SQL Injection su DVWA - Livello Medium

Introduzione e Contesto

Scopo dell'Esercizio

L'obiettivo di questa esercitazione era dimostrare la vulnerabilità di **SQL Injection** presente nell'applicazione **DVWA** configurata con livello di sicurezza **medium**, con lo specifico scopo di recuperare la password dell'utente **Pablo Picasso**.

Ambiente di Testing

- Target: Macchina **Metasploitable** con **DVWA** installata
- Livello Sicurezza: **Medium**
- Obiettivo: Password di Pablo Picasso username: **pablo**)

Configurazione Iniziale

Prima di iniziare il testing, è stato necessario configurare correttamente l'ambiente:

- Accesso alla **DVWA** tramite browser all'indirizzo **192.168.12.150/dvwa**
- Login con credenziali amministrative: username admin, password **password**
- Impostazione del livello di sicurezza da **Low** a **Medium** nella sezione **DVWA Security**
- Conferma delle modifiche tramite click su **Submit**

Motivazione Tecnica

La modifica del livello di sicurezza è fondamentale poiché attiva meccanismi di protezione aggiuntivi.
A differenza del livello **low** dove gli input non vengono sanitizzati, il livello **medium** implementa la funzione **mysql real escape string** che aggiunge **escape character**, backslash agli apici e altri caratteri speciali, rendendo necessarie tecniche di bypass più avanzate.

Identificazione delle Vulnerabilità

Accesso alla Pagina Vulnerabile

- Navigazione alla sezione **SQL Injection** del menu DVWA
- Identificazione del campo di input **User ID** vulnerabile

Test di Vulnerabilità Base

Input testato: **1**

Risultato Atteso vs Osservato

- **Atteso**: Messaggio di errore SQL che conferma la vulnerabilità
- **Osservato**: La protezione medium blocca l'apostrofo, ma la pagina risponde comunque in modo anomalo, indicando una vulnerabilità potenziale

Analisi della Protezione

La funzione **mysql real escape string** converte:

- **' in \'**
- **" in \"**
- **\ in **

  Questo previene **l'escape** semplice ma non blocca altre tecniche di injection.

Determinazione Struttura Database

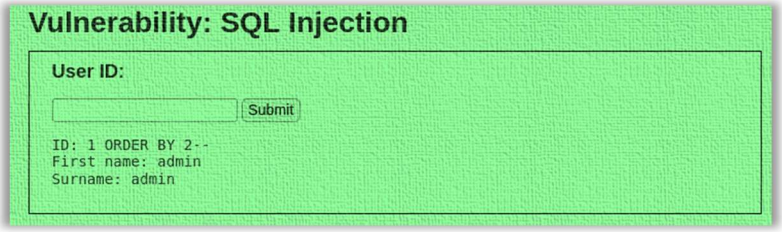
Scoperta Numero Colonne

Tecnica utilizzata: **ORDER BY** incrementale
Per identificare il numero di colonne presenti nella query **SQL** sottostante, è stata adottata la tecnica dell'**ORDER BY** incrementale.

Questo metodo consiste nell'inviare sequenze di richieste al server, aumentando progressivamente il valore del parametro utilizzato nell'istruzione ORDER BY.

Ad esempio, vengono testate le seguenti sequenze:

- **1 ORDER BY 1--**
- **1 ORDER BY 2--**
- **1 ORDER BY 3--**



Questi comandi permettono di osservare la risposta della pagina e di identificare il punto in cui si verifica un errore, indicando così il numero massimo di colonne gestite dalla query.
In questo modo si ottiene un'informazione fondamentale per proseguire con ulteriori fasi di analisi della vulnerabilità, sfruttando il comportamento anomalo della pagina per dedurre la struttura interna del database.

Risultati dell'Analisi tramite ORDER BY Incrementale

Durante l'applicazione della tecnica dell'ORDER BY incrementale, sono stati raccolti i seguenti risultati:

- **ORDER BY 1--: Successo** – la query è stata eseguita correttamente.
- **ORDER BY 2--: Successo** – la query è stata eseguita correttamente.
- **ORDER BY 3--: Errore** – il numero di colonne specificato eccede da quello effettivamente presente.

