OSCAR SIX RADAR

Security Assessment Report

Automated Penetration Testing & Vulnerability Analysis

httpbin.org

HIGH RISK

Urgent remediation needed

Assessment Date: January 22, 2026 at 12:45 UTC

Report ID: httpbin-org-20260122

Table of Contents

1. Summary Dashboard

Vulnerability Summary

0	1	6	4	2
CRITICAL	HIGH	MEDIUM	LOW	INFO

Detected Technologies

python · react · swagger ui · gunicorn:19.9.0 · jquery · gunicorn/19.9.0

Metric	Value
Target	**httpbin.org**
Total Findings	11
Tests Executed	13
Assessment Date	January 22, 2026
Overall Risk Level	**HIGH**

2. Executive Summary

Executive Summary

A comprehensive security assessment was conducted on httpbin.org, involving 13 distinct security tests to evaluate the application's defensive posture against common web-based attacks. The assessment identified a total of 13 security findings across various severity levels, with the majority classified as medium to low risk. While no critical vulnerabilities were discovered, one high-severity issue was identified that requires immediate attention, along with several medium-priority items that should be addressed to strengthen the overall security framework.

The most significant security risk identified is a Cross-Origin Resource Sharing (CORS) misconfiguration that could allow malicious websites to access and steal sensitive user data. This vulnerability occurs because the application improperly reflects arbitrary website origins in its security headers while allowing credentials to be transmitted. An attacker could exploit this flaw by tricking users into visiting a malicious website that then makes unauthorized requests to httpbin.org on their behalf, potentially exposing personal information or session data.

The remaining findings consist of six medium-severity issues and four low-severity concerns that, while not immediately exploitable, represent areas where security controls could be strengthened. These findings typically relate to information disclosure, missing security headers, and suboptimal configuration practices that could be leveraged by attackers as part of a broader attack strategy.

We recommend prioritizing the immediate remediation of the CORS vulnerability by implementing a strict whitelist of approved origins rather than reflecting arbitrary header values. Additionally, a systematic review and remediation of the medium-severity findings should be undertaken within the next 30-60 days to reduce the overall attack surface. Regular security assessments should be incorporated into the development lifecycle to identify and address similar issues proactively.

3. Technical Findings

# Technical Findings

## High Severity Findings (1)

### CORS Misconfiguration
A critical Cross-Origin Resource Sharing (CORS) vulnerability was identified on the target application at https://httpbin.org. The server exhibits dangerous origin

reflection behavior, where arbitrary Origin header values are directly mirrored in the Access-Control-Allow-Origin (ACAO) response header. Testing confirmed that when an Origin header value of "https://evil.com" was sent, the server responded with "Access-Control-Allow-Origin: https://evil.com" along with "Access-Control-Allow-Credentials: true".

This configuration creates a severe security exposure that allows any malicious website to make authenticated cross-origin requests on behalf of victims. An attacker can craft a malicious webpage that sends CORS requests to the vulnerable application, potentially accessing sensitive user data including authentication tokens, personal information, or performing unauthorized actions. Since credentials are explicitly allowed (ACAC: true), the attack can leverage the victim's existing session cookies, making it particularly dangerous for authenticated users.

The vulnerability violates the fundamental principle of CORS security controls and effectively negates the same-origin policy protection that browsers implement by default.

## Medium Severity Findings (6)

### Security Headers - Missing Transport and Content Protection
Multiple critical security headers are absent from the application's HTTP responses, significantly weakening the application's defense posture against common web attacks.

Transport Security Deficiencies:
The application lacks HTTP Strict Transport Security (HSTS) headers, failing to enforce HTTPS-only communication. Without HSTS, users remain vulnerable to protocol downgrade attacks where attackers can force connections over unencrypted HTTP, enabling man-in-the-middle attacks and cookie hijacking. Modern browsers rely on HSTS to prevent accidental HTTP connections and protect against SSL stripping attacks.

