Bug Bounty Report 01

https://nodejs.org

Vulnerability Title:

Personally Identifiable Information (PII) Disclosure

Vulnerability Description:

The response received from the following URL contains Personally Identifiable Information (PII), including credit card number (CC number) and sensitive data like social security number (SSN). This constitutes a serious data security breach and violates privacy regulations.

Affected Components:

The vulnerability is present in the response obtained from the following URL:

https://nodejs.org/en/blog/announcements/v20-release-announce

Impact Assessment:

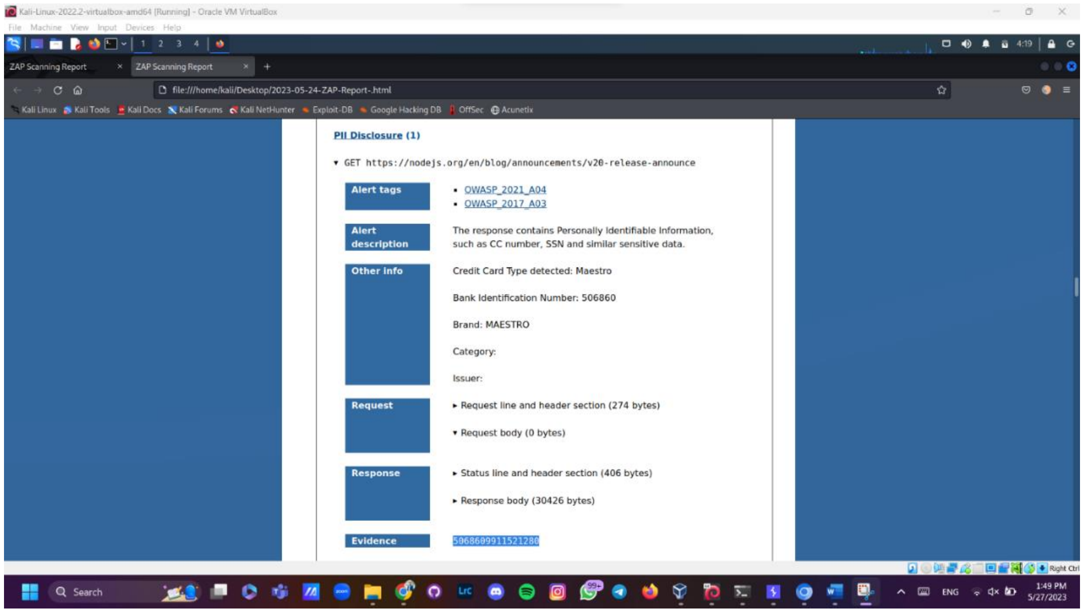
The disclosure of PII puts users at significant risk of identity theft, financial fraud, and other forms of abuse. Attackers can potentially exploit this sensitive information for unauthorized transactions, impersonation, and other malicious activities. The affected individuals may suffer financial losses and reputational damage.

Steps to Reproduce:

1. Send a GET request to the URL: https://nodejs.org/en/blog/announcements/v20-release-announce
2. Analyze the response received, focusing on the response body.
3. Look for Personally Identifiable Information (PII), such as credit card numbers, social security numbers, or any other sensitive data.
4. Observe the presence of the following PII:

- Credit Card Type: Maestro
- Bank Identification Number (BIN): 506860

Proof of Concept:



PII Disclosure (1)

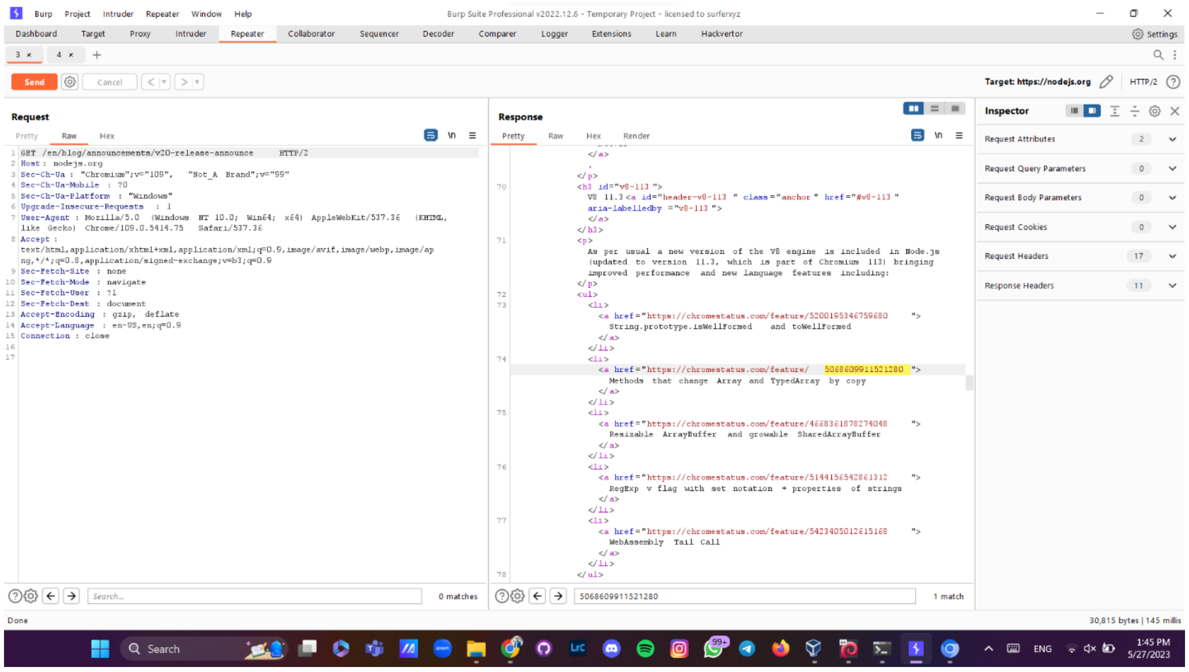
▾ GET https://nodejs.org/en/blog/announcements/v20-release-announce

Alert tags	• OWASP_2021_A04 • OWASP_2017_A03
Alert description	The response contains Personally Identifiable Information, such as CC number, SSN and similar sensitive data.
Other info	Credit Card Type detected: Maestro Bank Identification Number: 506860 Brand: MAESTRO Category: Issuer:
Request	▸ Request line and header section (274 bytes) ▾ Request body (0 bytes)
Response	▸ Status line and header section (406 bytes) ▸ Response body (30426 bytes)
Evidence	5068609911521280

1. Look for any instances of Personally Identifiable Information (PII) within the response body. In this case, the provided evidence is the number "5068609911521280".