Da queste osservazioni si può concludere che la query in esame restituisce **due** colonne.
Quando viene richiesto un ordinamento in base a una terza colonna **ORDER BY 3--,** il database restituisce un errore, segnalando che il numero di colonne richiesto supera quelle realmente disponibili.

Spiegazione Tecnica

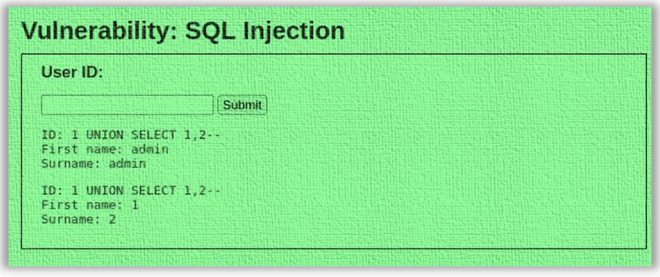
La clausola **ORDER BY** consente di ordinare i risultati della **query** in base a una o più colonne, specificate tramite il loro indice.
Se si tenta di ordinare i risultati utilizzando un indice di colonna superiore a quello effettivamente presente nel set di risultati, il database genera un errore.
Questo comportamento viene sfruttato per determinare in modo preciso il numero di colonne disponibili nella query, fornendo così una base solida per le successive fasi di analisi della vulnerabilità.

Identificazione delle Colonne Utili

Test UNION SELECT

Per individuare quali colonne della **query** risultano effettivamente visibili all'utente, è stato effettuato il test **UNION SELECT** tramite il seguente input:

- Input utilizzato: **1 UNION SELECT 1,2--**



Il risultato ottenuto dalla query è stato il seguente:

- **First name: 1**
- **Surname: 2**

Analisi del Risultato

Dall'analisi dell'output si nota che il valore **2** appare nella colonna **Surname**.
Questo indica che la seconda colonna è quella effettivamente visualizzata all'utente e, di conseguenza, può essere sfruttata per estrarre dati in fase di test.

Importanza della Clausola UNION SELECT

La clausola UNION consente di unire i risultati della query originale con quelli di una query arbitraria.
Perché questa operazione sia eseguita correttamente, è necessario che entrambe le query selezionino lo stesso numero di colonne.
Proprio per questo motivo, la determinazione preventiva del numero esatto di colonne si è rivelata una fase essenziale per la riuscita del test.

Scoperta del Nome del Database

Un passo fondamentale nell'analisi consiste nell'individuare il nome del database utilizzato dall'applicazione. Questo processo permette di contestualizzare meglio la struttura dei dati e di indirizzare le successive fasi di test verso le tabelle di interesse.

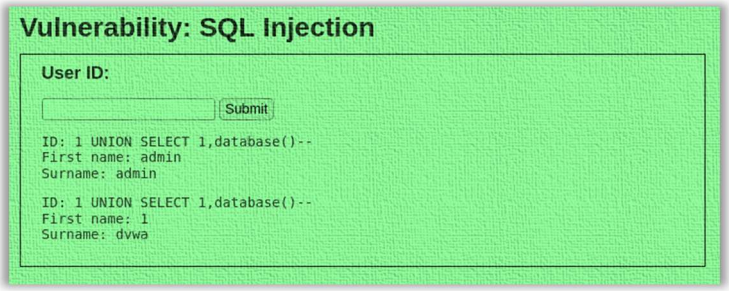
Estrazione del Nome del Database

Per identificare il nome del database, è stato utilizzato il seguente input nella query:

- Input utilizzato: **1 UNION SELECT 1, database () --**

Il risultato ottenuto dall'esecuzione di questa query è stato:

- **First name: 1**
- **Surname: dvwa**



Spiegazione:

La funzione **database ()** è una funzione **built-in di MySQL** che restituisce il nome del database attualmente in uso.
Questa funzione non prevede parametri e può essere invocata senza che siano necessari privilegi particolari.
Significato del Risultato
Dal risultato della **query**, si evince che il nome del database in uso è **dvwa**.
Questo database rappresenta il contenitore logico all'interno del quale sono memorizzate tutte le tabelle dell'applicazione, inclusa quella degli utenti che costituisce l'oggetto principale dell'attività di analisi condotta.

