



SAPIENZA
UNIVERSITÀ DI ROMA

SicuroMaNonTroppo

Dipartimento di Informatica
Corso: Sicurezza Informatica

Autore: Daniel Di Nella

Anno Accademico 2023/2024

Contents

1	Introduzione	3
1.1	Obiettivi del Progetto	3
1.2	Introduzione a SQL Injection	3
1.2.1	Attacco In-Band	3
1.3	Tecniche di SQL Injection Utilizzate	3
1.4	Struttura della Piattaforma	3
2	Implementazione	4
2.1	Configurazione Docker	4
2.1.1	Docker Compose	4
2.1.2	Dockerfile per PHP	4
2.1.3	Dockerfile per MySQL	4
2.2	Script di Setup e Cleanup	4
2.2.1	Script di Setup	4
2.2.2	Script di Cleanup	4
2.3	Progettazione dell'Interfaccia Utente	4
2.3.1	File HTML, PHP e CSS	5
2.4	Gestione delle Sessioni e Sicurezza	5
2.5	Integrazione del Backend con il Database	5
2.6	Passaggio di Dati tra le Pagine	5
3	Risultati Sperimentali	6
3.1	Accesso al Sistema tramite SQL Injection	6
3.2	Ottenimento delle Tabelle del Database	6
3.3	Ottenimento dei Dati degli Utenti	6
3.4	Modifica dei Dati degli Utenti	10
3.5	Cancellazione dei Dati nel Database	10
4	Conclusioni	13
4.1	Contromisure per Prevenire SQL Injection	13

1 Introduzione

1.1 Obiettivi del Progetto

Il progetto *SicuroMaNonTroppo* ha l'obiettivo di creare una piattaforma web che simula un'applicazione di gestione utenti con un livello di sicurezza volutamente basso. Questo ambiente controllato permette di esplorare e dimostrare vulnerabilità comuni, in particolare l'SQL Injection, e di esaminare le tecniche per mitigare tali vulnerabilità.

1.2 Introduzione a SQL Injection

L'SQL Injection è un tipo di attacco che sfrutta vulnerabilità nei campi di input di un'applicazione web per manipolare le query SQL eseguite sul database. Quando un'applicazione non valida correttamente l'input dell'utente, un attaccante può inserire del codice SQL malevolo che viene eseguito dal database, portando a una compromissione della confidenzialità, integrità o disponibilità dei dati.

1.2.1 Attacco In-Band

Il progetto si focalizza su un tipo di attacco SQL Injection chiamato **In-Band**. Questo tipo di attacco è caratterizzato dal fatto che l'attaccante utilizza lo stesso canale di comunicazione per inviare la SQL Injection e ricevere i risultati. È il metodo più comune e facile da eseguire, poiché i risultati dell'attacco sono immediatamente visibili all'attaccante.

1.3 Tecniche di SQL Injection Utilizzate

Nel corso del progetto, verranno utilizzate diverse tecniche di SQL Injection, tra cui:

- **Tautologia:** Questa tecnica consiste nell'inserire un'espressione booleana sempre vera all'interno della query SQL. Ad esempio, inserendo '`OR 1=1; --`' in un campo di input, l'attaccante può forzare la query a restituire tutti i record.
- **Query Piggybacked:** Questa tecnica permette all'attaccante di "agganciare" una query aggiuntiva a quella legittima, utilizzando il carattere ; per separare le due query. Questo consente di eseguire comandi SQL aggiuntivi, come il recupero di dati sensibili o la modifica dei dati.
- **Commenti di Fine Riga:** L'uso dei commenti di fine riga (--) è una tecnica che permette di ignorare la parte rimanente della query legittima, assicurando che solo la porzione malevola della query venga eseguita.

1.4 Struttura della Piattaforma

La piattaforma è composta da diverse pagine, tra cui una pagina di login, una dashboard per gli utenti autenticati e una pagina di ricerca con visualizzazione dei risultati. Il backend è sviluppato in PHP e interagisce con un database MySQL per la gestione dei dati degli utenti. Questa struttura rende la piattaforma un banco di prova ideale per sperimentare le tecniche di SQL Injection descritte.

2 Implementazione

2.1 Configurazione Docker

Per garantire un ambiente di sviluppo consistente e facilmente replicabile, il progetto utilizza Docker per containerizzare i servizi necessari. Docker Compose è stato impiegato per orchestrare i container e gestire le dipendenze tra i servizi.

2.1.1 Docker Compose

Il file `docker-compose.yml` definisce e gestisce i servizi necessari per il progetto, inclusi il server PHP e il database MySQL. Questo approccio consente di avviare e configurare tutti i servizi richiesti con un solo comando, semplificando la gestione dell'ambiente di sviluppo.

