JARWIS

PENETRATION TEST REPORT

TARGET APPLICATION

https://example-webapp.com

Web Application Security Assessment

REPORT ID JAR-20260104_194811-689642	ASSESSMENT DATE January 04, 2026
ENDPOINTS TESTED 14	ASSESSMENT TYPE Authenticated & Unauthenticated

Assessment Results

CRITICAL RISK

1 Critical

2 High

2 Medium

1 Low

0 Info

Introduction

Jarwis is an AI-powered penetration testing platform that delivers comprehensive security assessments at scale. Leveraging advanced artificial intelligence and machine learning algorithms, Jarwis automates the discovery and verification of security vulnerabilities across web applications, mobile applications, APIs, and cloud infrastructure.

The platform combines the thoroughness of manual penetration testing with the speed and consistency of automated scanning, providing organizations with actionable security insights in hours rather than weeks. Jarwis systematically tests against the OWASP Top 10 and other industry-standard vulnerability classifications, delivering detailed findings with proof-of-concept evidence and remediation guidance.

Key Capabilities: Automated vulnerability discovery • AI-powered verification to minimize false positives • OWASP Top 10 coverage • Detailed proof-of-concept evidence • Contextual remediation recommendations • Multi-format reporting (PDF, HTML, JSON, SARIF)

Executive Summary

Example WebApp Inc. engaged Jarwis to conduct a comprehensive penetration test of their web application security assessment environment to evaluate security vulnerabilities and assess potential risks to their digital infrastructure. During the time allotted for this engagement, Jarwis systematically tested the application's defense mechanisms, focusing on industry-standard attack vectors aligned with the OWASP Top Ten vulnerabilities.

Purpose

The penetration test was conducted to provide Example WebApp Inc. with a thorough security assessment of their web application security assessment infrastructure. Jarwis evaluated the application's resilience against common attack vectors including injection flaws, cross-site scripting, access control bypasses, authentication weaknesses, and information disclosure vulnerabilities. The assessment aimed to identify exploitable security weaknesses that could compromise the confidentiality, integrity, and availability of Example WebApp Inc.'s systems and data.

Findings Overview

During the assessment, Jarwis identified **6 distinct security vulnerabilities** across the target environment. The findings include **1 critical-severity** vulnerability, **2 high-severity** vulnerabilities, **2 medium-severity** vulnerabilities, and **1 low-severity** vulnerability.

The **critical-severity finding** involves sql injection in authentication endpoint. The login form is vulnerable to SQL injection attacks. An attacker can bypass authentication or extract sensitive data from the database by injecting malicious SQL code into the username or password f

The **high-severity finding** involves reflected cross-site scripting (xss) in search. The search functionality reflects user input without proper encoding, allowing attackers to inject malicious JavaScript that executes in victims' browsers. This can lead to session hijacking, credenti

The **high-severity finding** involves insecure direct object reference (idor) in user api. The application allows users to access other users' sensitive data by modifying the user ID parameter in API requests. This vulnerability enables unauthorized access to personal information, account d

> The critical and high-severity vulnerabilities pose immediate risks to the organization's security posture, as they provide pathways for unauthorized data access and system compromise. Immediate remediation is strongly recommended.

Findings Summary

The following table provides an overview of all vulnerabilities identified during this assessment, organized by severity level.

SEVERITY	VULNERABILITY	CATEGORY	CWE
CRITICAL	SQL Injection in Authentication Endpoint	Injection	CWE-89
HIGH	Reflected Cross-Site Scripting (XSS) in Search	Injection	CWE-79
HIGH	Insecure Direct Object Reference (IDOR) in User AP	Broken Access Control	CWE-639
MEDIUM	Missing Security Headers	Security Misconfiguration	CWE-693
MEDIUM	Weak Password Policy Allows Common Passwords	Auth Failures	CWE-521
LOW	Sensitive Data Exposed in URL Parameters	Cryptographic Failures	CWE-598

Conclusion

The penetration test revealed several significant security vulnerabilities that require immediate attention. The presence of 3 critical and high-severity vulnerabilities represents substantial risks that could lead to unauthorized system access and data compromise. By promptly addressing these identified vulnerabilities, the organization can significantly enhance their application security and reduce exposure to potential cyber threats.