Significato del Risultato

Dal risultato della query, si evince che il nome del database in uso è **dvwa**.

Questo database rappresenta il contenitore logico all'interno del quale sono memorizzate tutte le tabelle dell'applicazione, inclusa quella degli utenti che costituisce l'oggetto principale dell'attività di analisi condotta.

Avendo identificato il database attivo, possiamo ora concentrare le nostre azioni sulle specifiche tabelle di interesse, proseguendo l'indagine sulle strutture dati e sulle informazioni sensibili archiviate all'interno di **dvwa**.

Enumerazione delle Tabelle del Database

Ricerca delle Tabelle

Per identificare tutte le tabelle presenti all'interno del database attualmente in uso, è stata eseguita la seguente **query**:

```
1 UNION SELECT 1,group_concat(table_name) FROM information_schema.tables WHERE table_schema=0x64767761—
```

Hex Encoding

In questo contesto, si è reso necessario utilizzare la rappresentazione esadecimale dei nomi dei database e delle tabelle per aggirare eventuali filtri o restrizioni imposte dalla funzione **mysql real escape string**, la quale blocca l'utilizzo degli apici nelle **query**.
La conversione viene eseguita trasformando ogni carattere della stringa nel corrispondente valore esadecimale.

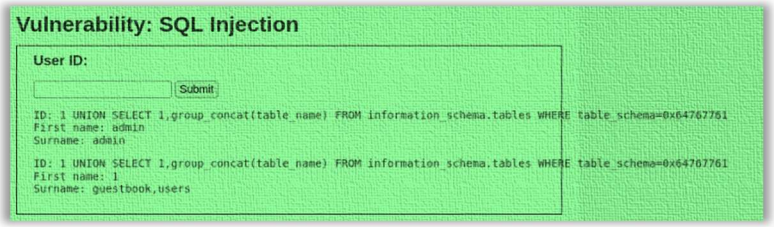
Ad esempio:

- **dvwa** diventa **0x64767761** (d = 64, v = 76, w = 77, a = 61)
- **users** diventa **0x7573657273**

Il prefisso **0x** comunica a **MySQL** che la sequenza successiva deve essere interpretata come un valore esadecimale.

Risultato della Query

L'output ottenuto dalla query è stato il seguente:



- **First name: 1**
- **Surname: guestbook, users**

Questo risultato mostra che nel database **dvwa** sono presenti almeno due tabelle: guestbook e users.
Funzione **group concat**
La funzione **group concat** svolge un ruolo fondamentale in questa fase, poiché consente di aggregare i valori restituiti da più righe in una singola stringa, separandoli mediante virgole.
In questo modo è possibile visualizzare in un'unica colonna tutti i nomi delle tabelle individuate dalla **query**.

Schema information schema

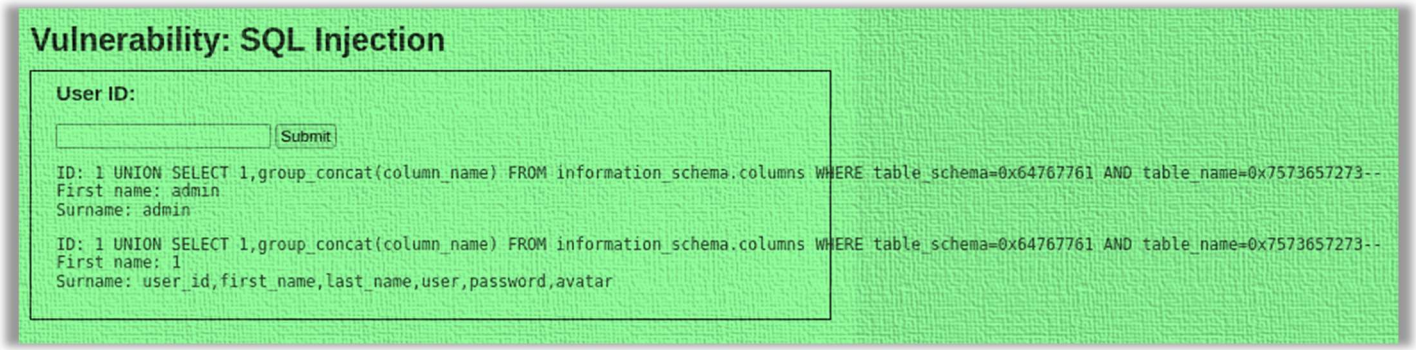
Per completare l'analisi, è importante sottolineare che **information schema** è il database di sistema di **MySQL** che contiene i metadati riguardanti tutti gli altri database, le relative tabelle e colonne.
Interrogando questo schema è possibile ottenere una panoramica strutturata delle informazioni archiviate nei database gestiti dall'istanza **MySQL**.