2. Note the type of PII mentioned in the report, such as credit card number (CC number), social security number (SSN), or any other sensitive data.

3. Verify that the PII identified in the response matches the type of PII mentioned in the report. In this case, it seems to be a credit card number.



Request

```
GET /en/blog/announcements/v20-release-announce HTTP/2
Host: nodejs.org
Sec-Ch-Ua: "Chromium";v="109", "Not_A Brand";v="99"
Sec-Ch-Ua-Mobile: ?0
Sec-Ch-Ua-Platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/109.0.5414.75 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
```

Response

```
        </a>
        .
      </p>
      <h3 id="v8-113">
        V8 11.3<a id="header-v8-113" class="anchor" href="#v8-113"
        aria-labelledby="v8-113">
        </a>
      </h3>
      <p>
        As per usual a new version of the V8 engine is included in Node.js
        (updated to version 11.3, which is part of Chromium 113) bringing
        improved performance and new language features including:
      </p>
      <ul>
        <li>
          <a href="https://chromestatus.com/feature/5200195346759680">
            String.prototype.isWellFormed and toWellFormed
          </a>
        </li>
        <li>
          <a href="https://chromestatus.com/feature/5068609911521280">
            Methods that change Array and TypedArray by copy
          </a>
        </li>
        <li>
          <a href="https://chromestatus.com/feature/4668361878274048">
            Resizable ArrayBuffer and growable SharedArrayBuffer
          </a>
        </li>
        <li>
          <a href="https://chromestatus.com/feature/5144156542861312">
            RegExp v flag with set notation + properties of strings
          </a>
        </li>
        <li>
          <a href="https://chromestatus.com/feature/5423405012615168">
            WebAssembly Tail Call
          </a>
        </li>
      </ul>
```

Search: 5068609911521280 — 1 match

Proposed Mitigation or Fix:

To address this vulnerability and prevent further exposure of PII, the following steps should be taken:

1. Implement strict data handling and storage practices to ensure the protection of PII. Apply encryption and access controls to sensitive information, such as credit card numbers and social security numbers.
2. Regularly review and audit the handling of PII to identify and address any potential vulnerabilities.
3. Follow industry best practices for secure coding and secure application development to minimize the risk of PII disclosure.
4. Conduct regular security assessments and penetration testing to identify and remediate vulnerabilities before they can be exploited.
5. Provide training and awareness programs to educate employees and developers about the importance of data privacy and security.
6. Comply with relevant data protection regulations and guidelines, such as the General Data Protection Regulation (GDPR) or local privacy laws.

It is recommended that the development team and website administrators promptly address this vulnerability and implement the proposed mitigations to protect user data and prevent potential harm.

Request

```
GET https://nodejs.org/en/blog/announcements/v20-release-announce HTTP/1.1
Host: nodejs.org
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: https://nodejs.org/sitemap.xml
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 24 May 2023 19:16:31 GMT
Content-Type: text/html
Connection: keep-alive
last-modified: Wed, 24 May 2023 15:56:21 GMT
Cache-Control: public, max-age=14400, s-maxage=14400
CF-Cache-Status: HIT
Strict-Transport-Security: max-age=31536000; includeSubDomains; preload
X-Content-Type-Options: nosniff
Server: cloudflare
CF-RAY: 7cc7dfa09abbb300-CMB
Content-Length: 30426
```

Bug Bounty Report 02

https://www.amazon.in

Vulnerability title:

Open Redirect Vulnerability in Amazon.in

Vulnerability description:

The application at https://www.amazon.in is affected by an open redirect vulnerability. This vulnerability allows an attacker to manipulate the "location" parameter in the URL, leading to unauthorized redirection of users to arbitrary external websites. Open redirects are considered a security risk as they can be exploited by spammers or used in phishing attacks.

Affected components:

The vulnerable component is the "redirect.html" page on the Amazon.in website.

Impact assessment:

The impact of this vulnerability includes potential phishing attacks, where an attacker can craft malicious links that appear to be legitimate Amazon URLs but redirect users to malicious websites. This can result in users unknowingly disclosing sensitive information or falling victim to scams. The vulnerability can also be exploited by spammers to deceive users and spread malicious content.

Steps to reproduce:

1. Obtain a URL of the form:
1. `https://www.amazon.in/gp/redirect.html?location=<malicious_url>&source=nav_linktree&token=<token_value>`
2. Replace `<malicious_url>` with the desired malicious destination URL.
3. Craft a deceptive message or disguise the URL as a legitimate Amazon link to trick users into clicking it.
4. Send the manipulated URL to a victim or host it in a location where it may be clicked by unsuspecting users.
5. When the victim clicks the link, they will be redirected to the malicious URL specified by the attacker.

Proposed mitigation or fix:

To mitigate the open redirect vulnerability, the following steps should be taken:

1. Implement input validation and sanitization for the "location" parameter to ensure that it only allows trusted and whitelisted URLs.
2. Avoid allowing direct user input to control offsite redirects. Instead, use a safe redirect mechanism that restricts redirection to relative URIs or a predefined list of trusted domains.
3. Regularly update and patch the application to address any security vulnerabilities and follow secure coding practices.
4. Conduct security testing, including vulnerability assessments and penetration testing, to identify and remediate any other potential security flaws.
5. Educate users about the risks of clicking on unfamiliar or suspicious links and encourage them to be cautious while browsing external websites.
6. Implement an active monitoring system to detect and mitigate any attempted abuse or exploitation of the open redirect vulnerability.

It is recommended that the Amazon.in development team addresses this vulnerability promptly to ensure the security and trustworthiness of their website.

Bug Bounty Report 03

https://moonpay.com

Vulnerability title:

Cloud Metadata Exposure in MoonPay NGINX Server

Vulnerability description:

The NGINX server configured on moonpay.com is vulnerable to a Cloud Metadata Attack. This vulnerability allows an attacker to abuse the misconfigured NGINX server to access the instance metadata maintained by cloud service providers such as AWS, GCP, and Azure. The metadata, which contains sensitive information, is typically accessible via the internal unroutable IP address '169.254.169.254'. Improper NGINX configuration can expose this IP address and allow unauthorized access to the cloud metadata.

Affected components:

The vulnerable component is the NGINX server configuration on moonpay.com.

