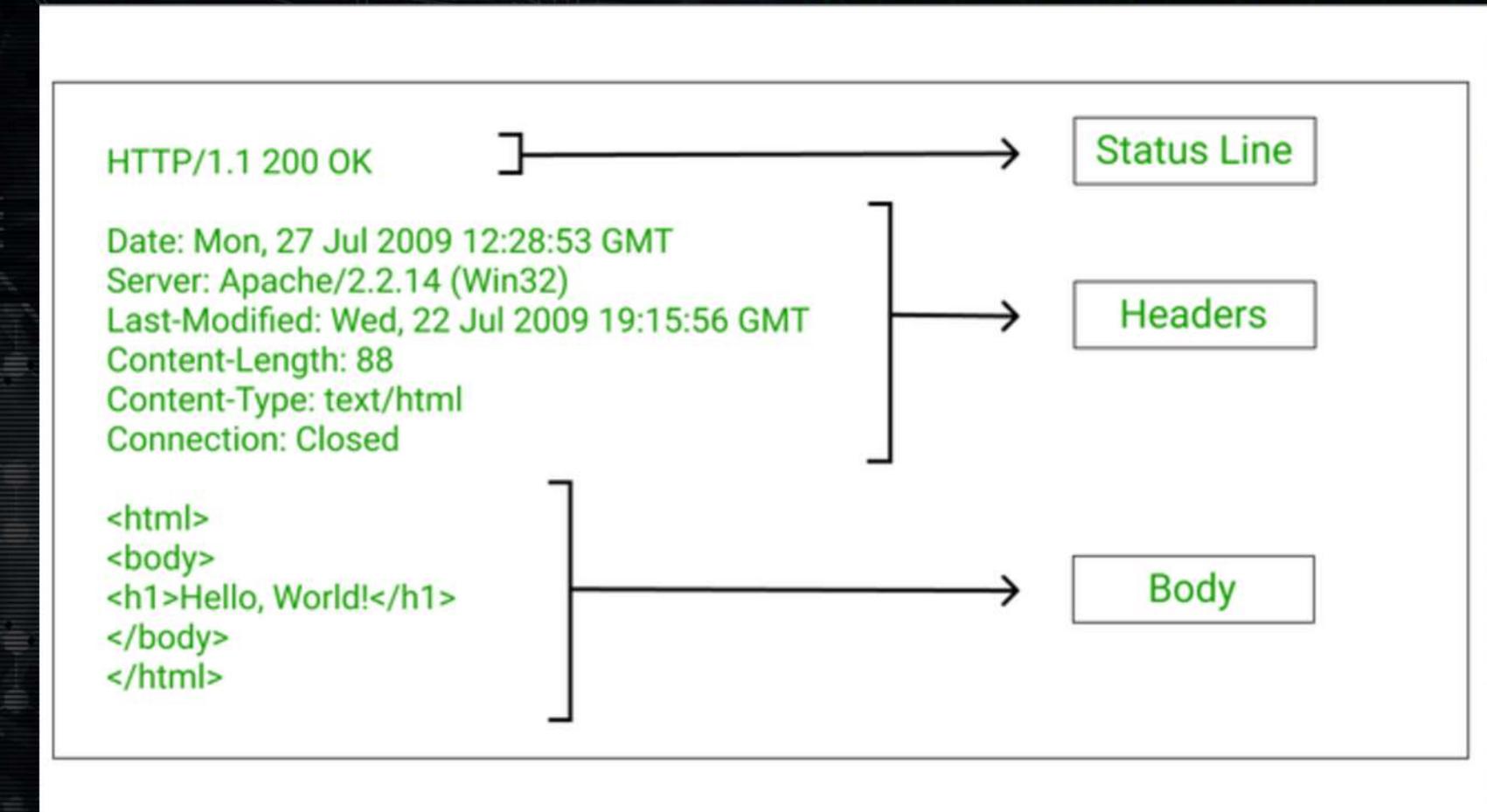
La richiesta HTTP è composta da 3 componenti:

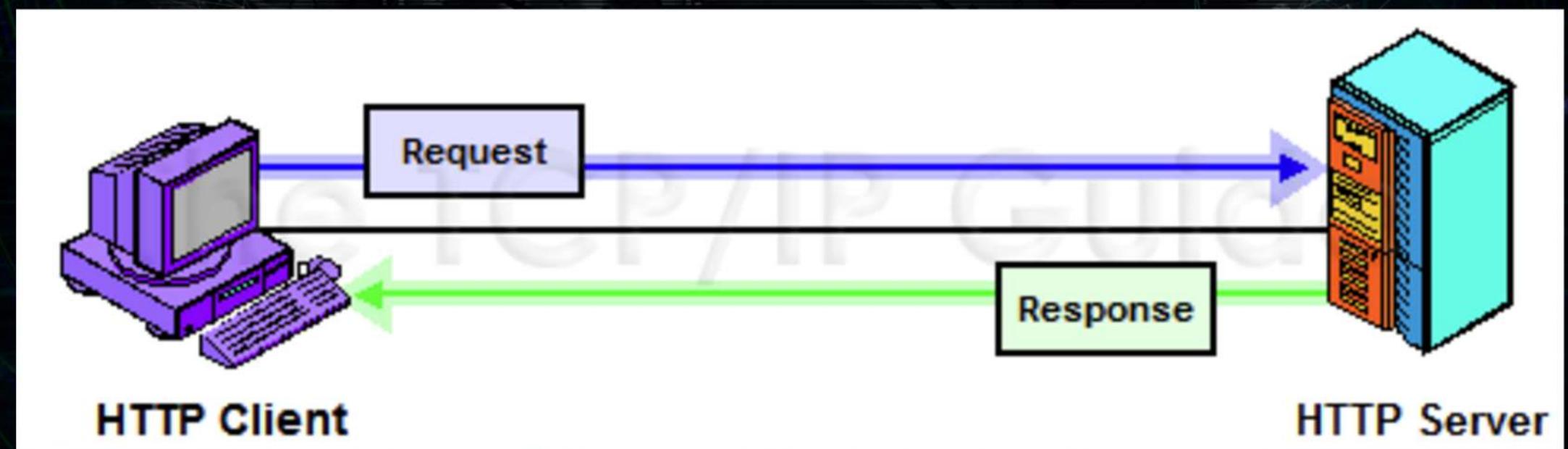
- **Request line:** Method, Path, Version
- **Headers:** Informazioni che arricchiscono la richiesta
- **Payload:** Dati che si inviano al server

# HTTP RESPONSE

La risposta HTTP è composta da 3 componenti:

- **Status Line:** Version, Status Code, Status Message
- **Headers:** Informazioni riguardante la risposta.
- **Payload:** Dati che si inviano al client





# HTTP METHODS

- **GET**
- **POST**
- **PUT**
- **HEAD**
- **DELETE**
- **OPTIONS**

```
1 GET /index.html HTTP/1.1
2 Host: rootgram.dev
3 Accept-Language: it-IT;q=0.9,en-us;q=0.7
4 Accept-Encoding: gzip, deflate, br
5 Connection: Keep-Alive
6
```

