// HALBORN

Ithaca - Backend

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **October 13th, 2023 - October 27th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	10/27/2023
0.2	Draft Review	10/29/2023
0.3	Draft Review	10/30/2023
1.0	Remediation Plan	02/22/2024
1.1	Remediation Plan Review	02/22/2024
1.2	Remediation Plan Review	02/26/2024

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Carlos Polop	Halborn	Carlos.Polop@halborn.com
Afaq Abid	Halborn	Afaq.Abid@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Ithaca engaged Halborn to conduct a security assessment on their Web applications and its respective API and services, beginning on October 13th, 2023 and ending on October 27th, 2023. The security assessment was scoped to the web applications and underlying APIs and backend services. Halborn was provided access to the application, and its respective source code to conduct security testing using tools to scan, detect, validate possible vulnerabilities found in the application and report the findings at the end of the engagement.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided a timeline for the engagement and assigned a full-time security engineer to review the security of the assets in scope. The security engineer is a penetration testing expert with advanced knowledge in web, mobile, recon, discovery & infrastructure penetration testing.

During the penetration testing engagement, numerous vulnerabilities were identified across the option trading platform, representing a varied range of risks and impacts. High-risk findings include an insecure CORS policy that potentially exposes authentication tokens, and a Cross-Site Request Forgery (CSRF) vulnerability leading to the cancellation of user orders.

Additionally, the platform is susceptible to another potential vulnerability Blockchain network configuration injection via CSRF and Clickjacking, and another separate clickjacking vulnerability that can lead to order cancellation. Other notable findings include a lack of resources and rate limiting, vulnerable JavaScript dependencies, and missing security headers. Less critical, yet still noteworthy, are the issues of reflected unsanitized input, verbose error responses on exceptions, cacheable HTTPS responses, and NGINX version disclosure. These findings have lower impact and likelihood ratings, but still necessitate attention

to bolster the overall security posture of the platform.

The Ithaca team has successfully addressed all identified issues, implementing appropriate checks and policies to enhance security.

Security Review and Remediation Summary for Ithaca Team
During the period between October 19th and October 27th, the Ithaca team underwent a comprehensive review of their security framework. This review was a critical step in evaluating the effectiveness of their existing security measures and identifying potential enhancements to their infrastructure.

Remediation Confirmation: February 23rd, 2024
Following the review, the team embarked on a significant overhaul of their security and infrastructure systems. The remediation efforts included a series of substantial security and infrastructure upgrades based on the findings from the initial review. The Ithaca team was meticulous in their approach, ensuring that all code updates since the review period were incorporated into the remediation process.
The team's commitment to creating a robust security framework was evident in their decision to extend the remediation timeline beyond the standard period. This allowed for thorough testing and verification of the new measures implemented. The remediation was confirmed on February 23rd, signalling the completion of this phase and the readiness of the upgraded security framework to withstand potential threats.

The Ithaca team's efforts have not only strengthened their security posture but have also set a precedent for continuous improvement and proactive risk management within their organization.

1.3 SCOPE

IN-SCOPE:
The following URLs/APIs and its respective repositories were in scope:

- https://www.dev.ithaca.finance/
- https://api.dev.ithaca.finance/
- https://webconsole.dev.ithaca.finance/
- https://console.dev.ithaca.finance/
- alpha-0.26
- v0.3.002-alpha

OUT-OF-SCOPE:
- External Libraries

1.4 TEST APPROACH & METHODOLOGY

Halborn followed Whitebox and Blackbox methodology as per the scope and performed a combination of manual and automated security testing with both to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the pentest. While manual testing is recommended to uncover flaws in logic, process and implementation; automated testing techniques assist enhance coverage of the infrastructure and can quickly identify flaws in it.

Throughout the assessment, we followed the following phases, but not limited to:

- Mapping Content and Functionality of APIs
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Rate Limitations Tests
- Brute Force Attempts
- Input Handling
- Response Manipulation
- Source Code Review
- Fuzzing of all input parameters
- Multiple Type of Injection (SQL/JSON/HTML/Command)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	5	3	1

LIKELIHOOD

IMPACT

			(HAL-01) (HAL-02)	
		(HAL-03)		
		(HAL-04) (HAL-05) (HAL-06) (HAL-07)		
	(HAL-08) (HAL-10)	(HAL-09)		
(HAL-11)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) INSECURE CORS POLICY LEAD TO EXPOSED AUTHENTICATION TOKENS	High	SOLVED - 12/20/2023
(HAL-02) CROSS-SITE REQUEST FORGERY LEAD TO CANCELLATION OF ORDERS	High	SOLVED - 02/23/2024
(HAL-03) BLOCKCHAIN NETWORK CONFIGURATION INJECTION VIA CSRF AND CLICKJACKING	Medium	SOLVED - 02/23/2024
(HAL-04) CLICKJACKING CAUSE ORDER CANCELLATION	Medium	SOLVED - 02/22/2024
(HAL-05) LACK OF RESOURCES AND RATE LIMITING	Medium	SOLVED - 02/21/2024
(HAL-06) VULNERABLE JAVASCRIPT DEPENDENCY	Medium	SOLVED - 02/21/2024
(HAL-07) MISSING SECURITY HEADERS	Medium	SOLVED - 02/23/2024
(HAL-08) REFLECTED UNSANITIZED INPUT	Low	SOLVED - 02/23/2024
(HAL-09) VERBOSE ERROR RESPONSE ON EXCEPTIONS	Low	SOLVED - 02/23/2024
(HAL-10) CACHEABLE HTTPS RESPONSE	Low	SOLVED - 02/15/2024
(HAL-11) NGINX VERSION DISCLOSURE	Informational	SOLVED - 12/18/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) INSECURE CORS POLICY LEAD TO EXPOSED AUTHENTICATION TOKENS - HIGH

Description:

A comprehensive security evaluation of the backend has revealed a serious vulnerability concerning the **Cross-Origin Resource Sharing (CORS)** settings. The current configuration inaccurately reflects the `Origin` header in the CORS response, coupled with the allowance of credentials through the `Access-Control-Allow-Credentials: true` setting.

This misconfiguration could have serious impact as it directly affects the way the platform handles cross-origin HTTP requests, and it jeopardizes the security of the user's authentication tokens specifically in the API endpoint `/api/v1/auth/getSessionInfo` An attacker could exploit this vulnerability by creating a malicious website that sends cross-origin HTTP requests to that API endpoint. Given that credentials are allowed, the cookie will be sent along to that request and the return HTTP response contains the auth token, which can also be stolen through an attacker controlled server. This token is a sensitive piece of information as it is used to authenticate users in `WebSocket` connections, facilitating real-time communication between the client and the server.

Given the high-paced nature of option trading, the potential impact of this vulnerability is severe. An attacker with access to a user's authentication token could gain unauthorized access to their trading activities, listen to their communications, and potentially manipulate trades. This scenario poses significant financial risks to users, compromises the integrity of our trading activities, and risks damaging the overall reputation of our platform.

Proof of Concept:

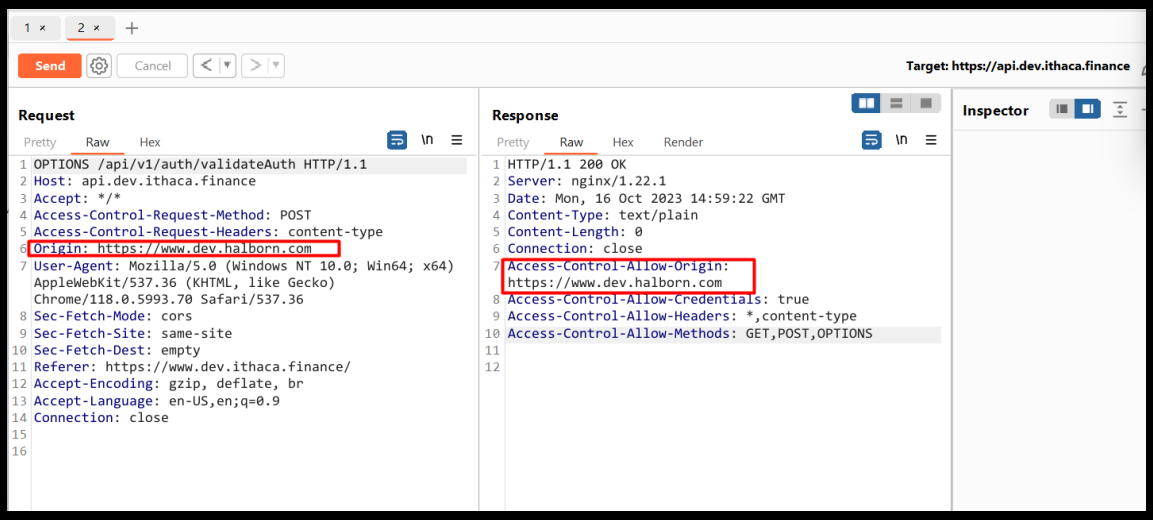


Figure 1: Insecure CORS Configuration

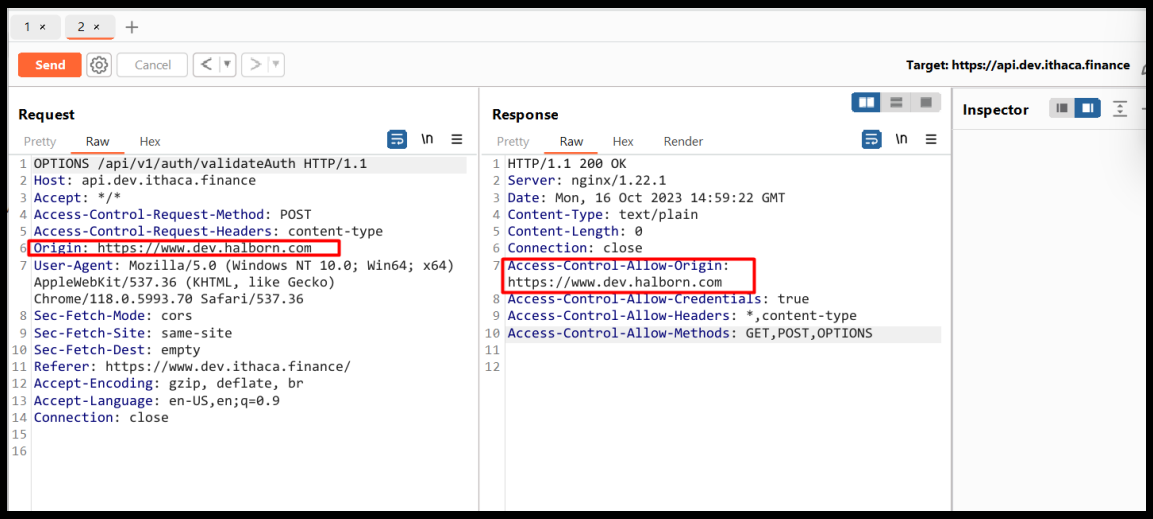


Figure 2: Attacker's hosted malicious site which going to steal the authentication token and send to another attacker's controlled server

