E-commerce Website Security Assessment & Remediation Plan

Prepared by: Arpan Patel
Date: 13/09/2025
Scenario: Small e-commerce website on LAMP stack (DVWA in lab)

1. Executive Summary

This assessment evaluates a fictional e-commerce website (DVWA lab on LAMP) against core Cyber Essentials Plus principles with a focus on web application security. Using AI assistance for rapid analysis and a controlled Kali → DVWA lab for demonstration, we identified critical weaknesses: SQL Injection, Broken Authentication, and Stored Cross-Site Scripting (XSS). Immediate remediation is required for SQL Injection and authentication flaws; security headers and output encoding address XSS. Defense-in-depth and operational controls are recommended to reduce risk and improve resilience.

2. Top 10 Common Web Application Vulnerabilities (AI-Assisted)

• A01: Broken Access Control: Failures in enforcing user permissions allow unauthorized access or actions. Impact: data exposure, privilege escalation.

• A02: Cryptographic Failures: Sensitive data not protected in transit/at rest; weak/legacy crypto. Impact: credential theft, privacy breaches.

• A03: Injection: Untrusted input interpreted as code/queries (SQL/NoSQL/OS/LDAP). Impact: data exfiltration, RCE.

• A04: Insecure Design: Missing threat modeling or insecure business logic. Impact: workflow bypass, systemic weaknesses.

• A05: Security Misconfiguration: Default accounts, verbose errors, open S3/buckets, bad perms. Impact: lateral movement, information disclosure.

• A06: Vulnerable & Outdated Components: Unpatched frameworks/servers/libs. Impact: known exploits → takeover/RCE.

• A07: Identification & Authentication Failures: Weak passwords/sessions, no MFA, flawed resets. Impact: account takeover.

• A08: Software & Data Integrity Failures: Unsigned/unchecked updates/CI-CD artifacts; insecure deserialization. Impact: supply-chain compromise.

• A09: Security Logging & Monitoring Failures: Insufficient/unguarded logs, lack of alerting. Impact: undetected breaches.

• A10: Server-Side Request Forgery (SSRF): Server fetches attacker-controlled URLs. Impact: access to internal services/metadata.

3. AI-Assisted Vulnerability Identification (Provided Snippets & Config)

3.1 Snippet 1 – PHP Login

Code summary:
Receives username/password from POST, interpolates directly into an SQL query, and compares plaintext values.

Identified issues & OWASP mapping:

- SQL Injection (A03: Injection): Direct string concatenation into SQL query enables injection (e.g., ' OR '1'='1).

- Plaintext Password Handling (A02: Cryptographic Failures): Compares plaintext passwords; no hashing, salting, or secure verification.

- Hard-coded Credentials (A05: Security Misconfiguration): Database credentials embedded in source; risk of leakage and reuse.

- Excessive DB Privileges (A05: Security Misconfiguration): Likely overly-permissive DB user.

- Lack of Brute Force Controls (A07: Identification & Authentication Failures): No rate limiting/lockouts/MFA on login.

- Verbose Responses (A05: Security Misconfiguration): Clear success/failure messages aid attackers.

Initial remediation:

• Use prepared statements/parameterized queries (PDO/mysqli).

• Store passwords with Argon2id (or bcrypt) and verify using password_verify().

• Move secrets to environment variables/secret manager; rotate regularly.

• Create least-privilege DB user; revoke unsafe privileges (DROP/ALTER).

• Implement login throttling, account lockout, and optional MFA.

• Generic error messages; detailed errors only in logs.

3.2 Snippet 2 – JavaScript Search (DOM Rendering)

Code summary:
Directly writes user-provided input into the DOM via innerHTML, rendering attacker-controlled markup/scripts.

Identified issues & OWASP mapping:

- DOM-based XSS (A03: Injection (XSS)): Using innerHTML with unsanitized input allows script execution.

Initial remediation:

• Use textContent or safe templating; avoid innerHTML for untrusted data.

• Sanitize input with a vetted library (e.g., DOMPurify).

• Set a strict Content-Security-Policy (CSP) to block inline scripts and restrict sources.

3.3 Configuration Detail – Apache Logs

Apache logs rotate daily under /var/log/apache2 with default permissions.

Identified issues & OWASP mapping:

- Overly Permissive Log Access (A09: Security Logging & Monitoring Failures): Defaults may leave logs readable to unintended users; logs often contain sensitive data.

- Lack of Centralized Monitoring (A09: Security Logging & Monitoring Failures): No forwarding/alerting to detect brute force or anomalies.

- Potential Log Injection/Poisoning (A05: Security Misconfiguration): If application logs raw input, attackers may inject control characters.

Initial remediation:

• Restrict permissions (e.g., chmod 640; owner root, group adm).

• Forward logs to a central SIEM; enable alerting for auth failures and anomalies.

• Sanitize/encode user input before logging; avoid sensitive data in logs.

• Protect log rotation scripts; verify correct logrotate config.

4. Prioritized Remediation & Mitigation Strategy (Personal Input)

Prioritization is based on exploitability and business impact:

• P1 – SQL Injection in login (Critical): trivial exploitation → full DB compromise.

• P2 – Authentication weaknesses (High): brute force, plaintext passwords → account takeover.

• P3 – DOM/Stored XSS (High): session hijack, phishing, supply-chain script injection risks.

• P4 – Hard-coded secrets & misconfiguration (Medium): secret leakage; privilege misuse.

• P5 – Logging/Monitoring weaknesses (Medium): delayed detection → larger blast radius.

5. Detailed Multi-Layered Mitigation (Top 3)

5.1 SQL Injection – Defense in Depth

Code changes (PHP – PDO example):

```
<?php
// Use environment variables for secrets
$dsn = getenv("DB_DSN"); // e.g., mysql:host=127.0.0.1;dbname=mydb;port=3306
$user = getenv("DB_USER");
$pass = getenv("DB_PASS");
$pdo = new PDO($dsn, $user, $pass, [PDO::ATTR_ERRMODE =>
PDO::ERRMODE_EXCEPTION]);
$stmt = $pdo->prepare("SELECT id, password_hash FROM users WHERE username = ?");
$stmt->execute([$username]);
$row = $stmt->fetch(PDO::FETCH_ASSOC);
if ($row && password_verify($password, $row["password_hash"])) {
   // success: regenerate session id
   session_regenerate_id(true);
   echo "Login successful";
} else {
   echo "Invalid credentials";
}
?>
```

Configuration & infra:

• Create least-privilege DB role: only SELECT on authentication tables; no schema changes.

• Enable parameterized queries everywhere; ban concatenated SQL in code reviews.

• Deploy ModSecurity + OWASP CRS to block common SQLi payloads.

• Turn off verbose DB errors in production; log securely with correlation IDs.

• Automated SAST/DAST in CI/CD to catch regressions.

5.2 Authentication Hardening

Code & policy:

• Hash passwords with Argon2id (memory-hard) or bcrypt; use password_hash() / password_verify().

• Enforce strong password policy; add MFA (TOTP/WebAuthn) for privileged users.

• Rate limit login attempts (e.g., 5 per 15 minutes per account/IP) and add exponential backoff.

• Lock out and alert after repeated failures; provide safe self-service unlock.

• Regenerate session IDs after login; set cookies HttpOnly, Secure, SameSite=Lax/Strict; force HTTPS.

• Implement CSRF tokens on all state-changing actions.

Server headers (Apache example):
Header always set Strict-Transport-Security "max-age=31536000; includeSubDomains"
Header always set X-Frame-Options "DENY"
Header always set X-Content-Type-Options "nosniff"

5.3 XSS Prevention

Code changes: replace dangerous DOM sinks and encode output:

```
// BAD
results.innerHTML = 'Showing results for: ' + query;

// GOOD
results.textContent = 'Showing results for: ' + query;
// Or sanitize first: results.innerHTML = DOMPurify.sanitize(userHtml);
```

Security headers (Apache):
Header set Content-Security-Policy "default-src 'self'; script-src 'self'; object-src 'none'; base-uri 'self'; frame-ancestors 'none'"
Header set Referrer-Policy "no-referrer"

• Ban inline scripts and event handlers; use nonces or hashes if inline is unavoidable.

