

Project:3

Automated vs Manual Testing Project

DVWA Security Assessment

This project compares the effectiveness of automated security scanners with manual testing techniques using DVWA as the target application.

Objective

The objectives of this project were:

- To perform automated security testing on DVWA using common tools
 - To manually test the same application using interception and crafted payloads
 - To document and compare findings from both approaches
 - To analyze strengths and weaknesses of automated scanners
-

Scope of Testing

The scope of this assessment included:

- DVWA running locally on Kali Linux
- Web application testing only

All testing was performed in a controlled lab environment.

Tools Used

- Kali Linux
 - DVWA
 - Nikto
 - OWASP ZAP
 - sqlmap
 - Burp Suite
 - Firefox Browser
-

Methodology

The testing methodology followed these phases:

- Automated vulnerability scanning
 - Manual interception and testing
 - Validation of findings
 - Comparison and analysis
-

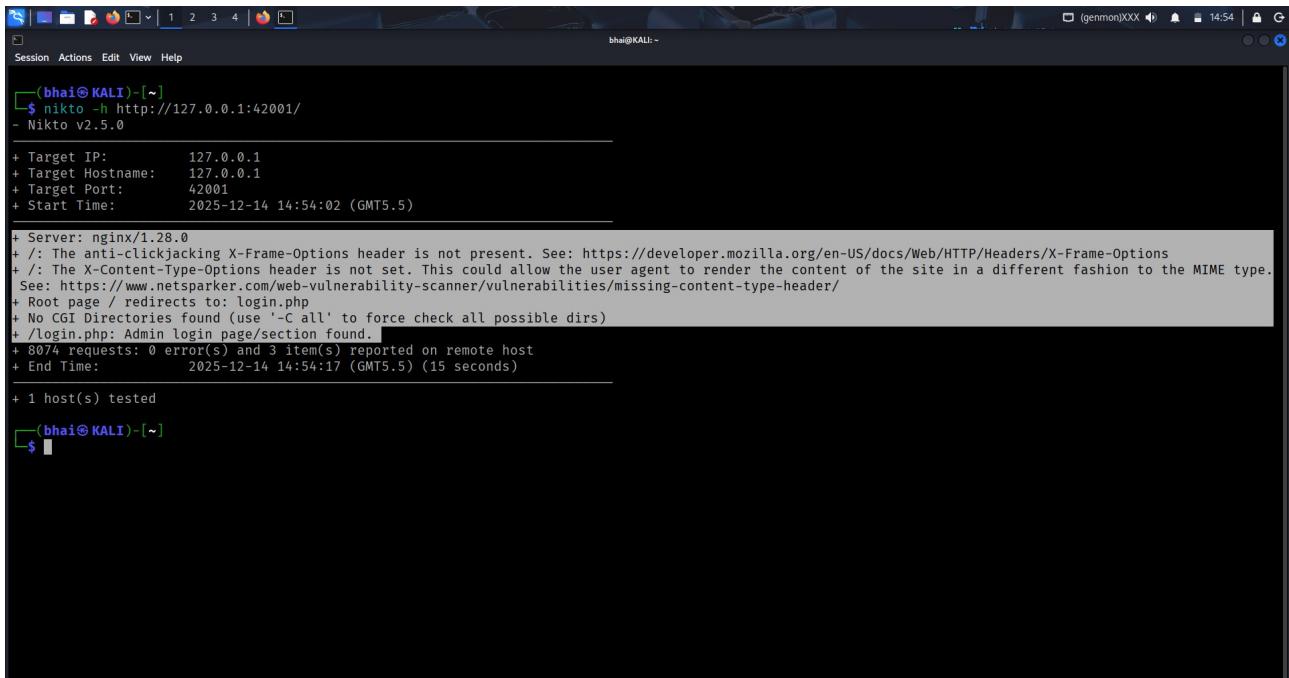
Automated Testing on DVWA

Nikto Scan

Nikto was used to identify web server misconfigurations and known issues.