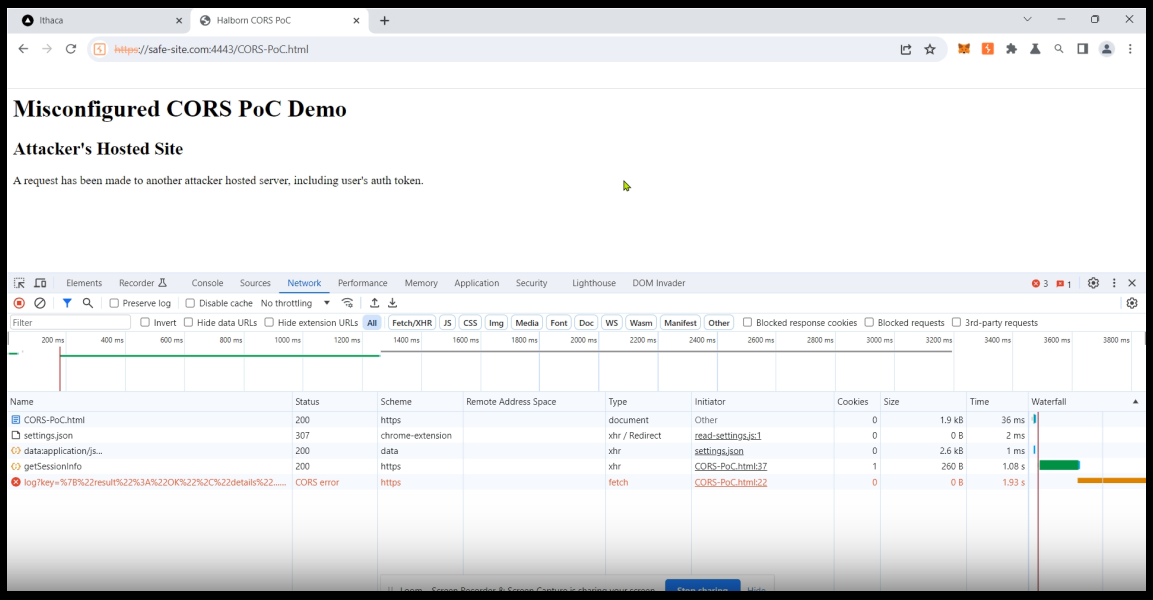


Figure 3: Attacker's hosted malicious site which going to steal the authentication token and send to another attacker's controlled server

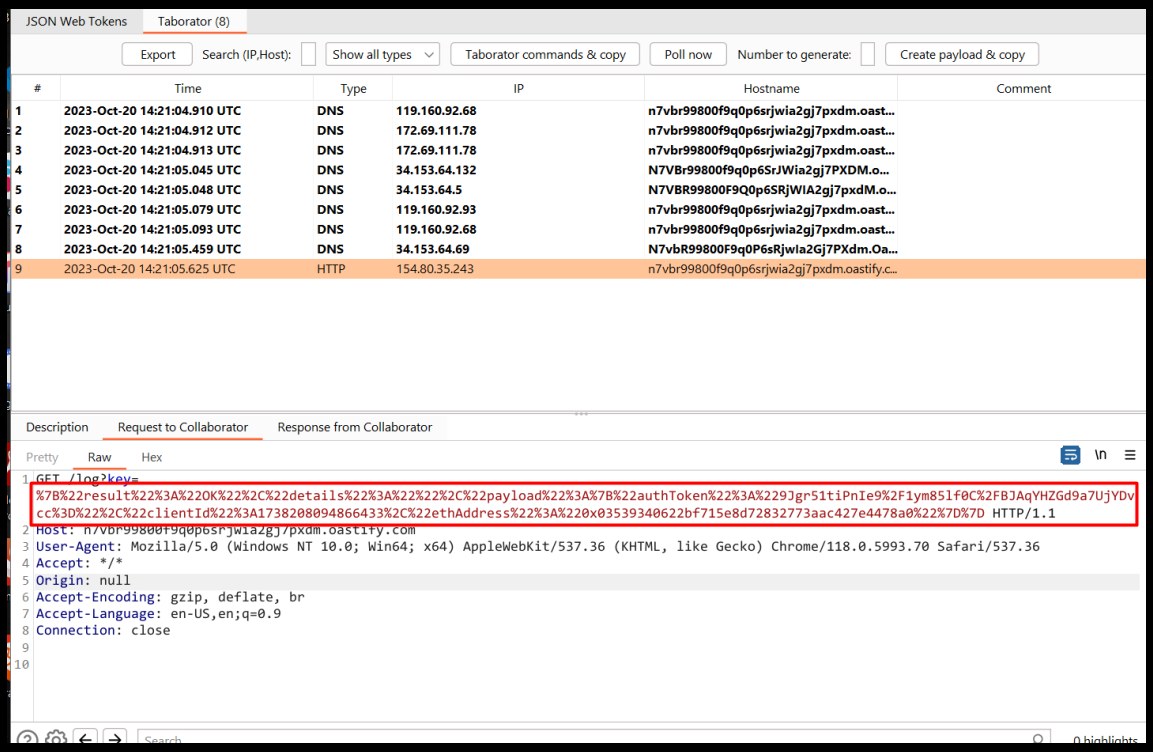


Figure 4: Stolen tokens on another attacker's hosted server containing user's authentication token

Figure 5: User's authentication token

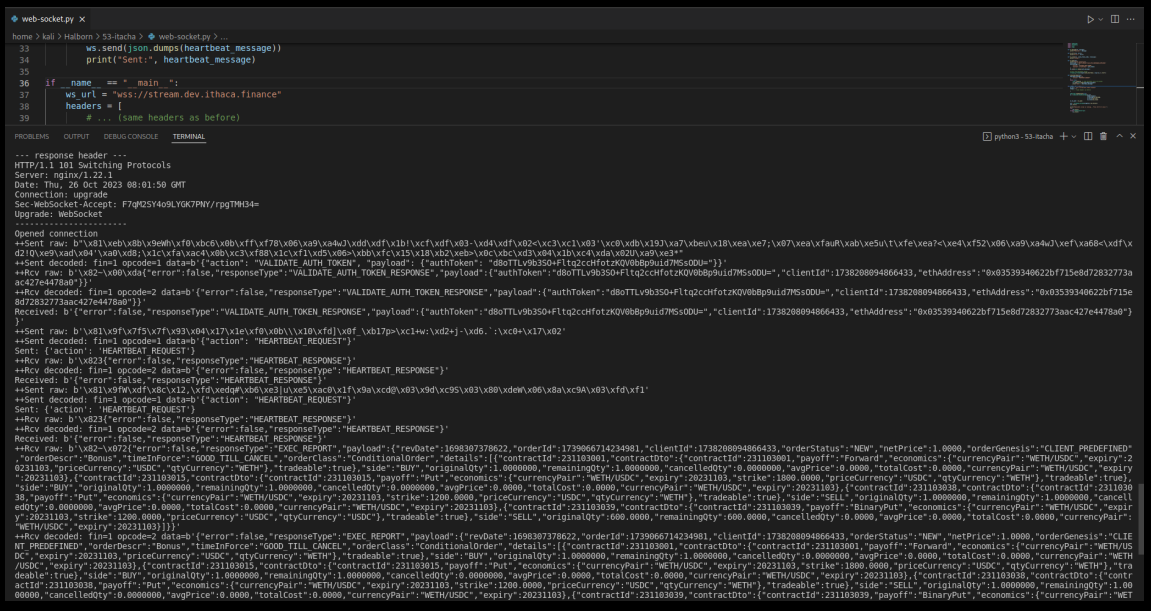


Figure 6: By using that stolen token authentication with server's websockets connection on the behalf of the user to steal further information

For more details, please review the attached proof of concept video below:

Loom Video: CORS misconfiguration lead to steal authentication tokens

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

It is recommended to reconfigure the CORS settings for the backend API to ensure that the Access-Control-Allow-Origin header does not reflect arbitrary origins, and set the Access-Control-Allow-Credentials header to false.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the appropriate CORS header policy.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

- CORS Security in Web Applications
- Secure Handling of Access Tokens

3.2 (HAL-02) CROSS-SITE REQUEST FORGERY LEAD TO CANCELLATION OF ORDERS - HIGH

Description:

During a thorough security assessment of the backend, another vulnerability was identified in relation to Cross-Site Request Forgery (CSRF) which could lead to the unauthorized cancellation of user orders.

The platform is currently susceptible to CSRF attacks, whereby an attacker could deceive a logged-in user into submitting a malicious request that cancels all of their placed orders on the platform. This vulnerability can be user's specific as well with chaining to CORS misconfiguration that allows an attacker to steal the WebSocket token, enabling them to listen to communications and potentially steal order IDs to cancel specific orders selectively as well.

The impact of this vulnerability is severe. Not only does it put our users' financial assets at risk by allowing unauthorized order cancellations, but it also exposes sensitive communication that could be exploited for further malicious activities, damaging the integrity of the trading platform and eroding trust in our services.

Proof of Concept:

. This proof of concept is specifically tied to the /api/v1/clientapi/allOrdersCancel endpoint.

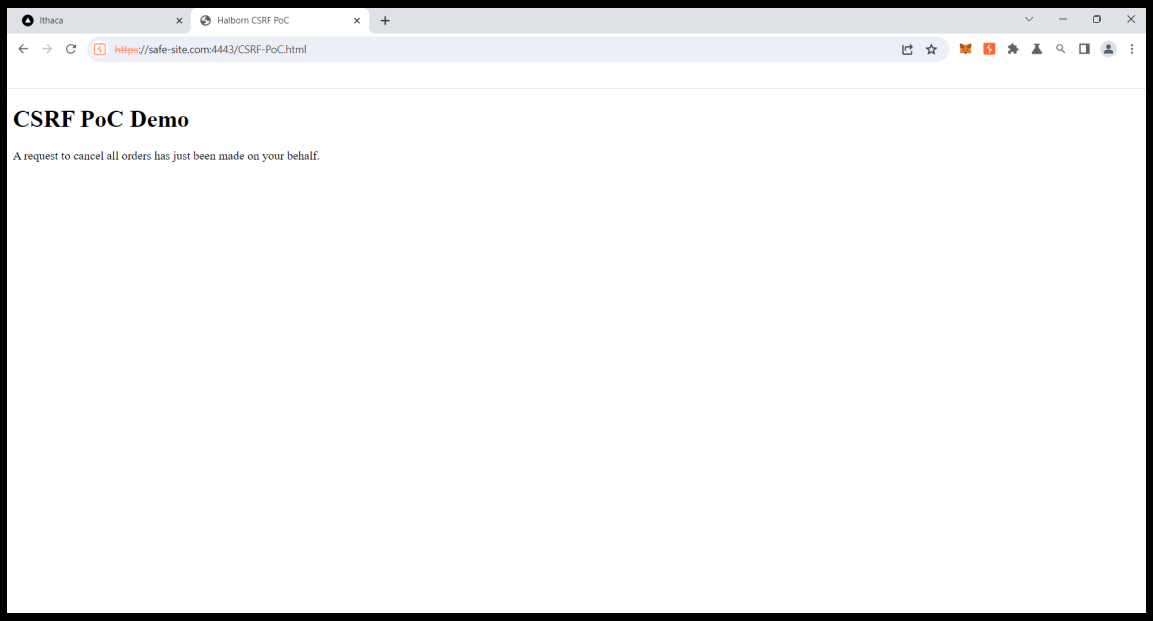
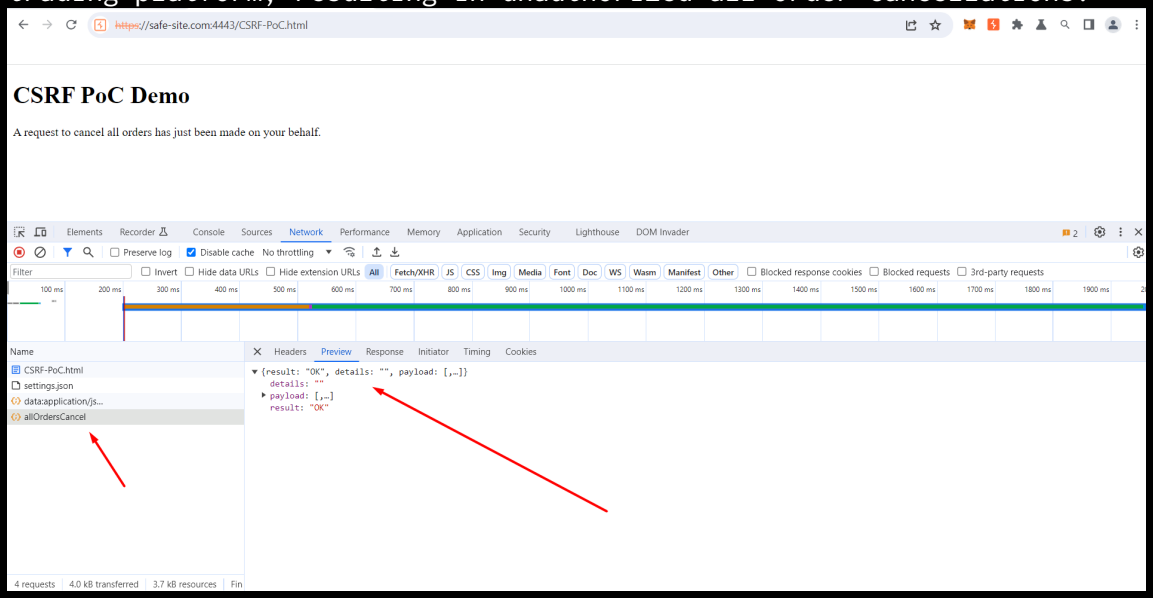


Figure 7: A malicious site hosted by an attacker

CSRF Attack:

- A user is deceived into visiting a malicious site while logged into the trading platform, resulting in unauthorized all order cancellations.



