Website Security Audit Report

Comprehensive Security Assessment

Target Environment:	Metasploitable2 - DVWA
Target IP Address:	192.168.1.4
Assessment Type:	Web Application Security Audit
Assessment Date:	January 29, 2026
Report Generated:	February 01, 2026
Auditor:	Sumit
Classification:	CONFIDENTIAL

1. Executive Summary

A comprehensive web application security assessment was conducted against the Damn Vulnerable Web Application (DVWA) hosted on Metasploitable2. The primary objective of this assessment was to identify security vulnerabilities, validate exploitation paths, and evaluate the overall security posture of the web application within a controlled laboratory environment.

The assessment identified multiple **Critical** and **High** severity vulnerabilities across various attack vectors. These vulnerabilities include Remote Code Execution (RCE), OS Command Injection, SQL Injection variants, Insecure File Upload mechanisms, Cross-Site Scripting (XSS), and Cross-Site Request Forgery (CSRF). Successful exploitation of these vulnerabilities enables complete compromise of the web application, underlying operating system, and sensitive data stores.

Overall Risk Rating	**CRITICAL**

Immediate Action Required: The vulnerabilities identified pose severe security risks and require immediate remediation before any similar configuration is deployed in a production environment. Critical vulnerabilities should be addressed within 24-48 hours, while high severity issues should be resolved within one week.

2. Scope of Assessment

2.1 In Scope

- DVWA (Damn Vulnerable Web Application) - all modules and functionalities
- HTTP/HTTPS-based application endpoints and API interfaces
- Authentication mechanisms and session management controls
- Input validation testing across all user-controllable parameters
- File upload and handling functionalities
- Client-side security controls and browser-based vulnerabilities
- Authorization and access control mechanisms
- Server configuration and security headers

2.2 Out of Scope

- Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) attacks
- Physical security controls and infrastructure security
- Social engineering and phishing assessments
- Wireless network security testing
- Internal network penetration beyond the target system

2.3 Testing Environment

Attacker Machine:	Kali Linux (Latest)
Target Machine:	Metasploitable2
Target IP:	192.168.1.4
Network Type:	Isolated Lab Environment
Testing Duration:	January 29, 2026

3. Assessment Methodology

The security assessment followed industry-recognized best practices and methodologies, including the OWASP Web Security Testing Guide (WSTG), OWASP Top 10, and SANS Top 25. The assessment combined automated scanning tools with comprehensive manual testing to ensure thorough coverage and validation of identified vulnerabilities.

3.1 Reconnaissance and Information Gathering

- Service enumeration and version detection
- Web technology stack identification
- Directory and file discovery
- Application mapping and endpoint identification

3.2 Authentication and Access Control Testing

- Default credential testing
- Password policy evaluation

- Session management analysis
- Authorization bypass testing

3.3 Input Validation and Injection Testing

- SQL Injection (classic and blind variants)
- OS Command Injection
- LDAP Injection
- XML/XXE Injection testing

3.4 File Handling and Upload Assessment

- Unrestricted file upload testing
- Path traversal vulnerabilities
- File inclusion vulnerabilities (LFI/RFI)
- Executable upload and execution testing

3.5 Client-Side Vulnerability Testing

- Cross-Site Scripting (Reflected and Stored)
- Cross-Site Request Forgery (CSRF)
- Clickjacking vulnerabilities
- DOM-based vulnerabilities

3.6 Manual Verification and Exploitation

- Proof-of-concept development
- Impact assessment and demonstration
- False positive elimination
- Vulnerability chaining analysis

4. Tools and Technologies Used

Tool/Technology	Purpose	Version
OWASP ZAP	Automated vulnerability scanning and active security testing	v2.17.0
Nikto	Web server misconfiguration and outdated software detection	v2.5.0
Web Browser	Manual testing and validation of exploitation paths	Firefox/Chrome
Burp Suite	HTTP/HTTPS proxy and manual request manipulation	Community
Linux CLI Tools	Payload crafting, verification, and post-exploitation	Various
cURL	HTTP request testing and API endpoint validation	Latest
Python Scripts	Custom exploit development and automation	Python 3.x

5. Findings Summary

The security assessment identified a total of 10 unique vulnerabilities across various severity levels. The distribution of findings demonstrates significant security weaknesses that require immediate attention, particularly in the areas of input validation, authentication, and file handling.

