**Cyber Security Task 1 Report**

## Web Application Security Testing

By CHITIRALA SA MADHU KEERTHI  
Future Interns Program – 2/July/2025



# FUTURE INTERNS

## Introduction

This report presents the findings from a web application security assessment conducted as part of the **Cyber Security Internship Program by Future Interns**. The primary objective of this task was to evaluate the security posture of a vulnerable web application by identifying and documenting real-world security flaws using ethical hacking tools and OWASP standards.

The assessment was carried out on the **Damn Vulnerable Web Application (DVWA)**, an intentionally insecure web app designed for learning and practicing security testing techniques. Industry-standard tools such as **OWASP ZAP**, **web browsers**, and a **Kali Linux/Windows environment** were used to simulate realistic attacks and uncover exploitable weaknesses.

The testing methodology focused on identifying vulnerabilities mapped to the **OWASP Top 10**—a globally recognized list of the most critical web application security risks. The scope included both passive and active scanning phases to detect misconfigurations, outdated components, insecure cross-origin policies, and improper error handling.

This report details **four verified vulnerabilities**, each supported with technical evidence, risk evaluation, and actionable remediation strategies. The goal is not only to simulate a real-world client assessment but also to strengthen practical understanding of web application threats, ethical hacking, and secure coding practices.

**Tools Used**

| Tool : | Purpose |
| --- | --- |
| OWASP ZAP : | Vulnerability scanning (passive and active) |
| DVWA : | Test web application |
| Browser : | Application interaction |
| Windows/Kali Linux : | Environment for setup |

**Vulnerability Summary Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| SNO | Vulnerability Name | Risk | Confidence | OWASP Ref | CWE ID | Page/Endpoint |
| **1** | Application Error Disclosure | Medium | Medium | A05:2021 (Security Misconfiguration) | CWE-550 | http://192.168.1.59/DVWA/dvwa/ |
| **2** | Cross-Domain Misconfiguration (CORS) | Medium | Medium | A01:2021 (Broken Access Control) | CWE-264 | https://www.zaproxy.org/main.98e1bf.css |
|  | Vulnerable JavaScript Library | Medium | Medium | A06:2021 (Vulnerable & Outdated Components) | CWE-1395 | https://cwe.mitre.org/includes/jquery.js |
| **4** | Cross-Domain JavaScript Source File Inclusion | Low | Medium | A08:2021 (Software and Data Integrity Failures) | CWE-829 | https://www.zaproxy.org/docs/alerts/10  109/ |

**Detailed Vulnerability Report**

**Vulnerability 1: Application Error Disclosure**

* **URL:** http://192.168.1.59/DVWA/dvwa/
* **Risk Level:** Medium
* **Confidence:** Medium
* **CWE ID:** [CWE-550: Exposure of Sensitive Information to an Unauthorized Actor](https://cwe.mitre.org/data/definitions/550.html)
* **WASC ID:** 13 - Information Leakage
* **OWASP Reference:** [OWASP A05:2021 – Security Misconfiguration](https://owasp.org/Top10/A05_2021-Security_Misconfiguration/)
* **Source:** OWASP ZAP Passive Scan (Alert ID: 90022)
* **Evidence Found:** Parent Directory displayed in response

**Risk Assessment**

Risk: Medium

Confidence: Medium

Impact: May enable attackers to gain insight into server-side code structure or exploit stack trace information for lateral attacks.

**Root Cause Analysis**

* Custom error pages not implemented.
* Default server error handling leaks directory structures or file paths.
* Insecure error handling policy across the web application.

**Remediation Guidance**

* Custom Error Pages: Replace default error responses with user-friendly messages that do not disclose internal details.
* Error Logging: Use server-side logging with reference IDs; do not expose log details to users. |
* Security Headers: Set headers like `X-Content-Type-Options`, `X-Frame-Options`, and `Strict-Transport-Security`.
* Stack Trace Removal: Ensure exception traces and debug information are hidden from production environment.
* Code Review: Check for improperly handled exceptions across all controller/routes. |

**Conclusion**

The disclosed error page is vulnerable to information leakage, potentially aiding attackers in mapping the server-side environment. The issue can be resolved through improved error handling and secure configuration practices, aligning with OWASP and CWE guidelines.