Command used:

```
nikto -h http://127.0.0.1:42001/
```



```
(bhai㉿KALI)-[~]
$ nikto -h http://127.0.0.1:42001/
- Nikto v2.5.0

+ Target IP:      127.0.0.1
+ Target Hostname: 127.0.0.1
+ Target Port:    42001
+ Start Time:    2025-12-14 14:54:02 (GMT5.5)

+ Server: nginx/1.28.0
+ /: The anti-clickjacking X-Frame-Options header is not present. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /: The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type.
See: https://www.netsparker.com/web-vulnerability-scanner/vulnerabilities/missing-content-type-header/
+ Root page / redirects to: login.php
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ /login.php: Admin login page/section found.
+ 8074 requests: 0 error(s) and 3 item(s) reported on remote host
+ End Time:        2025-12-14 14:54:17 (GMT5.5) (15 seconds)

+ 1 host(s) tested

(bhai㉿KALI)-[~]
$
```

Findings:

- Missing HTTP security headers
- Outdated server configuration
- Information disclosure issues, page redirection

Nikto focused mainly on server-level weaknesses rather than application logic.

OWASP ZAP Scan

OWASP ZAP was used to perform an automated web application scan.

Steps performed:

- DVWA opened in browser through ZAP
- Spidering enabled to discover pages
- Active scan initiated

The screenshot shows the OWASP ZAP 2.16.1 interface running on a Kali Linux VM. The main window displays a table of network traffic captured during a scan of the DVWA application at http://127.0.0.1:42001. The table includes columns for ID, Req. Timestamp, Resp. Timestamp, Method, URL, Code, Reason, RTT, Size Resp. Header, and Size Resp. Body. The scan progress bar at the top indicates 40% completion. The bottom status bar shows current status metrics like CPU, RAM, and disk usage.

The screenshot shows the OWASP ZAP 2.16.1 interface running on a Kali Linux VM. The main window displays the 'Alerts' tab, which lists a single issue: 'Content Security Policy (CSP) Header Not Set'. The alert details include the URL (http://127.0.0.1:42001/sitemap.xml), risk level (Medium), confidence (High), and a detailed description of the CSP header issue. The bottom status bar shows current status metrics like CPU, RAM, and disk usage.

Findings:

- Content policy header missing
 - Cookie security issues without samesite parameter
 - Missing security headers
 - Server Leak version

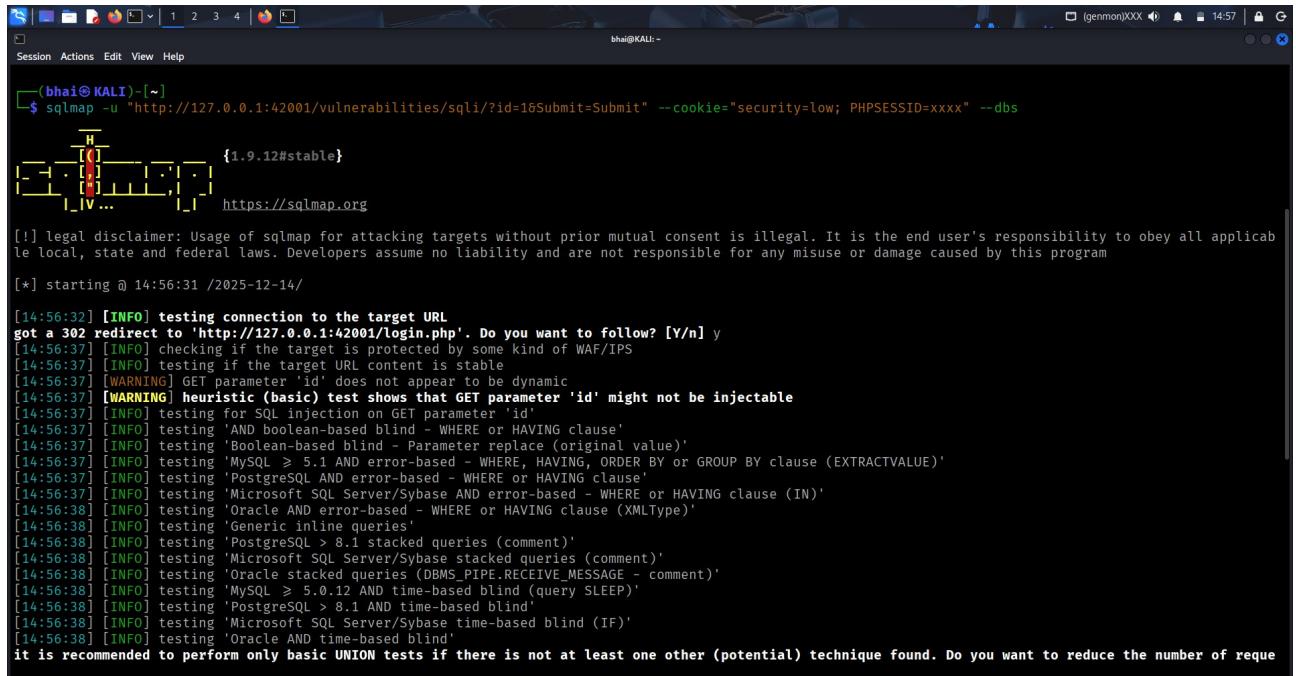
ZAP provided broad coverage but also generated some low-confidence alerts.

sqlmap Scan

sqlmap was used to test for SQL Injection vulnerabilities.