ID	Vulnerability	Severity	CVSS	Status
F-01	Default Credentials	Critical	9.8	Confirmed
F-02	OS Command Injection	Critical	9.8	Confirmed
F-03	SQL Injection	High	8.6	Confirmed
F-04	Blind SQL Injection	High	8.2	Confirmed
F-05	Insecure File Upload (RCE)	Critical	9.8	Confirmed
F-06	Reflected Cross-Site Scripting	Medium	6.1	Confirmed
F-07	Stored Cross-Site Scripting	High	7.1	Confirmed
F-08	Outdated Apache & PHP	High	7.5	Confirmed
F-09	Missing Security Headers	Medium	5.3	Confirmed
F-10	Cross-Site Request Forgery	Medium	6.5	Confirmed

5.1 Risk Distribution

Severity Level	Count	Percentage
Critical	3	30%
High	4	40%
Medium	3	30%
Low	0	0%
Informational	0	0%

6. Detailed Findings

F-01: Default Credentials

Severity	**Critical**
CVSS Score	9.8 (Critical)
Affected Component	DVWA Login Interface - /login.php

Description:

The application allows authentication using well-known default credentials that are publicly documented and widely known. The credentials (admin/password) are commonly used in default installations and represent a critical security flaw.

Proof of Concept:

```
Credentials tested: admin / password

Access URL: http://192.168.1.4/dvwa/login.php

Result: Successful authentication with full administrative privileges
```

Impact:

An unauthenticated attacker can gain immediate administrative access to the application without requiring any exploitation techniques. This provides complete control over the application, including the ability to: • Access all application functionality and sensitive data • Modify application settings and configurations • Create, modify, or delete user accounts • Execute administrative functions • Potentially pivot to underlying system access

Remediation:

1. Remove all default credentials before deployment 2. Implement strong password policy requiring: - Minimum 12 characters - Complexity requirements (uppercase, lowercase, numbers, special characters) - Password expiration and rotation policies 3. Implement account lockout after failed login attempts 4. Enable multi-factor authentication (MFA) for administrative accounts 5. Implement password history to prevent reuse 6. Force password change on first login

F-02: OS Command Injection

Severity	**Critical**
CVSS Score	9.8 (Critical)
Affected Component	/dvwa/vulnerabilities/exec/

Description:

The application executes system-level commands using unsanitized user input passed directly to shell command execution functions. This allows attackers to inject arbitrary operating system commands that are executed with the privileges of the web server process.

Proof of Concept:

```
Input supplied: 127.0.0.1; whoami

Command executed: ping -c 4 127.0.0.1; whoami

Result: The application returned the output of both the ping command and the whoami
command,

confirming command injection and showing execution as user 'www-data'
```

Impact:

This vulnerability allows an attacker to execute arbitrary operating system commands, leading to complete system compromise: • Read, modify, or delete any files accessible to the web server user • Install backdoors and maintain persistent access • Pivot to internal network systems • Exfiltrate sensitive data including database credentials • Launch attacks against other systems • Create reverse shells for interactive access

Remediation:

1. Never pass user input directly to system command execution functions 2. Use parameterized APIs that do not involve shell command execution 3. Implement strict input validation using allowlists 4. If system commands are absolutely necessary: - Sanitize all user input - Use escapeshellarg() and escapeshellcmd() in PHP - Implement command execution in restricted environments 5. Apply principle of least privilege to web server user 6. Implement application-level sandboxing

F-03: SQL Injection

Severity	**High**
CVSS Score	8.6 (High)
Affected Component	/dvwa/vulnerabilities/sqli/

Description:

The application does not properly sanitize user input before incorporating it into SQL queries. This allows attackers to manipulate SQL query logic and execute arbitrary SQL commands.