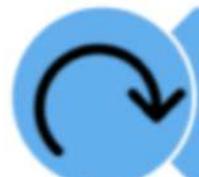
```
1 POST /create_user.php HTTP/1.1
2 Host: rootgram.dev
3 Accept-Language: it-IT;q=0.9,en-us;q=0.7
4 Accept-Encoding: gzip, deflate, br
5 Connection: Keep-Alive
6 Content-Type: application/json
7
8 {'ID':5, 'Nome':'Marco', 'Cognome':'Rossi'}
```



Informational 1xx (100 – 199)



Success 2xx (200 – 299)



Redirection 3xx (300 – 399)



Client Error 4xx (400 – 499)



Server Error 5xx (500 – 599)



# COOKIES

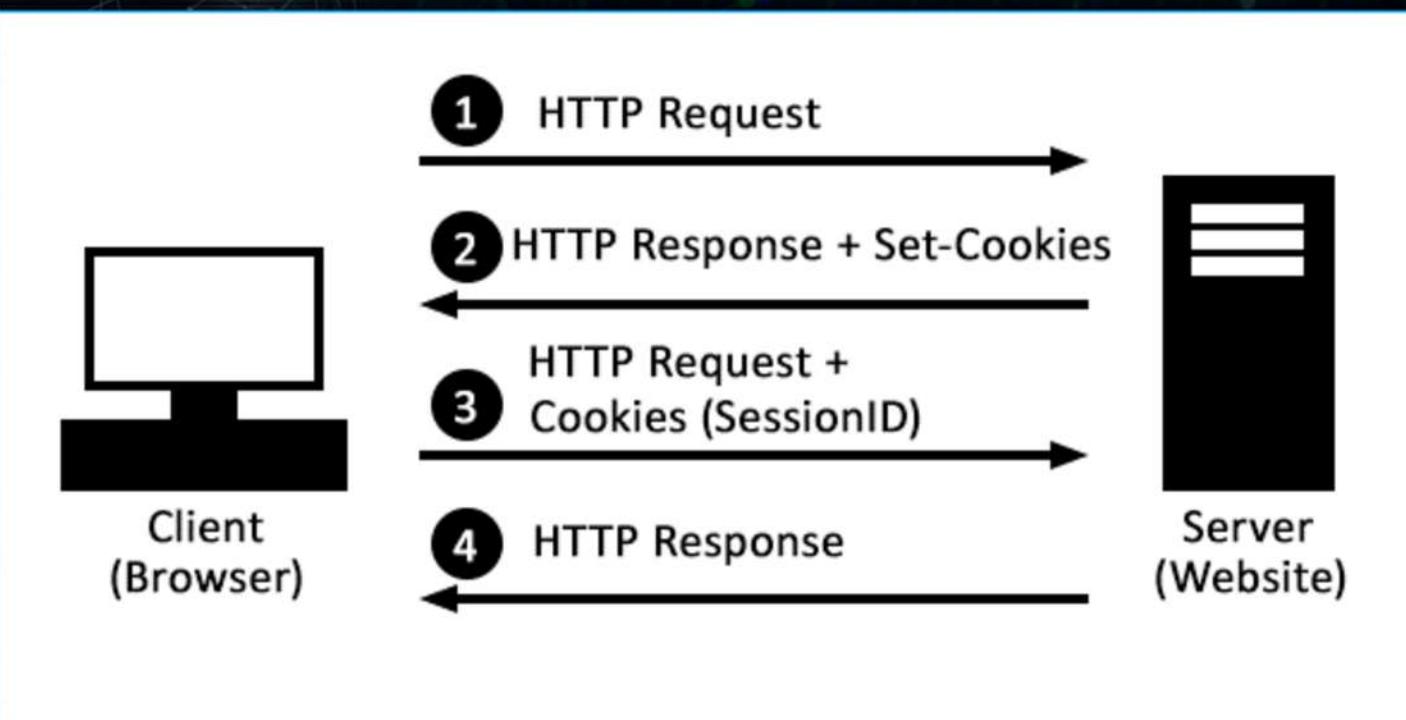
Poiché HTTP è stateless, ogni richiesta del browser è isolata: il server non sa chi sei o cosa hai fatto prima.

- Per "ricordare" l'utente tra una richiesta e l'altra, il server invia al browser un cookie: una piccola stringa di testo contenente informazioni (es. un ID univoco).
- Il browser memorizza il cookie e lo restituisce automaticamente nelle richieste successive.
- In questo modo, il server può riconoscere l'utente e mantenere una sessione attiva (es. dopo il login).

I Cookies sono delle coppie chiave-valore gestiti automaticamente dal browser. I cookie possono essere utilizzati principalmente per:

- Sessioni
- Preferenze utente

Quando vengono utilizzati per le Sessioni è molto importante che il loro valore non venga reso noto.





# COOKIES - PARAMETRI

Attraverso le HTTP RESPONSE possiamo definire diversi parametri sui cookie, come:

- Durata
- Dominio
- Path

Poichè cookie sono gestiti dal browser è molto importante che il server non si fidi ciecamente del loro valore.



# COOKIES - VULNERABILITÀ

Poichè cookie sono gestiti dal browser è molto importante che il server non si fidi ciecamente del loro valore. Chiunque navighi sul sito può visualizzarli e modificarli tramite strumenti come le DevTools. I rischi possono essere:

- Cookie Poisoning: manipolazione del loro contenuto

In certi casi il cookie poisoning può contenere anche attacchi del tipo SQLi o XSS.



# COOKIES - PREVENZIONE

Alcuni proprietà dei cookie sono:

- I cookie impostati su un determinato dominio sono inviati solo a quel dominio
- Un dominio non può impostare cookie per altri domini.

Attraverso l'header Set-Cookie possiamo usare dei meccanismi di difesa:

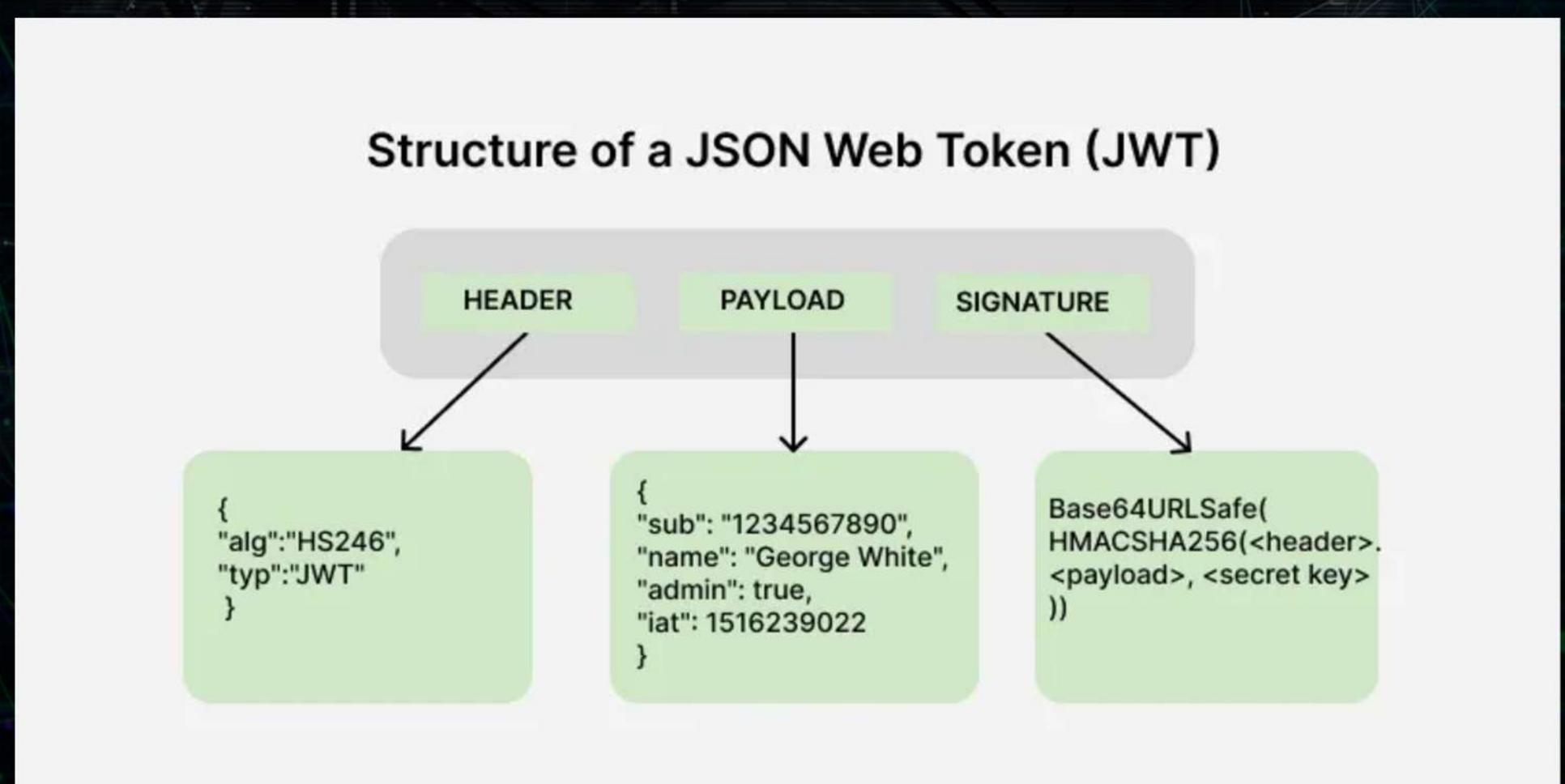
- Secure: il cookie viene inviato solo con HTTPS.
- HttpOnly: il cookie non è accessibile con JS.
- SameSite: Limita l'invio del cookie per richieste cross-site.
  - Strict
  - Lax
  - None



```
1 HTTP/1.1 200 OK
2 Content-Type: text/html; charset=UTF-8
3 Set-Cookie: sessionid=abc123;
4   HttpOnly;
5   Secure;
6   SameSite=Strict;
7   Path=/;
8   Domain=rootgram.dev
9
```



# COOKIES - PREVENZIONE





# COOKIES - PHP

## Reserved Characters

Characters	;	=	\$	,	<	>	^	'	\
Encoded form	%3A	%2F	%23	%3F	%24	%40	%25	%2B	%20
Characters	:	/	#	?	&	@	%	+	<space>
Encoded form	%3B	%3D	%26	%2C	%3C	%3E	%5E	%60	%5C
Characters	[	]	{	}		"			
Encoded form	%5B	%5D	%7B	%7D	%7C	%22			





# COOKIES - PHP

In PHP, i cookie si impostano attraverso la funzione `setcookie()`.

- La funzione `setcookie()` invia un'intestazione HTTP (Set-Cookie) al browser, che provvede a salvare il cookie localmente.
- Il valore del cookie viene codificato secondo l'URL encoding (percent encoding) per garantire che caratteri speciali non interferiscano con la trasmissione via HTTP.
- PHP memorizza i cookie ricevuti dal client nella superglobale `$_COOKIE`, che è disponibile per tutta la durata dello script.



# COOKIES - PHP

Un altro modo per impostare i cookie in PHP è attraverso la funzione `setrawcookie()`.

- A differenza di `setcookie()`, `setrawcookie()` non applica la Percent Encoding al valore del cookie.
- Questo significa che si possono impostare valori che contengono caratteri speciali (come spazi, simboli, ecc.) senza che vengano convertiti in notazione percentuale (%20, %3D, ecc.).
- Tuttavia, l'uso di caratteri non validi o non supportati può causare problemi nei browser, quindi bisogna usarla con attenzione.
- Come `setcookie()`, anche `setrawcookie()` invia l'intestazione HTTP `Set-Cookie`, ed è soggetta alle stesse regole deve essere chiamata prima di qualsiasi output HTML.



# COOKIES - PHP



```
1 setcookie(  
2     string $name,  
3     string $value = "",  
4     int $expires_or_options = 0,  
5     string $path = "",  
6     string $domain = "",  
7     bool $secure = false,  
8     bool $httponly = false  
9 ): bool
```



```
1 setrawcookie(  
2     string $name,  
3     string $value = "",  
4     int $expires_or_options = 0,  
5     string $path = "",  
6     string $domain = "",  
7     bool $secure = false,  
8     bool $httponly = false  
9 ): bool
```





# COOKIES - PHP - IMPOSTAZIONE

```
1 <?php
2   if (!isset($_COOKIE['username'])) {
3     setcookie("username", "Mario Rossi", time() + 3600 * 24 * 30);
4   }
5   if (!isset($_COOKIE['lingua'])) {
6     setcookie("lingua", "it", time() + 3600 * 24 * 30);
7   }
8   if (!isset($_COOKIE['tema'])) {
9     setcookie("tema", "scuro", time() + 3600 * 24 * 30);
10  }
11 ?>
```



# COOKIES - PHP - ELIMINAZIONE



```
1 <?php  
2 // Modo 1:  
3 setcookie('lingua', '', time() - 3600);  
4 // Modo 2:  
5 setcookie('lingua', '', 1);  
6 ?>
```



# PRACTICE TIME

**Tempo stimato:** 15 minuti

**Consegna:** Realizzare una web application consenta l'impostazione di un cookie che rappresenti la preferenza del tema della pagina di un utente. In particolare si vuole un form che permetta la scelta di due opzioni (Light, Dark). Il cookie deve avere la durata di un anno e deve essere visualizzato:

“Preferenza scelta: {theme}”

Nota:

Al fine di svolgere l'esercizio non è necessario l'uso di js.



# SESSION

Spesso è necessario riconoscere un utente specifico su un sito web, ad esempio, per mostrare una pagina personalizzata o offrire funzionalità riservate. Questo avviene tramite un processo di **autenticazione**, solitamente con username e password.

- La sessione inizia quando un utente naviga su un sito web.
- Lo username identifica l'utente, mentre la password conferma che è davvero chi dice di essere.
- Dopo l'autenticazione, il server memorizza questa informazione nella sessione dell'utente
- Il browser viene associato a quella sessione grazie a un cookie (di solito contenente un ID di sessione).

```
<?php
session_start(); // Inizia la sessione

// Simuliamo un login
$_SESSION["username"] = "Alice"; // Salva il nome
utente

echo "Login effettuato!";
?>

<?php
session_start(); // Inizia la sessione

if (isset($_SESSION["username"])) {
    echo "Bentornata, " . $_SESSION["username"];
} else {
    echo "Accesso non autorizzato.";
}
?>
```



# SESSION

HTTP è un protocollo stateless, quindi il web server non può sapere che nella precedente richiesta è stato effettuato un login.