For more details, please review the attached proof of concept video below:

Loom Video: CSRF Lead to all orders cancellation of end-users

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:H

Risk Level:

Likelihood - 4
Impact - 5

Recommendation:

To effectively mitigate the risk associated with the identified CSRF vulnerability, it is imperative to incorporate anti-CSRF tokens within all state-changing requests, especially those that can initiate financial transactions or alter user settings. These tokens ensure that only legitimate requests originating from our own platform are processed.

Moreover, when it comes to cookies, which are a common vector for CSRF attacks, it is crucial to set the SameSite attribute to 'Strict' or 'Lax'. This setting helps to prevent the browser from sending cookies along with cross-site requests, thereby adding another layer of defense against CSRF attacks. In cases where cookies are essential for the functionality of the site, the 'Lax' setting can be used, as it allows cookies to be sent in top-level navigation while still providing adequate protection.

Furthermore, ensuring that cookies are set as 'Secure' will make sure that they are only transmitted over secure HTTPS connections, which is vital for maintaining the confidentiality and integrity of the information contained within them. Implementing these recommendations will bolster the platform's resilience against CSRF attacks and contribute to the overall security of our users' data and financial assets.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the appropriate checks.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

- OWASP - Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet

3.3 (HAL-03) BLOCKCHAIN NETWORK CONFIGURATION INJECTION VIA CSRF AND CLICKJACKING - MEDIUM

Description:

During the assessment, it was identified a vulnerability on the admin side of the platform, related to the injection of malicious configurations into the blockchain network. The vulnerability arises from a combination of two issues: the absence of CSRF (Cross-Site Request Forgery) protections, and the lack of the X-Frame-Options header.

Attackers can exploit this vulnerability by creating a malicious website that frames the admin panel. When an admin logs in, the attacker can manipulate the session to inject a malicious network configuration into the blockchain network through a CSRF attack under the framed site. This can lead to severe consequences, including but not limited to, the disruption of network operations, financial losses, and compromise of network integrity.

The impact of this vulnerability is heightened due to the administrative level of access that is compromised, allowing the attacker to make critical changes to the blockchain network's configuration.

Proof of Concept:

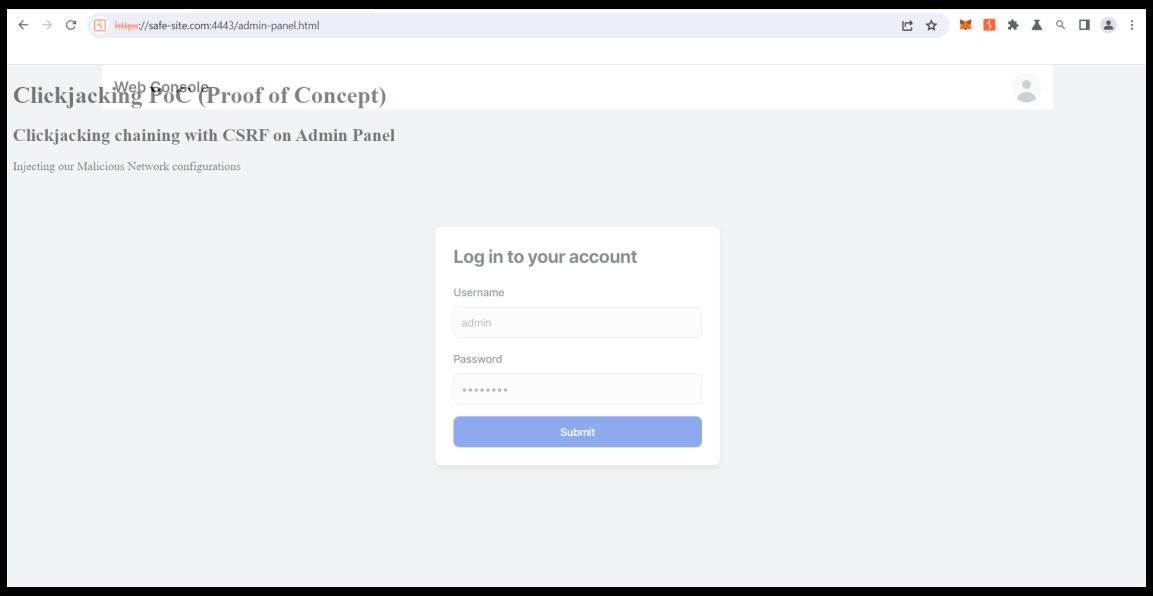


Figure 8: Framing the admin site due to lack of protection against clickjacking

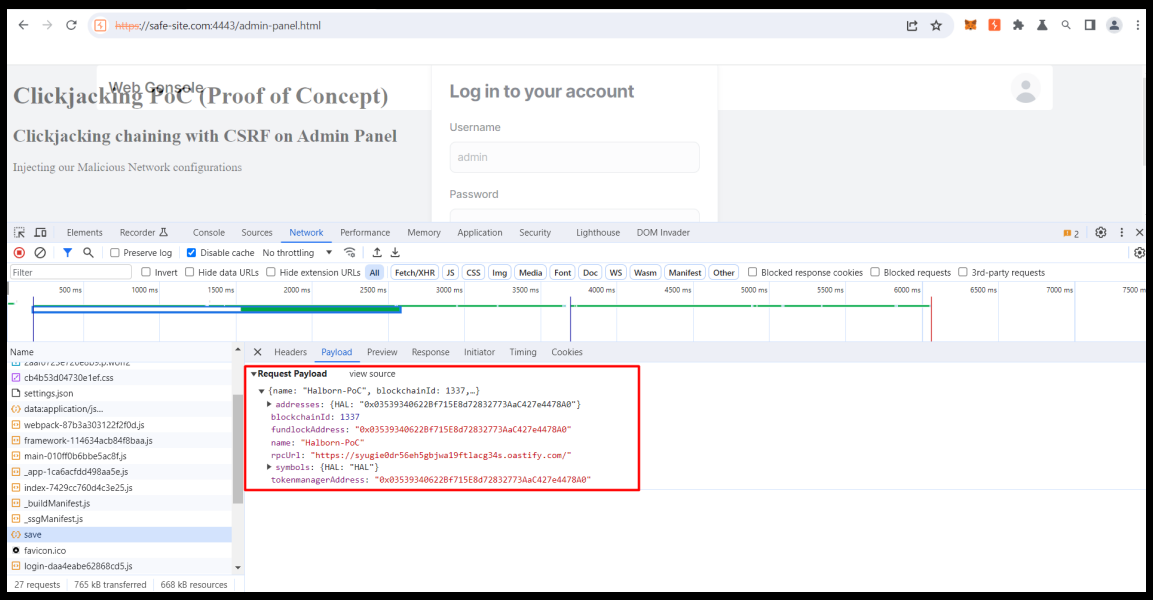


Figure 9: Injecting the network on the admin panel via malicious request

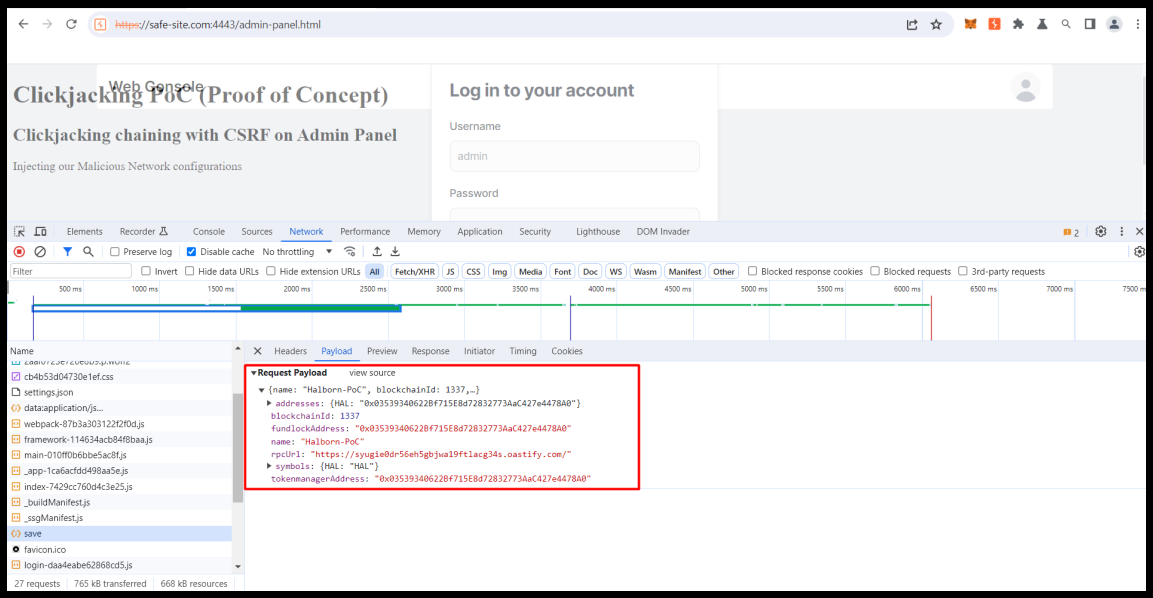


Figure 10: Network injection on the admin panel

Loom Video: Network Configuration Injection on Admin Side via CSRF + Clickjacking

Please review the attached proof of concept video above.

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:U/C:N/I:H/A:H

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

To remediate this vulnerability, several steps need to be taken:
- Introduce anti-CSRF tokens in all state-changing requests on the admin side to ensure that requests are legitimate and originated from trusted sources.
- Set the X-Frame-Options header to ‘DENY’ or ‘SAMEORIGIN’ to prevent

the admin panel from being framed within an iframe on malicious sites, effectively mitigating the risk of clickjacking attacks.
- Consider implementing additional layers of authentication security, such as multifactor authentication, to further protect admin accounts.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the appropriate checks.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

- OWASP - Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet
- OWASP - Clickjacking Defense Cheat Sheet

3.4 (HAL-04) CLICKJACKING CAUSE ORDER CANCELLATION - MEDIUM

Description:

Clickjacking, also known as a User Interface (UI) redress attack, involves an attacker manipulating a user into clicking on hidden elements on an overlaid malicious website, which can cause the user to perform actions on the underlying legitimate website without their consent.

A user could inadvertently execute trades, potentially resulting in substantial financial loss. For instance, a user might be tricked into selling a significant amount of a valuable cryptocurrency at a lower price or buying a less valuable cryptocurrency at a higher price, all benefiting the attacker.

