RELAZIONE SULL'HOMEWORK – TRACCIA 2: ATTACCHI SQL INJECTION (SQLi)

Membri del gruppo:

- Davide Cioeta, 2063098

- Gianmarco Cestari, 2053287

- Francesco Serva, 1809178

1. Introduzione

L'SQL Injection (SQLi) è una delle vulnerabilità più note e pericolose che affliggono le applicazioni web che interagiscono con un database. Questo tipo di attacco avviene quando un'applicazione consente l'inserimento di input non sanificati in una query SQL, permettendo a un attaccante di manipolare direttamente il comportamento del database. Le conseguenze possono essere estremamente gravi: accesso non autorizzato a dati sensibili, modifica o cancellazione dei dati, oppure a l'esecuzione di comandi distruttivi come il drop di tabelle, l'escalation dei privilegi, o addirittura il controllo completo del sistema nel caso di database mal configurati.

2. Tecniche di attacco SQLi

Nel corso di questo homework sono state analizzate e testate diverse tipologie di attacco SQLi. Di seguito si riportano i principali esempi con le relative finalità.

NOTA BENE: quando si inserisce nel campo username una delle seguenti tipologie di attacco che terminano con il commento di fine riga, è FONDAMENTALE lasciare uno spazio dopo l'ultimo trattito.

2.1 Bypass dell'autenticazione

' OR 1=1 --

Questo classico payload sfrutta una tautologia per forzare la condizione della query SQL a risultare sempre vera. In un form di login, permette di ottenere l'accesso come primo utente nel database, bypassando del tutto le credenziali.

2.2 Enumerazione del numero di colonne

' ORDER BY N --

Serve a determinare il numero di colonne attese dalla query. Si incrementa il valore di N finché non si riceve l' errore "Unknown column 'N' in 'order clause'", segno che il numero massimo di colonne è stato superato.

2.3 Identificazione delle colonne visualizzabili

' UNION SELECT 1, 2, ..., N-1 --

Con questa tecnica si verifica quali colonne vengono mostrate a schermo e possono essere utilizzate per veicolare informazioni arbitrarie.

2.4 Scoperta delle tabelle presenti

' UNION SELECT 1, table_name, 3, 4 FROM information_schema.tables WHERE table_schema = database() LIMIT 1 OFFSET 0 --

Utilizzando il database information_schema, è possibile ottenere l'elenco delle tabelle del database corrente, modificando opportunamente il valore dell'OFFSET.

2.5 Modifica dei dati

'; UPDATE nomeTabella SET ruolo='amministratore' WHERE username='...'; --

Una volta individuate le tabelle e i campi, è possibile modificarne i valori per ottenere vantaggi indebiti, ad esempio cambiando il ruolo di un utente in "amministratore".

2.6 Eliminazione di tabelle

'; DROP TABLE nomeTabella; --

Tra gli attacchi più distruttivi, consente la cancellazione di tabelle dal database, arrecando potenziali danni permanenti.

3. Struttura tecnica degli attacchi

Gli attacchi SQLi si basano spesso sulla chiusura prematura di un campo stringa all'interno della query, tipicamente con un apice ', per poi inserire codice malevolo. Ad esempio:
SELECT * FROM utenti WHERE username='$username' AND password='$password';

Diventa vulnerabile se $username contiene ' OR 1=1 --, facendo terminare prematuramente la condizione e rendendo la query sempre vera.

L'operatore -- serve per commentare il resto della query originale, annullando qualsiasi

controllo successivo come la verifica della password.

Gli attacchi descritti rientrano in due macro-categorie:
- Tautologie: rendono la condizione WHERE sempre vera.
- Piggybacked Queries: permettono l'esecuzione di comandi multipli, spesso pericolosi, in una singola richiesta.

4. Istruzioni per l'uso

Lancio del programma:

A seconda del sistema operativo, posizionarsi nel percorso relativo alla cartella contenente il progetto ed eseguire le seguenti istruzioni:

- docker-compose build
- docker-compose up

Poi, una volta che il sistema avrà scaricato tutto il necessario, farà partire le librerie e i programmi per eseguire l'applicazione (se si dispone di dockerDesktop, questo può essere più comodo).

Visualizzare l'applicazione:

Una volta lanciato il programma da terminale, aprire il browser e caricare la pagina seguente:

- http://localhost:8080 – Per visualizzare l'applicazione
- http://localhost:8082 – Per visualizzare il database

Il database richiederà delle credenziali per poter accedere, queste sono riportate qui di seguito:

- Server: mariadb
- Utente: user
- Password: userpass
- Database: testdb

5. Funzionamento dell'applicazione

L'applicazione prevede un sistema di accesso con autenticazione, che distingue gli utenti in base al ruolo assegnato: amministratore oppure utente tradizionale. In base a questo ruolo, l'interfaccia e le funzionalità disponibili variano.

5.1 Accesso e gestione dei ruoli

Dopo aver effettuato l'accesso:

- Gli amministratori accedono a un'area riservata che consente la gestione dei prodotti, dei magazzini e la visualizzazione dello shop.
- Gli utenti tradizionali accedono esclusivamente alla sezione shop, con possibilità di consultare i prodotti disponibili.

5.2 Area Riservata - Funzionalità per gli Amministratori

Gestione Prodotti

All'interno di questa sezione, l'amministratore può:

- Aggiungere nuovi prodotti.
- Decrementare la quantità dei prodotti esistenti.
- Eliminare prodotti dal sistema.

Per ogni prodotto è visibile anche la posizione, ovvero il magazzino in cui è attualmente stoccato.

Gestione Magazzini

L'amministratore può:

- Aggiungere nuovi magazzini.
- Eliminare magazzini esistenti.

Nel caso in cui venga eliminato un magazzino contenente dei prodotti, anche i prodotti in esso presenti verranno eliminati automaticamente.

5.3 Shop (visibile a tutti gli utenti)

La sezione shop è accessibile sia agli amministratori che agli utenti tradizionali e consente di:

- Cercare un prodotto tra quelli disponibili tramite un'apposita barra di ricerca.
- Visualizzare in una lista i prodotti corrispondenti.
- Per ciascun prodotto selezionato, viene mostrata la disponibilità nei vari magazzini, con indicazione della loro posizione.

6. Conclusioni

L'SQL Injection rappresenta un grave rischio per la sicurezza delle applicazioni web. È fondamentale per gli sviluppatori conoscere queste tecniche per prevenirle attraverso pratiche sicure di codifica, come l'utilizzo di query parametrizzate (prepared statements), la validazione degli input, e la minimizzazione dei privilegi concessi agli account del database.

Questo homework ci ha permesso di comprendere concretamente la pericolosità dell'SQLi e la facilità con cui, in assenza di protezioni, è possibile compromettere completamente un sistema.