SUDO SECURITY & CONSULTING LLC

PUBLIC DISTRIBUTION

Web Application Penetration Test

Case Study Report — Public Distribution Version

Client	[REDACTED CLIENT]
Prepared By	Sudo Security and Consulting LLC
Report Date	20 March 2026
Test Window	20 March 2026 @ 2200Z
Test Type	White-Box Web Application Penetration Test
Distribution	Public — All identifying information has been removed

This report has been sanitized for public distribution. All client-identifying information — including names, domains, IP addresses, and API credentials — has been redacted or replaced with [REDACTED] markers. Technical findings and methodologies are presented as a professional case study published by Sudo Security and Consulting LLC.

Table of Contents

1. Executive Summary

1.1 Confidentiality & Distribution Statement

This document has been sanitized for public distribution. All client-identifying information — including names, domains, IP addresses, and API credentials — has been removed or replaced with [REDACTED] markers. Technical findings and methodology are published as a professional case study by Sudo Security and Consulting LLC.

1.2 Report Overview

On 20 March 2026, Sudo Security and Consulting LLC conducted a white-box web application penetration test against the following target assets:

- https://[REDACTED]
- https://[REDACTED]

Engagement Objective

Identify exploitable vulnerabilities that could result in:

- Unauthorized access to application functionality or data
- Sensitive data exposure
- Application-layer compromise
- Direct targeting of backend infrastructure

Summary of Results

Severity	Count
Critical	0
High	1
Medium	4
Low	1

Key Observations

- Strong baseline practices observed in CI/CD pipeline configuration and deployment workflows.
- Weaknesses identified across the following control categories:
  - Credential management
  - Client-side security controls
  - Repository operational hygiene
  - Internal infrastructure identifier exposure

Risk Summary

1. Active API credential exposed in public repository commit history
2. Internal infrastructure identifier enabling direct origin server targeting
3. Client-side script injection risk via unsafe DOM sinks
4. Uncontrolled third-party dependency loading without integrity enforcement
5. Insecure cryptographic implementation present in deployed portfolio component

Remediation Priority

1. Revoke exposed credential and purge repository history (PT-1) — immediate action required
2. Remove infrastructure identifiers; implement CDN origin shielding (PT-3)
3. Replace unsafe DOM sinks; deploy Content Security Policy header (PT-4)
4. Pin third-party asset versions; implement Subresource Integrity (PT-5)
5. Isolate and label non-production cryptographic components (PT-2)
6. Add rel="noopener noreferrer" to all external link targets (PT-6)

1.3 Authorization & Scope

Testing was conducted under a signed penetration testing authorization agreement with the system owner. All activities remained within defined scope and constraints.

Authorized Target Assets

- [REDACTED] — primary web application
- Associated infrastructure owned and operated by the client

Permitted Activities

- Passive and active reconnaissance
- Port scanning and service enumeration
- Vulnerability scanning
- Web application testing (including XSS and input validation)
- Directory and file discovery
- Exploit attempts against confirmed vulnerabilities
- Privilege escalation testing (if initial access obtained)
- Proof-of-concept exploitation for validation purposes

Prohibited Activities

- Denial-of-Service (DoS/DDoS) attacks
- Social engineering or phishing
- Testing third-party services (e.g., CDN providers, external APIs)
- Data destruction or modification beyond proof-of-concept validation
- Lateral movement or pivoting outside owned infrastructure

Testing Window

- Authorized window: 20 February 2026 – 25 June 2026
- All testing performed: 20 March 2026 at 2200Z

2. Findings

2.1 Vulnerability Distribution

Severity	Count
Critical	0
High	1
Medium	4
Low	1

2.2 Master Findings Table

ID	Title	CWE	Severity	State
PT-1	Sensitive Data Exposure — Third-Party API Key	CWE-798	High	Unresolved
PT-2	Insecure Cryptographic Implementation	CWE-327	Medium	Unresolved
PT-3	Security Misconfiguration / CDN Bypass	CWE-16	Medium	Unresolved
PT-4	DOM-Based XSS via Unsafe Sinks	CWE-79	Medium	Unresolved
PT-5	Supply Chain Risk via Unpinned Assets	CWE-829	Medium	Unresolved
PT-6	Reverse Tabnabbing — Unsafe External Links	CWE-1022	Low	Unresolved

2.3 Detailed Findings

PT-1 Sensitive Data Exposure — Third-Party API Key | Severity: High | State: Unresolved

CWE: CWE-798 — Use of Hard-coded Credentials

Description

An active third-party weather service API key was identified embedded in client-side JavaScript and persisting in the public repository's Git commit history. The credential is accessible to any user who clones the repository or inspects deployed application source, enabling unauthorized use at the account owner's expense with no authentication bypass required.

