Project 3 — Automated vs. Manual Testing: DVWA

Author: Isha Bachhav
Course / Project: Automated vs Manual Testing Project

1. Executive Summary

This project compares automated vulnerability scanners (Nikto, OWASP ZAP, sqlmap) with manual testing techniques (Burp Suite, crafted payloads, manual inspection) against the Damn Vulnerable Web Application (DVWA). The goal is to document what each approach finds, identify gaps, and draw conclusions about when automation is sufficient and when manual testing is necessary.

High-level takeaway: Automated scanners quickly discover many surface-level issues and speed up reconnaissance, but they miss context-specific, chained, or logic flaws that manual testing uncovers. A combined approach yields the best coverage.

2. Scope & Rules of Engagement

Target: DVWA in isolated lab environment (e.g., http://10.0.2.20)

Tools: Nikto, OWASP ZAP, sqlmap (automation) — Burp Suite, manual payloads, browser inspection (manual)

Timeframe: [Insert test dates]

Permissions: Lab-only environment with explicit permission.

3. Environment & Setup Summary

Kali Linux (attacker) with required tools installed.

DVWA hosted locally via Docker or VM with security level set to Low/Medium/High depending on test phase.

Commands to run DVWA in Docker (quick repeat):

```
docker pull vulnerables/web-dvwa
docker run --rm -it -p 80:80 vulnerables/web-dvwa
```

4. Tools & Purpose

Automated:

Nikto: webserver misconfigurations, outdated server software, common vulnerabilities

OWASP ZAP: spidering, passive and active scanning, fuzzing, baseline vulnerability checks

sqlmap: automated SQL injection detection and exploitation

Manual:

Burp Suite (intercept & repeater): craft precise payloads, manipulate cookies and parameters, analyze logic flows

Browser tools: DOM inspection, JavaScript console

Custom payloads & manual verification: test chaining, business logic, and authentication flows

5. Methodology

Reconnaissance: Run quick Nmap and fingerprinting to confirm services.

Automated scans: Run Nikto, ZAP spider + active scan, sqlmap against known injection points.

Manual testing: Use Burp Suite to intercept forms and craft payloads for SQLi, XSS, command injection, and business logic faults.

Cross-check: Compare outputs, manually verify every automated finding, and document missed issues.

Summarize & conclude.

6. Commands & Run Examples

Nikto:

```
nikto -h http://10.0.2.20 -o nikto_dvwa.txt
```

OWASP ZAP (headless baseline + active):

```
# Start ZAP GUI or use docker/zap-baseline.py script
zap-baseline.py -t http://10.0.2.20 -r zap_report.html
```

sqlmap (example):

```
sqlmap -u "http://10.0.2.20/vulnerabilities/sqli/?id=1&Submit=Submit" --batch --dbs -o
sqlmap_output.txt
```

Burp (manual):

Intercept a login/post request, send to Repeater, inject 1' OR '1'='1 and analyze response.

Test XSS by injecting <script>alert(1)</script> in comments and verifying execution.

7. Automated Scan Findings (Examples and Interpretation)

Nikto

Found server banner, common files (robots.txt), and potential outdated components. Output included Warning: /phpinfo.php (example).

Notes: Good for quickly identifying server misconfigs and known bad endpoints.

OWASP ZAP

Spider discovered main application pages and forms.

Passive scan flagged missing security headers (Content-Security-Policy, X-Frame-Options) and missing HttpOnly cookie flag.

Active scan reported potential XSS and SQL injection in some form fields (requires manual verification).

sqlmap

Confirmed SQL injection on /vulnerabilities/sqli/ when allowed; dumped DB names and some tables when run against Low security.

Automated Findings — summary:

Fast detection of common issues: server banners, missing headers, some input-based vulnerabilities (SQLi when trivial), default files.

Generated a baseline report and suggestions for immediate fixes.

8. Manual Testing Findings (Examples and PoC)

8.1 SQL Injection — Deep verification & exploitation

Manual payload 1' OR '1'='1 in id parameter returned full table data in Low. Verified that UNION SELECT responses return columns, enabling data extraction.