**References**

1) OWASP. (2021). Testing for Stack Traces (WSTG-v42-ERRH-02). Retrieved from-

<https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/02-Testing_for_Stack_Traces>

2) OWASP. (2021). Testing for Improper Error Handling (WSTG-v42-ERRH-01). Retrieved from-

<https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/08-Testing_for_Error_Handling/01-Testing_For_Improper_Error_Handling>

3) MITRE. (2021). CWE-550: Information Exposure. Retrieved from-

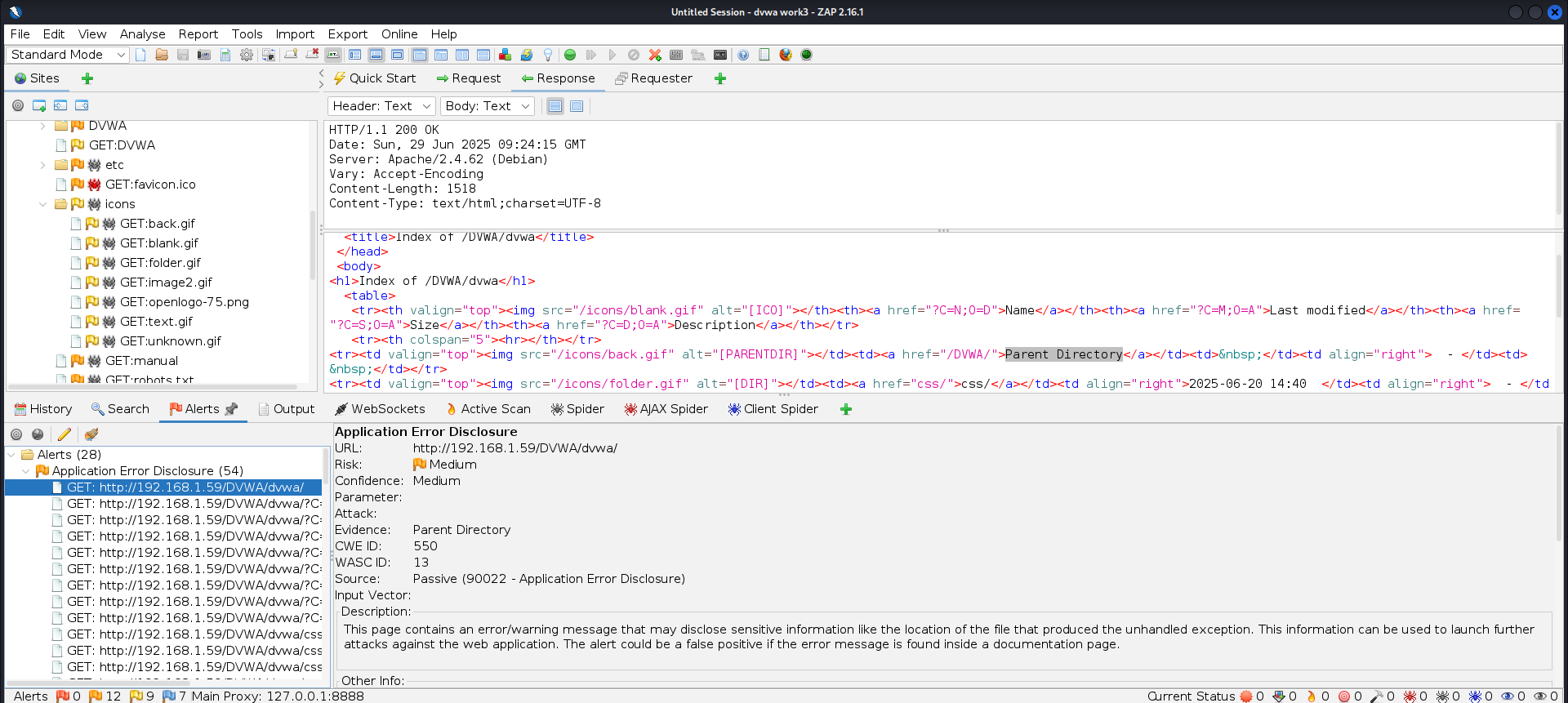
<https://cwe.mitre.org/data/definitions/550.html>

