

Web Security 2 Database

**Lorenzo
LEONARDINI**

Università di Pisa



<https://cybersecnatlab.it>

License & Disclaimer

2

License Information

This presentation is licensed under the
Creative Commons BY-NC License



To view a copy of the license, visit:

<http://creativecommons.org/licenses/by-nc/3.0/legalcode>

Disclaimer

- We disclaim any warranties or representations as to the accuracy or completeness of this material.
- Materials are provided “as is” without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.
- Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.

Obiettivi

3

- Comprendere il concetto di database ed il loro funzionamento
- Comprendere il concetto di injection
- Comprendere gli attacchi di tipo SQL injection

Argomenti

4

- I database
- Le SQL injection

Argomenti

5

- I database
- Le SQL injection

Database

6

- Spesso i server hanno bisogno di memorizzare dati:
 - Informazioni sull'utente
 - Messaggi scambiati
 - I post di un blog
- I **database** permettono di gestire efficacemente i dati

Database

7

- Perché non usare dei semplici file? I database:
 - Sono ottimizzati per essere più efficienti
 - Permettono di gestire dati strutturati
 - Hanno meccanismi di recupero dati in caso di crash
 - ACID
 - Sono standardizzati

Database

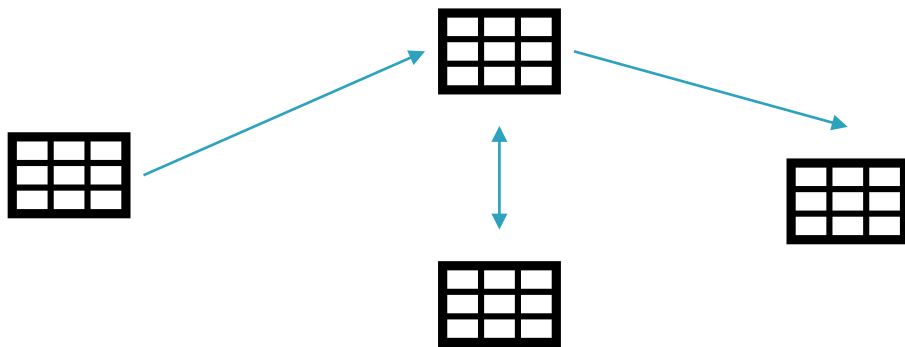
8

- I **DBMS** (Database Management System) sono programmi per gestire e interfacciarsi efficientemente con database. Tra i più comuni:
 - MySQL
 - MariaDB
 - PostgreSQL
 - SQLite
 - MongoDB

Database

9


- I **database relazionali** sono una tipologia di database che memorizzano i dati in tabelle, tra cui è possibile creare relazioni



Database

10

id	nome	email	password
1	Pippo	pippo@acme.com	topolino
2	Pluto	pluto@acme.com	gambadilegno
3	Paperino	paperino@acme.com	amaca



id	autore	titolo	testo
1	3	Come appendere un'amaca	L'amaca è indubbiamente il miglior supporto per riposare...

Database

11

- Per interagire con i database vengono utilizzati linguaggi specifici
- **SQL** (Structured Query Language) è il linguaggio più comune per i database relazionali
 - Ogni DBMS (SQLite, MySQL, PostgreSQL...) può estendere il linguaggio con tipi e funzionalità particolari
- I comandi inviati al database si chiamano **query**