2.1.2 Dockerfile per PHP

Il Dockerfile per PHP configura l'ambiente per l'esecuzione del server web. Specifica l'immagine base di PHP, le estensioni necessarie e le configurazioni di Apache. Utilizzando una versione specifica di PHP (7.4), per rendere l'ambiente più vulnerabile.

2.1.3 Dockerfile per MySQL

Il Dockerfile per MySQL imposta il database con le credenziali e le configurazioni iniziali richieste dal progetto. Viene utilizzato per gestire e memorizzare i dati necessari all'applicazione. Utilizzando una versione specifica di MySQL (5.4) si facilita la SQLInjection.

2.2 Script di Setup e Cleanup

2.2.1 Script di Setup

Il file `setup.sh` automatizza l'inizializzazione dell'ambiente Docker. Esegue la costruzione e l'avvio dei container, attende che il database sia pronto e applica eventuali script SQL iniziali. Questo script semplifica il processo di configurazione dell'ambiente di sviluppo.

2.2.2 Script di Cleanup

Il file `clean.sh` è utilizzato per rimuovere i container Docker e liberare le risorse associate. Ferma e rimuove i container, oltre a pulire volumi e reti Docker, garantendo che l'ambiente di sviluppo possa essere resettato facilmente o liberato da risorse non più necessarie.

2.3 Progettazione dell'Interfaccia Utente

L'interfaccia utente è stata progettata utilizzando HTML, CSS e Bootstrap. Il layout è responsivo e include un modulo di login e una dashboard con funzionalità di ricerca.

2.3.1 File HTML, PHP e CSS

La piattaforma è composta da file HTML per il frontend e file PHP per il backend, descritti di seguito:

- **index.html**: La pagina iniziale della piattaforma, che serve come punto di accesso per gli utenti.
- **dashboard.html**: La pagina principale della piattaforma dopo il login, che permette la navigazione tra le diverse funzionalità.
- **login.php**: Gestisce la procedura di login, verificando le credenziali degli utenti e reindirizzandoli alla dashboard.
- **dashboard2.php**: Visualizza i risultati delle ricerche effettuate dagli utenti in un container apposito e permette la navigazione come in dashboard.html.
- **search.php**: Processa le richieste di ricerca degli utenti e comunica con il database per restituire i risultati.
- **styles.css**: Il file CSS è stato utilizzato per personalizzare l'aspetto della piattaforma, includendo uno sfondo animato e stili per moduli e componenti come il carosello e la visualizzazione dei risultati.

2.4 Gestione delle Sessioni e Sicurezza

L'autenticazione degli utenti è gestita tramite sessioni PHP. Nonostante la natura volutamente insicura del sistema, sono state implementate alcune pratiche standard come la gestione delle sessioni.

2.5 Integrazione del Backend con il Database

Il backend PHP interagisce con un database MySQL. Le operazioni includono la verifica delle credenziali, la ricerca di utenti nel database e la gestione dei risultati della ricerca.

2.6 Passaggio di Dati tra le Pagine

I dati vengono passati tra le pagine utilizzando sia variabili di sessione che URL query parameters. Questo approccio è stato scelto per mostrare diverse tecniche di gestione dello stato e del flusso di dati all'interno di una web application.

3 Risultati Sperimentali

In questa sezione, descriveremo i passaggi eseguiti durante la sperimentazione, utilizzando attacchi SQL Injection per compromettere la sicurezza del sistema. Verranno mostrate le tecniche utilizzate e i principi del triangolo CIA (Confidentiality, Integrity, Availability) che sono stati violati.

3.1 Accesso al Sistema tramite SQL Injection

Il primo passo consiste nell'accedere al sistema utilizzando una SQL Injection. Utilizziamo una query malevola di tipo tautologia combinata con un commento di fine riga per bypassare il controllo delle credenziali di accesso.

- **Query Malevola:** ' OR 1=1; --
- **Tecnica Usata:** Tautologia, Commento di fine riga
- **Principio Violato: Confidenzialità** - L'accesso non autorizzato al sistema compromette la confidenzialità dei dati.

Dopo aver inserito questa query nel campo della password, siamo stati in grado di accedere al sistema come un utente autenticato.

3.2 Ottenimento delle Tabelle del Database

Una volta ottenuto l'accesso, abbiamo utilizzato un'altra SQL Injection per ottenere i nomi delle tabelle presenti nel database.

- **Query Malevola:** ' OR 1=1; SHOW TABLES; --
- **Tecnica Usata:** Tautologia, Query Piggybacked, Commento di fine riga
- **Principio Violato: Confidenzialità** - L'ottenimento non autorizzato dei nomi delle tabelle del database compromette ulteriormente la confidenzialità.