Exploitation Methodology

Steps to reproduce:

```
curl "https://[REDACTED-API-ENDPOINT]/data/2.5/weather?q=London&appid=<EXPOSED_KEY>"
```

1. Access the public repository or view deployed application JavaScript source.
3. Search for the `apiKey` parameter or equivalent weather API identifiers.
4. Extract the credential from the JavaScript file or review Git commit history for prior exposure.
5. Issue unauthorized API requests using the extracted key against the third-party endpoint.

Impact

- Unauthorized API consumption billed to the account owner
- Financial exposure through quota exhaustion or overage charges
- Rate limit exhaustion degrading legitimate application functionality for end users

Likelihood vs. Impact

Factor	Rating
Likelihood	**High**
Impact	**Medium**
Overall	**High**

Remediation

- Revoke and rotate the exposed API key immediately via the service provider dashboard.
- Remove all credential references from application source code.
- Migrate API calls to a server-side proxy — credentials must never be exposed client-side.
- Purge Git history using `git filter-branch` or BFG Repo Cleaner.
- Implement a secret scanning pre-commit hook (e.g., `detect-secrets`, `trufflehog`).

Detection

- Monitor API usage dashboards for anomalous request volume, geographic origin, or off-hours activity.
- Configure alerts on usage spikes or requests from unrecognized sources.
- Enable API key usage notifications through the service provider's alerting features.

Validation

Confirmed through manual review of deployed JavaScript and repository commit history. The extracted credential was used to issue a controlled test request against the third-party API endpoint. A valid data response was returned, confirming the key was active at the time of assessment.

PT-2 Insecure Cryptographic Implementation | Severity: Medium | State: Unresolved

CWE: CWE-327 — Use of a Broken or Risky Cryptographic Algorithm

Description

An RSA implementation embedded in the deployed application was identified as a portfolio demonstration component. The implementation does not meet minimum cryptographic standards and is unsuitable for any security-sensitive function. Specific weaknesses include:

- Use of small primes (<100), trivially factorable without specialized tooling
- Reliance on `Math.random()` as the entropy source — a non-cryptographic PRNG
- Private key material stored in `localStorage`, readable by any same-origin script

Risk Context

The component is accessible in the deployed application without contextual labeling indicating its non-production status. This creates risk of misinterpretation by visitors or future maintainers, and potential for inadvertent reuse in a security-sensitive context.

Exploitation Methodology

Steps to reproduce:

1. Navigate to the portfolio component containing the RSA demonstration.
3. Trigger key generation through the application interface.
4. Open browser developer tools and inspect `localStorage`.
5. Extract stored private key material and factor the weak primes trivially.

Impact

- No confidentiality or integrity guarantees if the component is reused or extended
- Reputational risk if the implementation is misrepresented as production-grade
- Risk of misuse by third parties who adopt the code without understanding its limitations

Likelihood vs. Impact

Factor	Rating
Likelihood	**Medium**
Impact	**Medium**
Overall	**Medium**

Remediation

- Add prominent labeling identifying the component as non-production demonstration code.
- Remove private key storage from `localStorage`.

- Replace `Math.random()` with `window.crypto.getRandomValues()` if the demo is retained.
- Isolate into a clearly scoped demo-only section of the application.
- Consider removing from the live deployment and serving from a sandboxed environment.

Detection

- Not applicable — this is a non-production component with no active user data in scope.

Validation

Confirmed through manual code review of the deployed application. Key generation was triggered via the interface and localStorage was inspected using browser developer tools. Private key material and prime factor values were visible in plaintext. Weak prime values were verified as trivially factorable without specialized tooling.

PT-3 Security Misconfiguration — CDN/WAF Bypass | Severity: Medium | State: Unresolved

CWE: CWE-16 — Configuration; CWE-284 — Improper Access Control

Description

The origin server IP address was identified through active reconnaissance using Burp Suite proxy interception and HTTP response header fuzzing, enabling direct access to the origin host outside of CDN/WAF protections. The identifier was discoverable via repository configuration artifacts and HTTP response metadata.

This is an access control failure, not passive information disclosure. The origin server accepts direct inbound connections, circumventing all security controls enforced at the CDN/WAF layer.