Idea:

- Il server chiede al client di mandare un identificativo allegato ad ogni richiesta.
- Ogni identificativo è univoco e rappresenta un client.
- Il client “allega” il proprio cookie con l’header Cookie.
- Il server assegna il cookie mediante l’header Set-Cookie.
- L’identificativo prende il nome di cookie di sessione



# SESSION





# SESSION

A seguito dell'autenticazione vengono spesso concesse determinate autorizzazioni all'utente, mentre l'autenticazione identifica l'utente, l'autorizzazione definisce cosa è permesso fare all'interno del sistema, assegnando dei ruoli come

- admin
- moderatore
- utente



# SESSION - PHP

- Le sessioni vengono inizializzate o recuperate con `session_start()`
- Tutte le variabili legate ad una sessione sono memorizzate nell'array `$_SESSION`
- Le sessioni vengono "distrutte" attraverso `session_destroy()`
- Tutte le variabili di una sessione vengono rimosse attraverso `session_unset()`



# SESSION - PHP

Una sessione rimane attiva fin quando:

- Il browser è aperto
- Non viene “distrutta”
- Non scade secondo i parametri di configurazione

In PHP ci sono due modi per gestire l'attività di una sessione, attraverso il `php.ini` o mediante script.

La sessione scade dopo un periodo di inattività, configurabile mediante `session.gc_maxlifetime`



# SESSION - PHP



```
1 // Tramite file di configurazione: php.ini  
2 session.gc_maxlifetime = 1800  
3  
4 // Tramite script:  
5 ini_set('session.gc_maxlifetime', 1800);  
6 session_start();  
7
```



```
1 // Tramite file di configurazione: php.ini  
2 session.cookie_lifetime = 1800  
3  
4  
5 // Tramite script:  
6 session_set_cookie_params([  
7     'lifetime' => 1800,  
8     'path' => '/',  
9     'secure' => true,  
10    'httponly' => true  
11]);  
12 session_start();  
13
```



# SESSION - PHP

```
1 <?php
2     session_start();
3
4     // Simulazione di un login
5     if (isset($_POST['username']) && $_POST['password'] === 'password123') {
6         $_SESSION['username'] = $_POST['username'];
7         $_SESSION['logged_in'] = true;
8         header('Location: dashboard.php');
9         exit;
10    } else {
11        echo 'Credenziali non valide.';
12    }
13 ?>
14
```

login.php



# SESSION - PHP

```
1 <?php
2     session_start(); // Recupero il cookie di sessione
3
4     if (!isset($_SESSION['logged_in'])) {
5         header('Location: login.php');
6         exit;
7     }
8
9     echo 'Benvenuto, ' . htmlspecialchars($_SESSION['username']) . '!';
10?
11
```

dashboard.php



# SESSION - PHP



```
1 <?php
2   session_start();      // Recupero la sessione
3
4   session_unset();      // Rimuove tutte le variabili di sessione
5   session_destroy();    // Distruggo la sessione
6
7   echo 'Sei stato disconnesso con successo.';
8 ?>
9
```

logout.php



# SESSION - PHP



```
1 HTTP/1.1 302 Found
2
3 Date: Mon, 06 May 2025 17:00:00 GMT
4 Server: Apache
5 Set-Cookie: PHPSESSID=abc123xyz456; path=/; HttpOnly
6 Location: dashboard.php
7 Content-Length: 0
8 Connection: close
9 Content-Type: text/html; charset=UTF-8
```

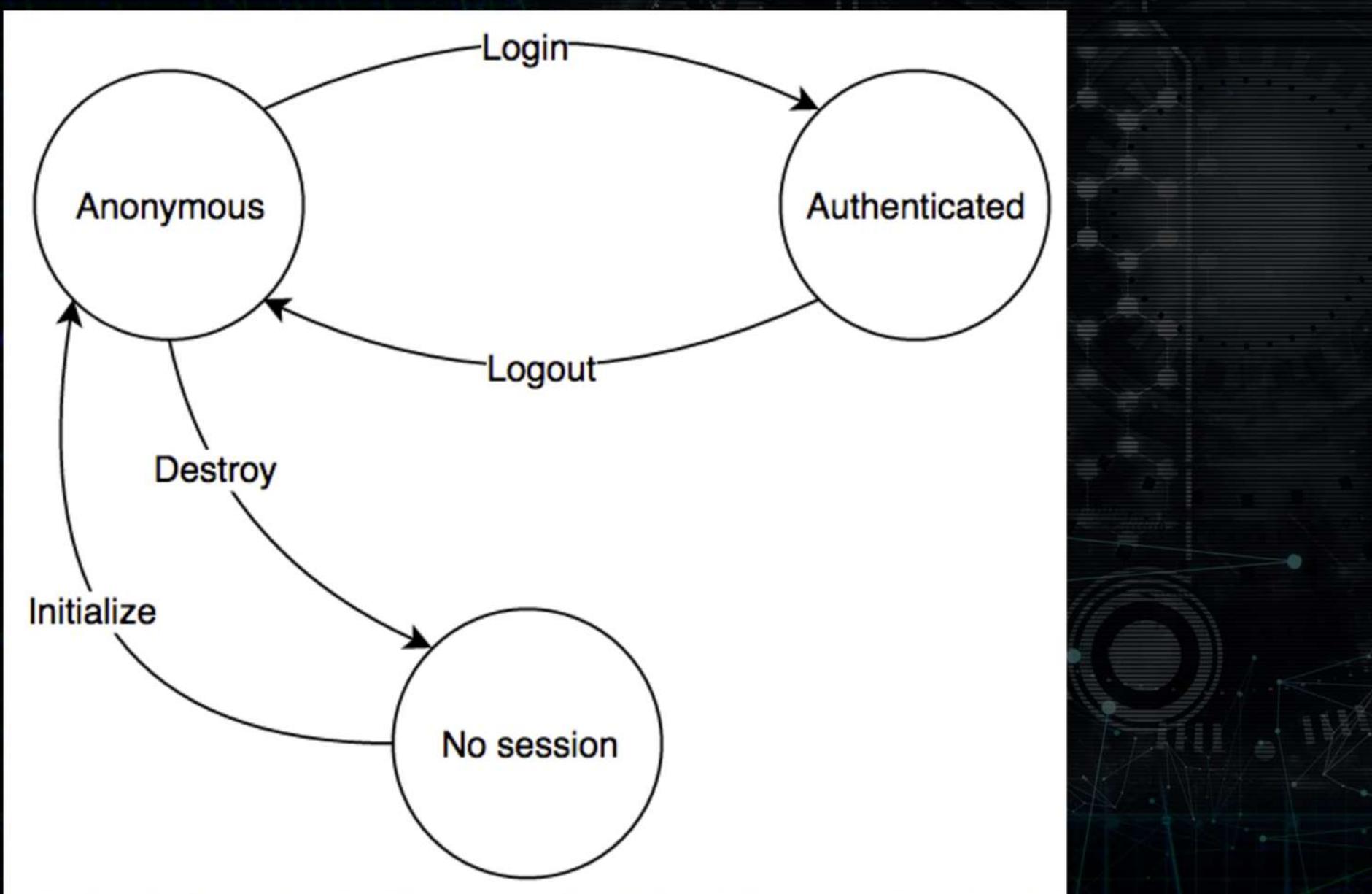


```
1 HTTP/1.1 302 Found
2
3 Date: Mon, 06 May 2025 17:00:00 GMT
4 Server: Apache
5 Set-Cookie: PHPSESSID=abc123xyz456; path=/; HttpOnly
6 Location: dashboard.php
7 Content-Length: 0
8 Connection: close
9 Content-Type: text/html; charset=UTF-8
10
11 Credenziali non valide.
```





# SESSION





# PRACTICE TIME

**Tempo stimato:** 10 minuti

**Consegna:** Realizzare una web application che imposti una sessione per il conteggio delle visite della pagina. In particolare si vuole:

"Salve, è l'array `$_SESSION[]` che parla, hai visitato la pagina {count} volte. Complimenti!"