3.3 Ottenimento dei Dati degli Utenti

Una volta ottenuto il nome della tabella utenti, ovvero users, abbiamo utilizzato un'altra SQL Injection per estrarre informazioni sensibili dal database, in particolare per ottenere tutti i dati degli utenti.

- **Query Malevola:** ' OR 1=1; SELECT * FROM users; --
- **Tecnica Usata:** Tautologia, Query Piggybacked, Commento di fine riga
- **Principio Violato: Confidenzialità** - L'estrazione non autorizzata dei dati sensibili degli utenti compromette ulteriormente la confidenzialità.

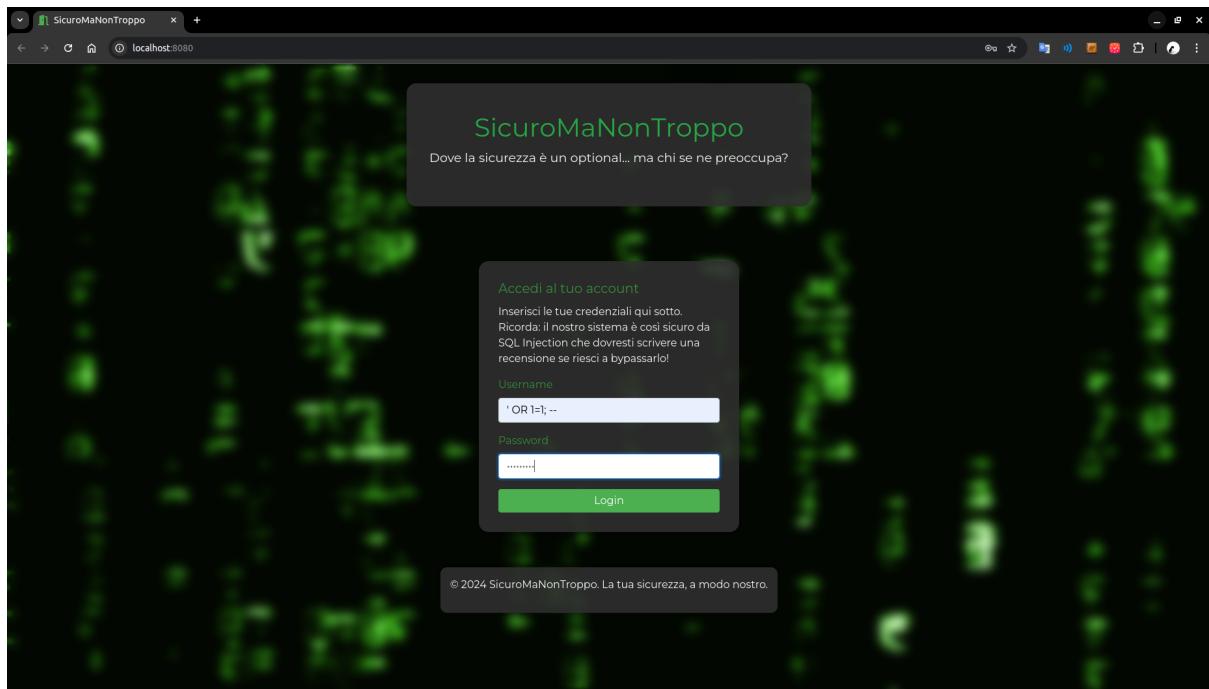


Figure 1: Accesso al sistema tramite SQL Injection.