Command used:

```
sqlmap -u "http://127.0.0.1/42001/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=xxxx" -dbs
```



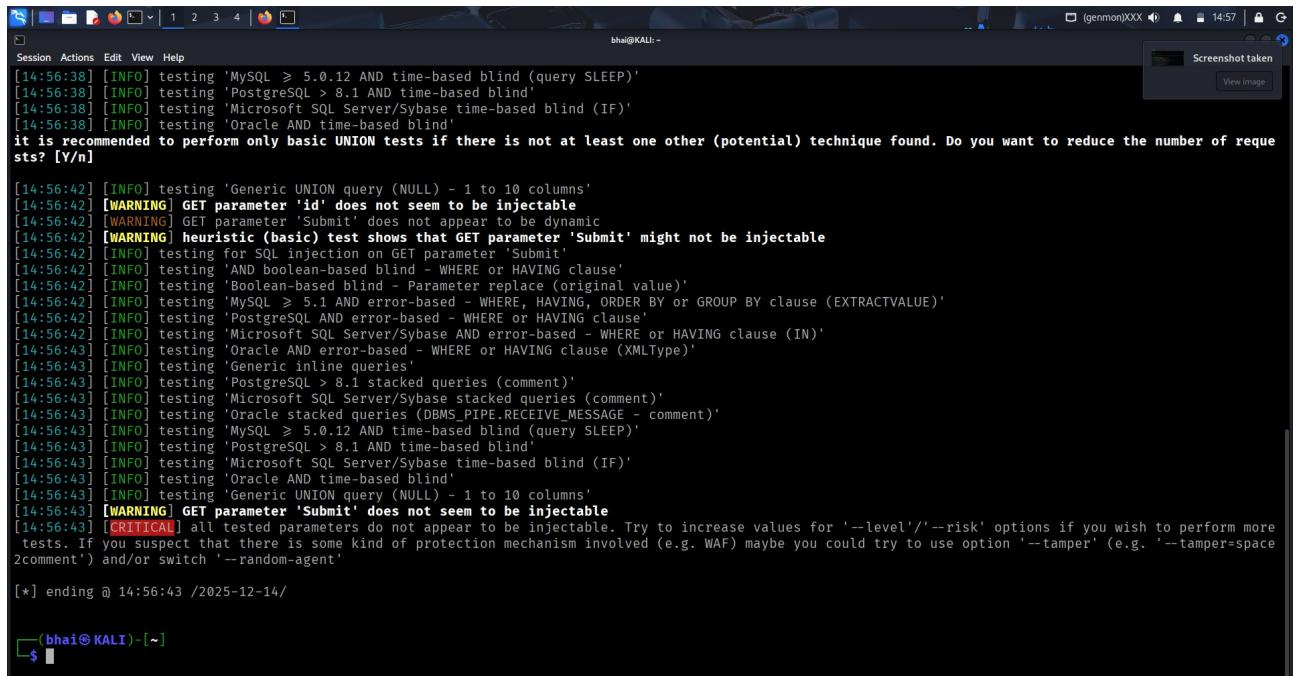
```
bhai@KALI: ~
$ sqlmap -u "http://127.0.0.1:42001/vulnerabilities/sqli/?id=1&Submit=Submit" --cookie="security=low; PHPSESSID=xxxx" -dbs

{1.9.12#stable}
https://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 are not responsible for any misuse or damage caused by this program

[*] starting @ 14:56:31 /2025-12-14

[14:56:32] [INFO] testing connection to the target URL
got a 302 redirect to 'http://127.0.0.1:42001/login.php'. Do you want to follow? [Y/n] y
[14:56:37] [INFO] checking if the target is protected by some kind of WAF/IPS
[14:56:37] [INFO] testing if the target URL content is stable
[14:56:37] [WARNING] GET parameter 'id' does not appear to be dynamic
[14:56:37] [WARNING] heuristic (basic) test shows that GET parameter 'id' might not be injectable
[14:56:37] [INFO] testing for SQL injection on GET parameter 'id'
[14:56:37] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:56:37] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:56:37] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[14:56:37] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[14:56:37] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[14:56:38] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[14:56:38] [INFO] testing 'Generic inline queries'
[14:56:38] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[14:56:38] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[14:56:38] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[14:56:38] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[14:56:38] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[14:56:38] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[14:56:38] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n]
```



```
bhai@KALI: ~
$ Screenshot
View image

[14:56:38] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[14:56:38] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[14:56:38] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[14:56:38] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n]

[14:56:42] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:56:42] [WARNING] GET parameter 'id' does not seem to be injectable
[14:56:42] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[14:56:42] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[14:56:42] [INFO] testing for SQL injection on GET parameter 'Submit'
[14:56:42] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[14:56:42] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[14:56:42] [INFO] testing 'MySQL > 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[14:56:42] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[14:56:42] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[14:56:43] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[14:56:43] [INFO] testing 'Generic inline queries'
[14:56:43] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[14:56:43] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[14:56:43] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[14:56:43] [INFO] testing 'MySQL > 5.0.12 AND time-based blind (query SLEEP)'
[14:56:43] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[14:56:43] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[14:56:43] [INFO] testing 'Oracle AND time-based blind'
[14:56:43] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[14:56:43] [WARNING] GET parameter 'Submit' does not seem to be injectable
[14:56:43] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=spacecomment') and/or switch '--random-agent'

[*] ending @ 14:56:43 /2025-12-14

(bhai@KALI)-[~]
$
```