Proof of Concept:

```
Input used: 1' OR '1'='1
```

```
Injected query (reconstructed):
```

```
SELECT * FROM users WHERE user_id = '1' OR '1'='1'
```

```
Result: The application returned multiple database records, confirming SQL query
manipulation and
```

```
bypassing the intended query logic
```

Impact:

SQL Injection enables attackers to: • Bypass authentication mechanisms • Access unauthorized database information • Extract sensitive data including user credentials, personal information • Modify or delete database records • Execute administrative operations on the database • In some cases, execute operating system commands through database features

Remediation:

1. Use parameterized queries (prepared statements) exclusively 2. Implement proper input validation and sanitization 3. Apply principle of least privilege to database accounts 4. Disable detailed error messages in production 5. Implement Web Application Firewall (WAF) rules 6. Use ORM frameworks that provide built-in SQL injection protection 7. Conduct regular code reviews focusing on database interactions

---

F-04: Blind SQL Injection

Severity	**High**

CVSS Score	8.2 (High)
Affected Component	/dvwa/vulnerabilities/sqli_blind/

Description:

The application is vulnerable to blind SQL injection through boolean-based logic manipulation. Unlike traditional SQL injection, this variant does not display database errors or results directly, but attackers can infer information based on application behavior differences.

Proof of Concept:

```
TRUE condition test:
Input: 1' AND '1'='1
Result: User exists message displayed
FALSE condition test:
Input: 1' AND '1'='2
Result: User missing message displayed
This confirms the ability to inject SQL logic and infer database information based on
application responses
```

Impact:

Blind SQL Injection allows attackers to: • Extract database information character by character • Enumerate database structure, tables, and columns • Determine database version and configuration • Extract sensitive data through boolean-based or time-based techniques • Potentially escalate to full SQL injection exploitation

Remediation:

1. Use prepared statements and parameterized queries 2. Implement strict input validation for all user-controllable parameters 3. Avoid detailed error messages that reveal database information 4. Implement consistent response times to prevent time-based exploitation 5. Use database access controls and least privilege principles 6. Monitor for unusual database query patterns

---

F-05: Insecure File Upload (Remote Code Execution)

Severity	**Critical**
CVSS Score	9.8 (Critical)
Affected Component	/dvwa/vulnerabilities/upload/

Description:

The application allows unrestricted file uploads without proper validation of file types, content, or extensions. Uploaded files are stored in a web-accessible directory with execute permissions, enabling remote code execution.

Proof of Concept:

```
File created: test.php containing:

<?php system($_GET['cmd']); ?>

Upload process: File uploaded successfully

Storage location: /dvwa/hackable/uploads/test.php

Access URL: http://192.168.1.4/dvwa/hackable/uploads/test.php?cmd=whoami

Result: The PHP file was executed, and the 'whoami' command output was displayed
```

Impact:

This vulnerability provides immediate remote code execution capabilities: • Execute arbitrary system commands with web server privileges • Upload web shells for persistent access • Read sensitive files including configuration files with database credentials • Modify application files and inject backdoors • Launch attacks against internal systems • Completely compromise the web server and potentially the underlying infrastructure

Remediation:

1. Implement strict file type validation based on content, not just extension 2. Use allowlists for permitted file types 3. Store uploaded files outside the web root directory 4. Rename uploaded files to prevent direct execution 5. Remove execute permissions from upload directories 6. Implement file size limits 7. Scan uploaded files for malware 8. Serve uploaded files through a separate domain or use Content-Disposition headers 9. Implement additional authentication for accessing uploaded files

---

F-06: Reflected Cross-Site Scripting (XSS)

Severity	**Medium**
CVSS Score	6.1 (Medium)
Affected Component	/dvwa/vulnerabilities/xss_r/

Description:

User input is reflected in the HTTP response without proper output encoding or sanitization, allowing attackers to inject malicious JavaScript code that executes in victims' browsers.

Proof of Concept:

```
Input supplied: <script>alert('XSS')</script>
```

```
URL: http://192.168.1.4/dvwa/vulnerabilities/xss_r/?name=<script>alert('XSS')</script>
```

```
Result: The JavaScript code executed in the browser, displaying an alert box
```

Impact:

Reflected XSS enables attackers to: • Steal session cookies and authentication tokens • Perform actions on behalf of the victim user • Redirect users to malicious websites • Capture user keystrokes and form data • Deface web pages • Deliver browser-based exploits

Remediation:

1. Implement proper output encoding for all user-controlled data 2. Use context-appropriate encoding (HTML, JavaScript, URL, CSS) 3. Implement Content Security Policy (CSP) headers 4. Set HTTPOnly flag on session cookies 5. Use modern frameworks with automatic XSS protection 6. Validate and sanitize all input 7. Use X-XSS-Protection header

---

F-07: Stored Cross-Site Scripting (XSS)

Severity	**High**
CVSS Score	7.1 (High)
Affected Component	/dvwa/vulnerabilities/xss_s/

Description:

The application stores user-supplied data without sanitization and displays it to other users without proper encoding. This creates a persistent XSS vulnerability that affects all users who view the compromised content.