• Template engines with automatic context-aware encoding (e.g., Mustache, Twig).

• Unit tests for output encoding; SAST rules for dangerous sinks (innerHTML, document.write).

6. Breaking Things Mindset – Chained Attack Scenario

1. **Step 1 – XSS attack:**
    - The attacker puts a malicious script in the search page.
    - A support agent clicks the attacker's link.
    - The script steals the agent's login cookie.
2. **Step 2 – Use stolen session:**
    - With the cookie, the attacker logs in as the agent.

- They now have access to the admin panel.

3. **Step 3 – File upload attack:**
   - The attacker uploads a fake "image" that is actually a PHP web shell.
   - Because the site does not check uploads properly, it accepts the file.
4. **Step 4 – Steal database info:**
   - The attacker uses the web shell to read hidden files (like .env).
   - This reveals database username and password.
   - They then steal all user data from the database.

🛡️ **How our defenses stop it**

- **CSP + output encoding** → block the XSS at the start.
- **Secure cookies (HttpOnly, SameSite) & MFA** → even if XSS runs, cookies can't be stolen or abused easily.
- **File upload validation** → only real images are allowed, not PHP scripts.
- **Least-privilege DB user & secret management** → even if someone gets in, damage is very limited.

7. Use of AI Tools & Personal Contribution

AI tools (ChatGPT) were used to accelerate identification of common vulnerabilities, propose initial remediations, and draft secure code patterns. Limitations observed: AI suggestions can be generic, environment-agnostic, or omit operational context; all outputs were validated and tailored to the LAMP/DVWA setup. Personal contributions included: lab build and validation (Kali ↔ DVWA), hands-on exploitation (sqlmap, Hydra, XSS), prioritization based on impact to an e-commerce context, creation of multi-layer defenses (code, headers, WAF, ops), and assembling this report.

8. Tools Used

• AI: ChatGPT (analysis and drafting).

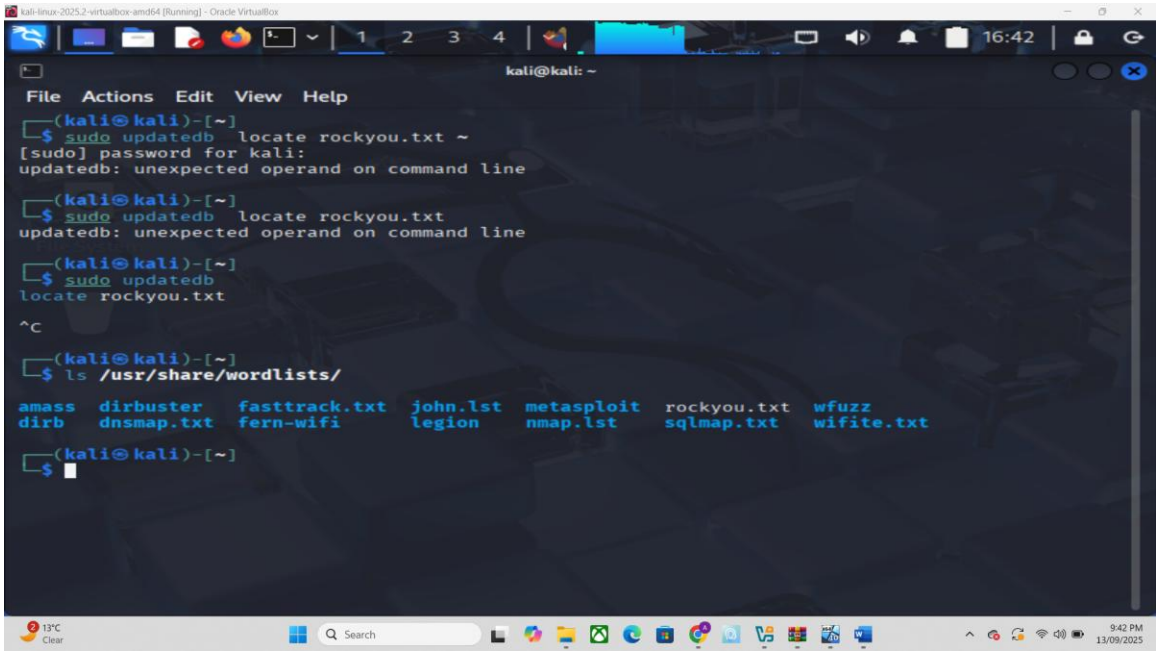
• Attack tools: sqlmap (SQLi), Hydra (brute force).

• Platform: Kali Linux, DVWA on Windows (WAMP), Oracle VirtualBox.

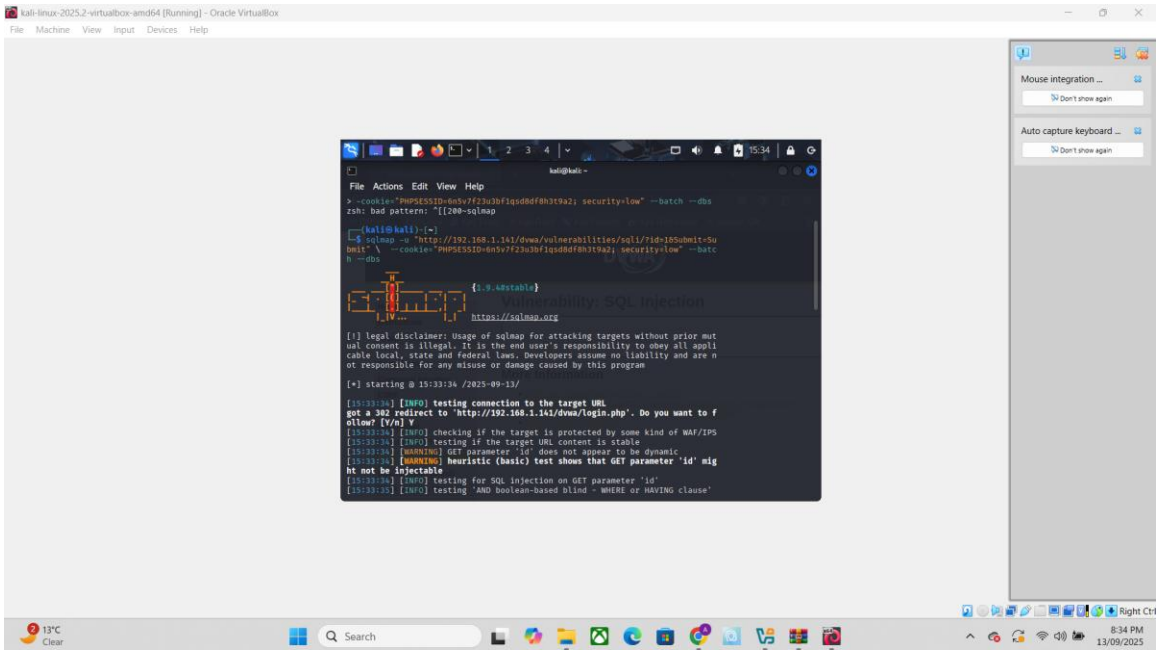
• References: OWASP Top 10 2021, PHP manual, Apache HTTP Server docs.