Findings:

- GET parameter 'Submit' does not seem to be injectable

- GET parameter 'id' does not seem to be injectable
- GET parameter 'Submit' does not appear to be dynamic
- heuristic (basic) test shows that GET parameter 'Submit' might not be injectable

sqlmap successfully automated exploitation of SQL Injection without manual payload crafting.

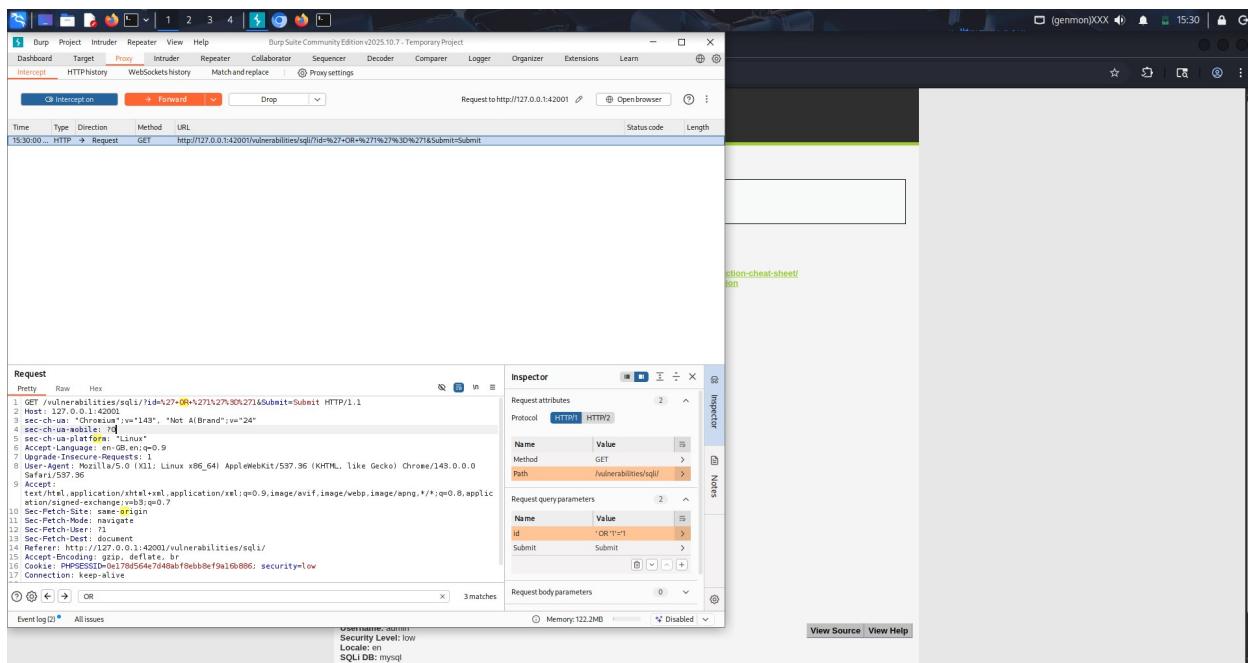
Manual Testing on DVWA

Manual SQL Injection Testing

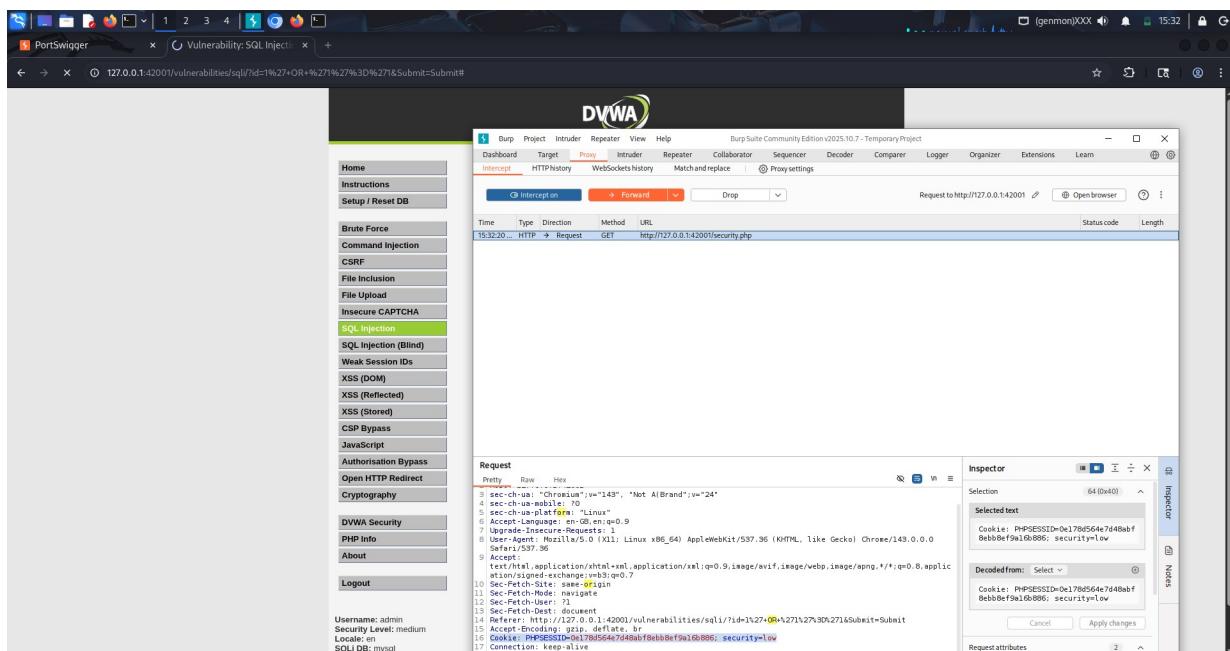
Manual SQL Injection testing was performed using crafted payloads.

Payload used:

```
' OR '1'='1
```



The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A request is being sent to the URL `http://127.0.0.1:42001/vulnerabilities/sql?id=1%27%27OR%27%27&Submit=Submit`. The 'Inspector' tool window is open, showing the raw request and its attributes. The 'Request parameters' section has the 'id' parameter set to '`' OR '1'='1`'. The response pane is empty, indicating no immediate results.



The screenshot shows the DVWA application's 'SQL Injection' page. The sidebar menu is expanded, showing various attack types, with 'SQL Injection' currently selected. The main content area displays the exploit payload being sent via Burp Suite. The Burp Suite interface shows the request being sent to the URL `http://127.0.0.1:42001/vulnerabilities/sql?id=1%27%27OR%27%27%27%27&Submit=Submit`. The 'Inspector' tool window in Burp Suite shows the raw request and its attributes, with the 'id' parameter set to '`' OR '1'='1`'. The response pane is empty.

Observation:

- Authentication bypass was successful
- Database results were manipulated

Manual testing helped understand how the vulnerability actually worked.

Manual Cross-Site Scripting Testing

Manual XSS testing was performed using input fields.

