SERVIZIO FILM_DB CTF 2019

Team Jelly Hinge

Introduzione

Film_DB era uno dei servizi web da exploitare durante la UniCTF 2019, avente una vulnerabilità di tipo SQL Injection su DB SQLite3.

Il servizio

Il servizio presentava un'interfaccia a riga di comando ed era accessibile via **netcat**. Il menù permetteva la scelta di sei opzioni tra cui:

- Inserimento film
- Ricerca di un film via filmaker
- Ricerca di un film via titolo
- Ricerca di un film via tipologia
- Scelta randomica di un film
- Exit

Indizi

Dalla sola interfaccia disponibile non risultano esserci evidenti indizi.

La vulnerabilità

Osservando il codice è possibile notare che per ogni scelta è associata una query che prevede un controllo sull'input inserito dall'utente. Tutte le query, tranne l'ultima, sono di inserimento e prevedono l'escaping dei soli apici doppi. Per questo motivo è possibile fare attacchi di tipo SQL Injection che sfruttano l'utilizzo di apici singoli.

Attenzionando le singole query notiamo che solo quella relativa alla prima scelta esegue un inserimento nel database mentre tutte le altre eseguono delle selezioni.
Osservando l'ultima query capiamo che viene selezionato un **filmmaker** in modo randomico, la parte di codice è la seguente:

```
filmmaker = random.choise(SELECT DISTINCT
filmmaker FROM films);
```

dove successivamente viene eseguita la seguente query:

```
"SELECT title, filmmaker, kind FROM films WHERE
filmmaker = '{}'".format(filmmaker)"
```

Sfruttando la possibilità di inserimento nel database, il core dell'exploit sarà quello di far eseguire una query malevola.

L’Exploit

Selezioniamo la prima scelta ed inseriamo rispettivamente:

```
fimmaker: ‘ OR 1=1 --
title: TITOLOFILM
kind: TIPOLOGIA
```

Essendo questo l’unico record presente, la scelta randomica numero 5, lo selezionerà con certezza. L’esecuzione della query a seguire è un mezzo che ci aiuterà a capire se la vulnerabilità ipotizzata è sfruttabile o meno:

```
“SELECT title, filmmaker, kind FROM films WHERE filmmaker = ‘’ OR 1=1 --’
```

In questo modo, la parte relativa alla clausola WHERE, grazie alla parte in blu, sarà **vera** e tutto quello che segue i trattini (parte in rosso) verrà vista come un commento da SQLite3.
Il risultato di tale artificio è quello di ottenere una query equivalente a: `SELECT title, filmmaker, kind FROM films`
Che banalmente restituirà tutti i record presenti in **films**.
Per convalidare tale ipotesi bisogna comunque aggiungere altri record nel database poiché (‘ **OR 1=1 - -**) essendo l’unico, non ci aiuterà a capire il reale effetto.
Scegliendo l’opzione 5, qualora verrà selezionato il record malevolo inserito e verranno visualizzati tutti i record della tabella films, allora avremmo la certezza di tale vulnerabilità.

A fronte di ciò, possiamo pensare di estendere la query usando l'operatore UNION, per eseguire una query totalmente personalizzata (N.B l'operatore UNION permette di eseguire una sotto SELECT con vincolo che, quest'ultima, abbia lo stesso numero di parametri della SELECT principale).
Osservando il codice, tra le prime righe, notiamo che la flag si trova in una tabella cui nome e attributo che la contengono, sono generati in modo randomico:

```
nc=randomString(8)
nt=randomString(8)
c.execute("CREATE TABLE "+nt+"("+nc+" text)")
c.execute("INSERT INTO "+nt+" VALUES ('{}')".format(flag))
```

Non essendo a conoscenza del nome di tale tabella, sfruttiamo una funzionalità di base di SQLite3 che ci darà la lista di tutte le tabelle presenti nel database.
Per fare ciò aggiungiamo un altro record, tramite l'opzione 1, così composta nel campo filmmaker:

```
[1]  'UNION SELECT 1,2,tbl_name FROM sqlite_master
WHERE type='table
```

Selezioniamo l'opzione 5 fin quando la scelta randomica cadrà sul record **[1]**.
Il risultato sarà quello di dare in output i nomi di tutte le tabelle del database (compresa quella cui nome è stato generato in modo randomico che contiene la flag). Dalla tabella avente il nome randomico, dobbiamo estrarne i suoi campi. Quindi, allo stesso modo, inseriamo una query che ce li restituisca:

`[2] 'UNION SELECT 1,2,sql FROM sqlite_master WHERE name='tabella_random`

Selezioniamo nuovamente l'opzione 5 fin quando la scelta randomica cadrà sul record **[2]**.

Preleviamo il nome del campo random e aggiungiamo un altro record così formato:

[3] 'UNION SELECT 1,2,attr_rand FROM tabella_random'

Rieseguiamo l'opzione 5 fin quando la scelta randomica cadrà sul record **[3]**.

Il risultato darà in output la flag.

La patch

Una possibile patch per poter risolvere è quella di prevedere un escaping dei caratteri in input, precisamente negli apici singoli e doppi.

Link

Link alla pagina github dell'evento: https://github.com/unictf/unictf-2019/tree/master/services/film_db