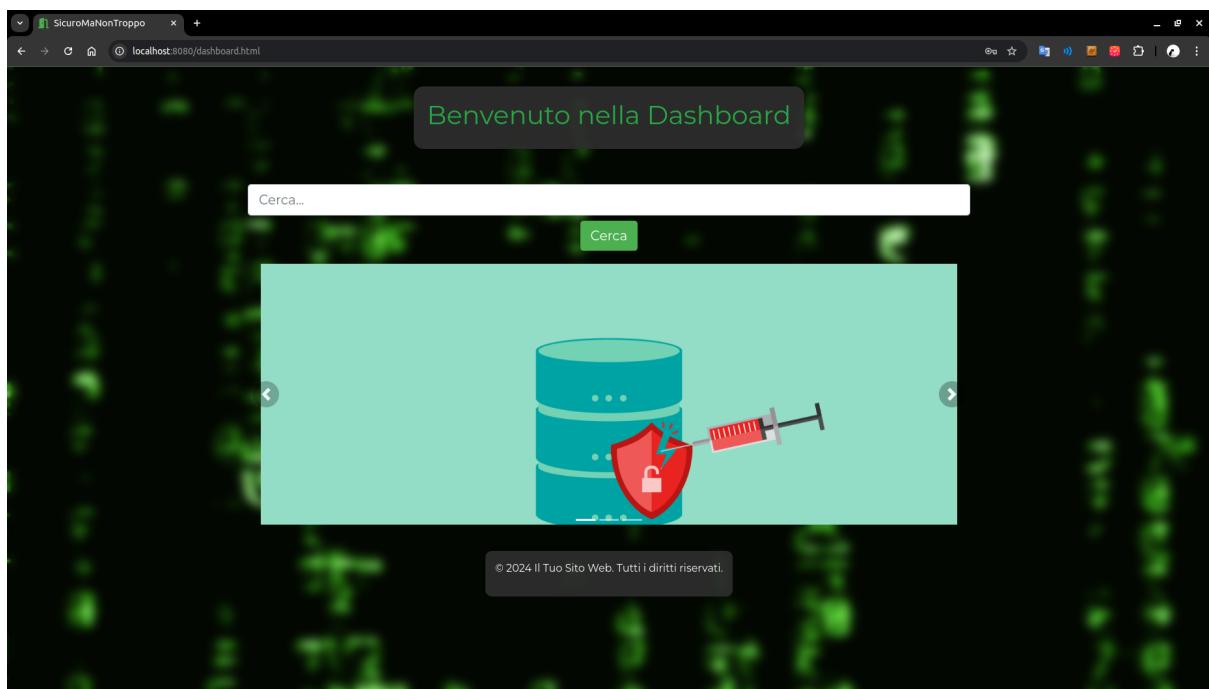


Figure 2: Accesso completato.

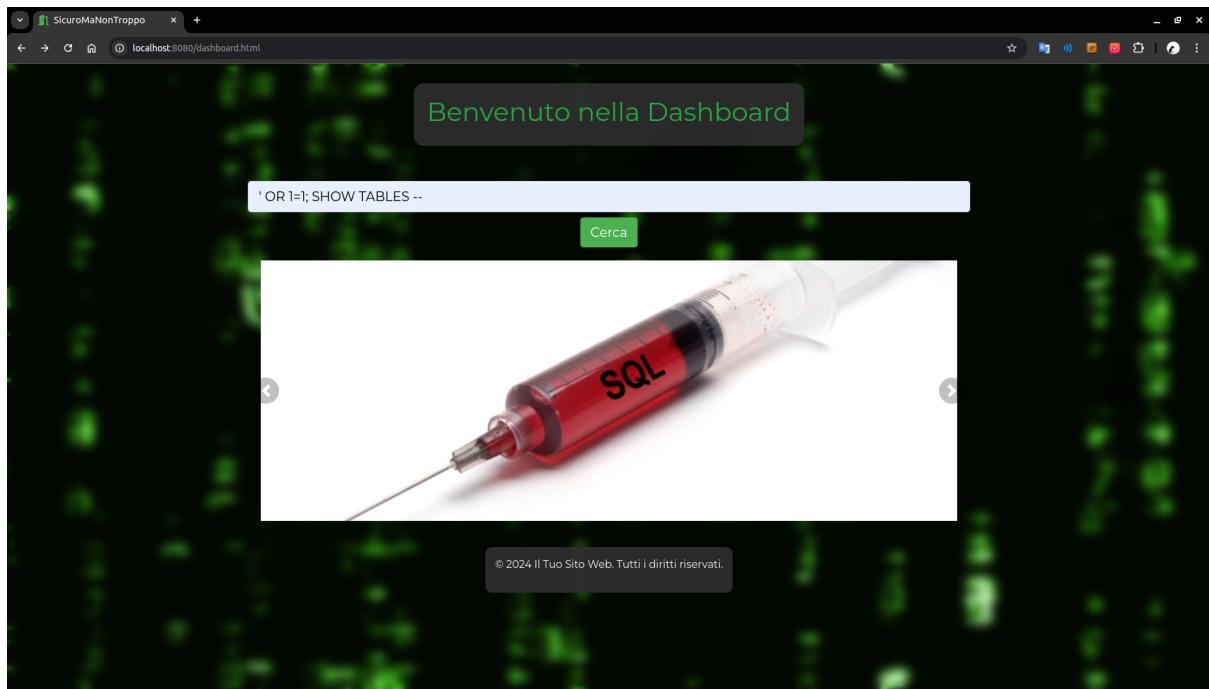


Figure 3: Inserimento della query malevola per ottenere tutte le tabelle.