An attacker could orchestrate such a scenario by creating a deceptive website or application with hidden iframes overlaying the sites and leverage it. Users could be lured to this malicious site through social engineering tactics such as phishing emails, fake advertisements, or even search engine poisoning.

Proof of concept:

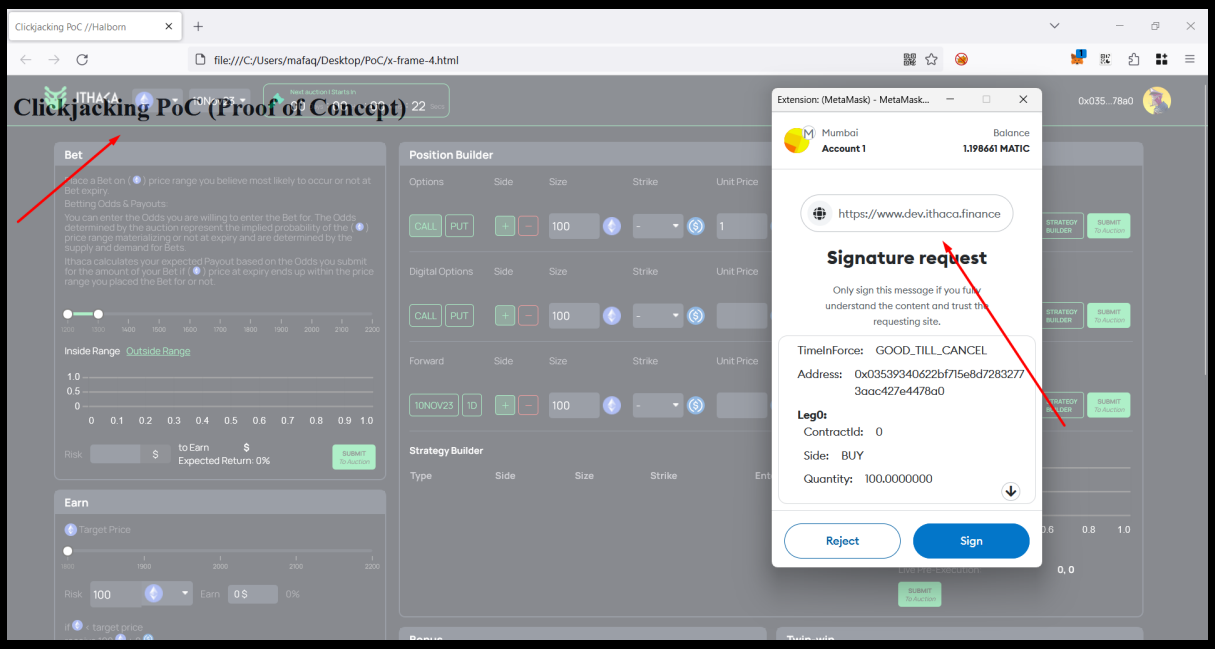


Figure 11: A frame is set over the Frontend www.dev.ithaca.finance with opacity low to create a PoC

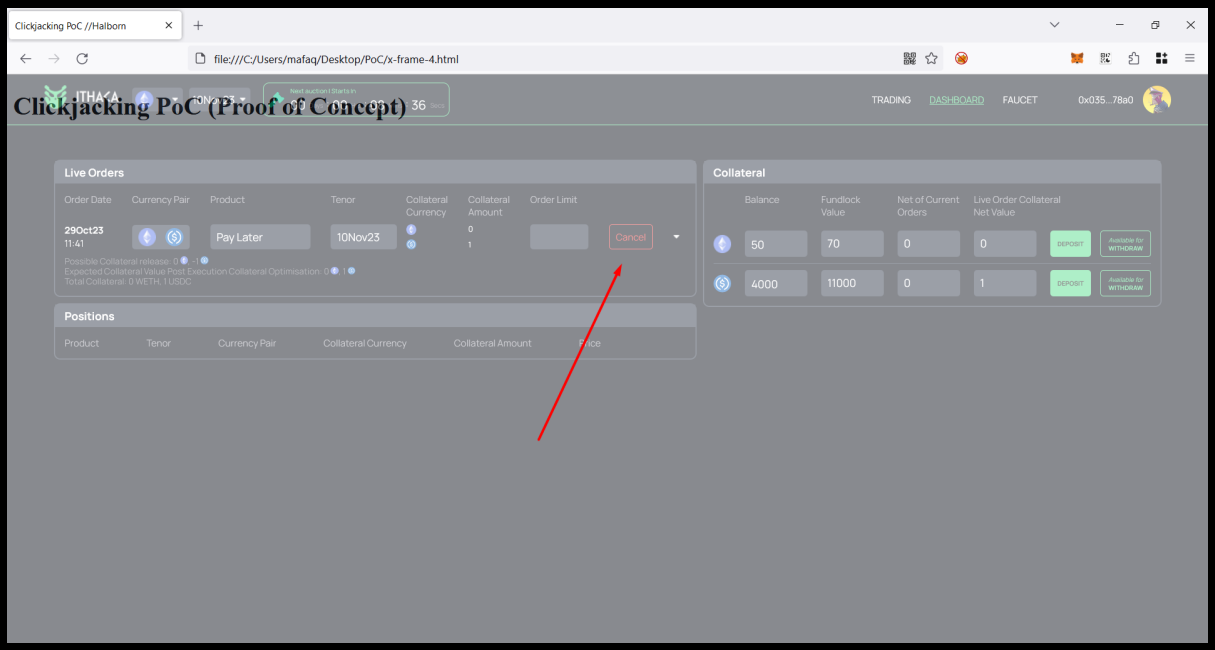
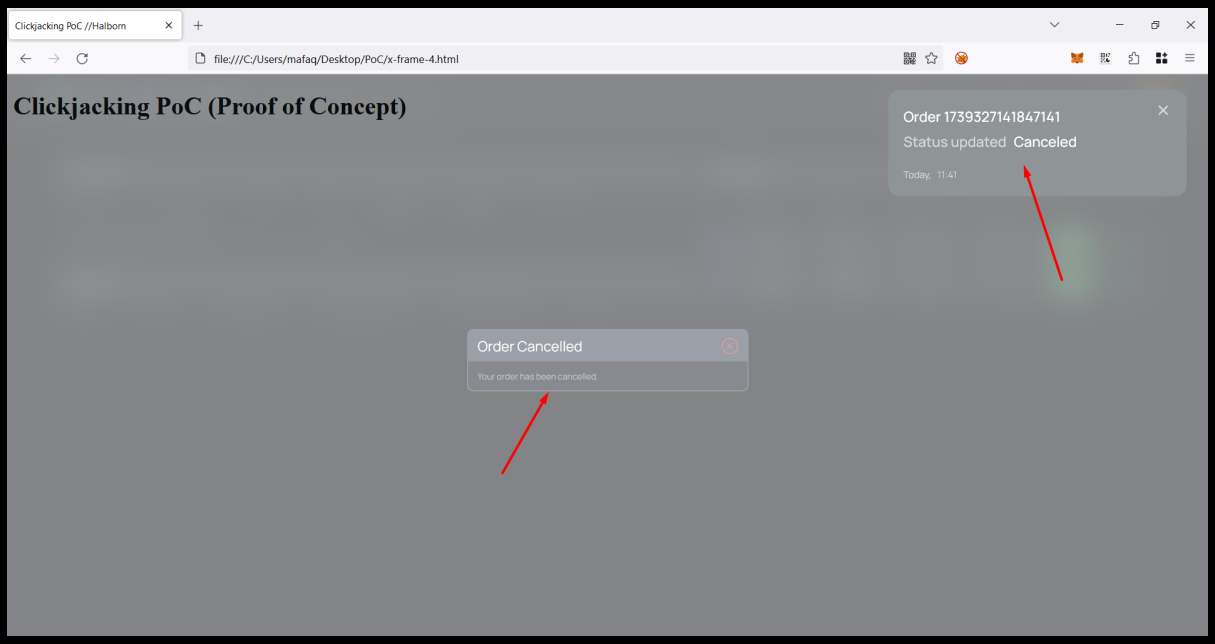


Figure 12: User's orders could be cancelled via leveraging the clickjacking attack



CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:L/I:L/A:L

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Implement X-Frame-Options HTTP header: Set this header to indicate whether a browser should be allowed to render your page in an <iframe>, <frame>, <embed> or <object>. A Content Security Policy (CSP) should also be defined with a frame-ancestors directive to indicate which websites are permitted to embed your page.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the security headers with appropriate policies.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

X-Frame-Options
Content Security Policy
Clickjacking

3.5 (HAL-05) LACK OF RESOURCES AND RATE LIMITING - MEDIUM

Description:

API requests consume resources such as network, CPU, memory, and storage. This vulnerability occurs when too many requests arrive simultaneously, and the API does not have enough compute resources to handle those requests.
During the assessment, no rate limitation was found on the API service. An attacker could exploit this vulnerability to overload the API by sending more requests than it can handle. As a result, the API becomes unavailable or unresponsive to new requests.

Proof of concept:

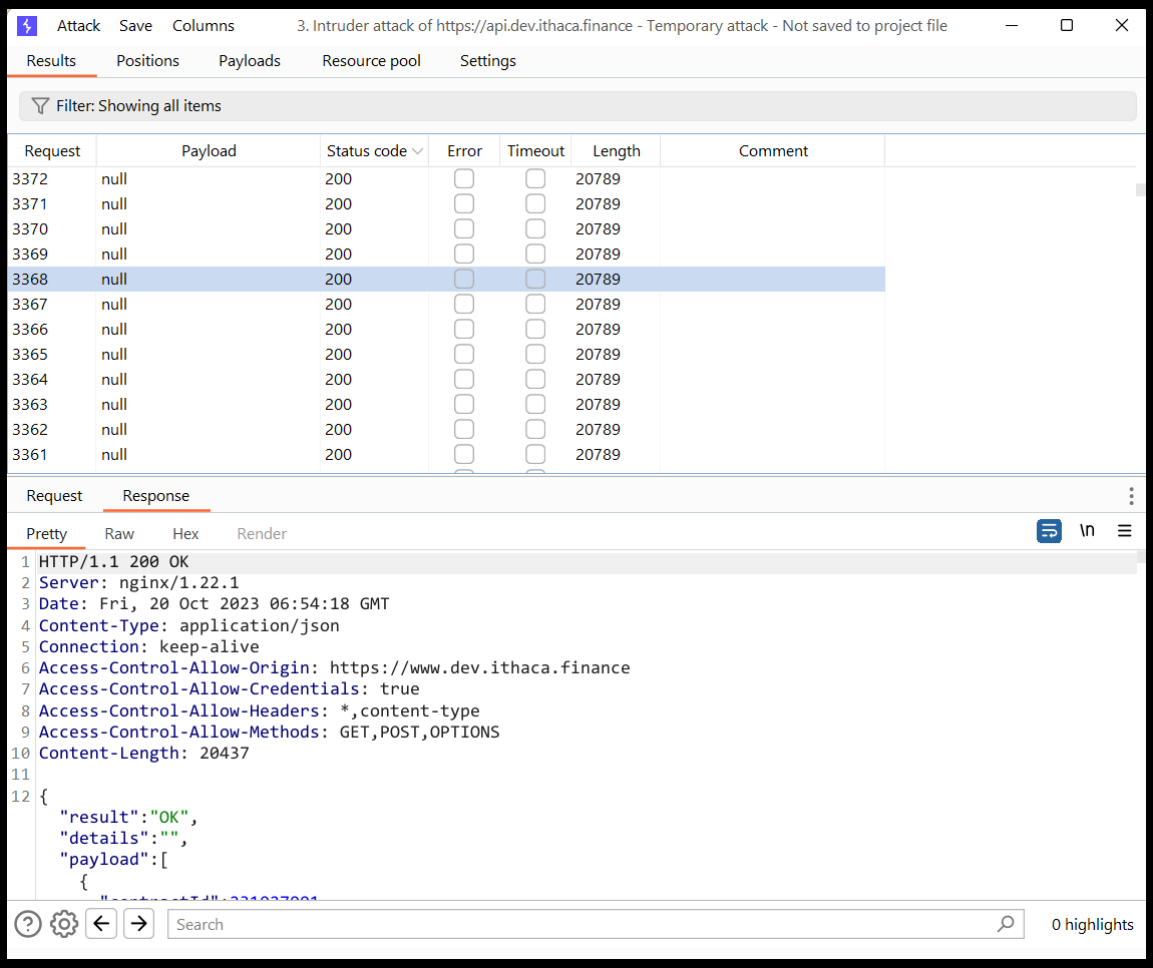


Figure 13: No rate limitation found on API

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

Risk Level:

Likelihood - 3

Impact - 3

Recommendation:

This vulnerability is due to the application accepting requests from users at a given time without performing request throttling checks. It is recommended to follow the following best practices:

- Implement a limit on how often a client can call the API within a defined timeframe.
- Notify the client when the limit is exceeded by providing the limit number and the time the limit will be reset.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the rate limitation protection against the API endpoints.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

CWE-770: Allocation of Resources Without Limits or Throttling

3.6 (HAL-06) VULNERABLE JAVASCRIPT DEPENDENCY - MEDIUM

Description:

The use of third-party JavaScript libraries can introduce a range of DOM-based vulnerabilities, including some that can be used to hijack user accounts like DOM-XSS.
Common JavaScript libraries typically enjoy the benefit of being heavily verified. This may mean that bugs are quickly identified and patched upstream, resulting in a steady stream of security updates that need to be applied. Although it may be tempting to ignore updates, using a library with missing security patches can make your website exceptionally easy to exploit. Therefore, it's important to ensure that any available security updates are applied promptly.
Some library vulnerabilities expose every application that imports the library, but others only affect applications that use certain library features. Accurately identifying which library vulnerabilities apply to your website can be difficult, so we recommend applying all available security updates regardless.

We observed a vulnerable JavaScript library. We detected Next.js version 12.0.3, which has the following vulnerabilities:
CVE-2022-23646: Improper CSP in Image Optimization API
CVE-2022-21721: DOS Vulnerability for self-hosted next.js apps
CVE-2021-43803: Unexpected server crash in Next.js versions

Proof of concept:

Figure 14: Outdated Nextjs version identified

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Update your application dependency to the latest stable version. It is also recommended to perform an automated analysis of the dependencies from the birth of the project and if they contain any security issues. Developers should be aware of this and apply any necessary mitigation measures to protect the affected application.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by updating the dependency to the latest.
commit:
930614d1ecdf69f17198e0e4fb54c4f27ba2b42d

3.7 (HAL-07) MISSING SECURITY HEADERS - MEDIUM

Description:

Important security HTTP Headers are missing from the scoped web applications. These headers used by the client browser improve the security of end users against common attacks.

Missing important security headers;
X-Frame-Options, **Strict-Transport-Security** and **Content-Security-Policy** response headers.

- X-Frame-Options header is not present in the response of the scoped websites, which makes them vulnerable to the Clickjacking attack. Clickjacking is a malicious technique to trick a web user into clicking something besides what the user perceives them to be doing, potentially revealing sensitive information or taking control of their computer while clicking seemingly innocuous web pages.
- Content-Security-Policy is an effective measure to protect your site from XSS attacks. By whitelisting sources of approved content, it is possible to prevent the browser from loading malicious assets.
- Strict-Transport-Security HTTP Strict Transport Security is an excellent feature to support on your site and strengthens your implementation of TLS by getting the User Agent to enforce the use of HTTPS. Recommended value “Strict-Transport-Security: max-age=31536000; includeSubDomains”.
- X-Content-Type-Options stops a browser from trying to MIME-sniff the content type and forces it to stick with the declared content-type. The only valid value for this header is “X-Content-Type-Options: nosniff”.

Proof of concept:

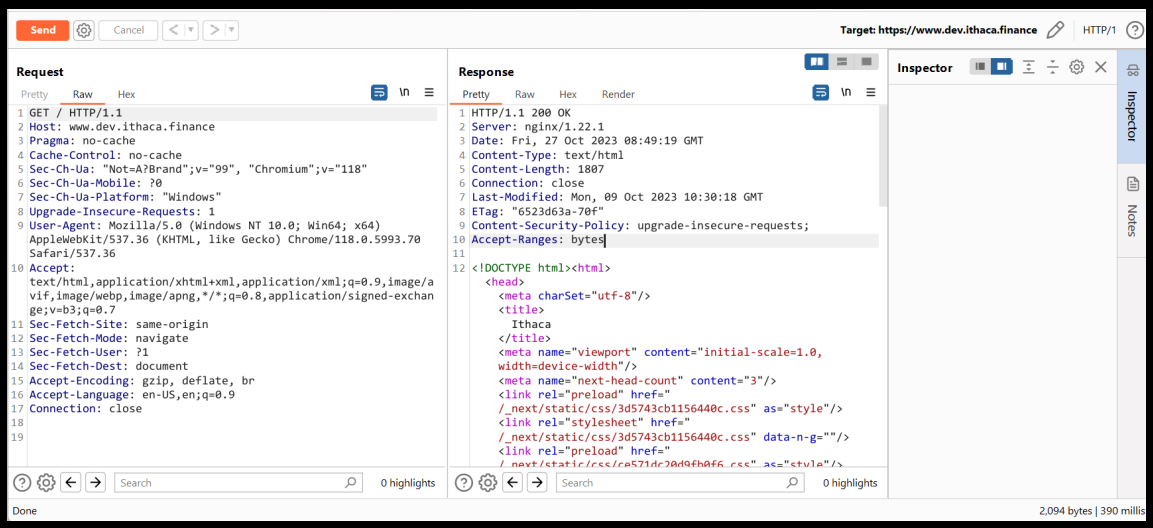


Figure 15: missing security headers www.dev.ithaca.finance

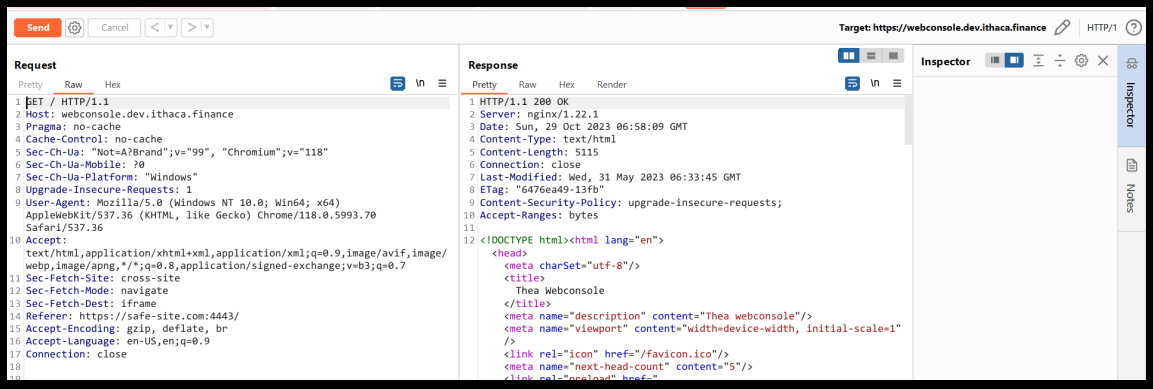


Figure 16: missing security headers webconsole.dev.ithaca.finance

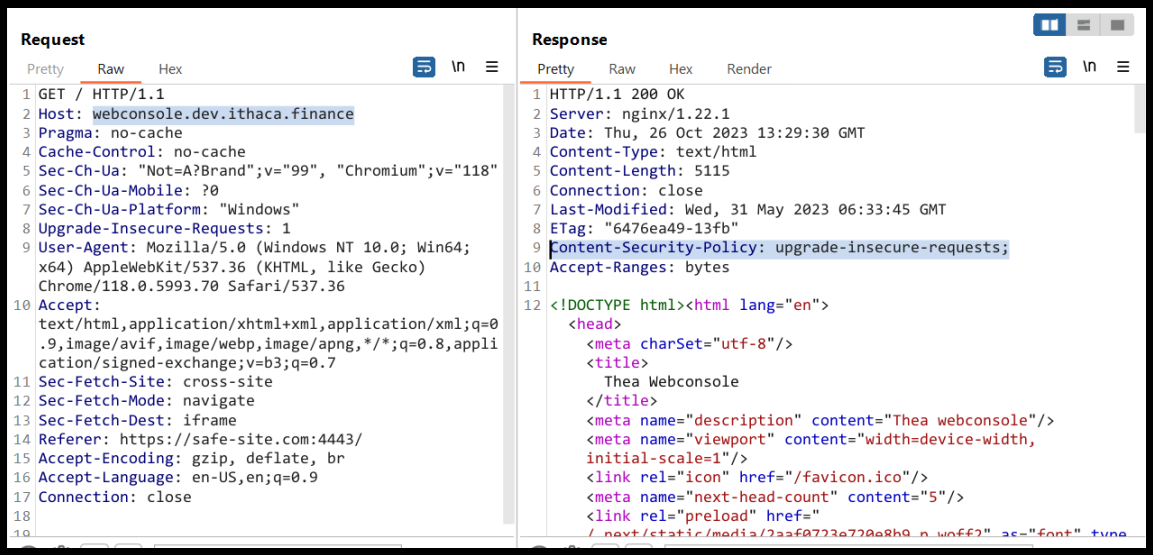


Figure 17: Misconfigured CSP in webconsole.dev.ithaca.finance

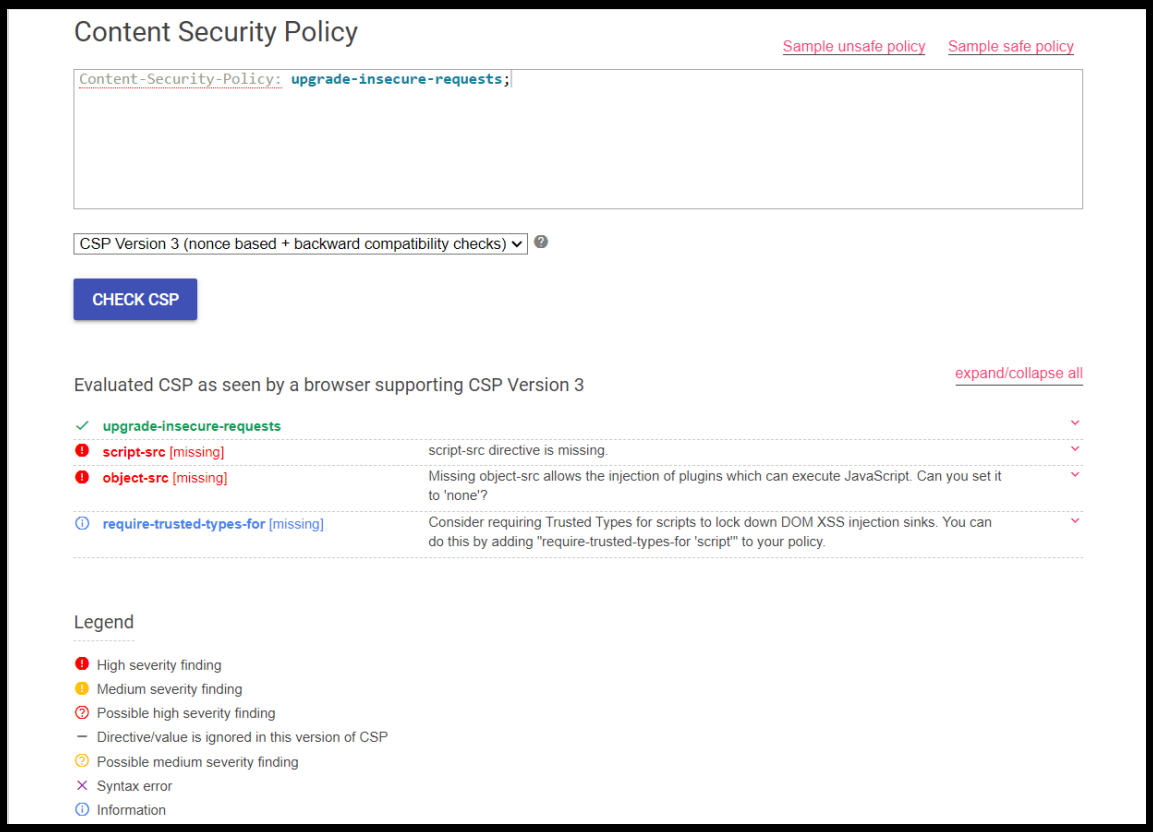


Figure 18: Misconfigured CSP in webconsole.dev.ithaca.finance

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:R/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

It is recommended to define Strict-Transport-Security Content-Security-Policy, X-Frame-Options and X-Content-type-options response headers with appropriate policies.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the security headers with appropriate policies.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

X-Frame-Options
Content Security Policy
Content-Type
Strict-Transport-Security
x-content-type-options

3.8 (HAL-08) REFLECTED UNSANITIZED INPUT - LOW

Description:

We have identified user's inputs reflected in the response of API request. It is worth noting that these values didn't cause any XSS type injection attack currently on the UI. This, however, does not mean that the response is not displayed, or will not be displayed in the future, on other web pages, making it a potential **Cross Site Scripting** attack vector.