Database

12

- La sintassi completa delle query SQL è estremamente complessa, ma gli usi più comuni sono limitati e molto semplici

```
[ WITH [ RECURSIVE ] with_query [ , ... ] ]
SELECT [ ALL | DISTINCT [ ON ( expression [ , ... ] ) ] ]
[ * | expression [ [ AS ] output_name [ , ... ] ] ]
[ FROM from_item [ , ... ] ]
[ WHERE condition ]
[ GROUP BY [ ALL | DISTINCT ] grouping_element [ , ... ] ]
[ HAVING condition ]
[ WINDOW window_name AS ( window_definition ) [ , ... ] ]
[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
[ ORDER BY expression [ ASC | DESC ] USING operator ] [ NULLS ( FIRST | LAST ) ] [ , ... ] ]
[ LIMIT { count | ALL } ]
[ OFFSET start [ ROW | ROWS ] ]
[ FETCH { FIRST | NEXT } [ count ] [ ROW | ROWS ] { ONLY | WITH TIES } ]
[ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [ , ... ] ] [ NOWAIT | SKIP LOCKED ] [ ... ] ]

where from_item can be one of:

[ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [ , ... ] ) ] ]
[ TABLESAMPLE sampling_method ( argument [ , ... ] ) [ REPEATABLE ( seed ) ] ]
[ LATERAL ] ( select ) [ AS ] alias [ ( column_alias [ , ... ] ) ]
with_query_name [ [ AS ] alias [ ( column_alias [ , ... ] ) ] ]
[ LATERAL ] function_name ( [ argument [ , ... ] ] )
[ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [ , ... ] ) ] ]
[ LATERAL ] function_name ( [ argument [ , ... ] ] ) [ AS ] alias ( column_definition [ , ... ] )
[ LATERAL ] function_name ( [ argument [ , ... ] ] ) AS ( column_definition [ , ... ] )
[ LATERAL ] ROWS / ROWS ( function_name ( [ argument [ , ... ] ] ) [ AS ( column_definition [ , ... ] ) ] [ , ... ] )
[ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [ , ... ] ) ] ]
from_item join_type from_item ( ON join_condition | USING ( join_column [ , ... ] ) [ AS join_using_alias ] )
from_item NATURAL join_type from_item
from_item CROSS JOIN from_item

and grouping_element can be one of:

( )
expression
( expression [ , ... ] )
ROLLUP ( [ expression | ( expression [ , ... ] ) ] [ , ... ] )
CUBE ( [ expression | ( expression [ , ... ] ) ] [ , ... ] )
GROUPING SETS ( grouping_element [ , ... ] )

and with_query is:

with_query_name [ ( column_name [ , ... ] ) ] AS [ [ NOT ] MATERIALIZED ] ( select | values | insert | update | delete )
[ SEARCH ( FORTH | DEPTH ) FIRST BY column_name [ , ... ] SET search_seq_col_name ]
[ CYCLE column_name [ , ... ] SET cycle_mark_col_name [ TO cycle_mark_value DEFAULT cycle_mark_default ] USING
cycle_path_col_name ]

TABLE [ ONLY ] table_name [ * ]
```

Database

13

- Per leggere dati da una tabella si usa la query **SELECT**

```
SELECT * FROM tabella
```

- I risultati possono essere filtrati

```
SELECT * FROM utenti WHERE email='pippo@acme.com'
```

Database

14

- Per inserire dati in una tabella si usa la query **INSERT**

```
INSERT INTO utenti (nome, email, password) VALUES ('Pippo', 'pippo@acme.com', 'topolino')
```

- Per modificare i dati in una tabella si usa la query **UPDATE**

```
UPDATE utenti SET password='cx67xctvt73' WHERE email='pippo@acme.com'
```

Database

15

- I server interagiscono con i database utilizzando specifiche librerie e funzioni:

```
$users = mysql_query("SELECT * FROM users");
```

Argomenti

16

- I database
- Le SQL injection

Injection

17

- Le injection sono una macro-categoria di vulnerabilità
- Esistono perché i programmi hanno la necessità di interagire con altri software:
 - Database → **SQL injection**
 - Comandi → **command injection**
- In queste interazioni spesso vengono utilizzati dati inseriti da utenti

SQL injection

18

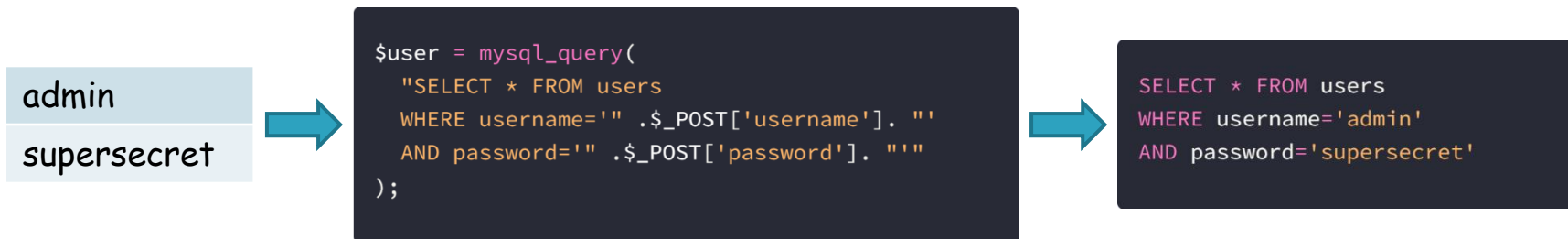
- Prendiamo in analisi il seguente codice utilizzato per fare il login:

```
$user = mysql_query(  
    "SELECT * FROM users  
    WHERE username='" . $_POST['username'] . "'  
    AND password='" . $_POST['password'] . "'" .  
);
```

SQL injection

19

- In una richiesta normale i valori di username e password vengono inseriti nella query tramite concatenazione di stringhe:



SQL injection

20

- Cosa succede se aggiungo un apice nell'username?
 - Errore di sintassi!