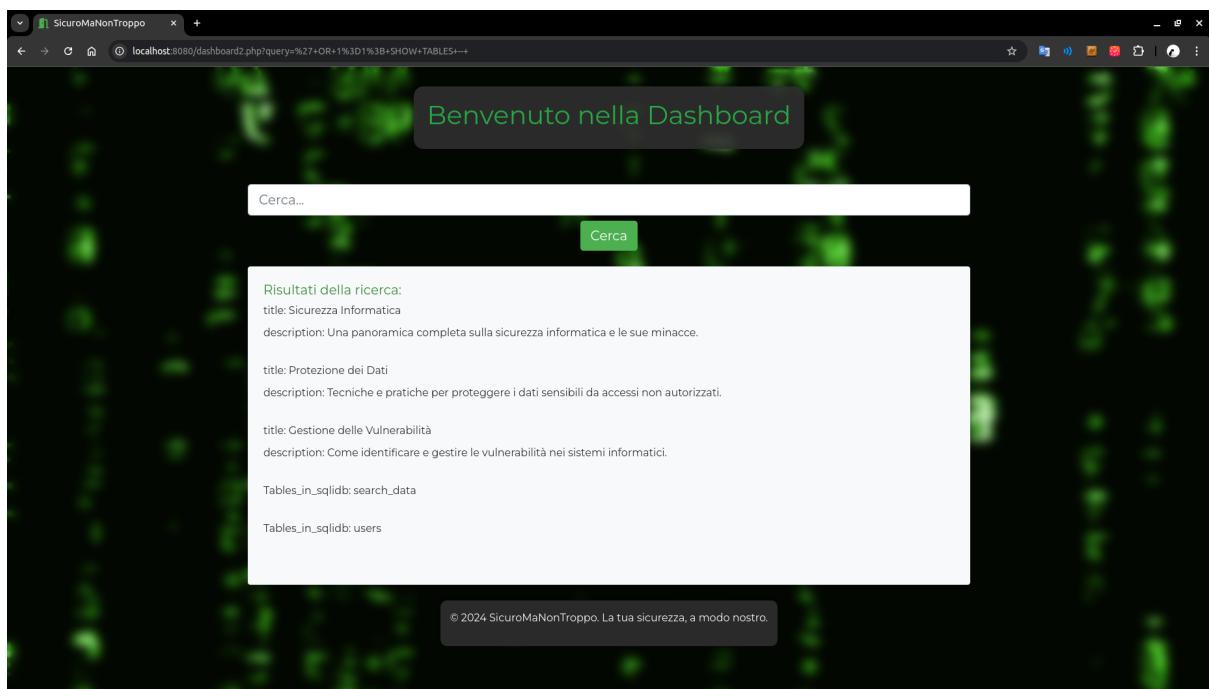


Figure 4: Risultati ottenuti dalla query malevola.