Nota:

Al fine di svolgere l'esercizio non è necessario l'uso di css e js.



# PRACTICE TIME

**Tempo stimato:** 10 minuti

**Consegna:** Implementare nei codici visti in precedenza l'uso di un database per l'effettuazione del login.

**Nota:**

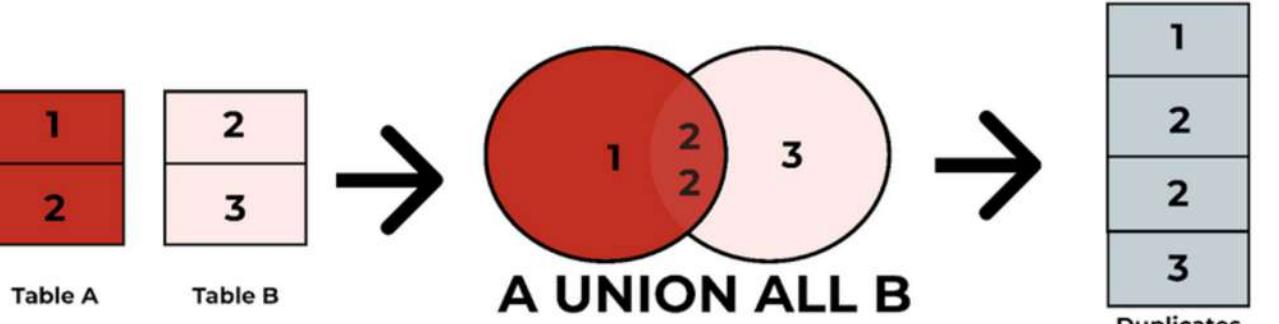
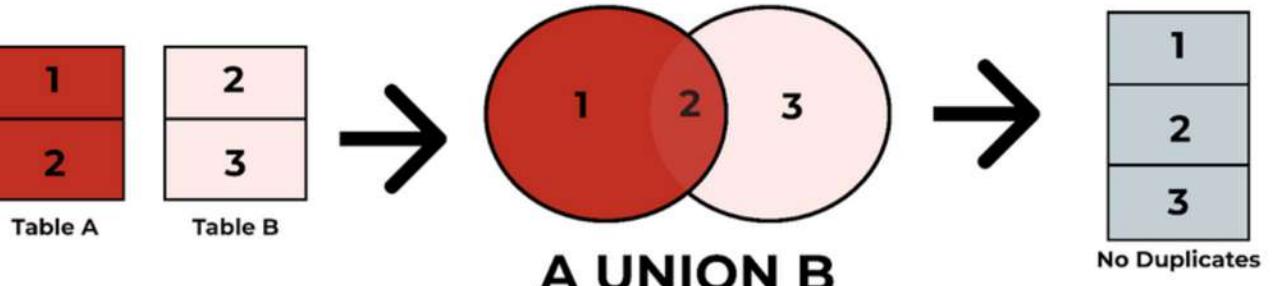
Al fine di svolgere l'esercizio non è necessario l'uso di css, js e dei prepared statements. Considerata la gradualità dell'esercizio si consiglia l'uso dei PDO.



# UNION

La clausola UNION in SQL permette di combinare i risultati di due query in un'unica tabella. L'importante è che entrambe le query restituiscano lo stesso numero di colonne. Quando si utilizza UNION, i risultati finali vengono restituiti senza duplicati. Se invece si desidera mantenere anche i record duplicati tra le query, si utilizza UNIONALL.