SQL injection

21

- Potendo *scappare* dagli apici, è possibile iniettare del codice SQL
- Questo tipo di vulnerabilità è chiamato **SQL Injection**



SQL injection

22

- Proviamo a bypassare il login!
 - Continuiamo ad avere un errore di sintassi :(



SQL injection

23

- In SQL il doppio trattino (--) denota l'inizio di un commento
- Accediamo senza conoscere la password!

admin
boh' OR True -- -



```
$user = mysql_query(  
    "SELECT * FROM users  
    WHERE username='" . $_POST['username'] . "'  
    AND password='" . $_POST['password'] . "'  
    );
```



```
SELECT * FROM users  
WHERE username='admin'  
AND password='boh' OR True -- -'
```

*MySQL richiede uno spazio dopo il doppio trattino, aggiungendo " -" siamo sicuri che l'attacco funzioni in tutti i DBMS

SQL injection

24

- Come proteggersi dalle injection?
 - Sanitizzare sempre l'input!
 - Utilizzare i *prepared statement*:

```
$stmt = $pdo->prepare('SELECT * FROM users WHERE username = ? AND password=?');  
$stmt->execute([$$_POST['username'], $_POST['password']]);
```


SQL injection

25

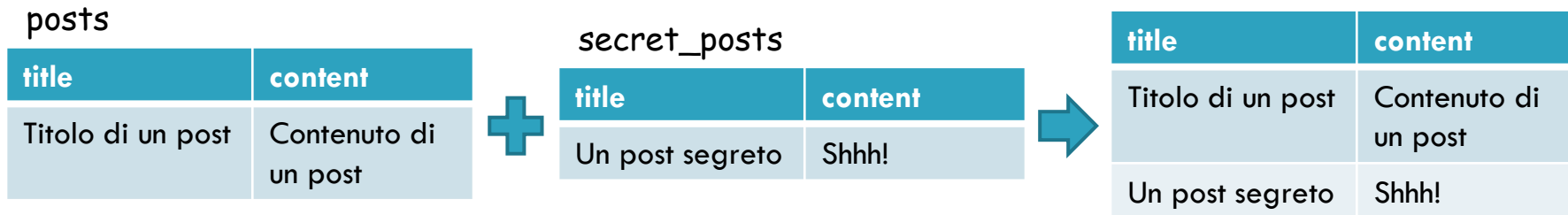
- Esistono diversi metodi per sfruttare le SQL injection:
 - **Logic** SQLi (come quella appena vista)
 - **UNION-based** SQLi
 - **Blind** SQLi:
 - **Error-based** SQLi
 - **Time-based** SQLi
 - ...

Union based SQL injection

26

- SQL permette di unire più tabelle nei risultati di una query:

```
SELECT title, content FROM posts  
UNION SELECT title, content FROM secret_posts
```



Union based SQL injection

27

- Immaginiamo una query di questo tipo:

```
$post = mysql_query("SELECT title, content FROM posts  
WHERE id=".$_GET['id']);
```

- Si potrebbe sfruttare per estrarre username e password dal database:

```
SELECT title, content FROM posts  
WHERE id=1 UNION SELECT username, password FROM users
```

Blind SQL injection

28

- A volte un server non ci mostra i risultati della query, ma ci fornisce solo messaggi di errore

GET /posts/42



200 Success

GET /posts/1337



404 Not found

Blind SQL injection

29

- Si può sfruttare per fare bruteforce di un qualche segreto da estrarre
 - L'operatore LIKE permette di verificare se una stringa inizia con un prefisso

Richiesta	Risposta
/posts/42 AND secret LIKE 'a%'	404
/posts/42 AND secret LIKE 'b%'	200
/posts/42 AND secret LIKE 'ba%'	404
/posts/42 AND secret LIKE 'bb%'	404
/posts/42 AND secret LIKE 'bc%'	200
/posts/42 AND secret LIKE 'bca%'	404
...	...

Time based SQL injection

30

- E se non abbiamo alcun tipo di output, neanche un errore?
- SQL ha un comando **SLEEP**, che possiamo chiamare nel caso in cui una condizione sia vera
- Si può fare un bruteforce misurando il tempo di esecuzione della query
 - L'idea è «*if secret starts with 'a' then sleep(1)*»

**Lorenzo
LEONARDINI**

Università di Pisa

Web Security 2 Database



<https://cybersecnatlab.it>