Why manual mattered: Tuned payloads and observed application context; exploited logic differences not caught by automated scans (e.g., parameterized filters in Medium that required advanced payload encoding).

8.2 Stored XSS — Contextual & DOM-based XSS

Injected <script>fetch('/admin/secret').then(r=>r.text()).then(t=>console.log(t))</script> in comment field to observe DOM-based behaviors and to test theft of dynamic content.

Why manual mattered: Automated scanners flagged generic XSS but missed DOM-based and context-dependent execution paths.

8.3 Command Injection

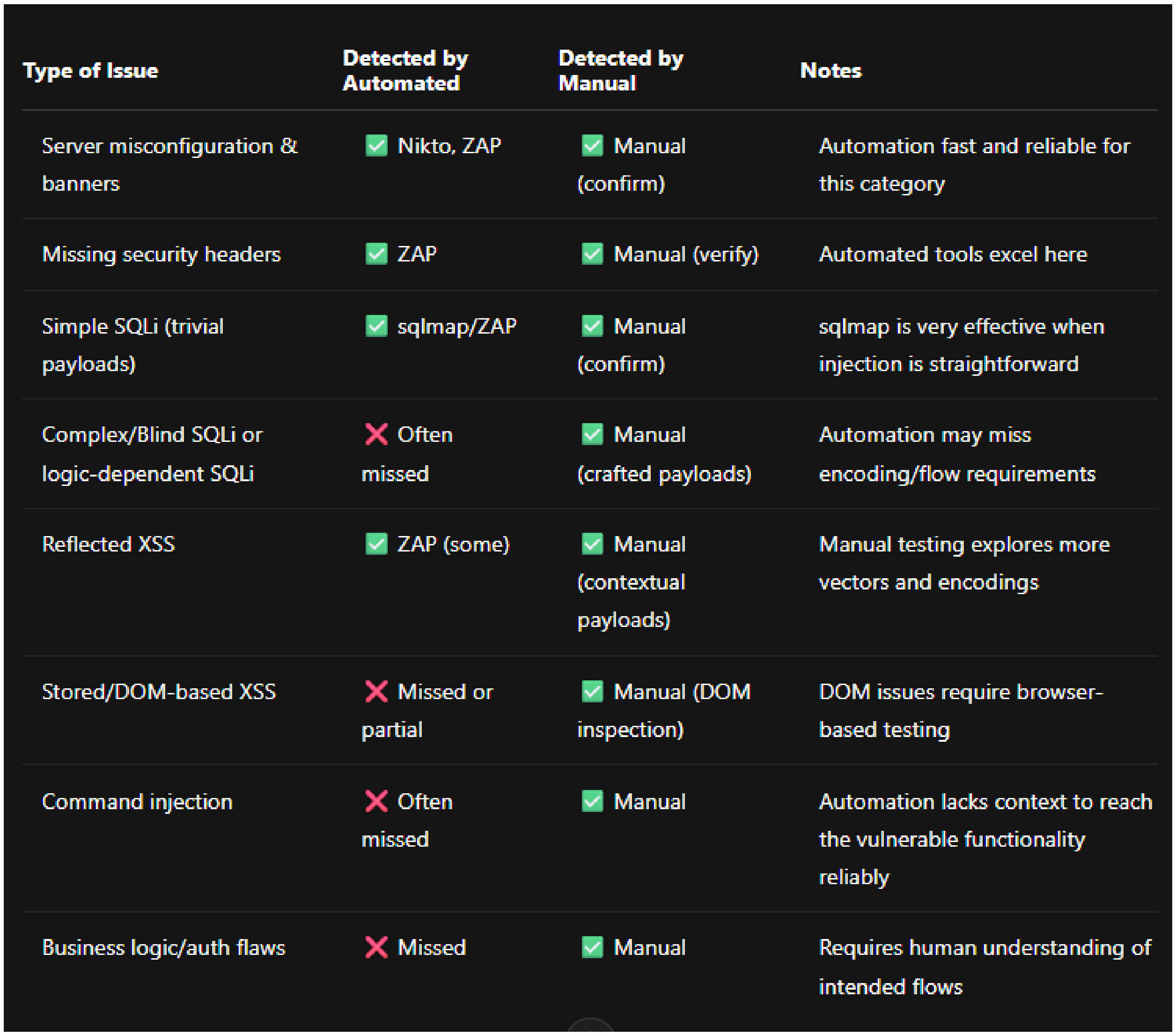
Ping functionality (/vulnerabilities/exec/) allowed 127.0.0.1; cat /etc/hostname. Manual injection confirmed OS command execution and file disclosure.