Content Security and Injection Protection:
No Content Security Policy (CSP) header was detected, removing a crucial defense layer against Cross-Site Scripting (XSS) attacks. CSP allows applications to define trusted sources for scripts, styles, and other resources, significantly reducing the impact of injection vulnerabilities. Without CSP, any XSS vulnerability in the

application can lead to complete compromise of user sessions and data theft.

Clickjacking Protection:
The absence of X-Frame-Options headers leaves the application vulnerable to clickjacking attacks. Malicious websites can embed the application in invisible or disguised iframes, tricking users into performing unintended actions while believing they are interacting with a different interface. This can lead to unauthorized transactions, account modifications, or data disclosure.

These missing headers represent fundamental security misconfigurations that should be addressed as standard security practices, regardless of the specific application functionality.

## Low Severity Findings (4)

### Security Headers - Additional Hardening Opportunities
Several supplementary security headers are missing, representing opportunities to further strengthen the application's security posture through defense-in-depth principles.

Content Type Protection:
The X-Content-Type-Options header is not implemented, allowing potential MIME type confusion attacks. Without the "nosniff" directive, browsers may incorrectly interpret file types, potentially executing malicious content that was uploaded as seemingly harmless file types. While less critical than other headers, this protection helps prevent certain file upload attack vectors.

Information Leakage Prevention:
No Referrer-Policy header was found, meaning the application does not control how much referrer information is shared with external sites when users navigate away from the application. This can lead to unintentional information disclosure through URL parameters or path structures being leaked to third-party sites.

Browser Feature Control:
The modern Permissions-Policy header (successor to Feature-Policy) is absent, missing an opportunity to restrict potentially dangerous browser APIs like geolocation, camera, and microphone access. While not immediately exploitable, this header provides additional control over browser features that could be abused

in conjunction with other vulnerabilities.

These findings, while lower in immediate risk, represent security best practices that contribute to a comprehensive security implementation and help prevent potential future attack vectors as the threat landscape evolves.

4. Remediation Recommendations

## Immediate Actions (Quick Wins)

---
1. CORS Origin Reflection Vulnerability
EVIDENCE: Origin: https://evil.com -> ACAO: https://evil.com, ACAC: true

REMEDIATION:
- Immediately implement a strict whitelist of allowed origins in your application
- Remove any code that directly reflects the Origin header in Access-Control-Allow-Origin
- For Python/Gunicorn applications, configure CORS middleware with explicit allowed origins only
- Never use `Access-Control-Allow-Origin: *` with credentials enabled

LEARN MORE: Search "CORS origin reflection vulnerability python flask remediation"

---
2. Cross-Site Scripting (XSS) Vulnerability
EVIDENCE: XSS found at https://httpbin.org/response-headers?page=FUZZ&Content-Type=text/html&Server=%3Cscript%3Ealert%28document.domain%29%3C%2Fscript%3E

REMEDIATION:
- Implement proper input validation and output encoding for all user inputs
- Use parameterized queries and escape HTML entities before rendering
- Add Content-Security-Policy header to mitigate XSS attacks
- Review all endpoints that accept user input, especially query parameters

LEARN MORE: Search "XSS prevention python web application input validation"

---

3. Outdated TLS Protocol Support

EVIDENCE:

- Server accepts TLS 1.0 connections at httpbin.org:443
- Server accepts TLS 1.1 connections at httpbin.org:443

REMEDIATION:

- Configure your web server/load balancer to disable TLS 1.0 and TLS 1.1
- For Gunicorn applications, update SSL configuration to support only TLS 1.2+
- Test thoroughly after changes to ensure legitimate clients can still connect
- Consider implementing TLS 1.3 for enhanced security

LEARN MORE: Search "disable TLS 1.0 1.1 gunicorn python web server configuration"

---

4. Missing Critical Security Headers

EVIDENCE: Multiple security headers missing from https://httpbin.org responses:

- Header 'strict-transport-security' was not present
- Header 'content-security-policy' was not present
- Header 'x-frame-options' was not present
- Header 'x-content-type-options' was not present