Appendix: Lab Evidence (Screenshots)

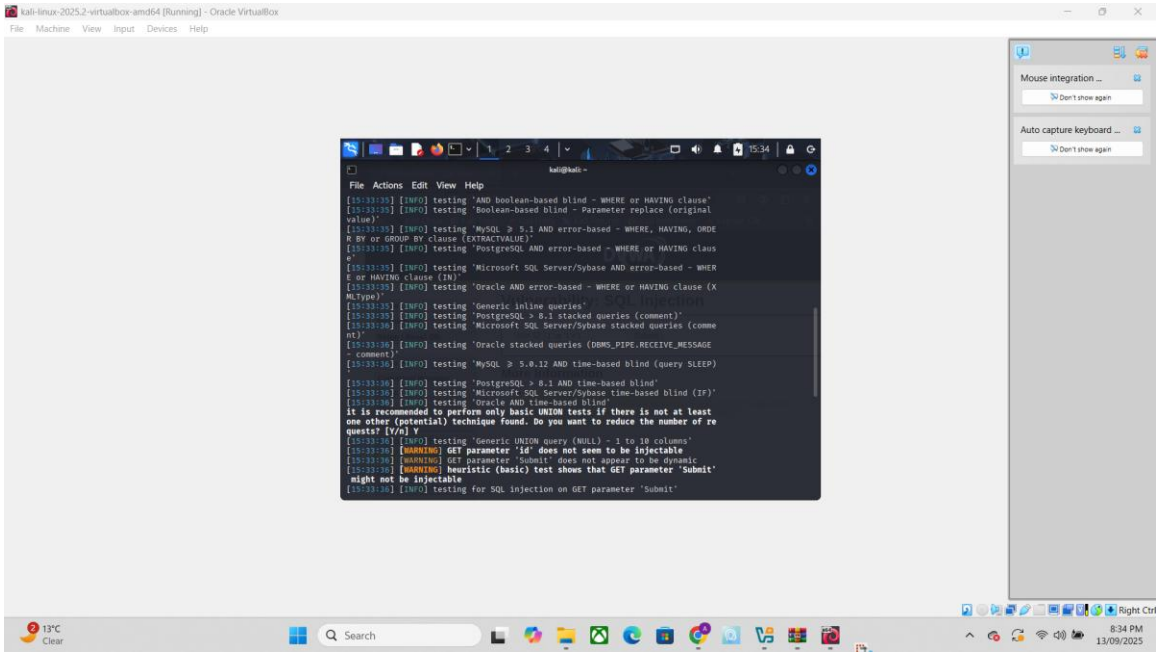
5b1ea63f-2332-4c07-9f46-338cf741dc94.png