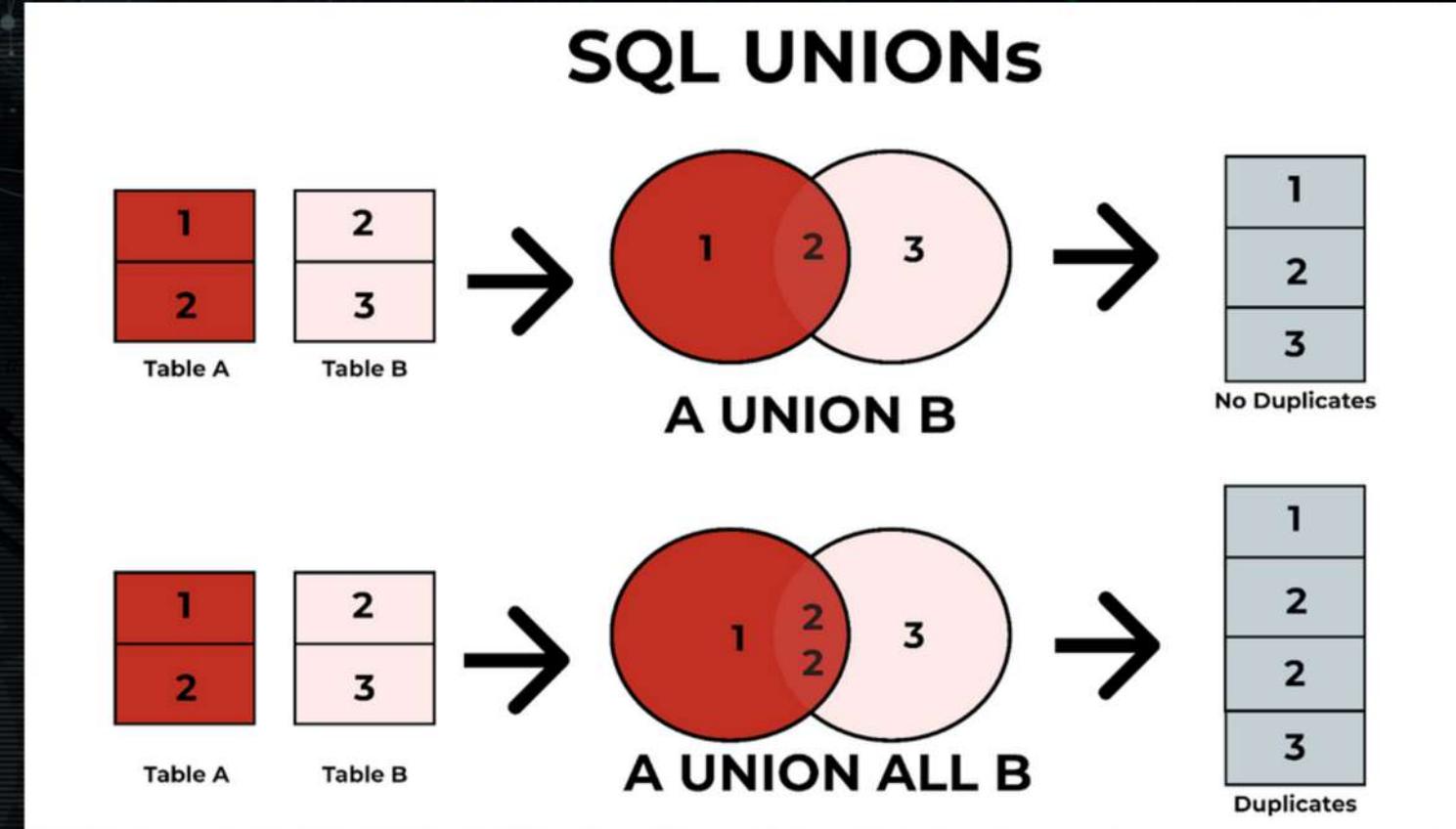
## SQL UNIONS

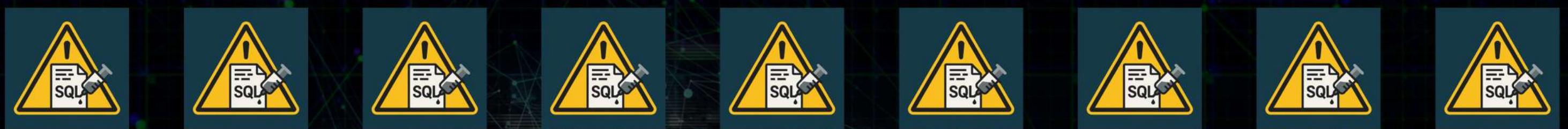




# FACCIAMO UN FULL JOIN ?

A seconda del DBMS non sempre è possibile utilizzare direttamente la clausola FULL OUTER JOIN, come ad esempio MySQL. Un modo per risolvere questo problema è attraverso la UNION.





# FACCIAMO UN FULL JOIN ?



```
1 SELECT Cognome, Nome, Titolo
2 FROM Dipendenti
3 LEFT JOIN Progetti ON Dipendenti.ID = Progetti.Dipendente
4 UNION
5 SELECT Cognome, Nome, Titolo
6 FROM Dipendenti
7 RIGHT JOIN Progetti ON Dipendenti.ID = Progetti.Dipendente
8 ORDER BY Cognome, Nome;
```



Cognome	Nome	Titolo
Bianchi	Marco	Progetto 1
Rossi	Anna	Progetto 2
Neri	Luca	NULL
NULL	NULL	Progetto 3

# SQL INJECTION - VULNERABILITÀ

La SQL Injection (SQLi) è una vulnerabilità che permette a un attaccante di **iniettarre** codice SQL per scopo malevolo nei campi di input (es. login o ricerca), alterando il comportamento delle query inviate al database.

Rischi principali:

- Accesso non autorizzato ai dati
- Modifica o cancellazione di tabelle
- Esposizione di dati sensibili





# SQL INJECTION - VULNERABILITÀ



Esistono diversi metodi per sfruttare le SQLi:

- Logic SQLi
- UNION-based SQLi
- Blind SQLi:
  - Error-based
  - Time-based



# LOGIC - SQLI

Le SQL Injection di tipo "Logic-based" (o Logic SQLi) sfruttano il comportamento logico delle query SQL per manipolare il flusso di esecuzione o le condizioni che determinano i risultati di una query. Questo tipo di iniezione può essere utilizzato per saltare controlli, bypassare autenticazioni.

```
$stmt->query( "  
    SELECT *  
    FROM Utenti  
    WHERE Username = '$username'  
    AND Password = '$password'  
");  
$stmt->fetch();
```

*Esempio vulnerabile alle sqli*



# LOGIC - SQLI

**Username:**

**Password:**

**Login**

● ● ●

```
1 SELECT *
2 FROM Utenti
3 WHERE Username = 'admin'
4 OR '1'='1' AND Password = 'boh';
```

**Benvenuto, sql!**

Sei nella tua area riservata.

**Esci**



# UNION BASED - SQLI

Le Union-Based SQL Injection (SQLi) sono un tipo specifico di attacco di iniezione SQL che sfrutta l'operatore UNION per combinare il risultato di una query legittima con una query malevola. Principalmente questo tipo di SQLi sfrutta un database, l'information\_schema[].



# UNION BASED - SQLI

L'INFORMATION\_SCHEMA è un database speciale presente nei sistemi di gestione database come MySQL, MariaDB, Microsoft SQL Server e altri DBMS compatibili con lo standard SQL.

L'INFORMATION\_SCHEMA contiene tavole e viste di metadati che forniscono informazioni dettagliate sulla struttura del database stesso. Mediante l'INFORMATION\_SCHEMA si ha accesso all'intera struttura del database.

## TABLES

- TABLE\_SCHEMA
- TABLE\_NAME
- TABLE\_TYPE

## COLUMNS

- TABLE\_NAME
- COLUMN\_NAME
- DATA\_TYPE

## SCHEMATA

- SCHEMA\_NAME
- DEFAULT\_CHARACTER\_SET
- DEFAULT\_COLLATION

## VIEWS

- TABLE\_NAME
- VIEW\_DEFINITION

## ROUTINES

- ROUTINE\_NAME
- ROUTINE\_TYPE (PROC/FUNC)



# UNION BASED - SQLI



```
1 $id = $_GET['id'];
2 $post = mysql_query("
3 SELECT Titolo, Contenuto
4 FROM Post
5 WHERE ID = $id");
6
```



```
1 SELECT Titolo, Contenuto
2 FROM Post
3 WHERE ID = 1
4 UNION
5 SELECT 1, table_name
6 FROM information_schema.tables
7 WHERE table_schema = DATABASE();
```



Titolo	Contenuto
Ciao Mondo	Il mio primo post
1	post
1	utenti



# UNION BASED - SQLI

Secondo l'output precedente si nota che esiste la tabella Utenti, però non sempre i campi delle tabelle si conoscono.

Per risolvere questo problema sfruttiamo nuovamente l'information\_schema per visualizzare i campi della tabella trovata.



```
1 SELECT Titolo, Contenuto  
2 FROM Post  
3 WHERE ID = 1  
4 UNION  
5 SELECT column_name, 1  
6 FROM information_schema.columns  
7 WHERE table_name = 'Utenti';
```



Titolo	Contenuto
Ciao Mondo	Il mio primo post sul blog.
ID	1
Password	1
Username	1

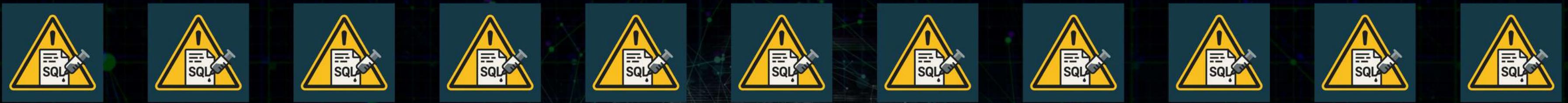


# UNION BASED - SQLI

```
1 SELECT Titolo, Contenuto  
2 FROM Post  
3 WHERE ID = 1  
4 UNION  
5 SELECT username, password  
6 FROM Utenti;
```



Titolo	Contenuto
Ciao Mondo	Il mio primo post sul blog.
alice	hash_pwd_1
bob	hash_pwd_2



«if secret starts with 'a' then sleep(1)»

# BLIND - SQLI

La Blind SQL Injection (SQLi cieca) è un tipo di attacco in cui:

- Il server non restituisce direttamente i risultati della query SQL, quindi l'attaccante non vede i dati.
- Ma può dedurre informazioni osservando i comportamenti (es. differenze nei tempi di risposta, errori HTTP, contenuti di pagine diversi, ecc.)