4) OWASP. (2021). A05:2021 - Security Misconfiguration. Retrieved from-

<https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>

5) OWASP. (2017). A6:2017 - Security Misconfiguration. Retrieved from-

<https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration.html>

**Output**

## Vulnerability 2: Cross-Domain Misconfiguration (CORS)

* **URL:** https://www.zaproxy.org/main.98e1bf.css
* **Risk Level:** Medium
* **Confidence:** Medium
* **CWE ID:** CWE-264: Permissions, Privileges, and Access Controls
* **WASC ID:** 14 – Server Misconfiguration
* **OWASP Reference:** OWASP A01:2021 – Broken Access Control
* **Source:** OWASP ZAP Passive Scan (Alert ID: 10098)
* **Evidence Found:** Access-Control-Allow-Origin header allows arbitrary origins

### Risk Assessment

* **Risk:** Medium
* **Confidence:** Medium
* **Impact:** A misconfigured CORS policy may allow untrusted third-party sites to access or read application responses, increasing the risk of data theft, session exposure, or misuse of unauthenticated but sensitive APIs.

### Root Cause Analysis

* The server sets an overly permissive Access-Control-Allow-Origin header, potentially allowing any domain to make cross-origin requests.
* The misconfiguration ignores the principles of the Same-Origin Policy (SOP), which normally protects data shared between browser and server.
* CORS policies may have been configured with wildcards (\*) or without domain validation.

**Remediation Guidance**

**1) Restrict Allowed Origins:**

Configure CORS headers to allow only **trusted** domains. Avoid using or reflecting arbitrary Origin headers in the response.

Example: Access-Control-Allow-Origin: https://trusted.example.com

**2) Remove CORS Headers if Not Required:**  
If the endpoint does not need to serve cross-domain requests, do not set any CORS-related headers.

**3) Validate Origin on the Server Side:**  
Use server-side logic to validate incoming requests and only return CORS headers for approved origins.

**4) Do Not Rely on IP Whitelisting Alone:**  
Avoid making sensitive, unauthenticated data publicly accessible even if behind IP whitelisting as CORS misconfigurations can expose it.

**5) Review All Endpoints for Unauthenticated Access:**  
Ensure sensitive APIs are behind authentication, not just access control headers.