Proof of Concept:

```
Payload submitted: <script>alert('Stored XSS')</script>
```

```
Storage: The malicious script was stored in the database
```

```
Execution: Every user viewing the guestbook page triggers the JavaScript execution
```

Impact:

Stored XSS is more severe than reflected XSS because: • Affects all users who view the compromised page • Does not require user interaction beyond normal browsing • Can create worm-like behavior in social applications • Enables mass credential harvesting • Can be used to deliver drive-by downloads • Facilitates large-scale phishing campaigns

Remediation:

1. Implement output encoding for all stored user content 2. Use Content Security Policy (CSP) with strict directives 3. Sanitize user input before storage 4. Implement proper input validation 5. Use HTTPOnly and Secure flags on cookies 6. Regular security scanning of stored content 7. Implement rate limiting on content submission

---

F-08: Outdated Apache & PHP Versions

Severity	**High**
CVSS Score	7.5 (High)
Affected Component	Server Infrastructure

Description:

The web server is running significantly outdated versions of Apache (2.2.8) and PHP (5.2.4), both of which have reached end-of-life and contain numerous known security vulnerabilities.

Proof of Concept:

```
Detected versions:
• Apache: 2.2.8 (EOL - current is 2.4.x)
• PHP: 5.2.4-2ubuntu5.10 (EOL - current is 8.x)
Detection method: HTTP response headers and banner grabbing
X-Powered-By: PHP/5.2.4-2ubuntu5.10
Server: Apache/2.2.8 (Ubuntu)
```

Impact:

Running outdated software exposes the system to: • Publicly known exploits and CVEs • Remote code execution vulnerabilities • Information disclosure issues • Authentication bypass vulnerabilities • Privilege escalation opportunities • Lack of modern security features and protections

Remediation:

1. Upgrade Apache to version 2.4.x or later 2. Upgrade PHP to version 8.1 or later (minimum 7.4) 3. Implement regular patch management processes 4. Subscribe to security mailing lists 5. Establish vulnerability scanning schedule 6. Test updates in staging environment before production deployment 7. Maintain software inventory and lifecycle tracking

---

F-09: Missing HTTP Security Headers

Severity	**Medium**
CVSS Score	5.3 (Medium)
Affected Component	HTTP Response Headers

Description:

The application fails to implement several critical HTTP security headers that provide defense-in-depth protection against various client-side attacks.

Proof of Concept:

```
Missing headers identified:
• X-Frame-Options (clickjacking protection)
• X-Content-Type-Options (MIME sniffing protection)
• Content-Security-Policy (XSS and injection protection)
• Strict-Transport-Security (HTTPS enforcement)
• X-XSS-Protection (XSS filter)
• Referrer-Policy (information leakage control)
```

Impact:

Missing security headers increase risk of: • Clickjacking attacks • MIME type confusion attacks • Cross-site scripting exploitation • Man-in-the-middle attacks • Information disclosure • Reduced defense-in-depth posture

Remediation:

Implement the following HTTP security headers: 1. X-Frame-Options: DENY or SAMEORIGIN 2. X-Content-Type-Options: nosniff 3. Content-Security-Policy: script-src 'self'; object-src 'none' 4. Strict-Transport-Security: max-age=31536000; includeSubDomains 5. X-XSS-Protection: 1; mode=block 6. Referrer-Policy: strict-origin-when-cross-origin 7. Permissions-Policy: geolocation=(), microphone=(), camera=()

F-10: Cross-Site Request Forgery (CSRF)

Severity	**Medium**
CVSS Score	6.5 (Medium)
Affected Component	Multiple endpoints lacking CSRF protection

Description:

The application does not implement anti-CSRF tokens for state-changing operations, allowing attackers to forge requests on behalf of authenticated users.