Exploitation Methodology

Steps to reproduce:

1. Proxy application traffic through Burp Suite to capture and analyze request/response headers.
3. Enumerate HTTP response headers and repository artifacts for infrastructure identifiers.
4. Extract origin IP address from configuration files, commit history, or server response metadata.
5. Issue HTTP requests directly to the origin IP, bypassing CDN routing entirely.
6. Confirm WAF bypass by receiving application responses absent of expected CDN inspection headers.
7. Conduct port scanning and service enumeration against the exposed origin host.

Impact

- Complete bypass of CDN-enforced WAF rules, DDoS mitigation, and rate limiting
- Direct targeting of origin infrastructure for exploitation or service disruption
- Expanded attack surface — origin services not intended for direct public exposure
- Enables host-level enumeration, service fingerprinting, and exploitation of unprotected services

Likelihood vs. Impact

Factor	Rating
Likelihood	**Medium**
Impact	**Medium**
Overall	**Medium**

Remediation

- Remove all origin IP references from public repositories, configuration files, and commit history.
- Enable CDN origin shielding; ensure all inbound traffic routes exclusively through the CDN.
- Restrict origin firewall rules to accept connections only from CDN provider IP ranges.
- Rotate the origin IP address if exposure cannot be fully remediated through configuration alone.
- Audit DNS records and passive DNS history (e.g., Shodan, SecurityTrails) for residual exposure.

Detection

- Monitor origin access logs for requests arriving outside CDN provider IP ranges.
- Alert on HTTP traffic missing expected CDN request headers (e.g., `CF-RAY`, `CF-Connecting-IP`).
- Implement rate limiting and access controls at the origin firewall as a defense-in-depth measure.

Validation

Origin IP was identified through Burp Suite proxy interception of HTTP response headers and manual inspection of repository configuration artifacts. Direct HTTP requests were issued to the identified origin host. Application responses were returned without traversing the CDN, confirmed by the absence of expected CDN response headers in direct-origin responses.

PT-4 DOM-Based XSS via Unsafe Sinks

Severity: Medium | **State: Unresolved**

CWE: CWE-79 — Improper Neutralization of Input During Web Page Generation (XSS)

Description

User-controlled input is written directly to the DOM via `innerHTML` without sanitization. This introduces a client-side injection vector exploitable for arbitrary JavaScript execution within the victim's browser session.

Exploitation Methodology

Vulnerable pattern identified in application source:

```
element.innerHTML = userInput;
```

1. Identify input fields or URL parameters reflected into the DOM via unsafe sinks.
3. Inject payload: `<img src=x onerror=alert(document.cookie)>`
4. Confirm JavaScript execution in the target browser context.
5. Escalate to session token exfiltration, credential harvesting, or DOM-based redirect.

Impact

- Session hijacking via authentication token theft
- Credential harvesting through DOM manipulation
- Arbitrary script execution in victim browser context
- Potential for drive-by malware delivery to application users

Likelihood vs. Impact

Factor	Rating
Likelihood	**Medium**
Impact	**Medium**
Overall	**Medium**

Remediation

- Replace all `innerHTML` assignments with `textContent` where HTML rendering is not required.
- Apply input sanitization via DOMPurify before any DOM insertion operation.
- Implement a strict Content Security Policy (CSP) header to restrict inline script execution.
- Perform a full codebase audit for unsafe DOM manipulation patterns.

Detection

- Monitor for CSP violation reports via the `report-uri` directive.
- Alert on unexpected inline script execution events in browser telemetry.
- Implement WAF rules targeting common XSS payload signatures.

Validation

Confirmed through manual code review. The innerHTML assignment pattern was identified in deployed application JavaScript. A controlled proof-of-concept payload was injected into the identified sink and JavaScript execution was confirmed in a controlled browser environment. No Content Security Policy header was present in application HTTP responses.

PT-5 Supply Chain Risk via Unpinned Third-Party Assets

Severity: Medium | **State: Unresolved**

CWE: CWE-829 — Inclusion of Functionality from Untrusted Control Sphere

Description

Third-party JavaScript dependencies are loaded via CDN using mutable version specifiers (e.g., `@latest`) without Subresource Integrity (SRI) hash enforcement. An upstream compromise of any loaded dependency results in automatic delivery of malicious code to all application users without client-side detection.

Exploitation Methodology

Steps to reproduce:

1. Identify third-party script tags loaded from external CDNs using mutable version specifiers.
3. Compromise or substitute the upstream dependency package (supply chain attack vector).
4. Malicious code is automatically served to all users on subsequent page load.
5. No integrity check prevents execution of the tampered resource.