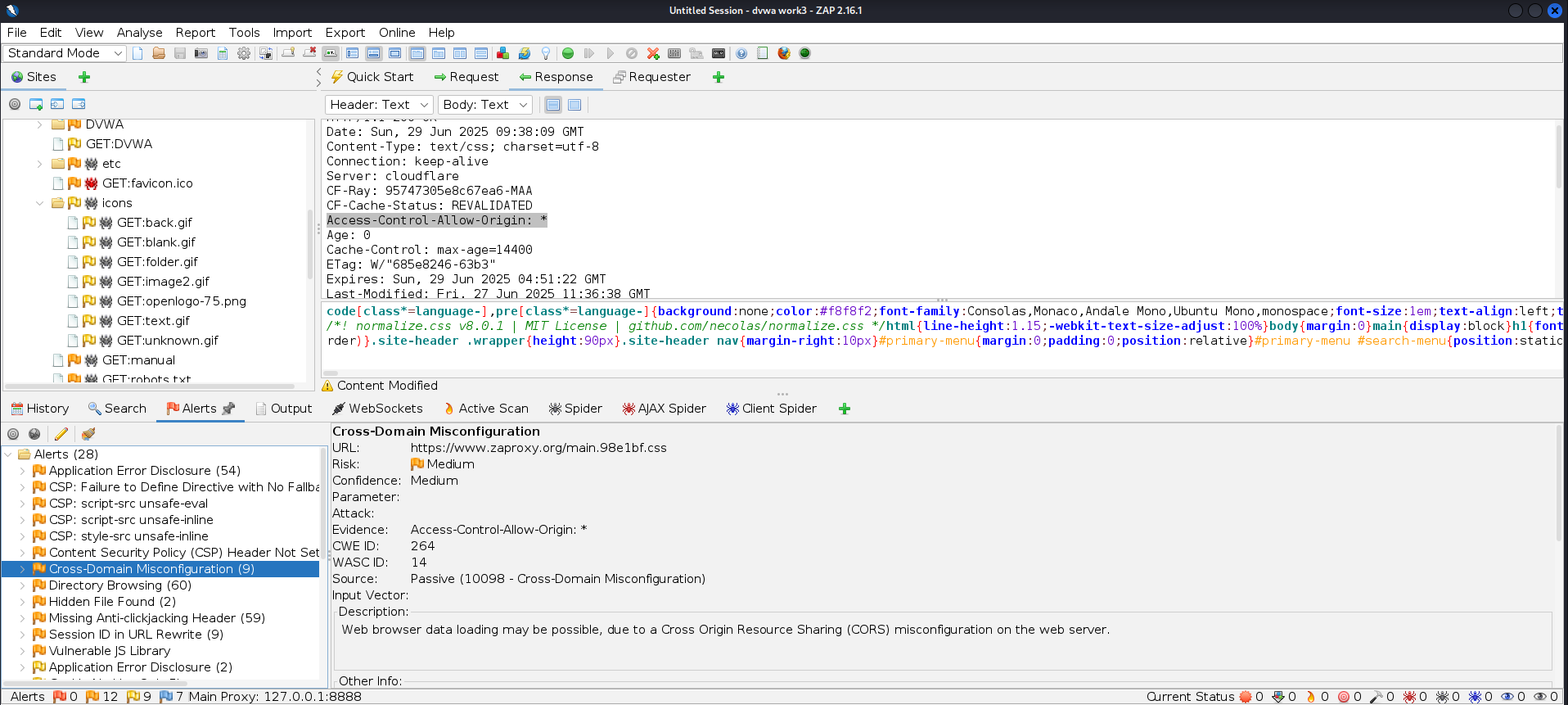
**Conclusion**

The presence of a misconfigured Access-Control-Allow-Origin header poses a security risk by weakening browser-enforced access controls. While modern browsers prevent unauthorized access to authenticated APIs, any publicly available sensitive resources may still be exposed. This issue aligns with **OWASP A01:2021 – Broken Access Control** and **CWE-264**. Properly scoping allowed domains and validating access at the server level is necessary to maintain strong access control.

### References

1. OWASP. (2021). [A01:2021 – Broken Access Control](https://owasp.org/Top10/A01_2021-Broken_Access_Control/)
2. MITRE. (2021). [CWE-264: Permissions, Privileges, and Access Controls](https://cwe.mitre.org/data/definitions/264.html)
3. OWASP. (2017). [A5:2017 – Broken Access Control](https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html)
4. Fortify. (n.d.). [Overly Permissive CORS Policy](https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.html5_overly_permissive_cors_policy)
5. Mozilla. (n.d.). [CORS Documentation](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS)

**Output**

****

## Vulnerability 3: Vulnerable JavaScript Library

* **URL:** https://cwe.mitre.org/includes/jquery.js
* **Risk Level:** Medium
* **Confidence:** Medium
* **CWE ID:** CWE-1395 – Use of Deprecated or Unsafe Third-Party Components
* **OWASP Reference:** OWASP A06:2021 – Vulnerable and Outdated Components
* **Source:** OWASP ZAP Passive Scan (Alert ID: 10003)
* **Evidence Found:** /\*! jQuery v1.10.2
* **Library Detected:** jquery version 1.10.2
* **CVEs Associated:**
  + [CVE-2020-11023](https://nvd.nist.gov/vuln/detail/CVE-2020-11023)
  + [CVE-2020-11022](https://nvd.nist.gov/vuln/detail/CVE-2020-11022)
  + [CVE-2015-9251](https://nvd.nist.gov/vuln/detail/CVE-2015-9251)
  + [CVE-2019-11358](https://nvd.nist.gov/vuln/detail/CVE-2019-11358)

### Risk Assessment

* **Risk:** Medium
* **Confidence:** Medium
* **Impact:** Vulnerable or outdated JavaScript libraries like jQuery v1.10.2 may be exploited by attackers to perform cross-site scripting (XSS), DOM manipulation, or even privilege escalation via chained vulnerabilities.

### Root Cause Analysis

* The application includes an outdated and vulnerable version of jQuery (v1.10.2), which contains several publicly disclosed vulnerabilities.
* It may have been included without regular updates or monitoring, which is a common issue in long-term production deployments.

### Remediation Guidance

**1) Upgrade jQuery:**

* Replace version 1.10.2 with the latest stable version of jQuery (e.g., 3.6.x or later).
* Ensure compatibility by testing all front-end scripts after the upgrade.

**2) Use a CDN with Subresource Integrity (SRI):**  
To prevent tampering and ensure authenticity:

<script src="https://code.jquery.com/jquery-3.6.0.min.js"

integrity="sha384-KyZXEAg3QhqLMpG8r+Knujsl5+5hb7O4FJt8vU1h8U9XaKf5f2zGxgIew6TwRCUc"

crossorigin="anonymous"></script>

**3) Monitor Dependencies:**Use tools like:

* [Snyk](https://snyk.io/)
* [Retire.js](https://retirejs.github.io/retire.js/)
* [npm audit](https://docs.npmjs.com/cli/v7/commands/npm-audit)  
  To track outdated libraries.