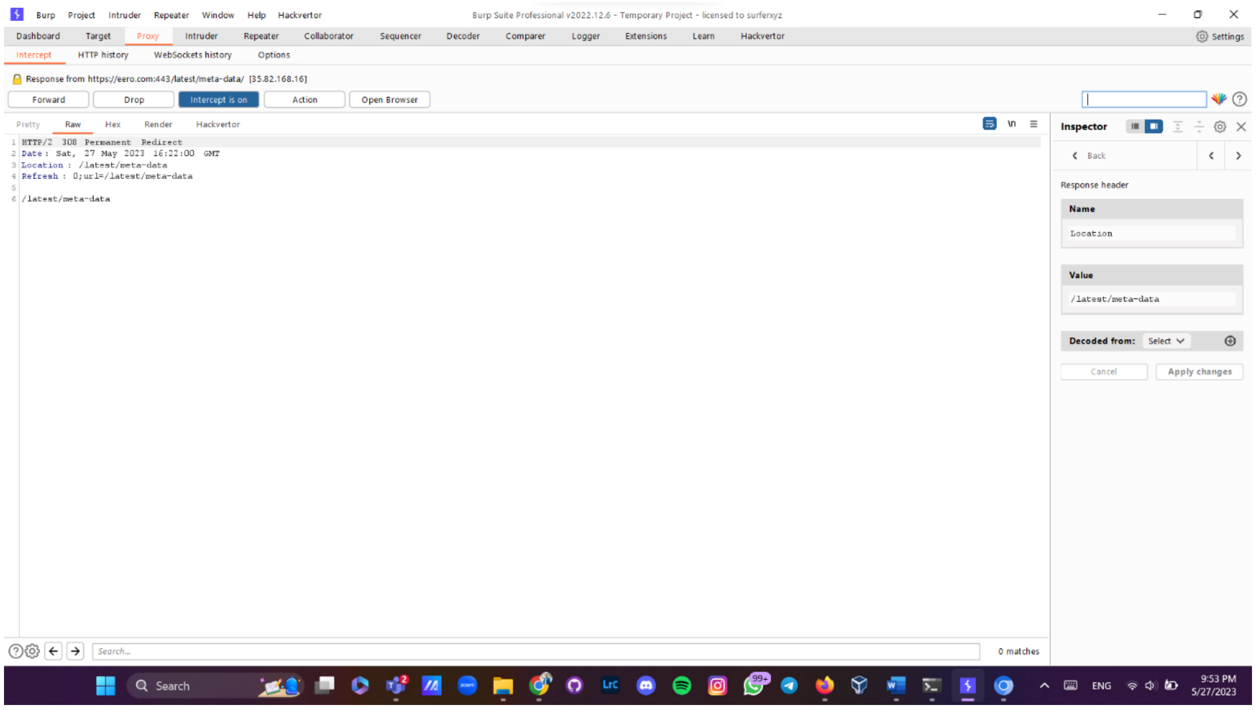
Impact assessment:

The impact of this vulnerability is severe as it can potentially lead to a complete compromise of the system. Accessing the cloud metadata can provide an attacker with sensitive information about the underlying cloud infrastructure, including access tokens, security credentials, instance details, and potentially other confidential data. With this information, an attacker can escalate their privileges, perform lateral movement, and launch further attacks on the compromised system.

Steps to reproduce:

1. Craft an HTTP request with the following details:
   - Method: GET
   - URL: https://moonpay.com/latest/meta-data/
   - Host Header: 169.154.169.254
   - User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0
   - Pragma: no-cache
   - Cache-Control: no-cache
2. Send the crafted request to the target server (moonpay.com).

Proof of concept:


Burp Suite Professional v2022.12.6 - Temporary Project - licensed to surferxyz
Response from https://eero.com:443/latest/meta-data/ [35.82.168.16]
Forward
Drop
Intercept is on
Action
Open Browser
Inspector
Response header

```
HTTP/2  308  Permanent  Redirect
Date :  Sat,  27 May 2023  16:22:00  GMT
Location :  /latest/meta-data
Refresh :  0;url=/latest/meta-data

/latest/meta-data
```

Name	Location
Value	/latest/meta-data

attempt to reproduce the Cloud Metadata Attack and check if the server responds with the cloud metadata information. It's important to note that this attack relies on a misconfigured NGINX server, so the success of the attack may vary depending on the target server's configuration.

Proposed mitigation or fix:

To mitigate the Cloud Metadata Exposure vulnerability, the following steps should be taken:

1. Review and update the NGINX server configuration to ensure that the 'Host' header is not blindly trusted or used to set variables without proper validation.
2. Remove any reliance on user-controlled input, such as the 'Host' header, in NGINX configurations.
3. Implement strict access controls to restrict access to the internal unroutable IP address '169.254.169.254' to trusted sources only.
4. Regularly monitor and review NGINX configuration for any misconfigurations or insecure practices.
5. Keep NGINX and all other software components up to date with the latest security patches.
6. Conduct regular security assessments and penetration testing to identify and remediate any potential vulnerabilities in the NGINX server configuration.

7. Educate developers and administrators about secure NGINX configuration practices and the risks associated with misconfigured servers.
8. Implement proper logging and monitoring mechanisms to detect and respond to any attempts of unauthorized access to cloud metadata.

It is recommended that the MoonPay team addresses this vulnerability promptly by securing their NGINX server configuration to prevent unauthorized access to cloud metadata.

Bug Bounty Report 04

https://metamask.io

Vulnerability title:

Hash Disclosure - Mac OSX Salted SHA-1 in MetaMask Website

Vulnerability description:

The MetaMask website at metamask.io has a vulnerability that discloses a hash. Specifically, the web server is leaking a Mac OSX salted SHA-1 hash. Hashes are used to protect credentials or other sensitive resources, and their disclosure can lead to potential security risks.

Affected components:

The vulnerability exists in the web server configuration of metamask.io.

Impact assessment:

The impact of this vulnerability depends on the sensitivity of the hashed information and the strength of the hashing algorithm used. While SHA-1 is considered weak for hashing passwords and sensitive data, its disclosure can still provide attackers with useful information for targeted attacks. With the disclosed hash, an attacker could potentially attempt to crack it offline using hash cracking techniques, such as rainbow tables or brute force methods. If successful, the attacker may gain unauthorized access to user accounts or sensitive information associated with those accounts.

Steps to reproduce:

1. Access the following URL: https://metamask.io/files/pentest-report_metamask.pdf
2. Observe the response data for any disclosed hashes.