Impact

- Full client-side compromise via upstream supply chain attack
- No application-layer detection without SRI enforcement
- All users served the application during the attack window are affected

Likelihood vs. Impact

Factor	Rating
Likelihood	**Low**
Impact	**High**
Overall	**Medium**

Remediation

- Pin all third-party dependencies to specific, audited version hashes.
- Implement Subresource Integrity (SRI) hashes on all externally loaded scripts and stylesheets.
- Establish a dependency review process prior to version updates.
- Consider self-hosting critical dependencies to eliminate CDN trust dependency.

Detection

- Monitor for SRI hash mismatches via browser console and CSP violation reports.
- Alert on unexpected changes to loaded script content hashes.
- Implement automated dependency integrity checks in the CI/CD pipeline.

Validation

Confirmed through review of application HTML source. Script tags referencing third-party CDN assets were identified using mutable version specifiers without accompanying SRI hash attributes. No integrity enforcement was present on any externally loaded resource.

PT-6 Reverse Tabnabbing — Unsafe External Links | Severity: Low | State: Unresolved

CWE: CWE-1022 — Use of Web Link to Untrusted Target with window.opener Access

Description

External links throughout the application use `target="_blank"` without the `rel="noopener noreferrer"` attribute. This allows the destination page to access and manipulate the originating window via `window.opener`, enabling phishing attacks through tab redirection.

Exploitation Methodology

Vulnerable pattern identified in application HTML:

```
<a href="https://[REDACTED]" target="_blank">Link</a>
```

1. User clicks an external link that opens in a new tab.
3. Destination page executes: `window.opener.location = "https://[attacker-controlled]"`
4. The originating application tab silently redirects to the attacker-controlled page.
5. User returns to the original tab and is presented with a credential harvesting page.

Impact

- Phishing vector enabling credential harvesting against application users
- Silent tab redirection without user awareness
- Applicable to any external link present in the application

Likelihood vs. Impact

Factor	Rating
Likelihood	**Low**
Impact	**Low**
Overall	**Low**

Remediation

- Apply rel="noopener noreferrer" to all external link anchor tags.
- Audit all templates for `target="_blank"` usage.
- Implement a linting rule to enforce `noopener noreferrer` on all external anchors.
- Consider a CSP `navigate-to` directive to restrict navigation targets.

Detection

- Monitor for abnormal navigation events in browser telemetry.
- Audit application HTML for missing `rel` attributes during code review.

Validation

Confirmed through manual review of application HTML source. Multiple external anchor elements were identified using target="_blank" without rel="noopener noreferrer". The window.opener reference was verified as accessible from the destination context.

Appendix A — Scope & Methodology

All testing was conducted under explicit written authorization with safe harbor protections in place. No testing was performed outside the defined target scope.

Methodologies Applied

- OWASP Testing Guide v4.2
- Penetration Testing Execution Standard (PTES)
- NIST SP 800-115 — Technical Guide to Information Security Testing

Testing Approach

- White-box testing with full source and repository access
- Manual validation of all findings prior to reporting
- Controlled exploitation limited to proof-of-concept scope
- No automated scanner output reported without manual confirmation

Appendix B — Vulnerability Coverage

Assessment coverage aligned with OWASP Top 10 (2021):

OWASP ID	Category	Findings
`A02:2021`	Cryptographic Failures	`PT-1, PT-2`
`A03:2021`	Injection (XSS)	`PT-4`
`A05:2021`	Security Misconfiguration	`PT-3`
`A06:2021`	Vulnerable & Outdated Components	`PT-5`
`A08:2021`	Software and Data Integrity Failures	`PT-5`

Appendix C — Glossary

API Key	A credential used to authenticate requests to an external service API.
CDN	Content Delivery Network — a distributed proxy layer that fronts origin infrastructure.
CSP	Content Security Policy — an HTTP header restricting script and resource execution.
CWE	Common Weakness Enumeration — a standardized catalog of software security weaknesses.

DOM-Based XSS	Cross-site scripting where the payload executes via client-side DOM manipulation.
OWASP	Open Web Application Security Project — industry-standard web security framework.
PII	Personally Identifiable Information.
Reverse Tabnabbing	An attack where a new tab gains control of the originating tab via window.opener.
SRI	Subresource Integrity — a browser mechanism to verify loaded resource integrity via hash.
WAF	Web Application Firewall — a security layer inspecting and filtering HTTP traffic.