Enumerazione delle Colonne della Tabella "Users"

Al fine di ottenere una visione dettagliata della struttura della tabella users all'interno del database **dvwa**, è stata condotta una ricerca sulle colonne utilizzando la seguente query SQL:

```
1 UNION SELECT 1,group_concat(column_name) FROM information_schema.columns WHERE table_schema=0x64767761 AND table_name=0x7573657273--
```

Questa istruzione sfrutta la funzione **group concat** per aggregare in un'unica stringa tutti i nomi delle colonne appartenenti alla tabella selezionata.



Questa analisi dettagliata della struttura della tabella **users**, ottenuta tramite la **query** illustrata, ha rappresentato un passaggio cruciale per identificare le informazioni sensibili presenti nel database e comprendere le potenziali implicazioni in termini di sicurezza.

Tale approccio dimostra quanto sia essenziale, per garantire una protezione efficace, adottare strategie di difesa multilivello e mantenere un controllo rigoroso sugli accessi ai dati più delicati.

Il risultato restituito dalla query è il seguente:

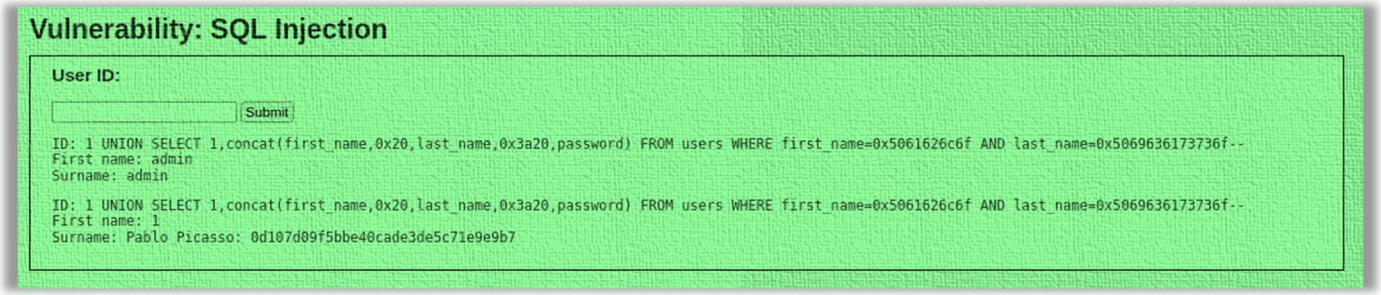
- **user_id:** Identificativo univoco dell'utente
- **first_name:** Nome dell'utente
- **last_name**: Cognome dell'utente
- **user:** Username utilizzato per il login
- **password: Hash** della password (campo di particolare interesse)
- **avatar:** Immagine profilo associata all'utente

Questa metodologia non solo ha evidenziato la vulnerabilità derivante dalla scarsa protezione dei dati, ma ha anche messo in luce come la conoscenza della struttura delle tabelle e delle relative colonne possa facilitare operazioni malevole di **data extraction.**
In tale contesto, il passo successivo consiste nell'analizzare le strategie di mitigazione più efficaci: è fondamentale limitare l'esposizione delle informazioni nel database, implementare controlli di accesso più stringenti e monitorare costantemente le attività sospette per prevenire possibili compromissioni.
Solo attraverso un approccio proattivo e una continua revisione delle politiche di sicurezza è possibile rafforzare la resilienza del sistema e garantire la tutela dei dati sensibili degli **utenti**.