Proposed mitigation or fix:

To mitigate the Hash Disclosure vulnerability, the following steps should be taken:

1. Ensure that hashes, especially those used to protect credentials or sensitive information, are not leaked by the web server or database.
2. Review the web server configuration to ensure that hashed values are not unintentionally exposed in the response.
3. Implement proper access controls to restrict access to hashed values to only authorized entities.
4. Consider using stronger and more secure hashing algorithms, such as SHA-256 or bcrypt, instead of the weak SHA-1 algorithm.
5. Implement salted hashes with a unique and random salt for each hashed value to increase the complexity and security of the hashes.
6. Regularly audit the application code, server configuration, and database to identify any potential information disclosure vulnerabilities.
7. Educate developers and administrators about secure coding practices and the risks associated with hash disclosure.
8. Monitor and log all access to hashed values to detect any suspicious or unauthorized activities.

It is recommended that the MetaMask development team addresses this vulnerability promptly by ensuring that the web server does not disclose hashes, especially those used to protect credentials or sensitive resources.

Bug Bounty Report 05

https://metamask.io

Vulnerability title:

External Redirect Vulnerability in MetaMask Website

Vulnerability description:

The MetaMask website at metamask.io has an external redirect vulnerability. The web server is using a redirect mechanism that allows an external URL to be set in the Location header of the response. This functionality can be abused by attackers to trick users into navigating to a site other than the intended destination, leading to potential social engineering or phishing attacks.

Affected components:

The vulnerability exists in the web server configuration of metamask.io.

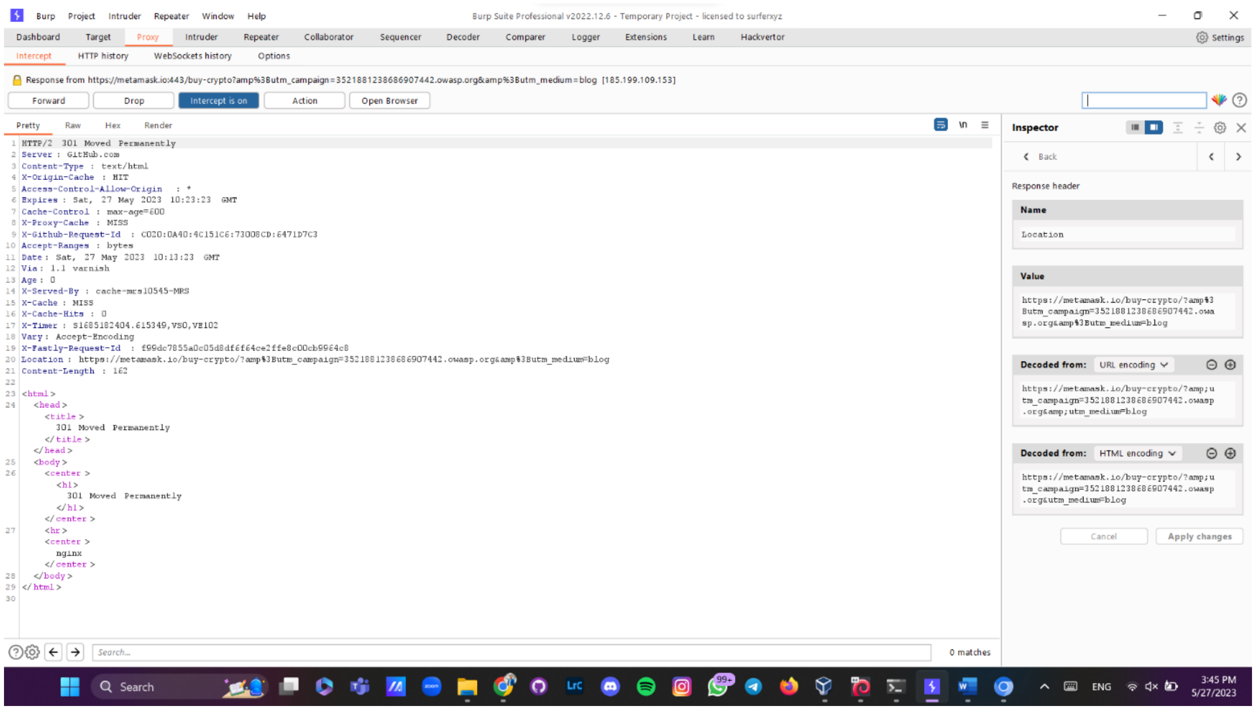
Impact assessment:

The impact of this vulnerability depends on the attacker's intent and the nature of the external URL provided in the redirect. If an attacker crafts a malicious URL and convinces users to click on it, they can redirect the users to a fraudulent website that mimics a legitimate one. This can lead to various security risks, such as phishing attacks, credential theft, malware distribution, or financial fraud. The impact can range from compromising user accounts to financial loss or damage to the reputation of MetaMask.

Steps to reproduce:

1. Access the following URL: https://metamask.io/buy-crypto?amp%3Butm_campaign=3521881238686907442.owasp.org&amp%3Butm_medium=blog
2. Observe the response data and inspect the Location header for the external URL.

Proof of concept:



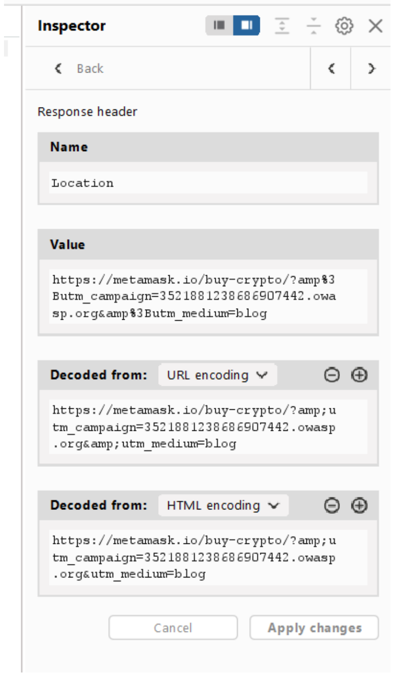
Response from https://metamask.io:443/buy-crypto?amp%3Butm_campaign=3521881238686907442.owasp.org&amp%3Butm_medium=blog [185.199.109.153]

```
HTTP/2 301 Moved Permanently
Server : GitHub.com
Content-Type : text/html
X-Origin-Cache : HIT
Access-Control-Allow-Origin  : *
Expires : Sat, 27 May 2023 10:23:23 GMT
Cache-Control : max-age=600
X-Proxy-Cache : MISS
X-Github-Request-Id  : C020:0A40:4C151C6:73008CD:6471D7C3
Accept-Ranges : bytes
Date : Sat, 27 May 2023 10:13:23 GMT
Via : 1.1 varnish
Age : 0
X-Served-By : cache-mrs10545-MRS
X-Cache : MISS
X-Cache-Hits : 0
X-Timer : S1685182404.615349,VS0,VE102
Vary : Accept-Encoding
X-Fastly-Request-Id  : f99dc7855a0c05d8df6f64ce2ffe8c00cb9964c8
Location : https://metamask.io/buy-crypto/?amp%3Butm_campaign=3521881238686907442.owasp.org&amp%3Butm_medium=blog
Content-Length : 162

<html>
  <head>
    <title>
      301 Moved Permanently
    </title>
  </head>
  <body>
    <center>
      <h1>
        301 Moved Permanently
      </h1>
    </center>
    <hr>
    <center>
      nginx
    </center>
  </body>
</html>
```

1. Check the response headers for the "Location" header. The report mentions that the response contains a redirect in its "Location" header, which allows an external URL to be set.