Recommended Actions: Remediate 1 critical vulnerabilities within 24-48 hours • Address 2 high-severity findings within 7 days • Plan remediation of 2 medium-severity issues within 30 days •

Implement security monitoring and logging for affected components • Conduct follow-up assessment to verify remediation effectiveness

Detailed Findings

The following section provides comprehensive technical details for each vulnerability identified during the assessment, including proof-of-concept evidence, AI-powered analysis, and specific remediation guidance.

JAR-WEB-001

CRITICAL

SQL Injection in Authentication Endpoint

A03 CWE-89

DESCRIPTION

The login form is vulnerable to SQL injection attacks. An attacker can bypass authentication or extract sensitive data from the database by injecting malicious SQL code into the username or password fields. This vulnerability enables complete database compromise including extraction of user credentials, personal information, and administrative access.

URL	METHOD	PARAMETER
`https://example-webapp.com/api/auth/login`	`POST`	`username`

EVIDENCE / PROOF OF CONCEPT

```
SQL error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version
```

🤖 AI Analysis

Jarwis AI detected SQL injection by injecting a single quote character which caused the database to return an SQL syntax error. This confirms that user input is being directly concatenated into SQL queries without proper sanitization or parameterized queries. The error message leakage also reveals the database type (MySQL).

HTTP DETAILS

REQUEST

```
POST /api/auth/login HTTP/1.1
Host: example-webapp.com
Content-Type: application/json

{"username": "admin' OR '1'='1'--", "password": "test"}
```

RESPONSE

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/json

{"error": "SQL error: You have an error in your SQL syntax..."}
```

✓ Remediation

Use parameterized queries (prepared statements) instead of string concatenation. Implement input validation and sanitization. Consider using an ORM like SQLAlchemy or Django ORM. Disable verbose error messages in production.

JAR-WEB-002

HIGH

Reflected Cross-Site Scripting (XSS) in Search

A03 CWE-79

DESCRIPTION

The search functionality reflects user input without proper encoding, allowing attackers to inject malicious JavaScript that executes in victims' browsers. This can lead to session hijacking, credential theft, and malware distribution.

URL	METHOD	PARAMETER
`https://example-webapp.com/search?q=<script>alert('XSS')</script>`	GET	q

EVIDENCE / PROOF OF CONCEPT

```
<script>alert('XSS')</script> reflected in response body without encoding
```

🤖 AI Analysis

Jarwis AI injected an XSS payload in the search parameter and found it was reflected verbatim in the HTML response without encoding. This allows arbitrary JavaScript execution in the context of the victim's session, enabling cookie theft and session hijacking.

HTTP DETAILS

REQUEST

```
GET /search?q=<script>alert('XSS')</script> HTTP/1.1
Host: example-webapp.com
Cookie: session=abc123...
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/html

<html><body><h2>Search results for: <script>alert('XSS')</script></h2></body></html>
```

✓ Remediation

Encode all user input before rendering in HTML using context-appropriate encoding. Implement Content-Security-Policy headers to restrict script execution. Use HTTP-only cookies to prevent JavaScript access to session tokens.

JAR-WEB-003

HIGH

Insecure Direct Object Reference (IDOR) in User API

A01 CWE-639

DESCRIPTION

The application allows users to access other users' sensitive data by modifying the user ID parameter in API requests. This vulnerability enables unauthorized access to personal information, account details, and potentially financial data.

URL	METHOD	PARAMETER
`https://example-webapp.com/api/users/12345/profile`	`GET`	`user_id`

EVIDENCE / PROOF OF CONCEPT