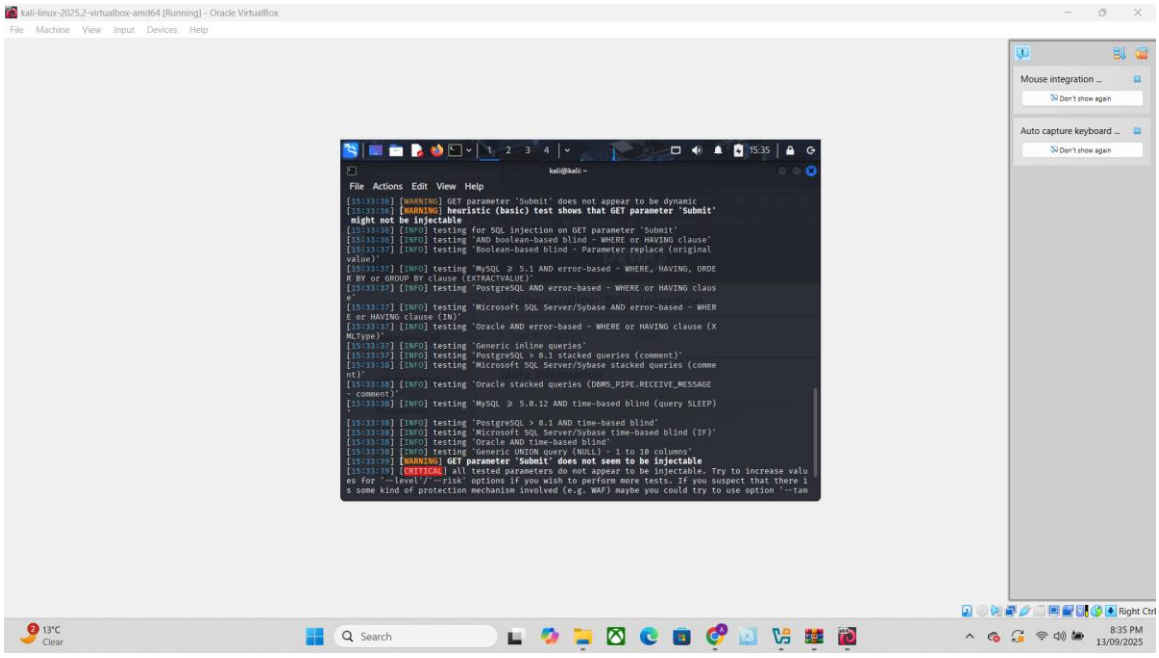
Screenshot 2025-09-13 203425.png



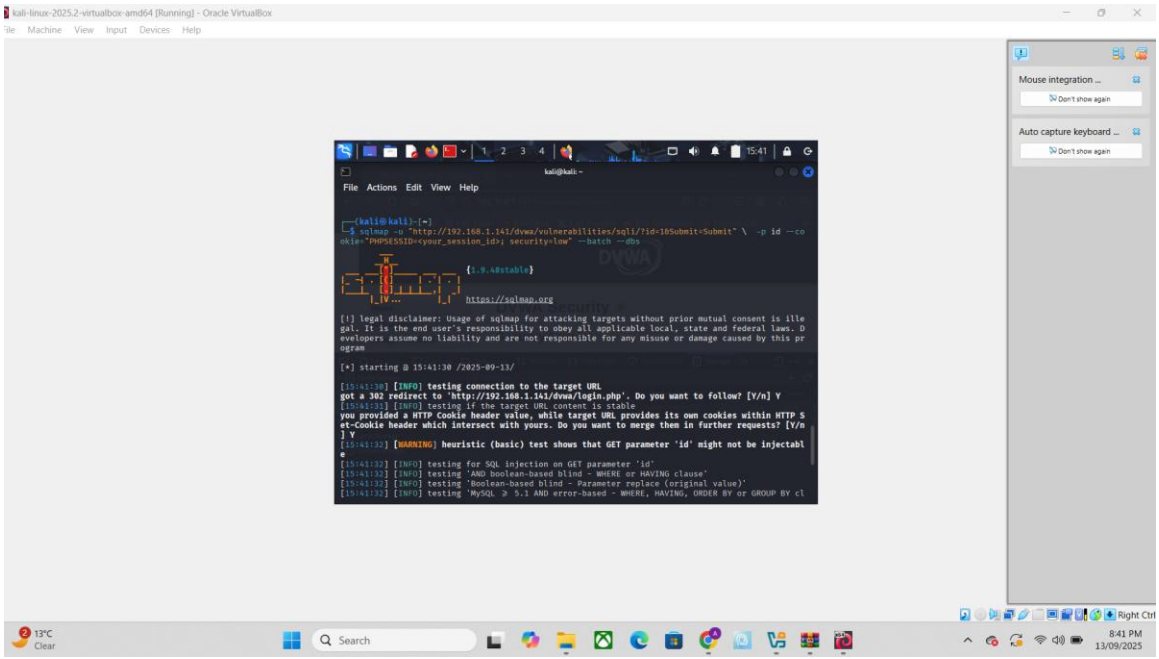
Screenshot 2025-09-13 203459.png



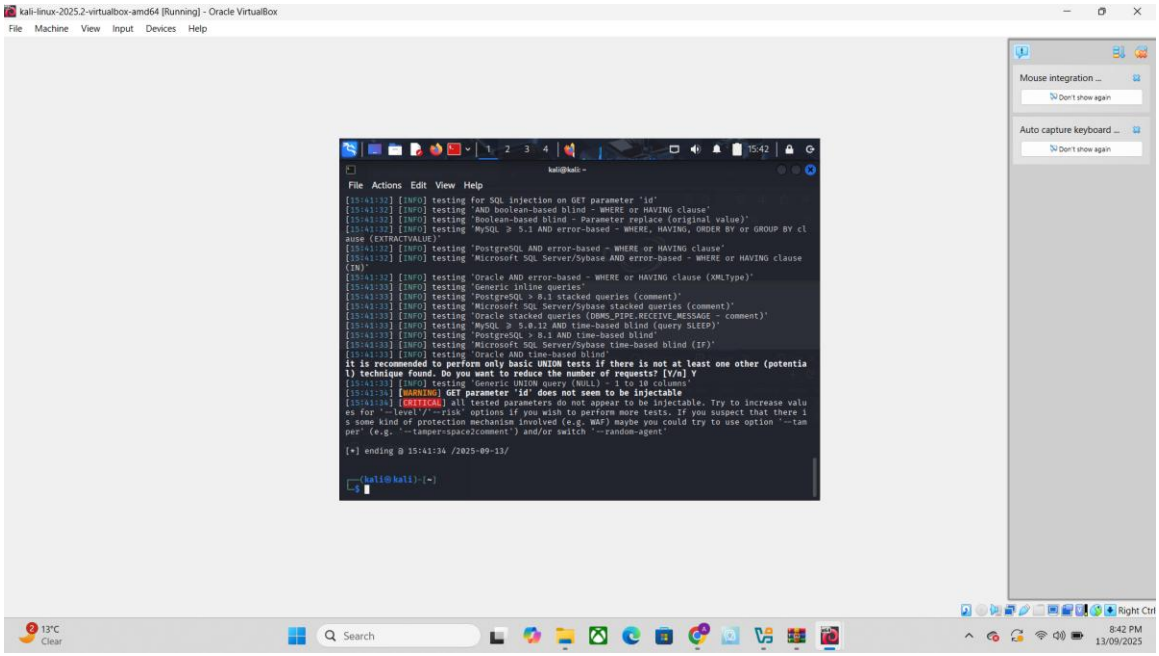
Screenshot 2025-09-13 203519.png



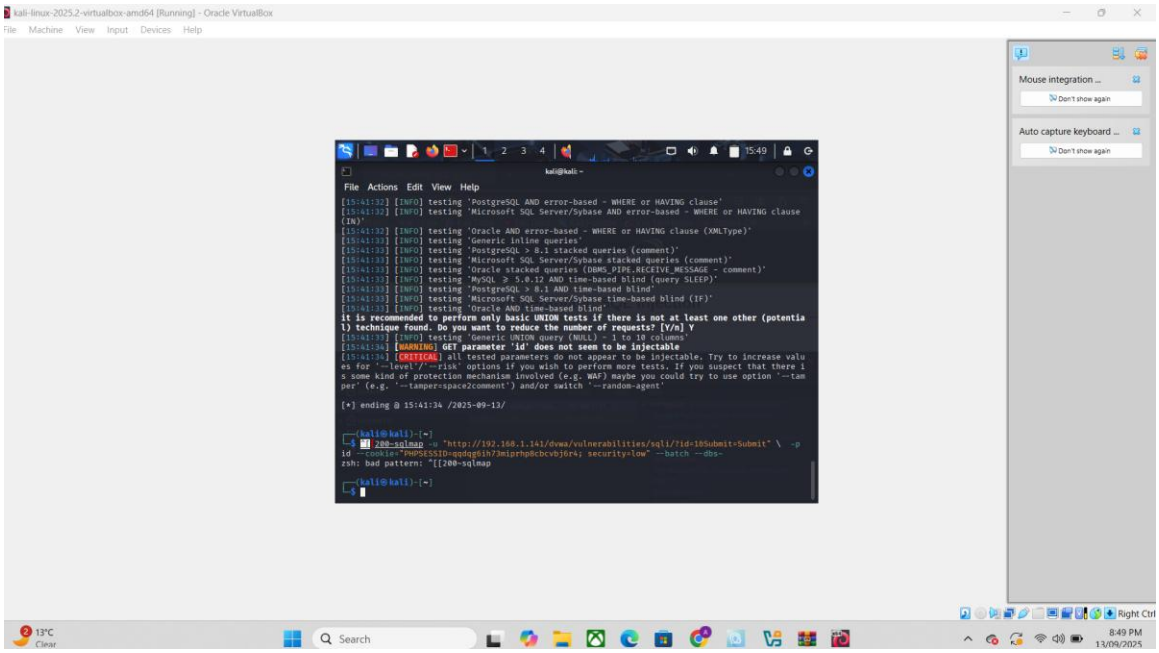
Screenshot 2025-09-13 204156.png



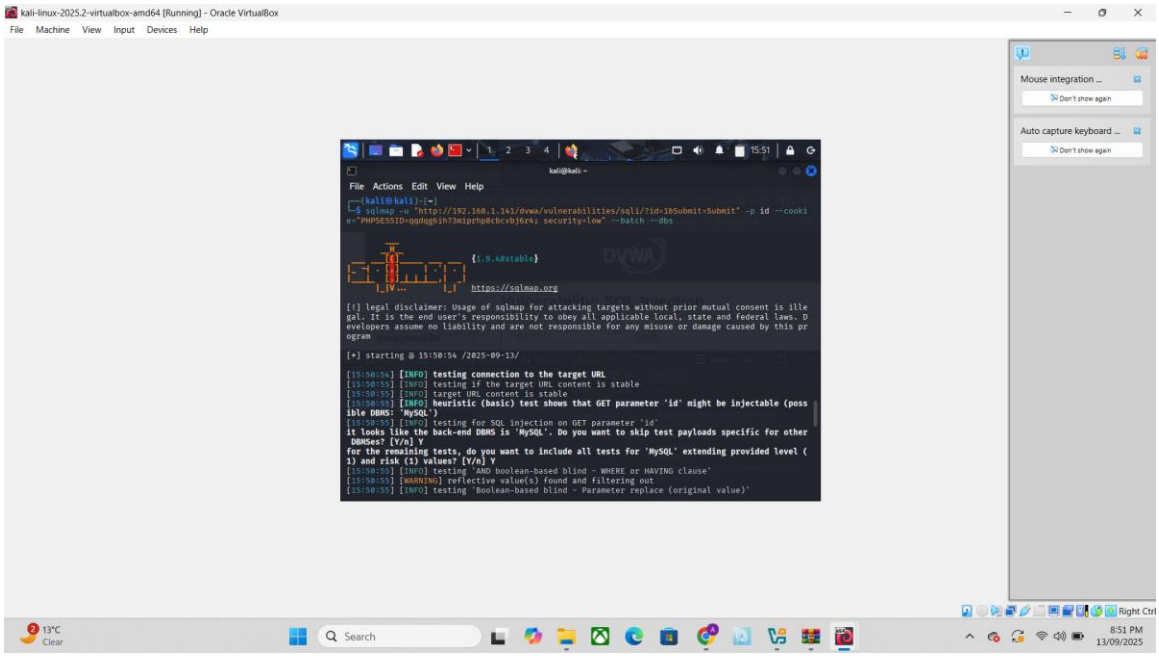
Screenshot 2025-09-13 204224.png



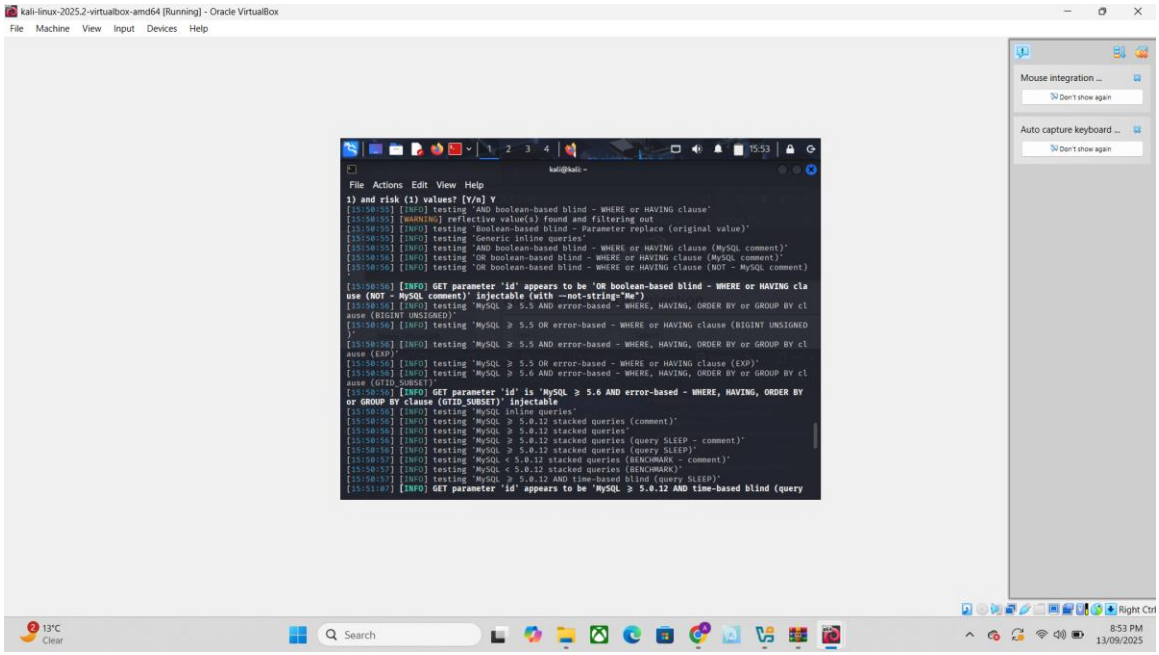
Screenshot 2025-09-13 204928.png



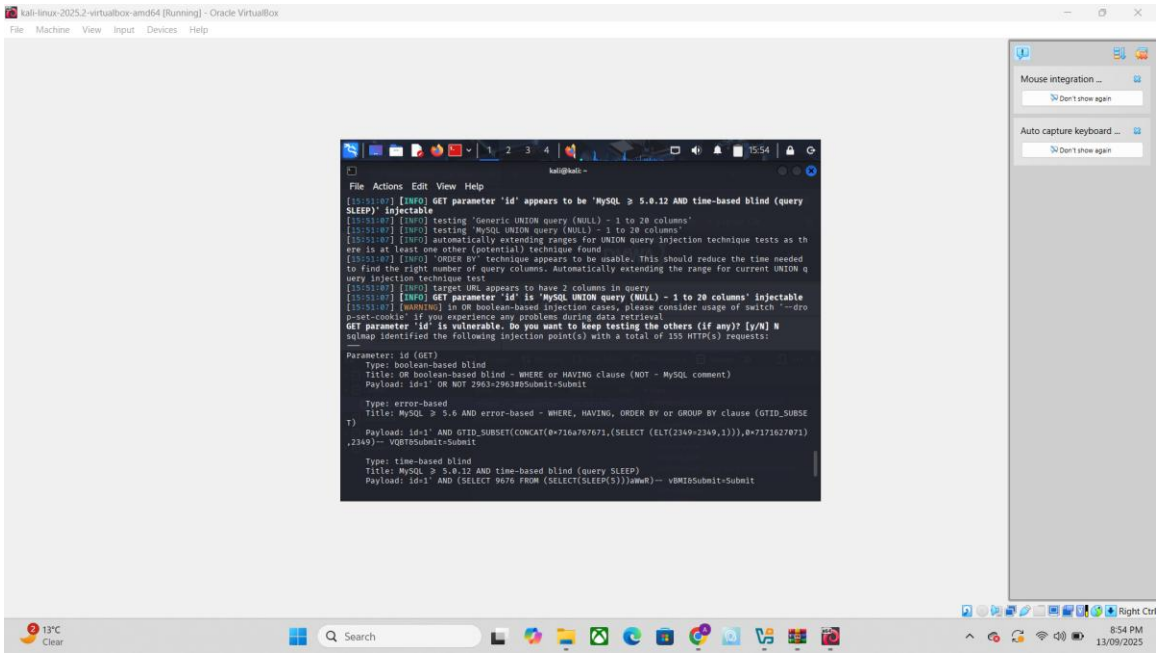