Screenshot:

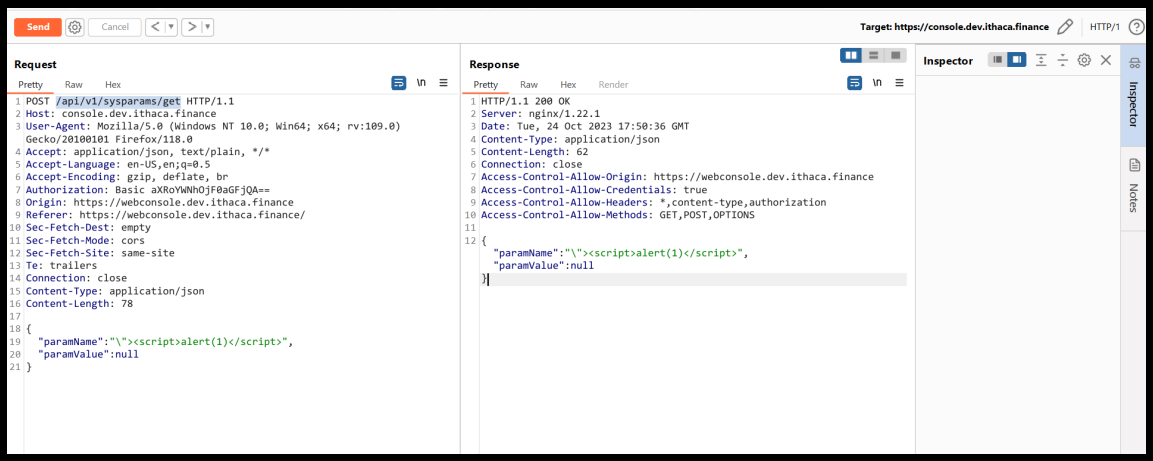


Figure 19: Reflected unsanitized input on API response

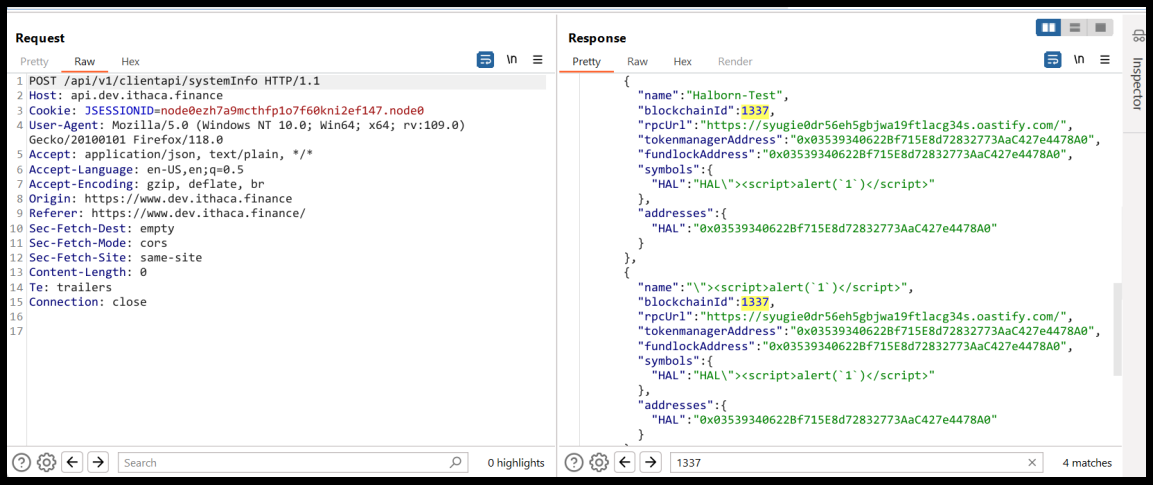


Figure 20: Reflected unsanitized input on API response

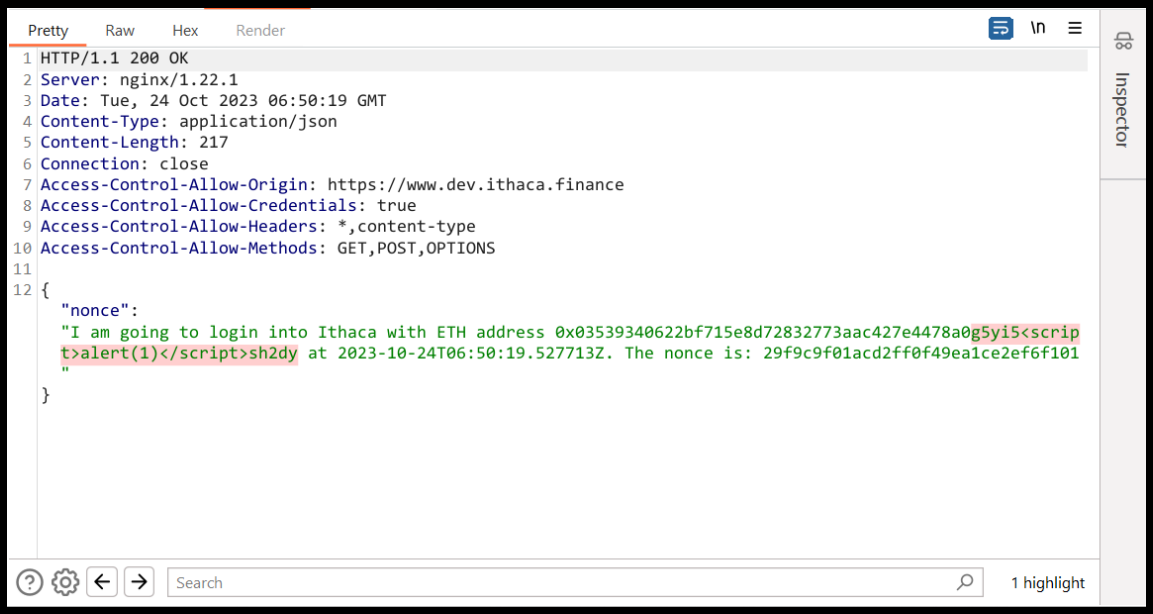


Figure 21: Reflected unsanitized input on API response

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:U/C:L/I:L/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is recommended to sanitize all user input on the frontend and backend on API level as well. One commonly used technique is HTML entity encoding, which converts special characters into their corresponding HTML entities. This approach ensures that user input is safely handled by the server and can be safely displayed in the browser without being interpreted as HTML or script code.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the appropriate checks.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

Cross Site Scripting Prevention Cheat Sheet

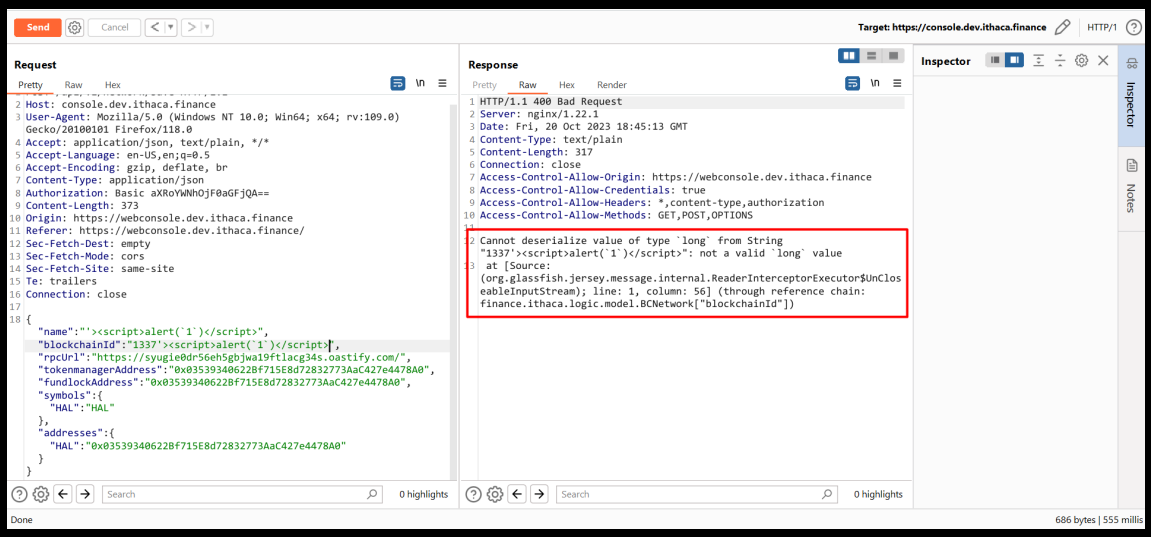
3.9 (HAL-09) VERBOSE ERROR RESPONSE ON EXCEPTIONS - LOW

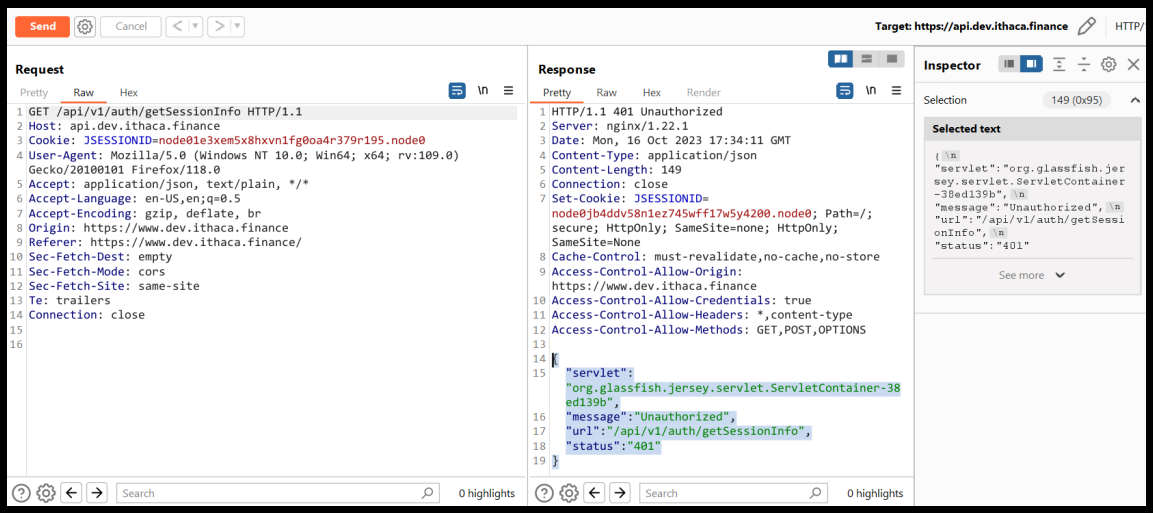
Description:

During the security assessment, it was observed that the application's error messages disclose internal implementation details. Specifically, when an unauthorized request is made to the /api/v1/auth/getSessionInfo endpoint, OR any exception cause against unexpected behavior the server responds with error and includes the servlet's information in the error message.