In particolare le Blind Time Based SQLi, si basano sul tempo di risposta della query (manipolato dall'attaccante). Le Blind Error Based SQLi, trovano superficie d'attacco quando il risultato di una query è booleano (successo, non successo).



# SQLI - PREVENZIONE

I prepared statements sono una tecnica di sicurezza comunemente utilizzata nei database. Si riferiscono a una sequenza di operazioni che separano la logica di esecuzione di una query SQL dalla parte dei dati che vengono effettivamente elaborati.

Anziché costruire manualmente una query SQL completa, si crea un modello con dei segnaposto, compilando la query una volta, cambiando solo i valori dei parametri per le successive esecuzioni.



# PREPARE STATEMENTS - PHP

Attraverso il linguaggio PHP si possono sfruttare 3 metodologie di connessione ad un database:

- Con approccio procedurale
- Con approccio a oggetti (mysqli, pdo)

Il PHP Data Object è uno dei metodi più opportuno da utilizzare poichè offre:

- Maggiore sicurezza
- Portabilità
- Gestione avanzata degli errori



# PREPARE STATEMENTS - PHP



```
1 $stmt = $pdo->prepare("SELECT * FROM utenti WHERE email = ? AND status = ?");  
2 $stmt->execute([$email, 'attivo']);
```

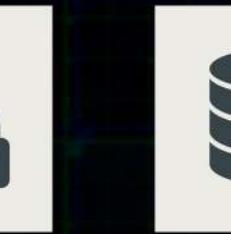


```
1 $stmt = $pdo->prepare("SELECT * FROM utenti WHERE email = :email AND status =  
:status");  
2 $stmt->execute([  
3     ':email' => $email,  
4     ':status' => 'attivo'  
5 ]);
```



# PREPARE STATEMENTS - PHP

```
1 $utenti = $stmt->fetchAll(PDO::FETCH_ASSOC);
2
3 if (count($utenti) > 0) {
4     foreach ($utenti as $utente) {
5         echo "Nome: " . $utente['nome'] . "<br>";
6         echo "Email: " . $utente['email'] . "<br>";
7     }
8 } else {
9     echo "Nessun utente trovato.";
10 }
```



# SALVATAGGIO SICURO DELLE PASSWORD

In questo caso, è necessario **proteggere le password** degli utenti.

Se le password sono salvate in chiaro, l'attaccante può riutilizzarle anche su altri siti (password reuse).

Per questo, le password vanno **sempre** salvate sotto forma di **hash**, e mai in testo leggibile. L'hash, per definizione, è una funzione *irriversibile*.

Abbiamo visto come proteggerci dalle SQL Injection, ma se un attaccante riuscisse comunque ad ottenere l'accesso al database?

BAD

Password in chiaro

password123

GOOD

Hash sicuro

94b0137fbfc90f  
1c14cadbb2c6ce5



# SALVATAGGIO SICURO DELLE PASSWORD - PHP

## `password_hash()`

Questa funzione genera un hash sicuro a partire dalla password dell'utente.

Cosa fa:

- Aggiunge automaticamente un salt
- Usa l'algoritmo più sicuro disponibile (es. bcrypt, o Argon2)
- Il risultato non è reversibile

PHP fornisce strumenti **sicuri** per gestire le password in modo corretto, come le funzioni:

- `password_hash`
- `password_verify`

## `password_verify()`

Serve per verificare se una password inserita dall'utente corrisponde all'hash precedentemente salvato.

Cosa fa:

- Confronta la password inserita con l'hash
- Funziona anche se l'algoritmo cambia nel tempo
- Protegge da confronti insicuri



# SALVATAGGIO SICURO DELLE PASSWORD - PHP

```
password_hash(  
    string $password,  
    string|int|null $algo  
): string
```

```
password_verify(  
    string $password,  
    string $hash  
): bool
```



# HASH SALTED - PHP

\$2y\$10\$6z7GKa9kpDN7KC3ICW1Hi.fd0/to7Y/x36WUKNP0IndHdkdR9Ae3K

Salt

Hashed password

Algorithm options (eg cost)

Algorithm



# SALVATAGGIO SICURO DELLE PASSWORD - COSA NON FARE

## 1. Troppo veloce

- MD5 è pensato per verifiche rapide, non per sicurezza.
- Gli attaccanti possono provare **milioni di combinazioni al secondo** (brute-force).

## 2. Nessun salt

- MD5 non include un valore casuale (salt).
- Due utenti con la stessa password → **stesso hash** nel database.
- Facilita attacchi con **rainbow tables** (database precalcolati di hash).

## 3. È rotto

- Esistono **collisioni**: due input diversi possono generare lo stesso hash.
- Già nel 2004 ne è stata dimostrata la vulnerabilità.

In passato, molti sviluppatori usavano **MD5** come algoritmo di hashing:

```
$hash = md5($password);
```

Perché non lo usiamo?



# PRACTICE TIME

**Tempo stimato:** 15 minuti

**Consegna:**

In riferimento al precedente esercizio, si vuole mitigare la vulnerabilità delle SQLi e aggiungere la possibilità di \*registrarsi alla web application.

Al fine di svolgere l'esercizio non è necessario l'uso di js e css.

\*Nel DB bisogna memorizzare l'hash delle password



# PRACTICE TIME

**Tempo stimato:** 20 minuti

**Consegna:**

Attraverso i precedenti codici, realizzare una web application che consenta di registrarsi ed effettuare il login, in una propria area personale che permetta la visualizzazione delle proprie note personali, con la possibilità di sceglierne il tipo, suddivise per data di creazione.

I codici si trovano in:

<http://10.50.0.161/5C/sperando/codici.zip>

XSS  
<JS>

JavaScript è un linguaggio di scripting che consente di eseguire codice sul browser dell'utente.

Tramite esso possiamo:

- Leggere e modificare il contenuto della pagina
- Leggere e modificare i cookie (eccetto i cookie HttpOnly)
- Rispondere ad eventi scatenati dall'utente
- ... e molto altro

Poiché JavaScript può accedere al contenuto della pagina, è importante che ci si possa fidare del codice eseguito

# CROSS-SITE SCRIPTING (XSS) - JavaScript

```
alert("Benvenuto nella pagina")  
  
document.getElementById('titolo')  
= "Titolo nuovo"  
  
document.getElementById('bottone')  
.addEventListener('click', () => {  
    alert("Hai cliccato!")  
})
```

XSS  
<JS>

# CROSS-SITE SCRIPTING (XSS)

## - Vulnerabilità

Esistono diversi tipi di XSS:

**Reflected XSS:** il payload viene inserito nella richiesta e viene riflesso nella risposta

**Stored XSS:** il payload è persistente in quanto viene salvato in un database e presentato a tutti gli utenti

**DOM-based XSS:** il payload viene eseguito a seguito di una modifica della pagina nel browser (DOM)

*Esempio vulnerabile per una DOM-based XSS*



```
<div id="msg"></div>
<script>
const params =
  new URLSearchParams(location.search)
document.write(params.get('text'))
</script>
```

XSS  
<JS>

Questo codice è vulnerabile a una **DOM-based XSS** poiché il parametro "text" viene inserito nella pagina senza essere controllato.

In questo modo, un attaccante può iniettare tag HTML, come il tag <script>.