```
Successfully retrieved profile data for user 12345 (including SSN, email) while authenticated as user 67890
```

🤖 AI Analysis

Jarwis AI tested IDOR by modifying the user ID in the API endpoint while authenticated as a different user. The server returned complete profile data for the requested user ID without proper authorization checks, indicating a broken access control vulnerability.

HTTP DETAILS

REQUEST

```
GET /api/users/12345/profile HTTP/1.1
Host: example-webapp.com
Authorization: Bearer eyJhbGc...(token for user 67890)
Cookie: session=user67890session
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json

{"user_id": 12345, "email": "victim@example.com", "name": "John Doe", "ssn": "123-45-6789", "phone":
"+1-555-123-4567"}
```

✓ Remediation

Implement proper authorization checks on the server side for every API request. Use indirect references (UUIDs) instead of sequential IDs. Verify that the authenticated user has permission to access the requested resource before returning data.

JAR-WEB-004

MEDIUM

Missing Security Headers

A05 CWE-693

DESCRIPTION

The application is missing critical security headers that protect against common web attacks including clickjacking, MIME-type sniffing, and cross-site scripting.

URL	METHOD	PARAMETER
https://example-webapp.com/	GET	N/A

EVIDENCE / PROOF OF CONCEPT

```
Missing headers: X-Frame-Options, X-Content-Type-Options, Content-Security-Policy, Strict-Transport-Security
```

🤖 AI Analysis

Jarwis AI analyzed the HTTP response headers and found several critical security headers missing. Without X-Frame-Options, the site is vulnerable to clickjacking. Without Content-Security-Policy, XSS attacks have no additional mitigation.

HTTP DETAILS

REQUEST

```
GET / HTTP/1.1
Host: example-webapp.com
Accept: text/html
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: nginx/1.18.0
Date: Mon, 01 Jan 2026 12:00:00 GMT

<!DOCTYPE html>...
```

✓ Remediation

Add the following headers to all responses: X-Frame-Options: DENY, X-Content-Type-Options: nosniff, Content-Security-Policy: default-src 'self', Strict-Transport-Security: max-age=31536000; includeSubDomains

JAR-WEB-005

MEDIUM

Weak Password Policy Allows Common Passwords

A07 CWE-521

DESCRIPTION

The application accepts weak passwords during registration, allowing users to set easily guessable passwords from common breach lists. This significantly increases the risk of account takeover through credential stuffing attacks.

URL	METHOD	PARAMETER
`https://example-webapp.com/api/auth/register`	`POST`	`password`

EVIDENCE / PROOF OF CONCEPT

```
Passwords '123456', 'password', 'qwerty123' all accepted during registration
```

🤖 AI Analysis

Jarwis AI tested the registration endpoint with passwords from the RockYou breach list and found they were accepted. This allows attackers to easily compromise accounts through brute force or credential stuffing attacks using common password lists.

HTTP DETAILS

REQUEST

```
POST /api/auth/register HTTP/1.1
Host: example-webapp.com
Content-Type: application/json

{"email": "test@test.com", "password": "123456", "name": "Test User"}
```

RESPONSE

```
HTTP/1.1 201 Created
Content-Type: application/json

{"message": "User created successfully", "user_id": 99999}
```

✓ Remediation

Implement strong password requirements: minimum 12 characters, mix of uppercase, lowercase, numbers, and special characters. Check passwords against HaveIBeenPwned API or similar breach databases. Implement rate limiting and account lockout on login attempts.

JAR-WEB-006

LOW

Sensitive Data Exposed in URL Parameters

A02 CWE-598

DESCRIPTION

Sensitive information including API keys and session tokens are passed in URL parameters, which are logged by web servers, proxies, and stored in browser history.

URL	METHOD	PARAMETER
https://example-webapp.com/api/data?api_key=sk_live_abc123xyz	GET	api_key

EVIDENCE / PROOF OF CONCEPT