**4) Secure Build Pipeline:**  
Ensure CI/CD processes include dependency scanning and alerts for known CVEs in third-party libraries.

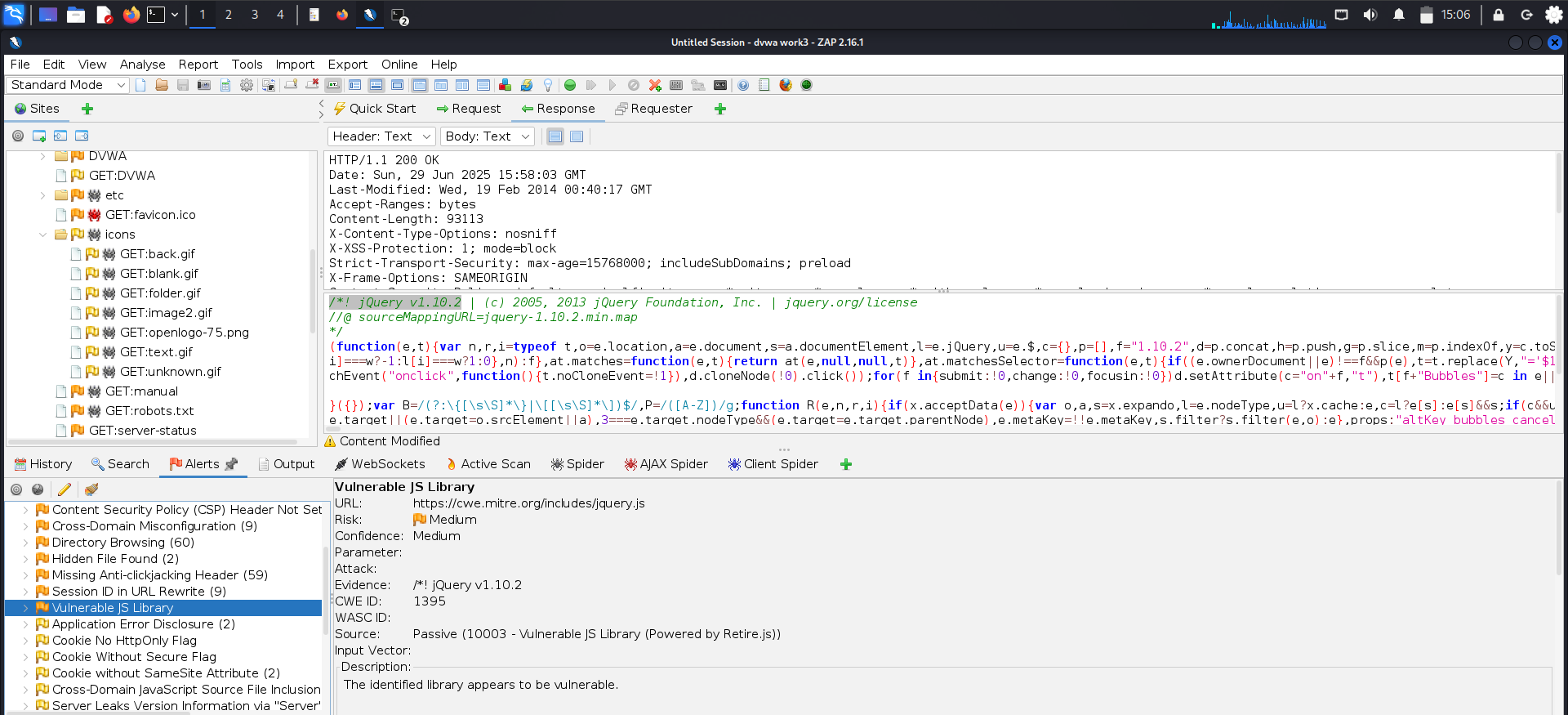
### Conclusion

Using jQuery v1.10.2 exposes the application to known client-side vulnerabilities. Keeping third-party components up to date is crucial to reduce attack surface and meet modern web security standards. This issue aligns with **OWASP A06:2021 – Vulnerable and Outdated Components** and **CWE-1395**.