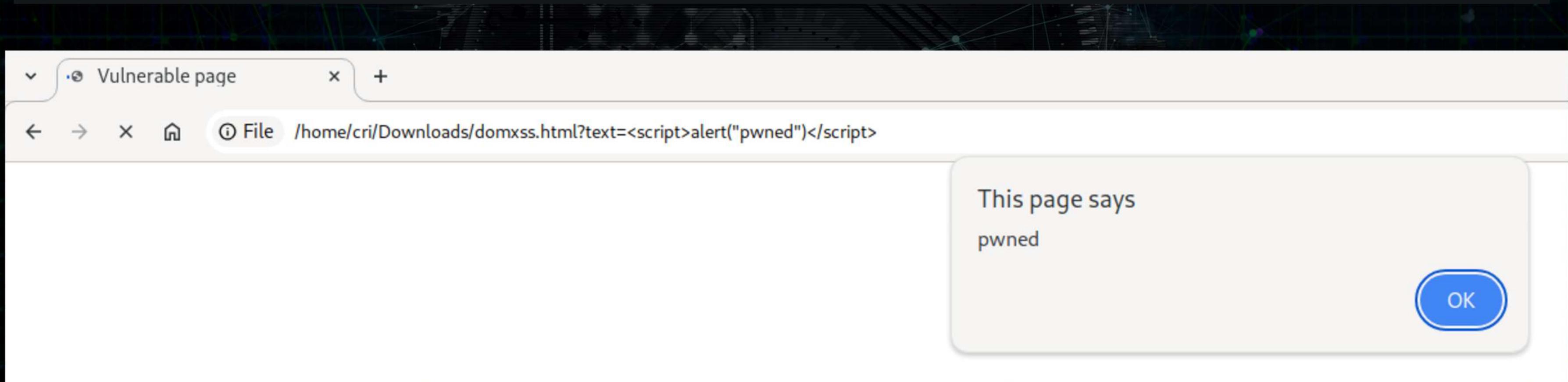
# CROSS-SITE SCRIPTING (XSS) - DOM-based XSS

*Esempio vulnerabile per una DOM-based XSS*

```
<div id="msg"></div>
<script>
const params =
  new URLSearchParams(location.search)
document.write(params.get('text'))
</script>
```

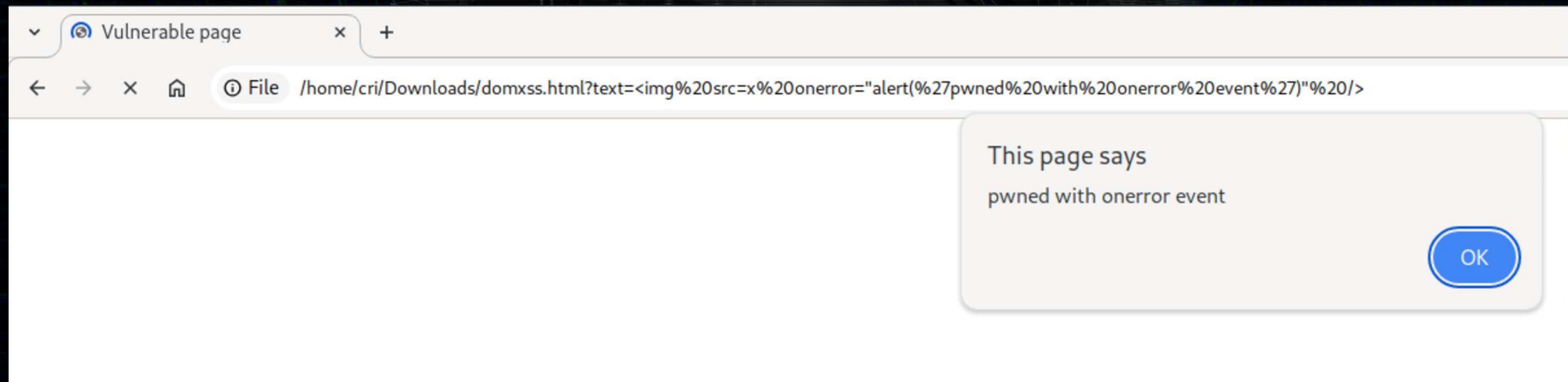
XSS  
<JS>

```
// Payload (da inserire nel parametro text)
<script>alert('pwned')</script>
```



XSS  
<JS>

```
// Payload (da inserire nel parametro text)
<img src=x onerror="alert('pwned with onerror event')"/>
```



XSS  
<JS>

In PHP, una **XSS** si verifica quando dati non filtrati dell'utente vengono inseriti in una pagina HTML.

Se l'output non è **sanificato**, un attaccante può iniettare **JavaScript maligno**.

# CROSS-SITE SCRIPTING (XSS) - Reflected XSS

*Esempio vulnerabile per Reflected XSS*



```
<?php  
$query = $_GET['q'];  
echo "Nessun risultato per:" . $query;  
?>
```

XSS  
<JS>

*Vulnerable*



```
<?php  
$query = $_GET['q'];  
echo "Nessun risultato per:" . $query;  
?>
```



*Non vulnerable*



```
<?php  
$query = $_GET['q'];  
echo "Nessun risultato per:" .  
htmlspecialchars($query);  
?>
```

La funzione **htmlspecialchars** trasforma i caratteri speciali in entità HTML, l'utente vede questi caratteri normalmente, ma il browser li interpreta semplicemente come testo

Carattere	Entità HTML
<	&lt
>	&gt
"	&quot
&	&amp

XSS  
<JS>

# CROSS-SITE SCRIPTING (XSS) - Prevenzione

Quindi come facciamo a  
prevenire questi  
attacchi?

Abbiamo visto che con le reflected possiamo usare **htmlspecialchars**,  
ma non è l'unico modo per prevenire le XSS.

La **CSP** (Content Security Policy) è un *header HTTP* che controlla quali  
risorse (script, immagini, stili, ecc.) possono essere **caricate** o **eseguite**  
da una pagina web.

È uno strumento **potente** per ridurre la superficie di attacco XSS,  
soprattutto nei casi in cui il contenuto viene inserito nel DOM in modo  
dinamico.

XSS  
<JS>

## Esempio di CSP



```
Content-Security-Policy: default-src 'self'; script-src 'self'
```

### Cosa fa:

- Permette di caricare tutto solo dal proprio dominio ('self')
- Blocca script inline (come quelli dentro `<script>alert[1]</script>`)
- Blocca script esterni non autorizzati

XSS  
<JS>

# CROSS-SITE SCRIPTING (XSS) - Altre tecniche di mitigazione

Sanitizzazione e validazione dell'input:

Usare librerie come DOMPurify per pulire HTML pericoloso.

- **Mai fidarsi dell'input utente**, anche se viene da URL o hash.
- Escape dell'output
  - Usare textContent, non innerHTML.
  - Sanitizzare dinamicamente i dati se devono essere inseriti come HTML, JS, attributi, ecc.
- Uso corretto delle API DOM
  - Evitare funzioni pericolose (innerHTML, document.write, eval, setTimeout con stringhe).

# GRAZIE PER LA VOSTRA ATTENZIONE



PER RIMANERE AGGIORNATI CON NOI FATE UN SALTO SU:  
<https://github.com/jim-bug>  
<https://github.com/Kuriix>



**GET IN TOUCH  
WITH US!**