2. Verify that the "Location" header contains the redirected URL or domain that you injected in the parameter value.



Inspector

Response header

Name: Location

Value:
```
https://metamask.io/buy-crypto/?amp%3Butm_campaign=3521881238686907442.owasp.org&amp%3Butm_medium=blog
```

Decoded from: URL encoding
```
https://metamask.io/buy-crypto/?amp;utm_campaign=3521881238686907442.owasp.org&amp;utm_medium=blog
```

Decoded from: HTML encoding
```
https://metamask.io/buy-crypto/?amp;utm_campaign=3521881238686907442.owasp.org&utm_medium=blog
```

Cancel | Apply changes

Proposed mitigation or fix:

To mitigate the External Redirect vulnerability, the following steps should be taken:

1. Implement strict input validation and sanitation for the redirect functionality. Treat all user-supplied input as potentially malicious.
2. Adopt an "accept known good" input validation strategy by using an allowlist of acceptable inputs that strictly conform to specifications.
3. Reject any input that does not strictly conform to specifications or transform it into a safe and expected format.
4. Avoid relying exclusively on denylists, as they may be insufficient to prevent all possible attacks.
5. Utilize an allowlist of approved URLs or domains to be used for redirection, ensuring that only trusted and authorized destinations are allowed.
6. Consider implementing an intermediate disclaimer page that warns users about leaving the current site and provides clear information about the external link they are about to visit.
7. Apply a long timeout or require user interaction (e.g., clicking a confirmation button) before the redirect occurs to give users the opportunity to review and validate the destination.
8. Be cautious when generating the disclaimer page to prevent cross-site scripting (XSS) vulnerabilities.
9. When possible, map fixed input values to specific URLs or filenames, and reject all other inputs that do not match the expected values.
10. Perform a comprehensive review of all areas where untrusted inputs can enter the application, such as parameters, cookies, network data, request headers, and external systems. Apply input validation and filtering mechanisms to prevent unauthorized redirects.
11. Educate developers about the risks associated with open redirects and the importance of implementing secure redirect mechanisms.

It is recommended that the MetaMask development team addresses this vulnerability promptly by implementing the proposed mitigation measures to prevent external redirect attacks and protect users from potential social engineering and phishing attempts.

Request

```
Host: metamask.io
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: https://metamask.io/page-data/news/latest/how-to-bridge-your-assets-
using-connexts-bridge-protocol/page-data.json
```

Response

```
HTTP/1.1 301 Moved Permanently
Connection: keep-alive
Content-Length: 162
Server: GitHub.com
Content-Type: text/html
Access-Control-Allow-Origin: *
expires: Thu, 25 May 2023 17:39:37 GMT
Cache-Control: max-age=600
x-proxy-cache: MISS
X-GitHub-Request-Id: D856:9AD7:65D02C4:9A65320:646F9B01
Accept-Ranges: bytes
Date: Thu, 25 May 2023 17:29:37 GMT
Via: 1.1 varnish
Age: 0
X-Served-By: cache-mrs10548-MRS
X-Cache: MISS
X-Cache-Hits: 0
X-Timer: S1685035778.635687,VS0,VE118
Vary: Accept-Encoding
X-Fastly-Request-ID: 0c6b1619a6ab37a928824838cf1430dd0cb35983
Location: https://metamask.io/buy-
crypto/?amp%3Butm_campaign=3521881238686907442.owasp.org&amp%3Butm_medium=blo
g
```

Bug Bounty Report 06

https://www.saytechnologies.com

Vulnerability title:

Time-Based SQL Injection in PostgreSQL

Vulnerability description:

The website at www.saytechnologies.com/contact/api-docs is vulnerable to time-based SQL injection attacks. By manipulating the "Company-2" parameter in the request, an attacker can inject malicious SQL code and control the query execution time. The vulnerability allows unauthorized access to the backend database and potential disclosure or manipulation of sensitive data.

Affected components:

The vulnerability resides in the contact form API of the www.saytechnologies.com website.

Impact assessment:

The impact of this vulnerability is severe. An attacker can exploit the SQL injection to execute arbitrary SQL commands in the backend database. This can lead to unauthorized access, data manipulation, data leakage, or even a complete compromise of the database. The potential consequences include unauthorized access to sensitive customer information, disclosure of proprietary data, unauthorized modifications, and potential damage to the reputation of Say Technologies.

Steps to reproduce:

1. Send a GET request to the following URL: https://www.saytechnologies.com/contact/api-docs
2. Modify the value of the "Company-2" parameter to inject time-based SQL code, such as ["case when cast(pg_sleep(15) as varchar) > '' then 0 else 1 end --].
3. Observe the delay in the response time, indicating successful injection and time-based execution of the injected SQL code.

Proof of concept:

In the evidence provided, the query time is controlled by the injected parameter value. The original unmodified query with the value "ZAP" took 3,307 milliseconds, while the modified parameter value ["case when cast(pg_sleep(15) as varchar) > '' then 0 else 1 end -- "] caused the request to take 20,318 milliseconds. This indicates a potential SQL injection vulnerability and the ability to control the query execution time.