Payload used:

```
<script>alert('XSS')</script>
```

The screenshot shows the DVWA Reflected XSS page. The URL is http://127.0.0.1:42001/vulnerabilities/xss_r/. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left, there's a sidebar with various attack types: 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) (which is highlighted in green), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, Cryptography, DVWA Security, PHP Info, About, and Logout. The main content area has a form with the placeholder "What's your name? <script>alert('XSS')</script>" and a "Submit" button. Below the form, there's a "More Information" section with links to XSS resources. At the bottom, it shows the user is "admin", security level is "low", locale is "en", and the database is "MySQL". There are "View Source" and "View Help" buttons at the bottom right. The status bar at the bottom says "https://en.wikipedia.org/wiki/Cross-site_scripting".

The screenshot shows the Burp Suite interface. The top bar shows "Burm Suite Community Edition v2025.10.7 - Temporary Project". The menu bar includes Burp, Project, Intruder, Repeater, View, Help. The toolbar has icons for Intercept, Forward, Drop, and a proxy settings dropdown. The main pane shows a timeline with a single entry: "15:34:55 14 Dec 2025 HTTP 1 Request GET http://127.0.0.1:42001/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27%27%3C%2Fscript%3E HTTP/1.1". The status bar at the bottom says "Request to http://127.0.0.1:42001 [P] [Open browser] [D] [E]".

This screenshot shows the Burp Suite Request and Inspector panes. The Request pane displays the raw HTTP request with the XSS payload: "GET /vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27%27%3C%2Fscript%3E HTTP/1.1". The Inspector pane shows the selected text as "GET /vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27%27%3C%2Fscript%3E HTTP/1.1". It also shows the decoded text: "GET /vulnerabilities/xss_r/?name=<script>alert('XSS')</script>". The Protocol tab in the Inspector pane is set to "HTTP/1". The status bar at the bottom says "Event log (2) All issues".

Observation:

- JavaScript executed in the browser
- Alert popup confirmed XSS

Manual testing allowed precise payload control and confirmation.

Burp Suite Interception

Burp Suite was used to intercept and modify HTTP requests.

The screenshot shows the Burp Suite interface with the 'Proxy' tab selected. A single request is listed in the main pane, which is a GET request to 'http://127.0.0.1:42001/vulnerabilities/sql盲'. The 'Inspector' panel on the right displays the request details, including the method (GET), path ('/vulnerabilities/sql盲'), and various headers such as User-Agent, Accept, and Sec-Fetch-Site. The 'Request' pane at the bottom shows the raw HTTP message. The status bar at the top right indicates the time as 15:38 and the host as (genmon)XXX.

Observation:

- Parameters could be manipulated
- Hidden inputs were identified
- Request and response behavior was clearly visible

Burp helped in understanding application logic and data flow.

Comparison of Automated and Manual Testing

Automated scanners quickly identified common vulnerabilities such as missing headers, SQL Injection, and basic XSS. They saved time and provided wide coverage of the application.

Manual testing allowed deeper understanding of vulnerabilities, precise payload crafting, and validation of exploitability. Some issues, such as logical flaws and parameter manipulation, were better identified manually.

Automated tools sometimes produced false positives, while manual testing required more skill and time.

Issues Identified by Automation Only

- Missing HTTP security headers
 - Server misconfigurations
 - Low-risk informational findings
 - Broad vulnerability patterns
-

Issues Identified by Manual Testing Only

- Precise SQL Injection behavior
 - Context-specific XSS payload execution
 - Parameter manipulation
 - Application logic understanding
-

Executive Summary

This assessment compared automated and manual security testing techniques on DVWA. Automated tools efficiently detected common vulnerabilities and configuration issues, while manual testing provided deeper insight and accurate validation of exploitability.

The results demonstrate that relying on only one approach is insufficient for comprehensive security testing.

Strengths and Weaknesses of Automated Scanners

Strengths:

- Fast and efficient scanning
- Broad coverage of application
- Useful for initial reconnaissance
- Easy to use for beginners

Weaknesses:

- False positives
- Limited understanding of application logic

- Cannot fully validate exploit impact
 - Miss complex or chained vulnerabilities
-

Conclusion

Automated scanners and manual testing complement each other in web application security assessments. Automated tools are ideal for initial discovery, while manual testing is essential for accurate validation and deeper analysis.

A balanced approach using both methods provides the most effective and reliable security testing results.

Learning Outcomes

- Understanding of automated and manual testing approaches
- Hands-on experience with Nikto, ZAP, sqlmap, and Burp Suite
- Improved vulnerability analysis skills
- Better understanding of tool limitations
- Professional security assessment reporting experience