REMEDIATION:

Add the following headers to all HTTP responses:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'; style-src
'self' 'unsafe-inline'
X-Frame-Options: DENY
X-Content-Type-Options: nosniff
Referrer-Policy: strict-origin-when-cross-origin
```

LEARN MORE: Search "security headers implementation python flask gunicorn middleware"

---

5. Server Version Information Disclosure

EVIDENCE: Server: gunicorn/19.9.0

REMEDIATION:

- Configure Gunicorn to suppress version information in HTTP headers
- Add `server_tokens = False` or equivalent configuration
- Consider updating Gunicorn from version 19.9.0 to the latest stable version
- Use a reverse proxy (nginx/Apache) to mask backend server details

LEARN MORE: Search "hide gunicorn server version header information disclosure"

## Long-term Security Improvements

1. Technology Stack Modernization

Your application runs on Gunicorn 19.9.0 (released in 2018), which may have known vulnerabilities. Plan a systematic upgrade of your Python web stack including Gunicorn, Flask/Django framework, and all dependencies. Implement automated dependency scanning in your CI/CD pipeline.

2. Implement Security Middleware Layer

Given your Python/React architecture, implement a comprehensive security middleware that automatically applies security headers, input validation, and rate limiting. Consider using libraries like Flask-Security or Django-Security for standardized security controls.

3. Content Security Policy Hardening

With React frontend and Swagger UI detected, develop a tailored CSP policy that supports your specific JavaScript requirements while blocking XSS attacks. Start with a restrictive policy and gradually whitelist necessary resources.

4. API Security Enhancement

Since Swagger UI is present, ensure your API endpoints implement proper authentication, authorization, input validation, and rate limiting. Consider implementing API versioning and deprecation strategies for security updates.

5. Automated Security Testing Integration

Establish automated security scanning in your development pipeline including SAST (static analysis), DAST (dynamic testing), and dependency vulnerability scanning. The current findings suggest manual security testing gaps that automation could address.

5. Risk Assessment

## Overall Risk Assessment

Overall Risk Rating: Medium

Based on the vulnerability distribution analysis, the organization maintains a Medium risk posture. While no critical vulnerabilities are present, the existence of one high-severity vulnerability combined with six medium-severity issues creates a moderate risk environment that requires structured attention.

## Key Risk Factors Identified

The primary risk driver is the single high-severity vulnerability, which likely represents a significant attack vector that could lead to substantial system compromise or data exposure. The concentration of six medium-severity vulnerabilities indicates systemic security gaps that, while individually manageable, collectively expand the organization's attack surface. These medium-risk items may serve as stepping stones for threat actors or could be exploited in combination to escalate privileges or maintain persistence within the environment.

The four low-severity vulnerabilities, while not immediately threatening, represent maintenance items that could degrade into more serious issues if left unaddressed over time.

## Business Impact Assessment

The current risk profile suggests moderate potential for business disruption. The high-severity vulnerability poses the greatest immediate threat, potentially enabling unauthorized access to sensitive systems or data, which could result in regulatory compliance violations, reputational damage, or operational disruptions. The medium-severity vulnerabilities create cumulative risk that could facilitate lateral movement within the network or provide multiple entry points for malicious actors.

However, the absence of critical vulnerabilities indicates that catastrophic system failures or complete security compromises are less likely under the current threat landscape.

## Recommended Risk Treatment Priorities

Immediate priority should focus on addressing the high-severity vulnerability through expedited patching, configuration changes, or compensating controls. This item should be resolved within the next 30 days with dedicated resources and executive oversight.

The six medium-severity vulnerabilities should be addressed through a structured remediation program over the next 60-90 days, prioritized based on asset criticality and exploitability factors. These items should be integrated into regular maintenance cycles with appropriate resource allocation.

Low-severity vulnerabilities should be incorporated into standard maintenance windows and addressed within the next quarter as part of routine security hygiene practices.

Continuous monitoring and regular vulnerability assessments should be maintained to prevent risk escalation and ensure the security posture remains stable or improves over time.

6. Attack Analysis

```
OVERALL SECURITY ASSESSMENT
----------------------------------------
Overall Risk Level: HIGH

Total findings: 13 (0 critical, 1 high, 6 medium, 4 low, 2 informational)

Priority: Implement missing security headers to improve baseline security
posture.
```

```
Priority: Address XSS vulnerabilities to prevent client-side attacks.