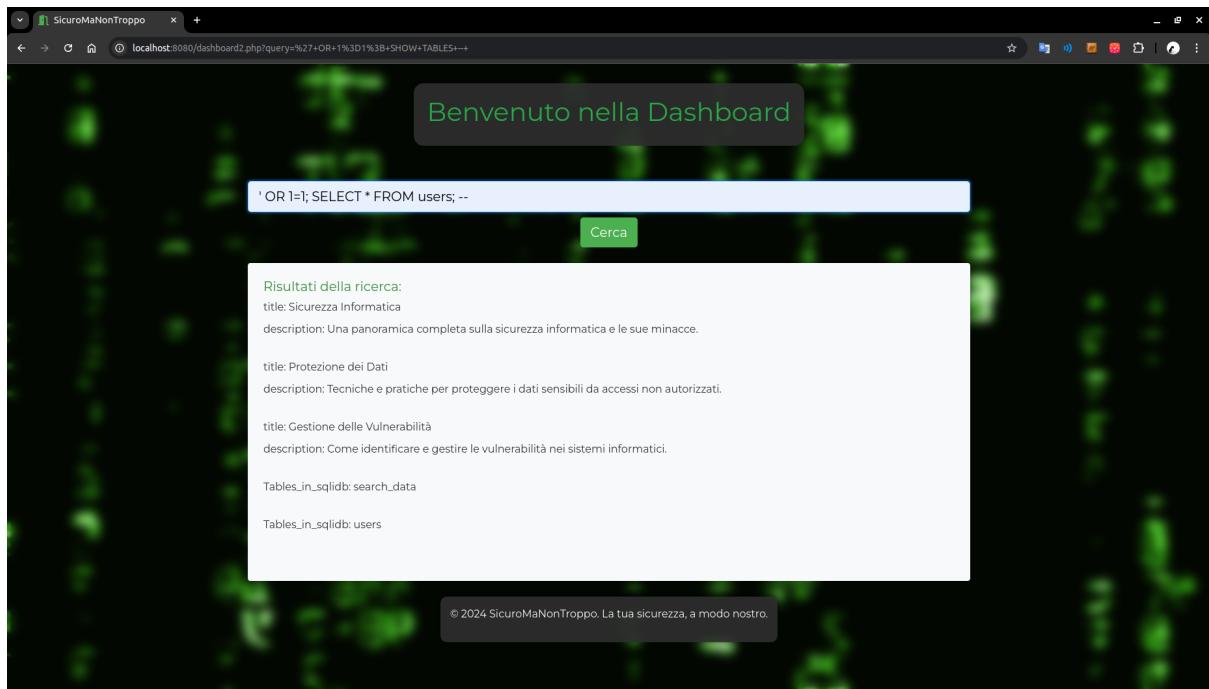


Figure 5: Inserimento della query malevola per ottenere i dati degli utenti.

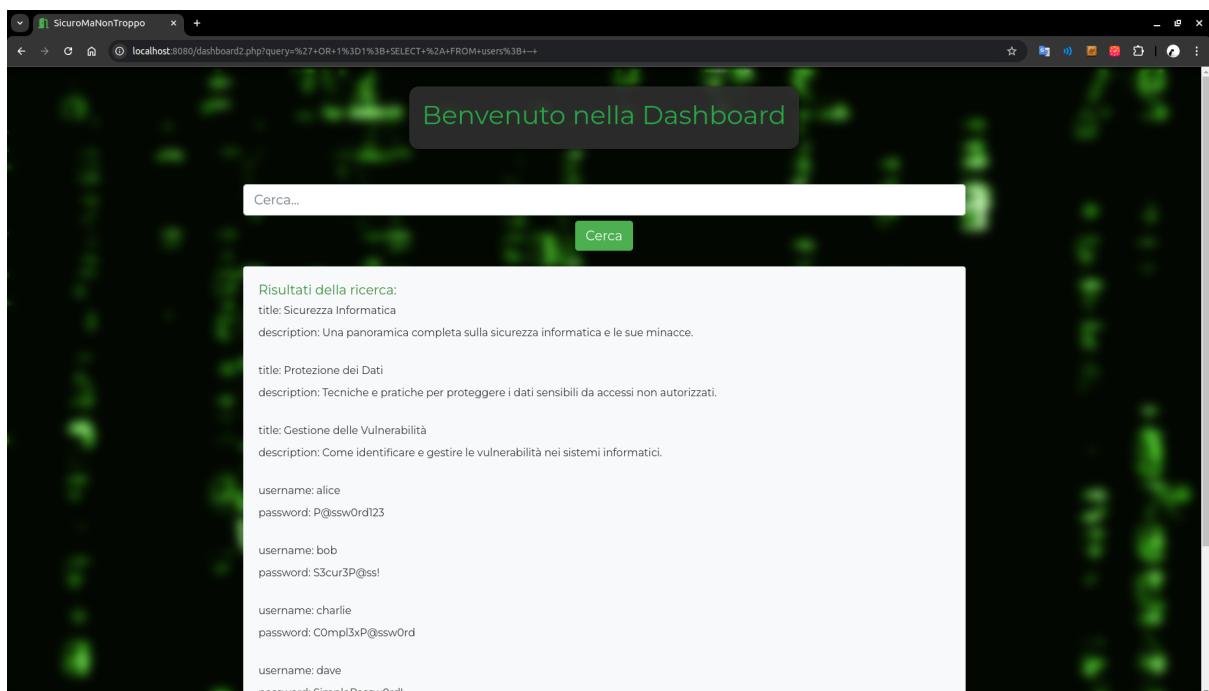


Figure 6: Ottenimento dei dati degli utenti tramite SQL Injection.

3.4 Modifica dei Dati degli Utenti

Dopo aver ottenuto i dati, abbiamo eseguito un attacco per compromettere l'integrità del database, modificando la password di uno degli utenti.

- **Query Malevola:** ' OR 1=1; UPDATE users SET password='newpassword' WHERE username='alice'; --
- **Tecnica Usata:** Tautologia, Query Piggybacked, Commento di fine riga
- **Principio Violato: Integrità** - La modifica non autorizzata dei dati viola l'integrità, poiché altera i dati esistenti nel database.

3.5 Cancellazione dei Dati nel Database

Infine, abbiamo eseguito una SQL Injection per eliminare una tabella dal database, compromettendo la disponibilità del sistema.