Proof of Concept:

```
Vulnerable request example:
GET /dvwa/vulnerabilities/csrf/?password_new=hacked&password_conf=hacked&Change=Change
```

```
No CSRF token validation present
```

```
Attack vector: Malicious website triggers request while user is authenticated
```

Impact:

CSRF vulnerabilities enable attackers to: • Change user passwords • Modify account settings • Perform unauthorized transactions • Create or delete content • Execute administrative functions • Chain with other vulnerabilities for greater impact

Remediation:

1. Implement anti-CSRF tokens for all state-changing operations 2. Use the Synchronizer Token Pattern 3. Validate CSRF tokens on the server side 4. Implement SameSite cookie attribute 5. Require re-authentication for sensitive operations 6. Implement custom request headers for AJAX requests 7. Use framework-provided CSRF protection mechanisms

---

7. Automated Scan Tool Correlation

The assessment incorporated findings from multiple automated security scanning tools to ensure comprehensive coverage. The results from OWASP ZAP and Nikto were correlated with manual testing to validate vulnerabilities and eliminate false positives.

7.1 OWASP ZAP Findings

OWASP ZAP (Zed Attack Proxy) version 2.17.0 was used for automated vulnerability discovery and active security testing. The scan identified the following key findings:

• **High Risk Alerts (2 findings):** Remote Code Execution vulnerabilities including CVE-2012-1823 affecting PHP-CGI, and Path Traversal issues

• **Medium Risk Alerts (6 findings):** CSRF vulnerabilities, insecure cookie configurations (missing HTTPOnly and Secure flags), SQL Injection in user input fields, and directory listing enabled

• **Low Risk Alerts (7 findings):** X-Frame-Options header missing, X-Content-Type-Options header missing, incomplete or missing charset declarations

• **Informational Alerts (14 findings):** Technology disclosure through HTTP headers, server version information leakage, timestamp disclosure, and default file discoveries

7.2 Nikto Findings

Nikto version 2.5.0 performed comprehensive web server scanning to identify misconfigurations, outdated software versions, and security issues. Key findings include:

• **Outdated Software:** Apache 2.2.8 detected (current version is 2.4.x, with 2.2 branch reaching EOL), PHP 5.2.4 detected through X-Powered-By header

• **Dangerous HTTP Methods:** HTTP TRACE method enabled, potentially allowing Cross-Site Tracing (XST) attacks

• **Information Disclosure:** phpinfo() accessible at /phpinfo.php exposing detailed system configuration, PHP easter eggs revealing version information through specific query strings

• **Directory Issues:** Directory indexing enabled on multiple directories (/doc/, /test/, /icons/), exposing file structure and potentially sensitive files

• **Default Files:** Apache default README files present, phpMyAdmin installation detected without proper access restrictions

• **Missing Security Headers:** X-Frame-Options header absent, X-Content-Type-Options header not configured

- **Sensitive Files:** WordPress configuration file backup (#wp-config.php#) found containing database credentials

7.3 Tool Correlation Summary

Cross-validation between automated tools and manual testing confirmed the accuracy of critical and high-severity findings. Multiple tools independently identified the same vulnerabilities, increasing confidence in the results:

- Outdated software versions confirmed by both Nikto (banner grabbing) and manual verification
- Missing security headers identified by both ZAP and Nikto scanning
- Information disclosure issues detected across multiple tools and validated manually
- All critical vulnerabilities (Command Injection, File Upload, SQL Injection) were manually verified with successful proof-of-concept exploitation
- False positives from automated scans were eliminated through manual validation

8. Remediation Roadmap and Recommendations

The following remediation roadmap prioritizes vulnerabilities based on severity, exploitability, and potential business impact. Immediate action is required for Critical and High severity findings.

8.1 Priority 1: Immediate Action Required (24-48 Hours)

- **F-01: Default Credentials** - Change immediately and enforce strong password policy
- **F-02: OS Command Injection** - Disable affected functionality or implement complete input sanitization
- **F-05: Insecure File Upload** - Disable file upload functionality or move uploads outside web root with strict validation

8.2 Priority 2: High Priority (Within 1 Week)

- **F-03: SQL Injection** - Implement prepared statements across all database queries
- **F-04: Blind SQL Injection** - Apply parameterized queries to all user input points
- **F-07: Stored XSS** - Implement output encoding and Content Security Policy
- **F-08: Outdated Software** - Plan and execute software upgrade to supported versions

8.3 Priority 3: Medium Priority (Within 2 Weeks)

- **F-06: Reflected XSS** - Implement input validation and output encoding
- **F-09: Missing Security Headers** - Configure HTTP security headers
- **F-10: CSRF** - Implement anti-CSRF tokens across all state-changing operations