This disclosure could provide an attacker with insights into the internal workings of the application, potentially aiding them in further attacks.

Proof of concept:





CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

To mitigate this vulnerability, it is recommended to configure the application server to provide generic error messages that do not disclose internal details to the client. Ensure that error handling is implemented consistently across the application to prevent information disclosure. Additionally, review and update the application's logging and monitoring capabilities to ensure that detailed error information is captured in the logs for debugging purposes, without exposing it to the end user.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by removing the verbose exceptions.

NOTE: Retest was conducted on the following URLs:

- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

Reference:

OWASP Information Disclosure
Jersey: How to configure error handling

3.10 (HAL-10) CACHEABLE HTTPS RESPONSE - LOW

Description:

The API service is currently returning cacheable HTTPS responses on certain endpoints. This behavior is risky because sensitive information sent over HTTPS can be cached on the client side, potentially exposing it to unauthorized users who have access to the client's device.

Caching behavior observed indicates that HTTP response headers from the API service do not adequately prevent caching, missing headers like:
- Cache-Control: no-store, no-cache, must-revalidate, private
- Pragma: no-cache
- Expires: 0

The absence of these headers allows client-side caching of potentially sensitive information, leading to risks such as data breaches and unauthorized access.

Proof of concept:

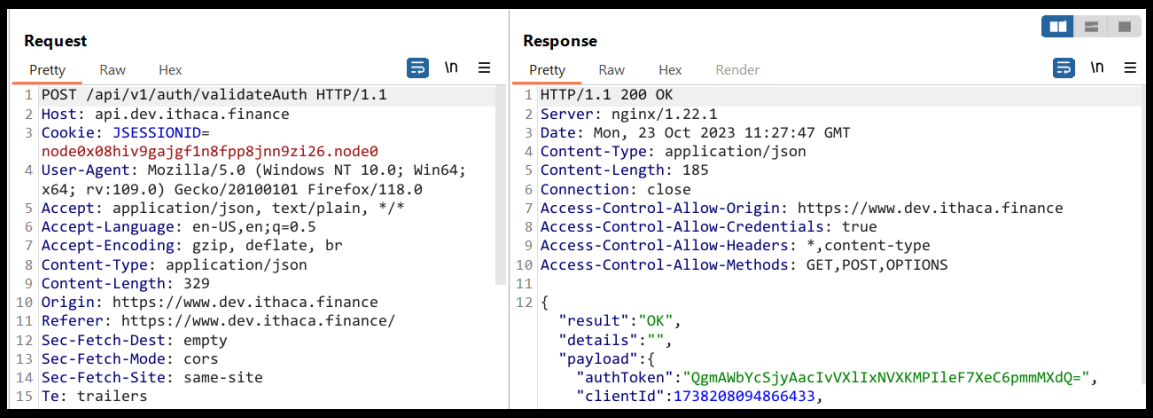


Figure 22: No cache directive defined on api.dev.ithaca.finance

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is advised to include the necessary HTTP headers on all sensitive endpoints to prevent caching of any sensitive information or responses that should not be stored on the client side. Specifically:
- Set Cache-Control to “no-store, no-cache, must-revalidate, private”
- Add Pragma: no-cache
- Use Expires: 0

These headers will instruct the client’s browser not to store the sensitive content, reducing the risk of data exposure.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by implementing the appropriate headers.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

References:

Cache-Control
Pragma
Expires

3.11 (HAL-11) NGINX VERSION DISCLOSURE - INFORMATIONAL

Description:

The scoped web applications revealed the exact version of the nginx server on the error page. This information is visible to anyone, and knowing the exact version of the server can help attackers to identify potential vulnerabilities and design attacks specifically tailored for this version of the application.

Attackers can use this information to search public vulnerability databases and other sources to find any known vulnerabilities that affect the version of NGINX in use. They can then use this information to craft targeted attacks that are more likely to succeed.

Proof of concept:

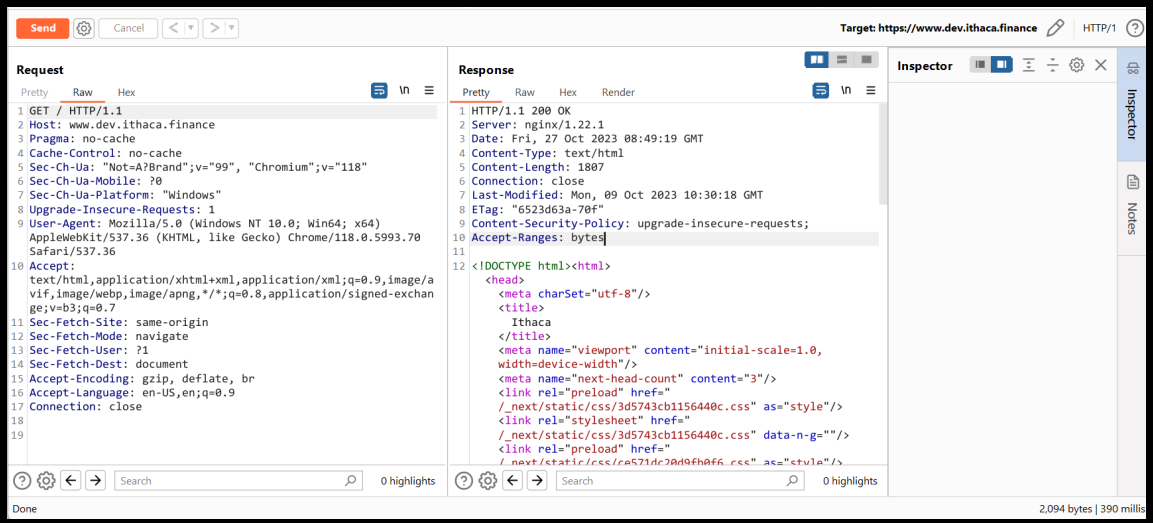


Figure 23: NGINX version disclose on RESPONSE

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

To remediate this vulnerability, we recommend updating nginx.conf by adding the following configuration:

Listing 1

```
http {
    ...
    server_tokens off;
    ...
}
```

The above configuration disables the server tokens and removes the version number from the server header or footer. This makes it more difficult for attackers to identify the version of the server and reduces the risk of attacks targeted specifically at this version.

In addition, it is recommended regularly update the server and application software to the latest version, as new security patches and updates are often released to address known vulnerabilities. Regular security testing should also be conducted to identify and address any other potential vulnerabilities in the application.

Remediation Plan:

SOLVED: The Ithaca team fixed the issue by removing the version disclosure header in response.

NOTE: Retest was conducted on the following URLs:
- https://app.canary.ithacanoemon.tech/
- https://canary.ithacanoemon.tech/
- https://admin.canary.ithacanoemon.tech/

ANNEX

4.1 Web App Security Testing Methodology

Planning:

1. Gather Scoping Information

After starting the project, scope/target information is collected from the client. In the case of web apps penetration testing, this information includes IP addresses and URLs, a definition files or documentation such as swagger or postman, the source code (if possible), authentication credentials, and/or API tokens (2 sets of credentials for each feature being tested) and a list of any sensitive or restricted endpoints/functionalities that should not be scanned or exploited.

Halborn and the client jointly review and acknowledge the penetration testing rules of engagement, confirm project scope and testing timeline, identify specific testing objectives, document any testing limitations or restrictions, and answer any questions related to the project.

Execution:

1. Information gathering

The information-gathering phase consists of gathering information about the app, including its purpose, functionality, documentation, endpoints, and authentication mechanisms. Understand the app's intended use and potential attack vectors.

It includes application footprinting, metafile leakage review, listing services, operating system functions, and application fingerprinting. This step maps the in-scope application to prepare for identifying exploitable vulnerabilities collectively.

During the Information Gathering phase, Halborn does:

- Use discovery tools to uncover information about the application passively.

- Identify entry points into the application, such as administration portals or backdoors.
- Perform application fingerprinting to identify the underlying development language and components.
- Send fuzzing requests for analysis of error codes that may disclose sensitive data that could be used to launch a more targeted cyberattack.
- Actively scan for open services and develop a test plan for the latter phases of the vulnerability assessment.

If the source code was provided, all this discovery could be done in a more effective way by reading it.

2. Threat Modeling

The threat modeling phase serves to evaluate the types of threats that may affect the target apps that are in scope. The types of attacks and likelihood of these threats materializing serve to inform risk rankings/priorities that are assigned to vulnerabilities throughout the assessment.

The perspective of the testing (external/internal, authenticated/unauthenticated, black box/crystal box, etc.) is also identified to ensure the validity of vulnerabilities discovered. This phase of the assessment also includes manual review of the exposed endpoints, determining business functionality of the endpoints, and identifying unauthenticated/authenticated endpoint attack surface.

With the information collected from the previous step, the testing process transitions to identifying security vulnerabilities in the app.

During this phase, Halborn does:

- Use open-source, commercial, and internally developed tools to identify and confirm well-known vulnerabilities.
- Swagger/Postman documents in-scope to effectively build a map of each feature, component, and area of interest.

- Send fuzzing requests to analyze error codes that may disclose valuable information that can launch a more targeted attack.
- Build the application's threat model using the information gathered in this and the previous phase to be used as a plan of attack for later stages of the penetration test.

If the source code was provided, every unexpected behavior detected will be checked on it to find out the exact reason and discover if it is exploitable in any way.

3. Vulnerability Analysis

The vulnerability analysis phase will encompass the enumeration of all in-scope targets/applications at both the network layer and the application layer. The phase involves documenting and analyzing vulnerabilities discovered due to Information Gathering and Threat Modeling. This step includes the analysis of output from the various security tools and manual testing techniques.

In the Vulnerability Analysis phase, Halborn does:

- Compile the list of areas of interest and develop a plan for exploitation.
- Search and gather known exploits from various sources.
- Analyze the impact and likelihood of each potentially exploitable vulnerability.
- Select the best methods and tools for adequately exploiting each of the suspected security vulnerabilities.

In this phase, Halborn performs an in-depth security source code review with 2 stages:

- Automatic review: The source code is verified using specialized tools/software to scan and examine the code for common security vulnerabilities and insecure coding practices. These tools employ

various techniques, such as pattern matching, data flow analysis, and control flow analysis, to identify potential security risks. The tools used will vary depending on the language of the source code. Additionally, these tools may not detect all types of security vulnerabilities, and manual security assessments and penetration testing are still essential for a comprehensive security evaluation of an application.

- Manual review: The source code is reviewed by Halborn security experts to identify security vulnerabilities and weaknesses. Unlike automatic security source code reviews, which rely on automated tools, a manual review involves a careful and in-depth examination of the code by experienced security professionals. In this case, Halborn security professionals will perform a thorough examination of the code and identify complex vulnerabilities that may not be easily detected by automated tools.

4. Exploitation

This phase involves taking all potential vulnerabilities identified in the previous phases of the assessment and attempting to exploit them as an attacker would. This helps to evaluate the realistic risk level associated with the successful exploitation of the vulnerability, analyze the possibility of exploit/attack chains, and account for any mitigating controls that may be in place.