Estrazione della Password dell'Utente "Pablo Picasso"

Per recuperare le credenziali dell'utente **Pablo Picasso**, è stata eseguita la seguente **query**:

```
1 UNION SELECT 1,concat(first_name,0x20,last_name,0x3a20,password) FROM users WHERE first_name=0x5061626c6f AND last_name=0x5069636173736f—
```



In questa istruzione, i valori **Pablo** e **Picasso** sono stati convertiti in **esadecimale** per essere utilizzati nella clausola **WHERE**:
Grazie a questa procedura, la concatenazione dei dati ha permesso di ottenere in modo semplice e diretto la stringa contenente il nome, il cognome e **l'hash** della password dell'utente selezionato, favorendo così l'identificazione e la successiva analisi **dell'hash** stesso.
La leggibilità del risultato, dovuta all'uso di separatori ben definiti, ha agevolato la fase di attacco: attraverso tecniche come il **brute force** o l'impiego di dizionari, si è potuto decifrare la stringa **MD5** ottenuta e risalire alla password in chiaro, compromettendo l'integrità del sistema di autenticazione.
Questo dimostra quanto sia rischioso esporre dati sensibili senza adeguate misure di sicurezza, sottolineando la necessità di adottare algoritmi di **hashing** più robusti e sistemi di autenticazione multifattoriale per tutelare gli account degli utenti.

- **Pablo** = **0x5061626c6f**
- **Picasso** = **0x5069636173736f**
- **Spazio** ( ) = **0x20**
- **Separatore** (:) = **0x3a20**

La funzione **concat**() unisce il nome, il cognome e **l'hash** della password in una stringa formattata, separando i dati tramite uno spazio e i due punti, per una lettura più chiara.

Il risultato ottenuto dalla query è:

Pablo Picasso: 0d107d09f5bbe40cade3de5c71e9e9b7

Grazie all'estrazione **dell'hash** della **password** tramite la **query** mirata, è stato possibile effettuare un attacco di tipo **brute force** o **dizionario sull'hash MD5** ottenuto, arrivando alla decifrazione della password corrispondente.

Una volta individuata la **stringa in chiaro**, la fase di autenticazione risulta compromessa: l'accesso al profilo dell'utente Pablo Picasso può avvenire impiegando direttamente le credenziali rilevate, dimostrando così la pericolosità dell'esposizione degli **hash** delle **password** e sottolineando l'importanza di adottare meccanismi di protezione robusti, come l'uso di algoritmi di **hashing** più sicuri e l'implementazione di sistemi di **autenticazione multifattoriale**.

Risultato e Decifrazione della Password

L'hash MD5 recuperato per l'utente **Pablo Picasso** è: **0d107d09f5bbe40cade3de5c71e9e9b7.**

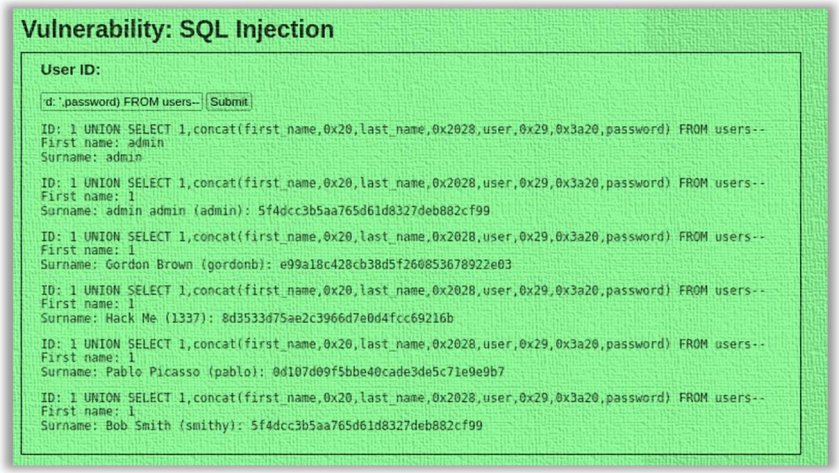
Password ottenuta: letmein

Le credenziali complete per l'utente Pablo Picasso sono quindi:

- Username: **pablo**
- Password: **letmein**
- Nome completo: **Pablo Picasso**

Questo esempio evidenzia chiaramente come la semplice esposizione **dell'hash** della **password** possa portare alla compromissione dell'intero sistema di autenticazione: una volta che un attaccante riesce a ottenere la corrispondenza in **chiaro** tramite tecniche di **brute force** o **dizionario**, l'accesso non autorizzato al profilo dell'utente diventa immediato, mettendo a rischio la sicurezza delle informazioni personali e sottolineando la necessità di adottare misure preventive più efficaci per la **protezione** delle **credenziali**.

Conclusioni e Considerazioni sulla Sicurezza
Vulnerabilità Dimostrate
Nel corso dell'esercitazione sono state evidenziate numerose vulnerabilità che mettono a rischio la sicurezza delle applicazioni web. In primo luogo, è stata riscontrata la possibilità di eseguire attacchi di tipo **SQL** Injection nonostante la presenza di protezioni di livello intermedio, dimostrando che queste misure possono essere aggirate da tecniche più sofisticate.



Un altro punto critico riguarda il bypass della funzione **mysql_real_escape_string**() tramite la codifica esadecimale **hex encoding**, che permette agli attaccanti di superare i controlli sugli input e accedere a dati che dovrebbero essere protetti.

Queste vulnerabilità hanno consentito l'accesso non autorizzato a informazioni sensibili, evidenziando la debolezza delle misure di sicurezza implementate e la necessità di adottare strategie più efficaci per la protezione dei **dati**.

Infine, è stata sottolineata la **criticità** dell'utilizzo di algoritmi di hashing deboli, come **MD5**, per la protezione delle password, che espongono le credenziali degli utenti a rischi concreti di **compromissione**.

Raccomandazioni di Sicurezza

Per rafforzare la sicurezza delle applicazioni **web** e prevenire **exploit** simili, è essenziale adottare alcune misure chiave.

In particolare, l'implementazione dei **prepared statements** rappresenta una soluzione efficace contro gli attacchi di **SQL Injection**, in quanto permette di separare i dati dai comandi **SQL** e riduce drasticamente la superficie di attacco.

Risulta altrettanto importante utilizzare algoritmi di **hashing** più robusti, come **bcrypt** e **Argon2**, che offrono una protezione significativamente superiore rispetto a **MD5** e rendono molto più difficile la decifrazione delle password da parte di **malintenzionati**.

Limitare i privilegi assegnati agli **account** del database è una pratica fondamentale per minimizzare i danni in caso di compromissione: ogni account dovrebbe avere solo i permessi strettamente necessari alle sue funzioni.

Un altro elemento chiave è l'implementazione di una **validazione** degli input su più livelli, che consente di filtrare e controllare i dati ricevuti dall'utente in modo rigoroso, riducendo le possibilità di **exploit**.

Infine, l'utilizzo di un **Web Application Firewall** (**WAF**) rappresenta un'ulteriore barriera contro gli attacchi, offrendo una protezione aggiuntiva che può rilevare e bloccare tentativi di intrusione in tempo reale.

Conclusione

Questa esercitazione ha dimostrato in modo chiaro ed efficace che le protezioni di livello intermedio non sono sufficienti per garantire la sicurezza delle applicazioni **web**.

Gli attaccanti possono sfruttare tecniche avanzate per aggirare le difese e accedere a dati sensibili.

Pertanto, è fondamentale adottare un approccio multilivello alla sicurezza, integrando diverse strategie e tecnologie per proteggere in modo completo le informazioni e le credenziali degli utenti.

Infine, come extra troviamo altri dati sensibili ES: Carte di credito