### References

1. OWASP. (2021). [A06:2021 – Vulnerable and Outdated Components](https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/)
2. MITRE. (2021). [CWE-1395: Use of Deprecated or Unsafe Third-Party Components](https://cwe.mitre.org/data/definitions/1395.html)
3. National Vulnerability Database. (2019). [CVE-2019-11358](https://nvd.nist.gov/vuln/detail/CVE-2019-11358)
4. National Vulnerability Database. (2020). [CVE-2020-11022](https://nvd.nist.gov/vuln/detail/CVE-2020-11022)
5. GitHub Advisory Database. [jQuery 1.x Vulnerability Report](https://github.com/advisories/GHSA-rmxg-73gg-4p98)

**Output**

****

## Vulnerability 4: Cross-Domain JavaScript Source File Inclusion

* **URL:** https://www.zaproxy.org/docs/alerts/10109/
* **Risk Level:** Low
* **Confidence:** Medium
* **Parameter:** https://cdnjs.cloudflare.com/ajax/libs/lunr.js/2.3.6/lunr.min.js
* **CWE ID:** CWE-829: Inclusion of Functionality from Untrusted Control Sphere
* **WASC ID:** 15 – Application Misconfiguration
* **OWASP Reference:** OWASP A08:2021 – Software and Data Integrity Failures
* **Source:** OWASP ZAP Passive Scan (Alert ID: 10017)
* **Evidence Found:** <script src="https://cdnjs.cloudflare.com/ajax/libs/lunr.js/2.3.6/lunr.min.js"></script>

### Risk Assessment

* **Risk:** Low
* **Confidence:** Medium
* **Impact:** Inclusion of JavaScript files from third-party sources introduces risk if those external sources are compromised. An attacker could inject malicious scripts, especially if content security policies (CSP) are weak or not enforced.

### Root Cause Analysis

* JavaScript is included from an external domain (cdnjs.cloudflare.com) not under the site's control.
* There is no evidence that Subresource Integrity (SRI) or CSP policies are enforced to validate the script’s authenticity.
* This violates secure software supply chain practices.

### Remediation Guidance

* **Use Trusted Sources Only:** Only load scripts from sources you control or from verified and secure CDNs.
* **Apply Subresource Integrity (SRI):** Add integrity hashes to external script tags.  
  Example:<script src="..." integrity="sha384-..." crossorigin="anonymous"></script>
* **Content Security Policy (CSP):** Enforce strict CSP headers to limit script execution sources.
* **Code Review:** Periodically audit external dependencies and their security advisories.