8.4 Strategic Security Recommendations

- **Secure Development Lifecycle:** Integrate security into the development process from design through deployment
- **Security Training:** Provide secure coding training for development team focusing on OWASP Top 10 vulnerabilities
- **Code Review Process:** Implement mandatory security-focused code reviews for all changes
- **Automated Security Testing:** Integrate SAST/DAST tools into CI/CD pipeline
- **Web Application Firewall:** Deploy WAF with rulesets for common attack patterns
- **Regular Assessments:** Conduct quarterly security assessments and penetration tests
- **Vulnerability Management:** Establish formal vulnerability management program with defined SLAs
- **Incident Response:** Develop and test incident response procedures for security breaches
- **Security Monitoring:** Implement logging and monitoring for security events and anomalies
- **Patch Management:** Establish regular patching schedule for all software components

9. Conclusion

This comprehensive security assessment of the Damn Vulnerable Web Application (DVWA) on Metasploitable2 has identified critical security vulnerabilities that demonstrate common weaknesses found in web applications. The assessment revealed 10 distinct vulnerabilities spanning three critical, four high, and three medium severity levels.

The most critical findings - default credentials, OS command injection, and insecure file upload - provide immediate pathways to complete system compromise. These vulnerabilities, combined with the identified SQL injection variants and cross-site scripting issues, represent severe security deficiencies that would enable attackers to gain unauthorized access, execute arbitrary code, steal sensitive data, and maintain persistent access to compromised systems.

This assessment underscores the critical importance of implementing secure coding practices, conducting regular security testing, maintaining current software versions, and following security best practices throughout the software development lifecycle. The vulnerabilities identified are representative of real-world security issues found in production web applications and serve as valuable learning opportunities.

While this assessment was conducted in a controlled laboratory environment for educational purposes, the security principles and remediation strategies outlined in this report are directly applicable to production web applications. Organizations should prioritize security throughout the development process and implement defense-in-depth strategies to protect against the constantly evolving threat landscape.

The assessment team remains available to provide clarification on any findings, assist with remediation efforts, or conduct follow-up verification testing once fixes have been implemented.

Prepared by:	Sumit
Title:	Security Auditor
Date:	February 01, 2026

Appendix A: CVSS Scoring Methodology

The Common Vulnerability Scoring System (CVSS) provides a standardized method for rating the severity of security vulnerabilities. This assessment uses CVSS v3.1 base scores.

Rating	CVSS Score	Description
Critical	9.0 - 10.0	Immediate threat requiring urgent remediation
High	7.0 - 8.9	Serious vulnerability requiring prompt remediation
Medium	4.0 - 6.9	Moderate risk requiring planned remediation
Low	0.1 - 3.9	Minor security concern with low exploitability

Appendix B: References and Resources

• OWASP Top 10 Web Application Security Risks - https://owasp.org/www-project-top-ten/

• OWASP Web Security Testing Guide (WSTG) - https://owasp.org/www-project-web-security-testing-guide/

• OWASP Zed Attack Proxy (ZAP) - https://www.zaproxy.org/

• Nikto Web Scanner - https://cirt.net/Nikto2

• Common Vulnerability Scoring System (CVSS) - https://www.first.org/cvss/

• CWE/SANS Top 25 Most Dangerous Software Errors - https://cwe.mitre.org/top25/

• NIST National Vulnerability Database - https://nvd.nist.gov/

• OWASP Code Review Guide - https://owasp.org/www-project-code-review-guide/

• Web Application Security Consortium (WASC) - http://www.webappsec.org/

• SANS Institute - Securing Web Application Technologies - https://www.sans.org/

Appendix C: Legal Disclaimer

This security assessment was conducted in a controlled laboratory environment using deliberately vulnerable systems designed for security testing and educational purposes. All testing was performed with explicit authorization on systems owned and controlled by the tester. This report and its contents are provided "as is" for educational and informational purposes. The findings and recommendations contained herein represent the assessment team's professional opinion based on testing conducted at a specific point in time. Security assessments provide a snapshot of the security posture and cannot guarantee the identification of all vulnerabilities. This report should not be used for malicious purposes or unauthorized security testing. Unauthorized access to computer systems is illegal and may result in criminal prosecution. Always obtain proper authorization before conducting security testing on any systems.

--- END OF REPORT ---