- **Query Malevola:** ' OR 1=1; DROP TABLE search_data; --
- **Tecnica Usata:** Tautologia, Query Piggybacked, Commento di fine riga
- **Principio Violato: Disponibilità** - La cancellazione della tabella `search_data` compromette la disponibilità, rendendo i dati non più accessibili agli utenti legittimi.

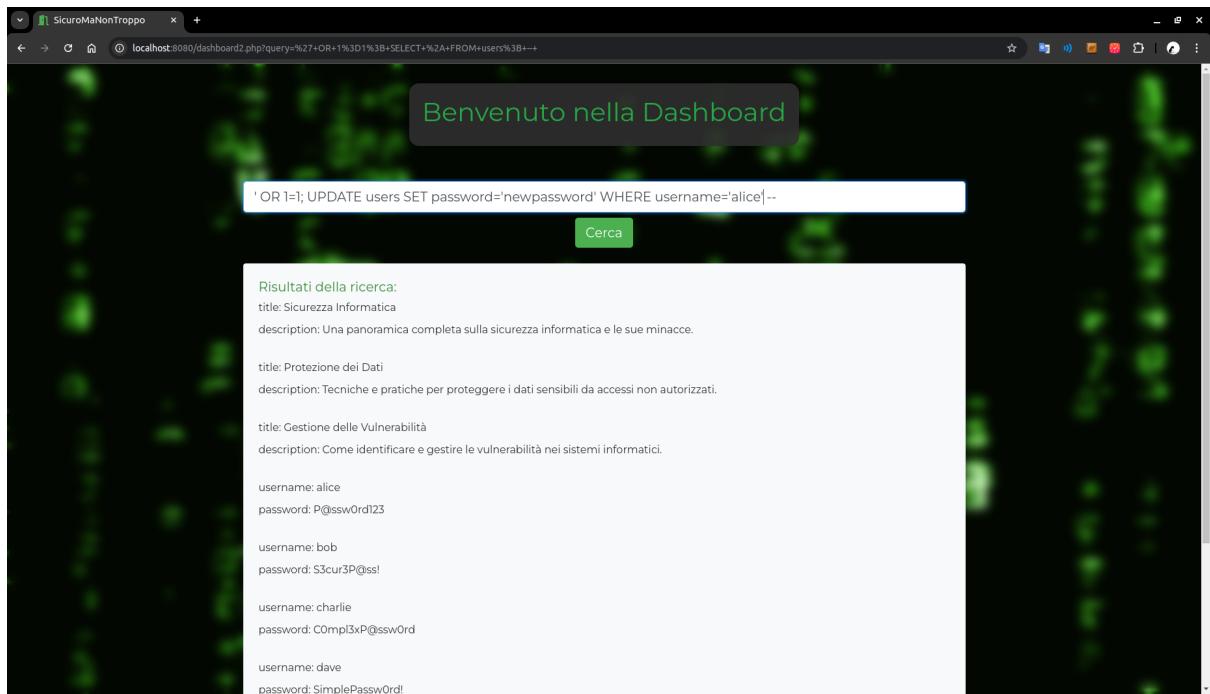


Figure 7: Modifica della password dell’utente tramite SQL Injection.

Risultati della ricerca:

title: Sicurezza Informatica
description: Una panoramica completa sulla sicurezza informatica e le sue minacce.

title: Protezione dei Dati
description: Tecniche e pratiche per proteggere i dati sensibili da accessi non autorizzati.

title: Gestione delle Vulnerabilità
description: Come identificare e gestire le vulnerabilità nei sistemi informatici.

username: alice
password: newpassword

username: bob
password: S3cur3P@ss!

username: charlie
password: C0mpl3xP@ssw0rd

username: dave
password: SimplePassw0rd!

Figure 8: Visualizzazione tramite SQL Injection della tabella users dopo la modifica della password.