Proposed mitigation or fix:

To mitigate the Time-Based SQL Injection vulnerability, the following steps should be taken:

1. Implement strict input validation and sanitization for all user-supplied data. Do not trust client-side input, even if client-side validation is implemented.
2. Employ server-side type checking and enforce strong data typing to ensure that only valid and expected data is processed by the database.
3. If using JDBC, utilize parameterized queries with PreparedStatements or CallableStatements, passing parameters using the '?' placeholder.
4. If using ASP, employ ADO Command Objects with strong type checking and parameterized queries.
5. Whenever possible, leverage stored procedures in the database to handle database interactions, ensuring that dynamic SQL generation is avoided.
6. Never concatenate strings directly into SQL queries within stored procedures or use potentially dangerous functions like 'exec' or 'exec immediate'.
7. Implement proper data escaping and input validation to neutralize any malicious input.
8. Utilize an "allow list" of allowed characters or a "deny list" of disallowed characters in user input to prevent injection attacks.
9. Apply the principle of least privilege by granting the minimum necessary database access privileges to the application.
10. Avoid using highly privileged database users such as 'sa' or 'db-owner' to minimize the impact of potential SQL injection attacks.
11. Implement a comprehensive database access control strategy, granting access only to the specific resources and operations required by the application.
12. Regularly update and patch the database software to ensure the latest security fixes and enhancements are in place.

It is strongly advised that Say Technologies addresses this vulnerability promptly by implementing the proposed mitigation measures to prevent SQL injection attacks and protect the integrity and confidentiality of their data.

Request

```
GET https://www.saytechnologies.com/contact/api-docs?Company-
2=%22case+when+cast%28pg_sleep%2815%29+as+varchar%29+%3E+%27%27+then+0+else+1
+end+--+&Contact+URL=https%3A%2F%2Fwww.saytechnologies.com%2Fcontact%2Fapi-
docs&First-name-2=ZAP&Last-name-2=ZAP&Message-2=&Next-Event-Date-
2=ZAP&Page+Type=This+lead+wants+access+to+our+broker-dealer+API+docs.&Phone-
number-2=9999999999&Work-email-2=foo-bar%40example.com HTTP/1.1
Host: www.saytechnologies.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```

Bug Bounty Report 07

https://eero.com

Vulnerability title:

SQL Injection in SQLite

Vulnerability description:

The website at eero.com is vulnerable to SQL injection attacks targeting the SQLite database. By manipulating the "utmcontent" parameter in the request, an attacker can inject arbitrary SQL code into the query, potentially leading to unauthorized access, data manipulation, or disclosure of sensitive information.

Affected components:

The vulnerability is present in the website's handling of the utmcontent parameter.

Impact assessment:

The impact of this vulnerability is significant. Successful exploitation of SQL injection in SQLite can allow an attacker to execute arbitrary SQL queries, leading to unauthorized access to the underlying database. This can result in unauthorized retrieval, modification, or deletion of sensitive data. Additionally, an attacker may escalate the attack to compromise other systems or launch further attacks on the website or its users.

Steps to reproduce:

1. Send a GET request to the following URL: https://eero.com/
2. Modify the value of the "utmcontent" parameter to include SQL injection code, such as "case randomblob(1000000) when not null then 1 else 1 end".
3. Observe the response time or any error messages that indicate successful SQL injection.

Proof of concept:

In the evidence provided, the query time is controlled by the injected parameter value. The original unmodified query with the value "What_to_know_when_youre_building_your_smart_home" took 2,552 milliseconds. However, when using the modified parameter values "case randomblob(1000000) when not null then 1 else 1 end", the request took 2,468 milliseconds. Another modified value "case randomblob(10000000) when not null then 1 else 1 end" caused the request to take 6,433 milliseconds. These variations in query execution time indicate a potential SQL injection vulnerability and the ability to control the query.

Proposed mitigation or fix:

To mitigate the SQL injection vulnerability in SQLite, the following measures should be implemented:

1. Employ strict input validation and sanitization on the server side for all user-supplied data. Do not rely solely on client-side validation.
2. Implement strong type checking and parameterized queries when interacting with the database.
3. If using SQLite with JDBC, utilize PreparedStatements or CallableStatements with parameter placeholders ('?') to prevent SQL injection.
4. If using ASP, employ ADO Command Objects with strong type checking and parameterized queries.
5. Avoid creating dynamic SQL queries using string concatenation, as it can introduce vulnerabilities. Instead, use parameterized queries or stored procedures.
6. Apply proper data escaping or encoding techniques to neutralize any potentially malicious input.
7. Implement an "allow list" of allowed characters or a "deny list" of disallowed characters for user input to prevent injection attacks.
8. Follow the principle of least privilege by granting the minimum necessary database access rights to the application.
9. Avoid using highly privileged database users, such as 'sa' or 'db-owner', as this minimizes the potential impact of SQL injection attacks.
10. Regularly update and patch the SQLite database to ensure the latest security fixes and improvements are applied.

It is recommended that the website administrators at eero.com address this vulnerability promptly by implementing the proposed mitigation measures. By doing so, they can protect the integrity and confidentiality of their data and prevent potential unauthorized access or manipulation.

Request

```
GET
https://eero.com/?utmcontent=case+randomblob%281000000%29+when+not+null+then+
1+else+1+end+&utmsource=blog HTTP/1.1
Host: eero.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
Referer: https://eero.com/sitemap.xml
```

Response