ATTACK SCENARIOS
----------------------------------------
## Attack Scenarios for httpbin.org

**Cross-Origin Data Theft via CORS Misconfiguration**
An attacker hosts a malicious website that makes JavaScript requests to
httpbin.org endpoints while a victim is logged in. Due to the CORS origin
reflection vulnerability, the malicious site can read sensitive API responses
and user data that should be protected by same-origin policy, potentially
exposing authentication tokens or personal information.

**Clickjacking Attack on API Testing Interface**
An attacker embeds httpbin.org's Swagger UI interface in an invisible iframe
on a popular website, overlaying it with fake content like a "Download"
button. When users click what they think is legitimate content, they're
actually interacting with the hidden API interface, potentially triggering
unintended API calls or data modifications without their knowledge.

PRIORITIZED REMEDIATION
----------------------------------------
1. CORS Origin Reflection Vulnerability
   Priority Score: 95
   Action: Restrict CORS to specific trusted origins
   Factors: CORS can lead to data theft, Strong evidence

2. HTTPBin - Cross-Site Scripting
   Priority Score: 65
   Action: Review and remediate as appropriate
   Factors: XSS is directly exploitable

3. Missing Security Header: HTTP Strict Transport Security (HSTS)
   Priority Score: 55
   Action: Add missing security headers to server configuration
   Factors: Strong evidence

4. Missing Security Header: Content Security Policy (CSP)
   Priority Score: 55
   Action: Add missing security headers to server configuration
   Factors: Strong evidence

5. Missing Security Header: X-Frame-Options (Clickjacking Protection)
```

```
Priority Score: 55
Action: Add missing security headers to server configuration
Factors: Strong evidence
```

7. Appendix

A. Raw Findings Data

```
APPENDIX A: Detected Technologies
----------------------------------------
  - python
  - react
  - swagger ui
  - gunicorn:19.9.0
  - jquery
  - gunicorn/19.9.0

APPENDIX B: Finding Summary
----------------------------------------
1. [MEDIUM] Missing Security Header: HTTP Strict Transport Security (HSTS)
   Location: https://httpbin.org

2. [MEDIUM] Missing Security Header: Content Security Policy (CSP)
   Location: https://httpbin.org

3. [LOW] Missing Security Header: X-Content-Type-Options
   Location: https://httpbin.org

4. [MEDIUM] Missing Security Header: X-Frame-Options (Clickjacking Protection)
   Location: https://httpbin.org

5. [INFO] Missing Security Header: X-XSS-Protection
   Location: https://httpbin.org

6. [LOW] Missing Security Header: Referrer-Policy
   Location: https://httpbin.org

7. [LOW] Missing Security Header: Permissions-Policy
   Location: https://httpbin.org

8. [HIGH] CORS Origin Reflection Vulnerability
   Location: https://httpbin.org

9. [MEDIUM] Clickjacking Vulnerability - No Frame Protection
   Location: https://httpbin.org

10. [INFO] Server Version Disclosure
   Location: https://httpbin.org

11. [MEDIUM] TLS 1.0 Protocol Enabled
   Location: httpbin.org:443

12. [LOW] TLS 1.1 Protocol Enabled
   Location: httpbin.org:443
```

```
13. [MEDIUM] HTTPBin - Cross-Site Scripting
   Location: https://httpbin.org/response-headers?page=FUZZ&Content-Type=text/
html&Server=%3Cscript%3Ealert%28document.domain%29%3C%2Fscript%3E
```