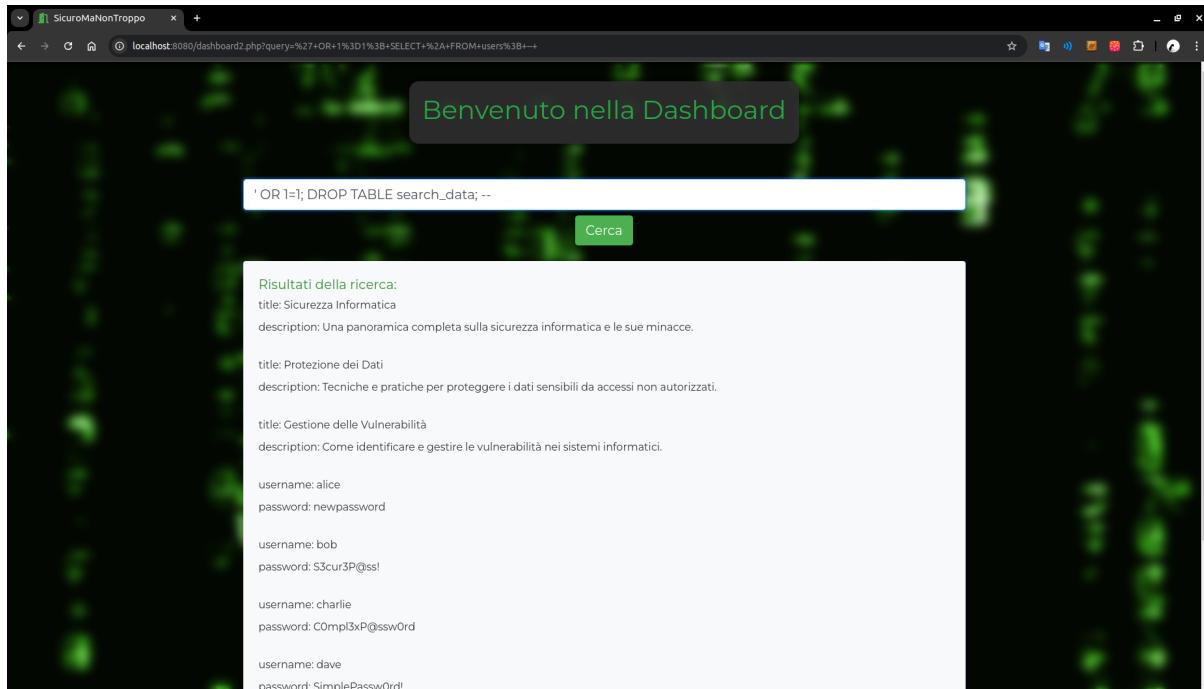


Figure 9: Cancellazione della tabella `search_data` tramite SQL Injection.



Figure 10: Tentativo di ricerca dopo la cancellazione della tabella `search_data`.

4 Conclusioni

Il progetto *SicuroMaNonTroppo* ha dimostrato l'importanza di implementare pratiche di sicurezza nelle web application. Le vulnerabilità identificate evidenziano i rischi associati a una gestione inadeguata della sicurezza.

4.1 Contromisure per Prevenire SQL Injection

Dopo aver analizzato gli attacchi SQL Injection condotti durante la sperimentazione, è essenziale discutere le contromisure che avrebbero potuto prevenire tali vulnerabilità. Di seguito sono elencate alcune pratiche e tecniche che avrebbero reso impossibile l'esecuzione di questi attacchi:

- **Parameter Binding:** L'utilizzo di prepared statements e parameter binding è una delle contromisure più efficaci contro SQL Injection. Con questa tecnica, i parametri delle query SQL vengono gestiti separatamente dal codice SQL, impedendo l'iniezione di codice malevolo.
- **Evitare Multi-Query:** Disabilitare l'esecuzione di multi-query, ovvero l'esecuzione di più query SQL in una singola richiesta, riduce il rischio di attacchi SQL Injection come quelli che sfruttano query piggybacked.
- **Validazione e Sanitizzazione degli Input:** Implementare una robusta validazione e sanitizzazione degli input dell'utente può prevenire l'iniezione di caratteri speciali che potrebbero alterare l'esecuzione delle query SQL.
- **Aggiornamento di MySQL e PHP:** Utilizzare versioni aggiornate e supportate di MySQL e PHP server. Le versioni più recenti includono patch di sicurezza che mitigano le vulnerabilità conosciute, rendendo più difficile l'esecuzione di SQL Injection.
- **Least Privilege Principle:** Configurare il database in modo che ogni applicazione o utente abbia i minimi privilegi necessari. Questo approccio limita i danni potenziali nel caso in cui un attacco SQL Injection riesca ad avere successo.
- **Configurazione Sicura del Server:** Configurare il server in modo sicuro, ad esempio disabilitando funzioni PHP pericolose, impostando opzioni di sicurezza aggiuntive in MySQL, come:
`sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES` per imporre una sintassi SQL più rigorosa.
- **Logging e Monitoraggio:** Implementare sistemi di logging e monitoraggio per rilevare e rispondere rapidamente a tentativi di SQL Injection. Questi sistemi possono aiutare a identificare comportamenti sospetti e a bloccare l'accesso prima che i danni siano fatti.

Queste contromisure, se implementate correttamente, aumentano significativamente la sicurezza del sistema, proteggendo l'integrità, la confidenzialità e la disponibilità dei dati contro gli attacchi SQL Injection.