```
HTTP/1.1 200 OK
Date: Wed, 24 May 2023 16:38:41 GMT
Content-Type: text/html; charset=utf-8
Connection: keep-alive
X-Powered-By: Next.js
Cache-Control: private, no-cache, no-store, max-age=0, must-revalidate
Vary: Accept-Encoding
Content-Length: 140089
```

Bug Bounty Report 08

http://ring.com

Vulnerability title:

Content Security Policy (CSP) Header Not Set

Vulnerability description:

The web application at http://ring.com/ does not have the Content-Security-Policy (CSP) header set in the server response. CSP is an important security measure that helps prevent various types of attacks, such as Cross Site Scripting (XSS) and data injection attacks. Without a proper CSP header, the website is more vulnerable to these types of attacks.

Affected components:

The affected component is the web server or application server hosting http://ring.com/.

Impact assessment:

The lack of a Content-Security-Policy header increases the risk of successful attacks on the website. Attackers can potentially exploit vulnerabilities like XSS to inject malicious scripts, steal user data, deface the website, or distribute malware. By setting a proper CSP header, the impact of such attacks can be significantly mitigated.

Steps to reproduce:

1. Visit http://ring.com/ using a modern web browser.
2. Inspect the server response headers.
3. Look for the absence of the Content-Security-Policy header.

Proof of concept:

No proof of concept is provided as this vulnerability is a configuration issue rather than an actively exploitable flaw.

Proposed mitigation or fix:

To address this vulnerability, the website owner or administrator should configure the web server, application server, or load balancer to include the appropriate Content-Security-Policy header in the server response. The recommended header names to achieve optimal browser support are:

- "Content-Security-Policy" for Chrome 25+, Firefox 23+, and Safari 7+
- "X-Content-Security-Policy" for Firefox 4.0+ and Internet Explorer 10+
- "X-WebKit-CSP" for Chrome 14+ and Safari 6+

By setting the Content-Security-Policy header, approved sources of content, such as JavaScript, CSS, HTML frames, fonts, images, and embeddable objects, can be declared, thereby restricting unauthorized content, and reducing the risk of various web-based attacks.

It is recommended to follow best practices and use a strict CSP policy tailored to the specific requirements of the website. Regular testing and monitoring should be performed to ensure the effectiveness of the implemented Content Security Policy.

Bug Bounty Report 09

https://www.saytechnologies.com

Vulnerability title:

SQL Injection - MsSQL

Vulnerability description:

The web application at https://www.saytechnologies.com/contact/broker is vulnerable to SQL injection. The attack payload "ZAP" WAITFOR DELAY '0:0:15' --" was able to manipulate the query time, indicating a potential vulnerability. SQL injection allows an attacker to execute malicious SQL statements, potentially leading to unauthorized access, data leakage, or even database compromise.

Affected components:

The affected component is the web application's backend code that interacts with the MsSQL database.

Impact assessment:

The impact of SQL injection can be severe. An attacker can manipulate the SQL query to extract sensitive data, modify or delete data, or perform other unauthorized actions. In this specific case, the delay introduced by the payload suggests a potential time-based blind SQL injection. However, the impact assessment depends on the specific actions permitted by the vulnerable SQL statement and the privileges associated with the database user.

Steps to reproduce:

1. Visit https://www.saytechnologies.com/contact/broker.
2. Intercept the request to the web server.
3. Modify the value of the "Company-2" parameter to include the SQL injection payload: "ZAP" WAITFOR DELAY '0:0:15' --".
4. Observe any delay or abnormal behavior in the response time, indicating a successful SQL injection.

Proof of concept:

No proof of concept is provided in the vulnerability description.

Proposed mitigation or fix:

To address this SQL injection vulnerability, the following measures can be implemented:

1. Input validation and sanitization: Implement server-side input validation and filtering to ensure that user-supplied data is properly sanitized before being used in SQL queries. Use whitelisting or parameterized queries to prevent unauthorized input from being executed as part of the SQL statement.

2. Prepared statements or stored procedures: Utilize prepared statements or stored procedures with parameterized queries. This helps prevent SQL injection by separating SQL code from user-supplied data, ensuring that data is treated as data and not executable code.

3. Least privilege principle: Apply the principle of least privilege by granting the minimum necessary database access rights to the application. Avoid using high-privileged database users like 'sa' or 'db-owner' to minimize the impact of potential SQL injection attacks.

4. Regular security updates: Keep the web application, web server, and database server up to date with the latest security patches and updates to mitigate known vulnerabilities.

5. Security testing: Perform regular security testing, including thorough vulnerability scanning and penetration testing, to identify and address any potential SQL injection vulnerabilities.

By implementing these measures, the web application can significantly reduce the risk of SQL injection attacks and enhance the overall security posture.

Bug Bounty Report 10

https://miro.com

Vulnerability title:

CSP: Wildcard Directive

Vulnerability description:

The web application at https://miro.com/ has a Content Security Policy (CSP) configuration issue related to wildcard directives. The following directives either allow wildcard sources (or ancestors), are not defined, or are overly broadly defined: script-src, style-src, img-src, connect-src, frame-src, frame-ancestors, font-src, media-src, object-src, manifest-src, worker-src, prefetch-src, form-action. This misconfiguration can potentially weaken the effectiveness of CSP and increase the risk of various types of attacks, including Cross Site Scripting (XSS) and data injection attacks.

Affected components:

The affected component is the web server or application server hosting https://miro.com/.

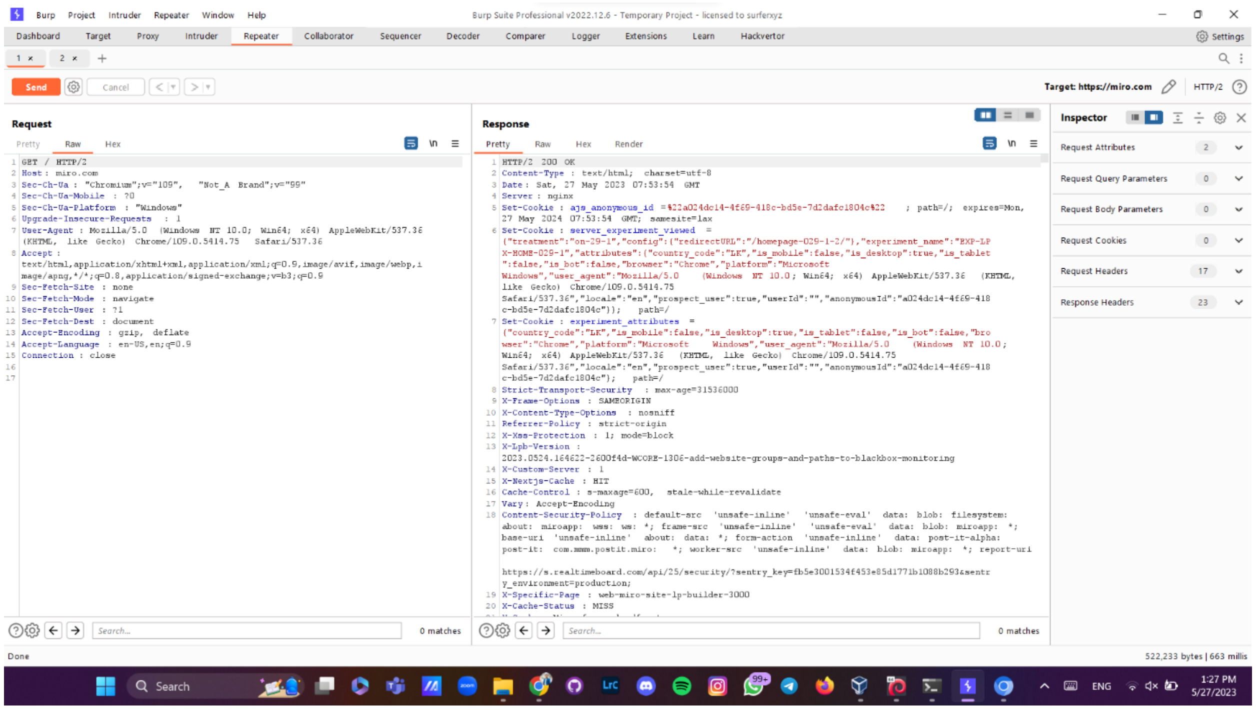
Impact assessment:

The impact of this misconfiguration depends on the specific CSP directives and the overall security posture of the web application. By allowing wildcard sources or not defining the directives properly, the web application is potentially allowing content from any source to be loaded, including potentially malicious or untrusted sources. This increases the risk of XSS attacks, data theft, site defacement, and distribution of malware.

Steps to reproduce:

1. Visit https://miro.com/ using a modern web browser.
2. Inspect the server response headers.
3. Look for the Content-Security-Policy header.
4. Check if the specified directives allow wildcard sources or are overly broadly defined.

Proof of concept:



```
GET / HTTP/2
Host : miro.com
Sec-Ch-Ua : "Chromium";v="109",   "Not_A  Brand";v="99"
Sec-Ch-Ua-Mobile  : ?0
Sec-Ch-Ua-Platform  : "Windows"
Upgrade-Insecure-Requests   : 1
User-Agent : Mozilla/5.0  (Windows  NT 10.0;  Win64;  x64)  AppleWebKit/537.36
(KHTML,  like  Gecko)  Chrome/109.0.5414.75   Safari/537.36
Accept :
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,i
mage/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site  : none
Sec-Fetch-Mode  : navigate
Sec-Fetch-User  : ?1
Sec-Fetch-Dest  : document
Accept-Encoding  : gzip,  deflate
Accept-Language  : en-US,en;q=0.9
Connection : close
```

```
HTTP/2 200 OK
Content-Type : text/html;  charset=utf-8
Date : Sat, 27 May 2023 07:53:54  GMT
Server : nginx
Set-Cookie : ajs_anonymous_id =%22a024dc14-4f69-418c-bd5e-7d2dafc1804c%22    ; path=/;  expires=Mon,
27 May 2024 07:53:54  GMT;  samesite=lax
Set-Cookie : server_experiment_viewed   =
{"treatment":"on-29-1","config":{"redirectURL":"/homepage-029-1-2/"},"experiment_name":"EXP-LP
X-HOME-029-1","attributes":{"country_code":"LK","is_mobile":false,"is_desktop":true,"is_tablet
":false,"is_bot":false,"browser":"Chrome","platform":"Microsoft
Windows","user_agent":"Mozilla/5.0    (Windows  NT 10.0; Win64;  x64)  AppleWebKit/537.36   (KHTML,
like  Gecko)  Chrome/109.0.5414.75
Safari/537.36","locale":"en","prospect_user":true,"userId":"","anonymousId":"a024dc14-4f69-418
c-bd5e-7d2dafc1804c"}};    path=/
Set-Cookie : experiment_attributes   =
{"country_code":"LK","is_mobile":false,"is_desktop":true,"is_tablet":false,"is_bot":false,"bro
wser":"Chrome","platform":"Microsoft    Windows","user_agent":"Mozilla/5.0    (Windows  NT 10.0;
Win64;  x64)  AppleWebKit/537.36   (KHTML,  like  Gecko)  Chrome/109.0.5414.75
Safari/537.36","locale":"en","prospect_user":true,"userId":"","anonymousId":"a024dc14-4f69-418
c-bd5e-7d2dafc1804c"};    path=/
Strict-Transport-Security   : max-age=31536000
X-Frame-Options  : SAMEORIGIN
X-Content-Type-Options   : nosniff
Referrer-Policy  : strict-origin
X-Xss-Protection  : 1; mode=block
X-Lpb-Version  :
2023.0524.164622-2600f4d-WCORE-1306-add-website-groups-and-paths-to-blackbox-monitoring
X-Custom-Server  : 1
X-Nextjs-Cache  : HIT
Cache-Control  : s-maxage=600,   stale-while-revalidate
Vary : Accept-Encoding
Content-Security-Policy   : default-src  'unsafe-inline'  'unsafe-eval'  data:  blob:  filesystem:
about:  miroapp:  wss:  ws:  *;  frame-src  'unsafe-inline'  'unsafe-eval'  data:  blob:  miroapp:  *;
base-uri  'unsafe-inline'  about:  data:  *;  form-action  'unsafe-inline'  data:  post-it-alpha:
post-it:  com.mmm.postit.miro:   *;  worker-src  'unsafe-inline'  data:  blob:  miroapp:  *;  report-uri

https://s.realtimeboard.com/api/25/security/?sentry_key=fb5e3001534f453e85d1771b1088b293&sentr
y_environment=production;
X-Specific-Page  : web-miro-site-lp-builder-3000
X-Cache-Status  : MISS
```

1. Note that the "default-src" directive allows a wildcard (*) as a source, which means any source is allowed for various types of content, including JavaScript, CSS, and images.

2. Also, the "frame-src" directive allows a wildcard (*) as a source, which means any website can be embedded in an iframe on the page.

3. Additionally, the "form-action" directive allows a wildcard (*) as a source, which means any URL can be used as the target for form submissions.

Proposed mitigation or fix:

To address this CSP misconfiguration, the following measures can be implemented:

1. Review and define appropriate directives: Analyze the web application's requirements and define specific and granular Content Security Policy directives for script-src, style-src, img-src, connect-src, frame-src, frame-ancestors, font-src, media-src, object-src, manifest-src, worker-src, prefetch-src, and form-action. Avoid using wildcard sources unless necessary and ensure that the defined sources are trusted and necessary for the web application's functionality.

2. Remove unsafe-inline and unsafe-eval: Avoid using 'unsafe-inline' and 'unsafe-eval' in the Content Security Policy directives as they can introduce security vulnerabilities. Instead, utilize strict Content Security Policy rules that require external files and prevent inline scripts and eval () functions.

3. Regular testing and monitoring: Regularly test and monitor the web application's Content Security Policy implementation to ensure that it remains effective and aligned with the application's security requirements. Conduct periodic security assessments and vulnerability scans to identify and address any misconfigurations or weaknesses in the CSP.

4. Follow best practices: Stay updated with the latest best practices for Content Security Policy implementation. Keep track of any new directives or changes in CSP standards and incorporate them into the web application's security configuration.

By implementing these mitigation measures, the web application can enhance its security by enforcing a stricter Content Security Policy that restricts unauthorized sources and reduces the risk of XSS attacks and other content-related vulnerabilities.

Request

```
GET https://miro.com/ HTTP/1.1
Host: miro.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0)
Gecko/20100101 Firefox/105.0
Pragma: no-cache
Cache-Control: no-cache
```