Exploitation involves establishing access to the application or connected components by bypassing security controls and exploiting vulnerabilities to determine their real-world risk through ethical hacking. Throughout this step, several manual tests will be performed simulating real-world exploits incapable of being achieved through automated means. During a Halborn penetration test, the exploitation phase involves heavy manual testing tactics and is often the most time-intensive.

This includes business logic flaws, authentication/authorization bypasses, direct object references, injection-style attacks (SQL, command, XPath, LDAP, XXE, XSS), error analysis, file uploads, parameter tampering, and session management etc.

The exploitation part includes:

- Authorization and Authentication Testing: Test the app's authentication mechanisms, such as API keys, tokens, OAuth, or other access control methods. Verify if proper authentication is enforced and that it cannot be easily bypassed or abused.
- Input Validation: Test for input validation vulnerabilities, such as injection attacks (SQL, command, or XML), cross-site scripting (XSS), Path, RCE, and cross-site request forgery (CSRF) etc. Send malicious input to the app endpoints to identify any security weaknesses.
- Error Handling: Check how the app handles various error conditions, such as invalid input, server-side errors, or exceptions. Determine if error messages reveal sensitive information that could aid attackers.
- Data Integrity and Confidentiality: Assess the app's ability to protect data integrity and confidentiality during transmission and storage. Evaluate the implementation of encryption, secure transport protocols (e.g., HTTPS), and secure storage practices.
- Session Management: Test how the app manages sessions, tokens, and state information. Identify any session-related vulnerabilities that could lead to unauthorized access or privilege escalation.
- Business Logic Testing: Analyze the app's business logic to identify vulnerabilities or flaws in the design and implementation. Test for authorization bypass, bypassing custom workflows, insecure direct object references, and other logical vulnerabilities.
- Rate Limiting and Throttling: Evaluate if the app has proper rate limiting and throttling mechanisms in place to prevent abuse and mitigate denial-of-service (DoS) attacks.
- Source code review: Potential vulnerabilities found in both automatic and manual source code review will be reviewed and exploited.

- Third-Party Integration: Assess the security of any third-party libraries, frameworks, or services used in the app implementation. Review their security controls and potential vulnerabilities.

In the Exploitation phase, Halborn does conduct:

- Attempt to manually exploit the vulnerabilities identified in the previous steps to determine the possible level of risk and level of exploitation.
- Capture and log evidence to provide proof of exploitation (images, screenshots, configs, etc.).
- Notify the client of findings.

5. Post Exploitation

After successful exploitation, analysis may continue, including infrastructure analysis, pivoting, sensitive data identification, data exfiltration, and identification of high-value targets/data. The information collected here is used in the prioritization and criticality ranking of identified vulnerabilities.

Furthermore, chaining different vulnerabilities can lead to proof-of-concept vulnerabilities with higher critically.

Post-Execution:

1. Reporting

After completing all the phases, Halborn formally documents the findings. The output provided generally includes an executive-level report and a technical findings report. The executive-level report is written for management consumption and includes a high-level overview of assessment activities, scope, most critical/thematic issues discovered, overall risk scoring, organizational security strengths, and applicable screenshots.

The technical findings report, on the other hand, includes all vulnerabilities listed individually, with details as to how to recreate

the issue, understand the risk, recommended remediation actions, and helpful reference links. Both include actionable recommendations for improving security.

2. Quality Assurance

All assessments go through a rigorous technical and editorial quality assurance phase. This may also include follow-ups with the client to confirm or deny details, as appropriate.

3. Presentation

The final activity in any assessment is a presentation of all documentation to the client. Halborn walks the client through the information provided, makes any updates needed, and addresses questions regarding the assessment output. Following this activity, new revisions of documentation and schedule any formal retesting are provided, if applicable.

ANNEX

5.1 Web App Security Testing Methodology

Planning:

1. Gather Scoping Information

After starting the project, scope/target information is collected from the client. In the case of web apps penetration testing, this information includes IP addresses and URLs, a definition files or documentation such as swagger or postman, the source code (if possible), authentication credentials, and/or API tokens (2 sets of credentials for each feature being tested) and a list of any sensitive or restricted endpoints/functionalities that should not be scanned or exploited.

Halborn and the client jointly review and acknowledge the penetration testing rules of engagement, confirm project scope and testing timeline, identify specific testing objectives, document any testing limitations or restrictions, and answer any questions related to the project.

Execution:

1. Information gathering

The information-gathering phase consists of gathering information about the app, including its purpose, functionality, documentation, endpoints, and authentication mechanisms. Understand the app's intended use and potential attack vectors.

It includes application footprinting, metafile leakage review, listing services, operating system functions, and application fingerprinting. This step maps the in-scope application to prepare for identifying exploitable vulnerabilities collectively.

During the Information Gathering phase, Halborn does:

- Use discovery tools to uncover information about the application passively.

- Identify entry points into the application, such as administration portals or backdoors.
- Perform application fingerprinting to identify the underlying development language and components.
- Send fuzzing requests for analysis of error codes that may disclose sensitive data that could be used to launch a more targeted cyberattack.
- Actively scan for open services and develop a test plan for the latter phases of the vulnerability assessment.

If the source code was provided, all this discovery could be done in a more effective way by reading it.

2. Threat Modeling

The threat modeling phase serves to evaluate the types of threats that may affect the target apps that are in scope. The types of attacks and likelihood of these threats materializing serve to inform risk rankings/priorities that are assigned to vulnerabilities throughout the assessment.

The perspective of the testing (external/internal, authenticated/unauthenticated, black box/crystal box, etc.) is also identified to ensure the validity of vulnerabilities discovered. This phase of the assessment also includes manual review of the exposed endpoints, determining business functionality of the endpoints, and identifying unauthenticated/authenticated endpoint attack surface.

With the information collected from the previous step, the testing process transitions to identifying security vulnerabilities in the app.

During this phase, Halborn does:

- Use open-source, commercial, and internally developed tools to identify and confirm well-known vulnerabilities.
- Swagger/Postman documents in-scope to effectively build a map of each feature, component, and area of interest.

- Send fuzzing requests to analyze error codes that may disclose valuable information that can launch a more targeted attack.
- Build the application's threat model using the information gathered in this and the previous phase to be used as a plan of attack for later stages of the penetration test.

If the source code was provided, every unexpected behavior detected will be checked on it to find out the exact reason and discover if it is exploitable in any way.

3. Vulnerability Analysis

The vulnerability analysis phase will encompass the enumeration of all in-scope targets/applications at both the network layer and the application layer. The phase involves documenting and analyzing vulnerabilities discovered due to Information Gathering and Threat Modeling. This step includes the analysis of output from the various security tools and manual testing techniques.

In the Vulnerability Analysis phase, Halborn does:

- Compile the list of areas of interest and develop a plan for exploitation.
- Search and gather known exploits from various sources.
- Analyze the impact and likelihood of each potentially exploitable vulnerability.
- Select the best methods and tools for adequately exploiting each of the suspected security vulnerabilities.

In this phase, Halborn performs an in-depth security source code review with 2 stages:

- Automatic review: The source code is verified using specialized tools/software to scan and examine the code for common security vulnerabilities and insecure coding practices. These tools employ

various techniques, such as pattern matching, data flow analysis, and control flow analysis, to identify potential security risks. The tools used will vary depending on the language of the source code. Additionally, these tools may not detect all types of security vulnerabilities, and manual security assessments and penetration testing are still essential for a comprehensive security evaluation of an application.

- Manual review: The source code is reviewed by Halborn security experts to identify security vulnerabilities and weaknesses. Unlike automatic security source code reviews, which rely on automated tools, a manual review involves a careful and in-depth examination of the code by experienced security professionals. In this case, Halborn security professionals will perform a thorough examination of the code and identify complex vulnerabilities that may not be easily detected by automated tools.

4. Exploitation

This phase involves taking all potential vulnerabilities identified in the previous phases of the assessment and attempting to exploit them as an attacker would. This helps to evaluate the realistic risk level associated with the successful exploitation of the vulnerability, analyze the possibility of exploit/attack chains, and account for any mitigating controls that may be in place.

Exploitation involves establishing access to the application or connected components by bypassing security controls and exploiting vulnerabilities to determine their real-world risk through ethical hacking. Throughout this step, several manual tests will be performed simulating real-world exploits incapable of being achieved through automated means. During a Halborn penetration test, the exploitation phase involves heavy manual testing tactics and is often the most time-intensive.

This includes business logic flaws, authentication/authorization bypasses, direct object references, injection-style attacks (SQL, command, XPath, LDAP, XXE, XSS), error analysis, file uploads, parameter tampering, and session management etc.

The exploitation part includes:

- Authorization and Authentication Testing: Test the app's authentication mechanisms, such as API keys, tokens, OAuth, or other access control methods. Verify if proper authentication is enforced and that it cannot be easily bypassed or abused.
- Input Validation: Test for input validation vulnerabilities, such as injection attacks (SQL, command, or XML), cross-site scripting (XSS), Path, RCE, and cross-site request forgery (CSRF) etc. Send malicious input to the app endpoints to identify any security weaknesses.
- Error Handling: Check how the app handles various error conditions, such as invalid input, server-side errors, or exceptions. Determine if error messages reveal sensitive information that could aid attackers.
- Data Integrity and Confidentiality: Assess the app's ability to protect data integrity and confidentiality during transmission and storage. Evaluate the implementation of encryption, secure transport protocols (e.g., HTTPS), and secure storage practices.
- Session Management: Test how the app manages sessions, tokens, and state information. Identify any session-related vulnerabilities that could lead to unauthorized access or privilege escalation.
- Business Logic Testing: Analyze the app's business logic to identify vulnerabilities or flaws in the design and implementation. Test for authorization bypass, bypassing custom workflows, insecure direct object references, and other logical vulnerabilities.
- Rate Limiting and Throttling: Evaluate if the app has proper rate limiting and throttling mechanisms in place to prevent abuse and mitigate denial-of-service (DoS) attacks.
- Source code review: Potential vulnerabilities found in both automatic and manual source code review will be reviewed and exploited.

- Third-Party Integration: Assess the security of any third-party libraries, frameworks, or services used in the app implementation. Review their security controls and potential vulnerabilities.

In the Exploitation phase, Halborn does conduct:

- Attempt to manually exploit the vulnerabilities identified in the previous steps to determine the possible level of risk and level of exploitation.
- Capture and log evidence to provide proof of exploitation (images, screenshots, configs, etc.).
- Notify the client of findings.

5. Post Exploitation

After successful exploitation, analysis may continue, including infrastructure analysis, pivoting, sensitive data identification, data exfiltration, and identification of high-value targets/data. The information collected here is used in the prioritization and criticality ranking of identified vulnerabilities.

Furthermore, chaining different vulnerabilities can lead to proof-of-concept vulnerabilities with higher critically.

Post-Execution:

1. Reporting

After completing all the phases, Halborn formally documents the findings. The output provided generally includes an executive-level report and a technical findings report. The executive-level report is written for management consumption and includes a high-level overview of assessment activities, scope, most critical/thematic issues discovered, overall risk scoring, organizational security strengths, and applicable screenshots.

The technical findings report, on the other hand, includes all vulnerabilities listed individually, with details as to how to recreate

the issue, understand the risk, recommended remediation actions, and helpful reference links. Both include actionable recommendations for improving security.

2. Quality Assurance

All assessments go through a rigorous technical and editorial quality assurance phase. This may also include follow-ups with the client to confirm or deny details, as appropriate.

3. Presentation

The final activity in any assessment is a presentation of all documentation to the client. Halborn walks the client through the information provided, makes any updates needed, and addresses questions regarding the assessment output. Following this activity, new revisions of documentation and schedule any formal retesting are provided, if applicable.

THANK YOU FOR CHOOSING

// HALBORN