```
Live API key (sk_live_abc123xyz) transmitted in URL query parameter
```

🤖 AI Analysis

Jarwis AI detected sensitive credentials (API key with 'sk_live_' prefix indicating production key) being transmitted in URL parameters. URLs are logged by web servers, proxies, CDNs, and stored in browser history, creating multiple exposure points.

HTTP DETAILS

REQUEST

```
GET /api/data?api_key=sk_live_abc123xyz HTTP/1.1
Host: example-webapp.com
Referer: https://example-webapp.com/dashboard
```

RESPONSE

```
HTTP/1.1 200 OK
Content-Type: application/json

{"data": [...], "count": 150}
```

✓ Remediation

Move sensitive data to request headers (Authorization header) or POST body. Implement proper API key rotation policies. Review and purge sensitive data from existing logs.

Methodology

Jarwis employs a systematic, multi-phase approach to penetration testing that combines automated scanning with AI-powered analysis to deliver comprehensive security assessments.

1 Reconnaissance

Automated discovery of endpoints, forms, APIs, and application structure using headless browser technology and intelligent crawling algorithms.

2 Pre-Auth Testing

Security testing of publicly accessible surfaces including login forms, registration flows, and unauthenticated API endpoints.

3 Authentication Analysis

Examination of authentication mechanisms, session management, token handling, and credential storage practices.

4 Post-Auth Testing

Authenticated testing including IDOR, CSRF, privilege escalation, and access control bypass attempts.

5 AI Verification

AI-powered analysis of all findings to eliminate false positives and provide contextual remediation recommendations.

6 Reporting

Generation of comprehensive reports with detailed findings, proof-of-concept evidence, and prioritized remediation guidance.

OWASP Top 10 Coverage

This assessment covers the OWASP Top 10 (2021) security risks, the industry-standard framework for web application security.

CODE	CATEGORY	STATUS
A01	Broken Access Control	✓ Tested

CODE	CATEGORY	STATUS
A02	Cryptographic Failures	✓ Tested
A03	Injection (SQL, XSS, Command)	✓ Tested
A04	Insecure Design	✓ Tested
A05	Security Misconfiguration	✓ Tested
A06	Vulnerable and Outdated Components	✓ Tested
A07	Identification and Authentication Failures	✓ Tested
A08	Software and Data Integrity Failures	✓ Tested
A09	Security Logging and Monitoring Failures	✓ Tested
A10	Server-Side Request Forgery (SSRF)	✓ Tested

Appendix

Discovered Endpoints

The following endpoints and components were identified during the reconnaissance phase of the assessment.

1. https://example-webapp.com/
2. https://example-webapp.com/api/auth/login
3. https://example-webapp.com/api/auth/register
4. https://example-webapp.com/api/auth/forgot-password
5. https://example-webapp.com/api/users/me
6. https://example-webapp.com/api/users/{id}/profile
7. https://example-webapp.com/api/products
8. https://example-webapp.com/api/products/{id}
9. https://example-webapp.com/api/cart
10. https://example-webapp.com/api/orders
11. https://example-webapp.com/search
12. https://example-webapp.com/admin/dashboard
13. https://example-webapp.com/admin/users
14. https://example-webapp.com/api/payments/checkout

Disclaimer

⚠️ Important Notice

This security assessment report is provided for informational purposes only. The findings contained herein represent the security posture of the target application at the time of testing. Jarwis AI Security Platform does not guarantee the completeness or accuracy of this assessment. Security is a continuous process, and new vulnerabilities may be discovered after this report is generated. The recipient of this report is responsible for implementing the recommended remediation measures and verifying their effectiveness.

Terms of Use

This report is confidential and intended solely for the authorized recipient. Unauthorized distribution, reproduction, or use of this report is strictly prohibited. All security testing was conducted within the agreed scope and with proper authorization. Any actions taken based on this report are the sole responsibility of the recipient.