Screenshot 2025-09-13 205319.png



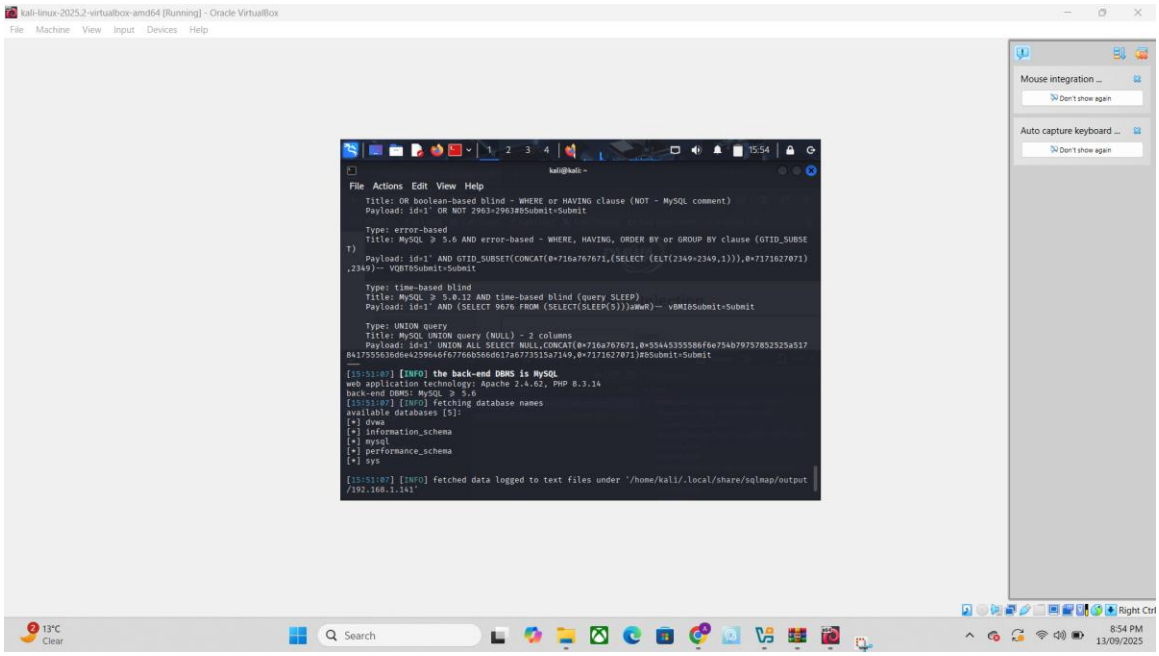
Screenshot 2025-09-13 205401.png



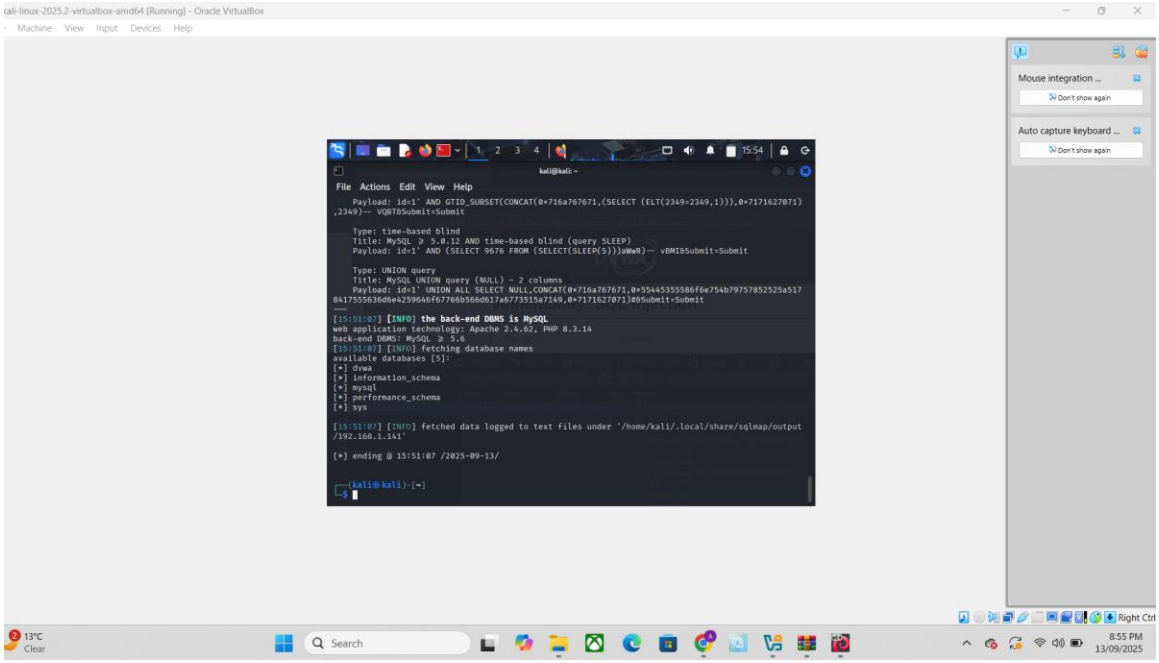
Screenshot 2025-09-13 205421.png



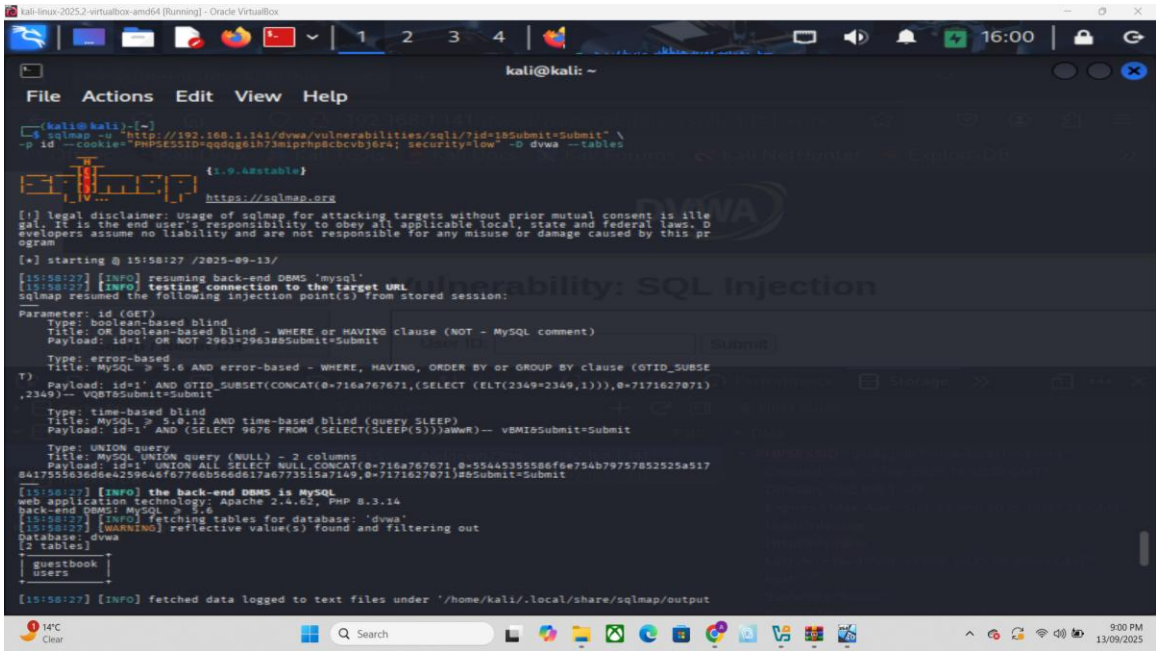
Screenshot 2025-09-13 205451.png



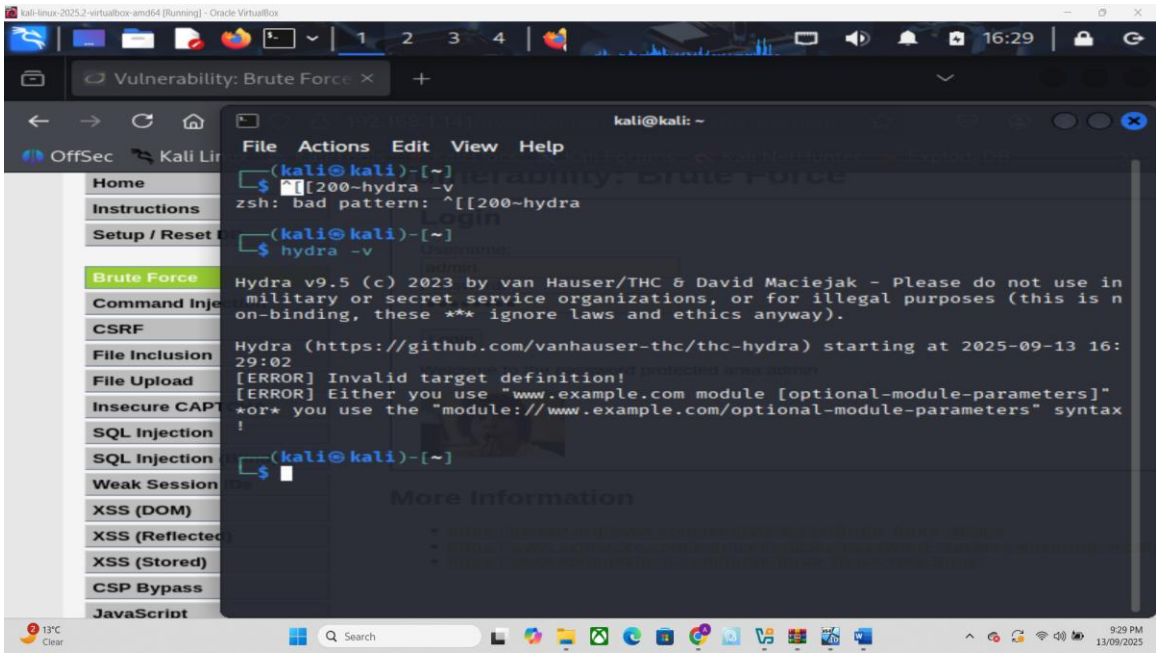
Screenshot 2025-09-13 205512.png



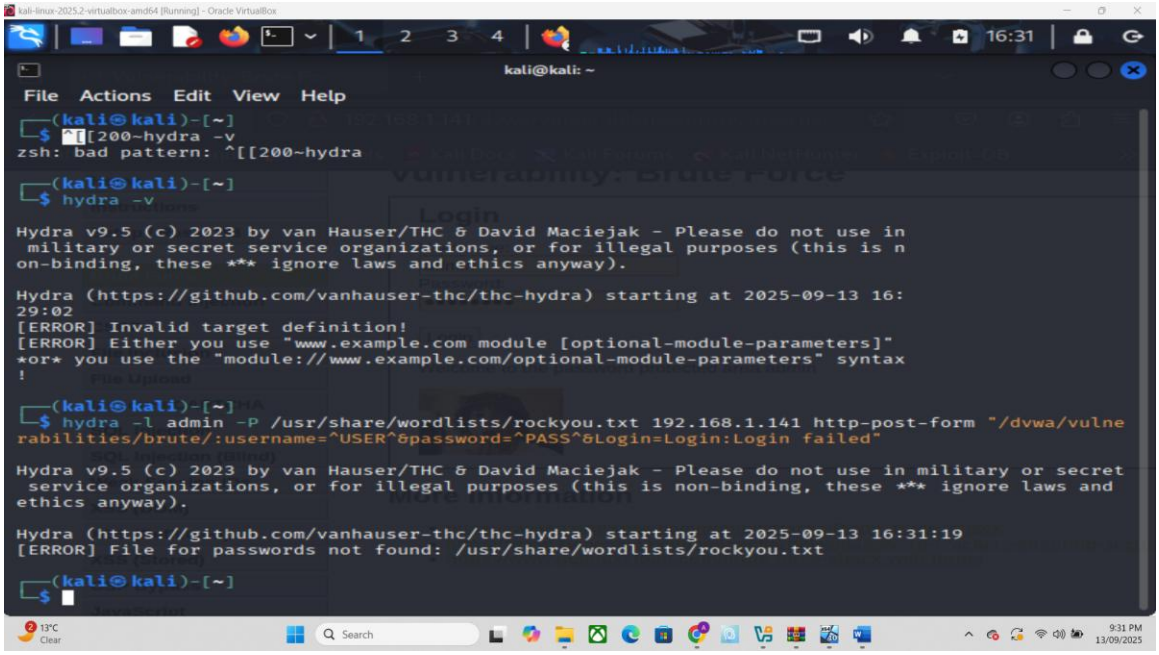
Screenshot 2025-09-13 210040.png



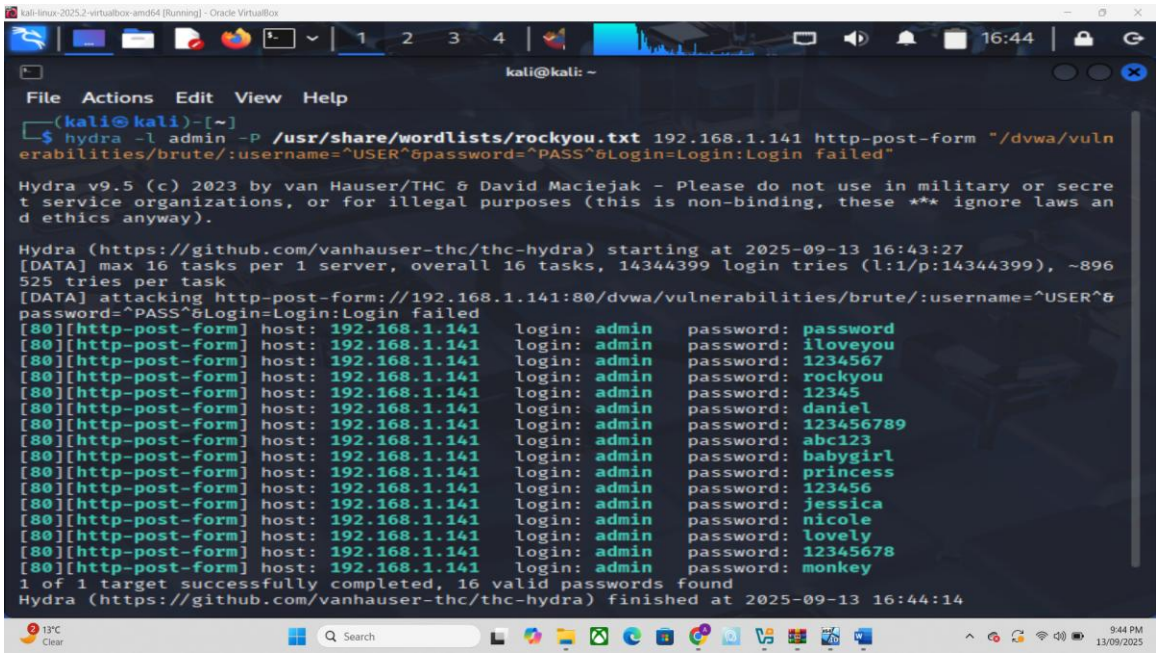
Screenshot 2025-09-13 212947.png