Why manual mattered: Automation sometimes misses command injection unless the exact pattern is known and reachable.

8.4 Authentication / Logic Flaws

Manual testing of password reset and session fixation showed logic gaps (e.g., predictable tokens at Low security). Automated scanners did not highlight these sufficiently.

9. Comparative Table (Automated vs Manual)



Type of Issue	Detected by Automated	Detected by Manual	Notes
Server misconfiguration & banners	✅ Nikto, ZAP	✅ Manual (confirm)	Automation fast and reliable for this category
Missing security headers	✅ ZAP	✅ Manual (verify)	Automated tools excel here
Simple SQLi (trivial payloads)	✅ sqlmap/ZAP	✅ Manual (confirm)	sqlmap is very effective when injection is straightforward
Complex/Blind SQLi or logic-dependent SQLi	❌ Often missed	✅ Manual (crafted payloads)	Automation may miss encoding/flow requirements
Reflected XSS	✅ ZAP (some)	✅ Manual (contextual payloads)	Manual testing explores more vectors and encodings
Stored/DOM-based XSS	❌ Missed or partial	✅ Manual (DOM inspection)	DOM issues require browser-based testing
Command injection	❌ Often missed	✅ Manual	Automation lacks context to reach the vulnerable functionality reliably
Business logic/auth flaws	❌ Missed	✅ Manual	Requires human understanding of intended flows

10. False Positives & False Negatives

False Positives (automation): ZAP reported some XSS/SQLi that were not exploitable (inputs sanitized server-side during subsequent processing). Manual verification cleared these.

False Negatives (automation): DOM-XSS, command injection, and complex auth logic were often missed.

11. Time & Efficiency Comparison

Automation: Rapid coverage of many endpoints; good for initial triage and baseline compliance checks.

Manual: Time-consuming but finds high-impact, context-specific vulnerabilities.

Recommendation: Use automation for broad coverage and prioritize findings, then use manual testing to validate and dig deeper into critical areas.

12. Conclusion — Strengths & Weaknesses of Automated Scanners

Strengths:

Speed and scale: scan large attack surface quickly.

Repeatability: consistent baseline scans.

Good at detecting misconfigurations, server issues, outdated software, and simple input-based vulnerabilities.

Helpful for continuous integration pipelines and routine checks.

Weaknesses:

Context-insensitive: can miss business logic, chaining, and complex flows.

Limited DOM/JavaScript analysis compared to an interactive browser-driven manual test.

False positives are common — each requires manual validation.

Cannot reason about application intent or detect subtle authorization flaws.

Final verdict: Automated scanners are essential for efficiency and baseline security checks, but they cannot replace human-driven manual testing for deep, high-impact vulnerabilities. A combined strategy maximizes security coverage.

13. Remediation Recommendations (Practical Steps)

Fix issues flagged by scanners quickly (headers, server banners, outdated components).

Use prepared statements and parameterized queries to mitigate SQLi.

Implement strict input/output encoding and CSP to limit XSS.

Harden application logic: implement proper authentication, token randomness, and authorization checks.

Schedule periodic automated scans and follow-up manual penetration tests for critical flows.

14. Artifacts & Appendices

Attach or include: Nikto output, ZAP report (HTML), sqlmap logs, Burp Suite request/response captures, screenshots of PoCs.