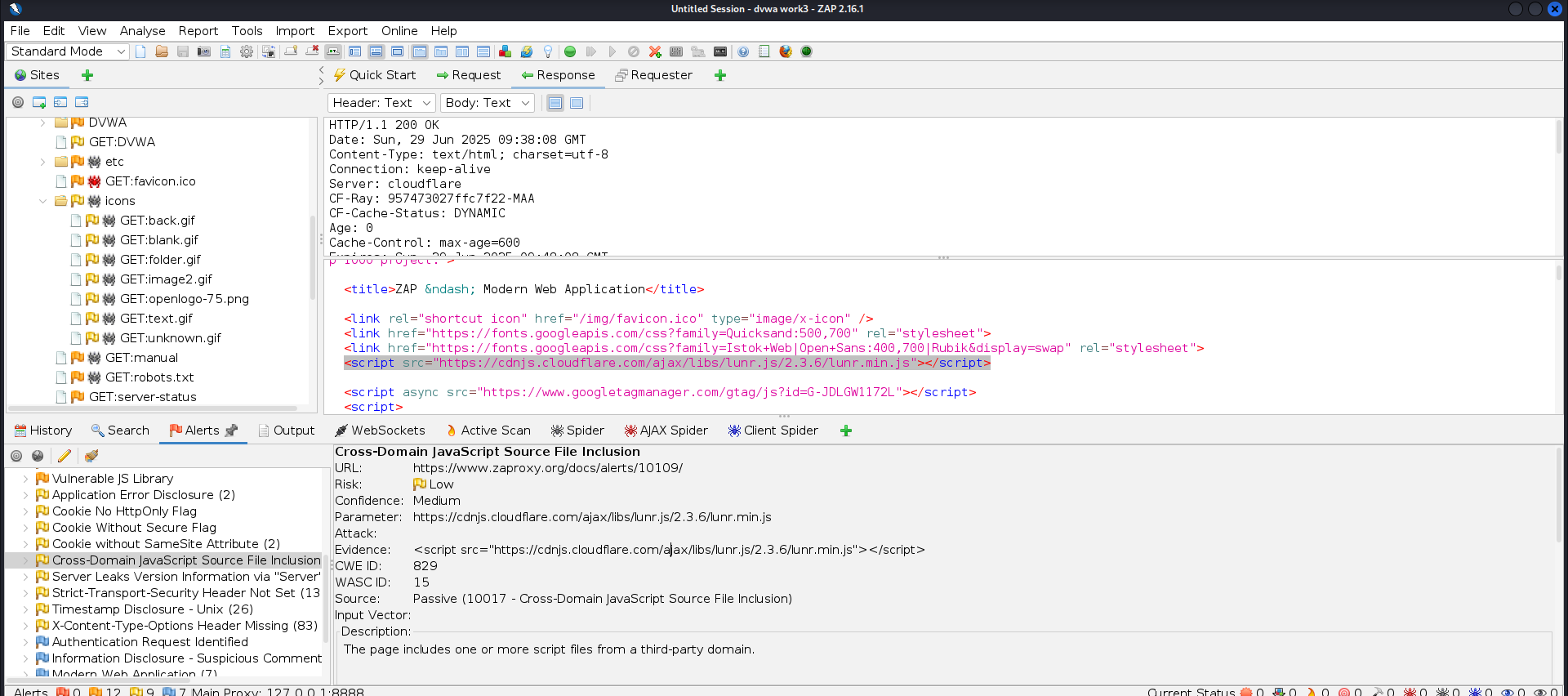
### Conclusion

Including JavaScript from a third-party source can create a software supply chain risk. This vulnerability is categorized under **OWASP A08:2021**, which focuses on software and data integrity failures. It is advisable to restrict script sources using **Subresource Integrity** and **CSP**, and avoid untrusted domains.

### References (APA Style)

1. CWE. (n.d.). CWE-829: Inclusion of Functionality from Untrusted Control Sphere. Retrieved from: <https://cwe.mitre.org/data/definitions/829.html>
2. OWASP. (2021). A08:2021 – Software and Data Integrity Failures. Retrieved from: <https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/>
3. OWASP Foundation. (n.d.). Content Security Policy (CSP) Cheat Sheet. Retrieved from: <https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html>

**Output**



**OWASP Top 10 Mapping Checklist Table**

|  |  |  |
| --- | --- | --- |
| OWASP Top 10 Issue | Detected | Reference in Report |
| A01: Broken Access Control |  | Vulnerability 2 |
| A05: Security Misconfig |  | Vulnerability 1 |
| A06: Vulnerable Components |  | Vulnerability 3 |
| A08: Software Integrity |  | Vulnerability 4 |
| Others (A02, A03, ...) |  | ----- |

**Appendix**

Including Full ZAP scan report pdf



### ****Executive Summary****

This security assessment was conducted as part of the **Cyber Security Task 1** for the Future Interns program. The objective was to evaluate the security posture of a test web application—**Damn Vulnerable Web Application (DVWA)**—using ethical hacking tools and techniques aligned with **OWASP Top 10** standards.Using **OWASP ZAP**, a series of passive scans were performed to detect real-world vulnerabilities that commonly affect modern web applications. The following **four security issues** were identified and documented:

1. **Application Error Disclosure** – Medium Risk
2. **Cross-Domain Misconfiguration (CORS)** – Medium Risk
3. **Use of a Vulnerable JavaScript Library** – Medium Risk
4. **Cross-Domain JavaScript Source Inclusion** – Low Risk

Each vulnerability was analyzed in depth with respect to its **CWE ID**, **OWASP classification**, risk level, and technical impact. Detailed remediation steps were provided to guide secure development and deployment practices.These issues highlight weaknesses such as insecure error handling, misconfigured access controls, reliance on outdated third-party components, and risks from untrusted content sources. If left unaddressed, these flaws could be exploited by attackers to gain unauthorized access, leak sensitive data, or compromise application integrity.The assessment reinforces the importance of secure coding, regular vulnerability scanning, and maintaining updated dependencies. The provided mitigation strategies serve as actionable guidance to strengthen the application’s defenses against common threats. Overall, the findings simulate real-world client

deliverables and reflect hands-on experience with penetration testing, secure development principles, and professional security reporting.