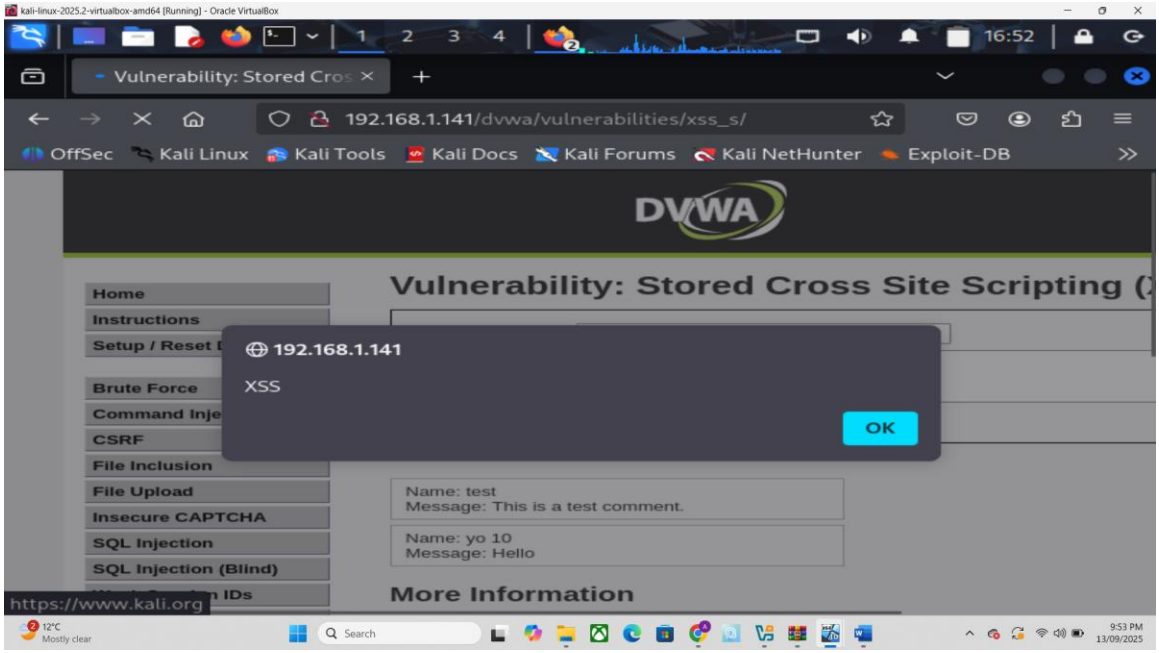
Screenshot 2025-09-13 213153.png

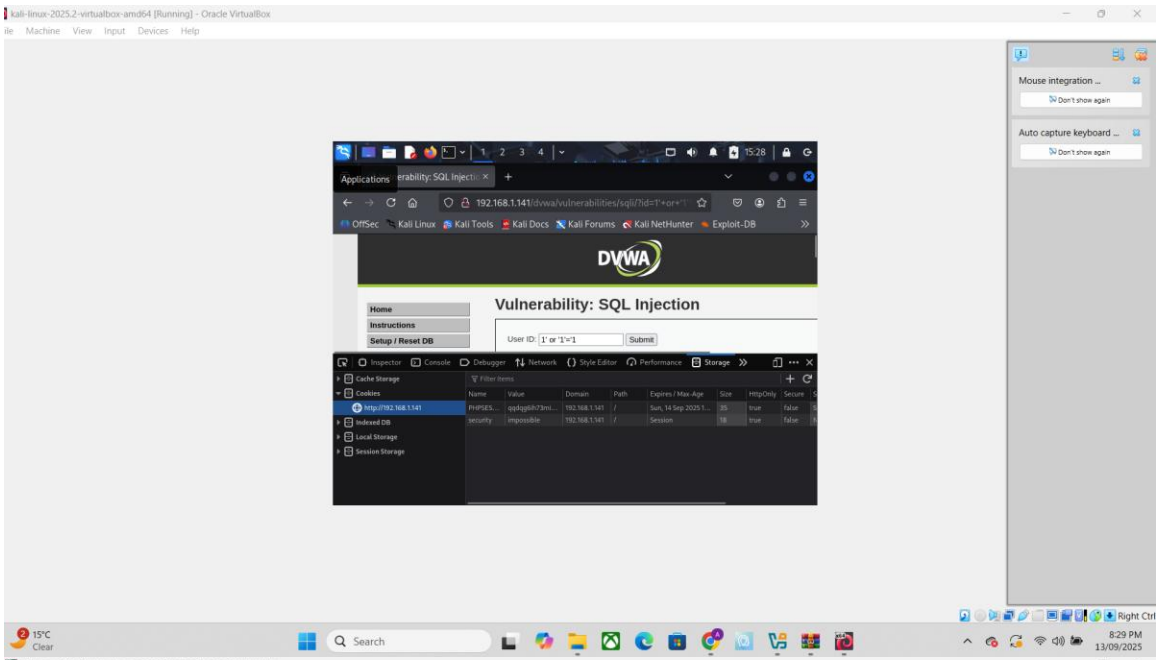


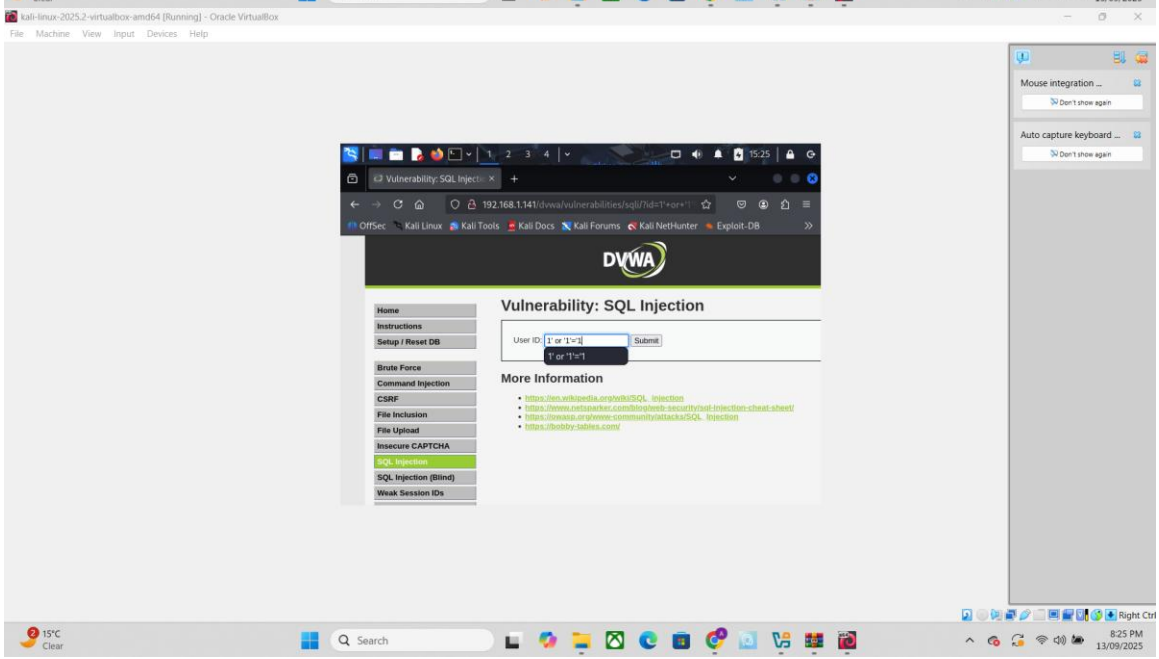
Screenshot 2025-09-13 214451.png

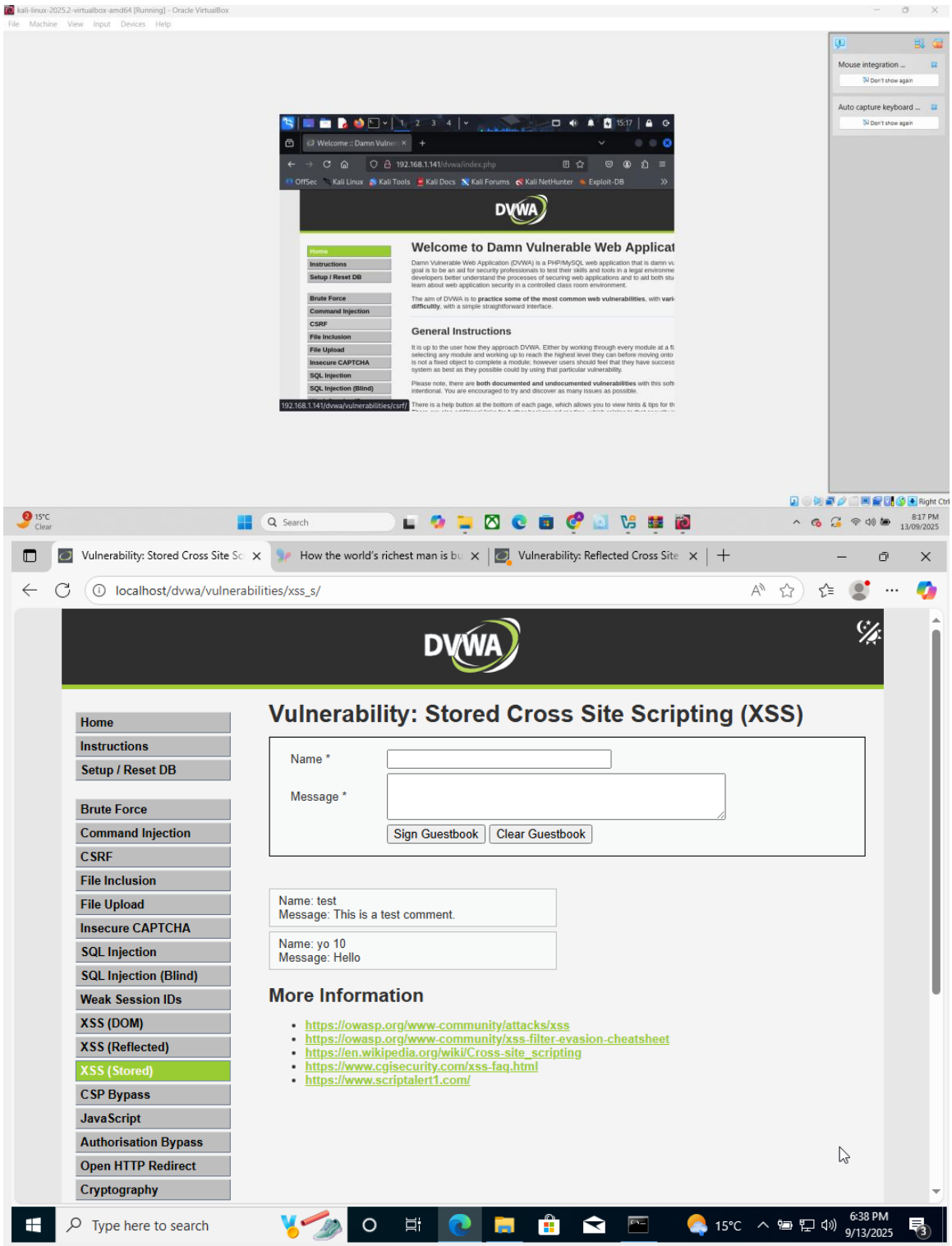


Screenshot 2025-09-13 215314.png




kali-linux-2025.2-virtualbox-amd64 [Running] - Oracle VirtualBox
Mouse integration ...
Auto capture keyboard ...
192.168.1.141/dvwa/vulnerabilities/sqli/?id=1'+or+'1
DVWA
Vulnerability: SQL Injection
Home
Instructions
Setup / Reset DB
User ID: 1' or '1'='1
Submit
Search
8:29 PM
13/09/2025


kali-linux-2025.2-virtualbox-amd64 [Running] - Oracle VirtualBox
Mouse integration ...
Auto capture keyboard ...
192.168.1.141/dvwa/vulnerabilities/sqli/?id=1'+or+'1
DVWA
Vulnerability: SQL Injection
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
User ID: 1' or '1'='1
Submit
1' or '1'='1
More Information
Search
8:25 PM
13/09/2025


Vulnerability: Stored Cross Site Scripting (XSS)
Name *
Message *
Sign Guestbook
Clear Guestbook
Name: test
Message: This is a test comment.
Name: yo 10
Message: Hello
More Information
https://owasp.org/www-community/attacks/xss
https://owasp.org/www-community/xss-filter-evasion-cheatsheet
https://en.wikipedia.org/wiki/Cross-site_scripting
https://www.cgisecurity.com/xss-faq.html
https://www.scriptalert1.com/
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript
Authorisation Bypass
Open HTTP Redirect
Cryptography
localhost/dvwa/vulnerabilities/xss_s/