ENIGMA DARK

Securing the Shadows

Security Review & Penetration Testing

Unhosted Wallet: Extension Core, Backend Services

February, 2025

Contents

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Unhosted Wallet: Extension Core & Backend Services
Unhosted Wallet is a next-generation, self-custody wallet built on Biconomy's Nexus account abstraction framework. It offers seamless integration with various providers, including fiat24 and Onramper, to enhance user experience.

Engagement Overview

Over the course of 3 weeks, beginning 27 January 2025, the Enigma Dark team conducted a security review of the Unhosted Wallet: Extension Core & Backend Services project. The review was performed by two Security Researchers: Jakub heba & N0xi0us.

The following repositories were reviewed at the specified commits:

Repository	Commit
Unhosted-Wallet/Unhosted/src/background	`d228fa585d25570c895e8103db27ccdfb74886b7`
Unhosted-Wallet/unhosted-wallet-backend	`48d5ebbcc5b368ba7ed9d132523f7026dd8f7f39`
Unhosted-Wallet/lib-unhosted-swap.js	`aaa093ab2616eba3e57f566e70d50fbaa44c9d38`
Unhosted-Wallet/lib-unhosted-signer.js	`57846236d4de7e37c24c562a11093f8c992af139`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the system.
Low	Minor code errors that may or may not impact the system.
Informational	Non-critical observations or suggestions for improving code quality, readability, or best practices.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	3	3	0
Medium	3	3	0
Low	2	2	0
Informational	2	2	0

Findings

Index	Issue Title	Status
H-01	Hardcoded AWS Secrets in Environment Variables	Fixed
H-02	Direct call to the `aa` service reveals the `bundlerSecret` and `pmSecret` in error path	Fixed
H-03	Lack of pagination enforcement leads to DoS	Fixed
M-01	Race condition allows to repeatedly claim quests	Fixed
M-02	Lack of authorization allows to claim arbitrary quests	Fixed
M-03	Password, seed phrases or private key might be extracted from the browser memory	Fixed
L-01	Improper address check in wallet creation	Fixed
L-02	Missing upper cache limit	Fixed
I-01	Backend implements never used services	Fixed
I-02	Unused WalletController handlers	Fixed

Detailed Findings

High Risk

H-01 - Hardcoded AWS Secrets in Environment Variables

Severity: High Risk

Technical Details:
During the assessment of the Kubernetes environment in Amazon EKS, it was discovered that AWS access credentials were exposed in the environment variables of running pods. This practice poses a significant security risk, as any process within the container—or an attacker with access to the pod—can extract and misuse these credentials.

Impact:
Privilege escalation and information disclosure.

Recommendation:
To mitigate this risk, AWS IAM Roles for Service Accounts (IRSA) should be implemented to securely grant AWS permissions to Kubernetes workloads without hardcoded credentials.

Developer Response:
Fixed at commit `bf6df8a`.

H-02 - Direct call to the `aa` service reveals the `bundlerSecret` and `pmSecret` in error path

Severity: High Risk

Technical Details:
It was found, that the `aa-paymaster-proxy` and `aa-bundler-proxy` services allows for unauthorized call to the, respectively, `/aa/paymaster/${chainId}/` and `/aa/bundler/${chainId}/` endpoints.

When specifying the proper `chainId`, for example 1 (Ethereum), request goes through the service, but due to the unknown reason, returns an error with secret included. Bundler:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>Cannot GET /api/v3/1/[REDACTED SECRET]/aa/bundler/1</pre>
</body>
</html>
```

Paymaster:

```
{"statusCode":404,"message":"Cannot GET /api/v2/1/[REDACTED
SECRET]","error":"Not Found"}
```

Impact:
Unauthorized access to the Paymaster and Bundler services using the secrets leaked.

Recommendation:
We recommend to analyze what is the reason of returning secret keys in case of error and mitigate this.

Developer Response:
Fixed at commit `3c2cb60`.

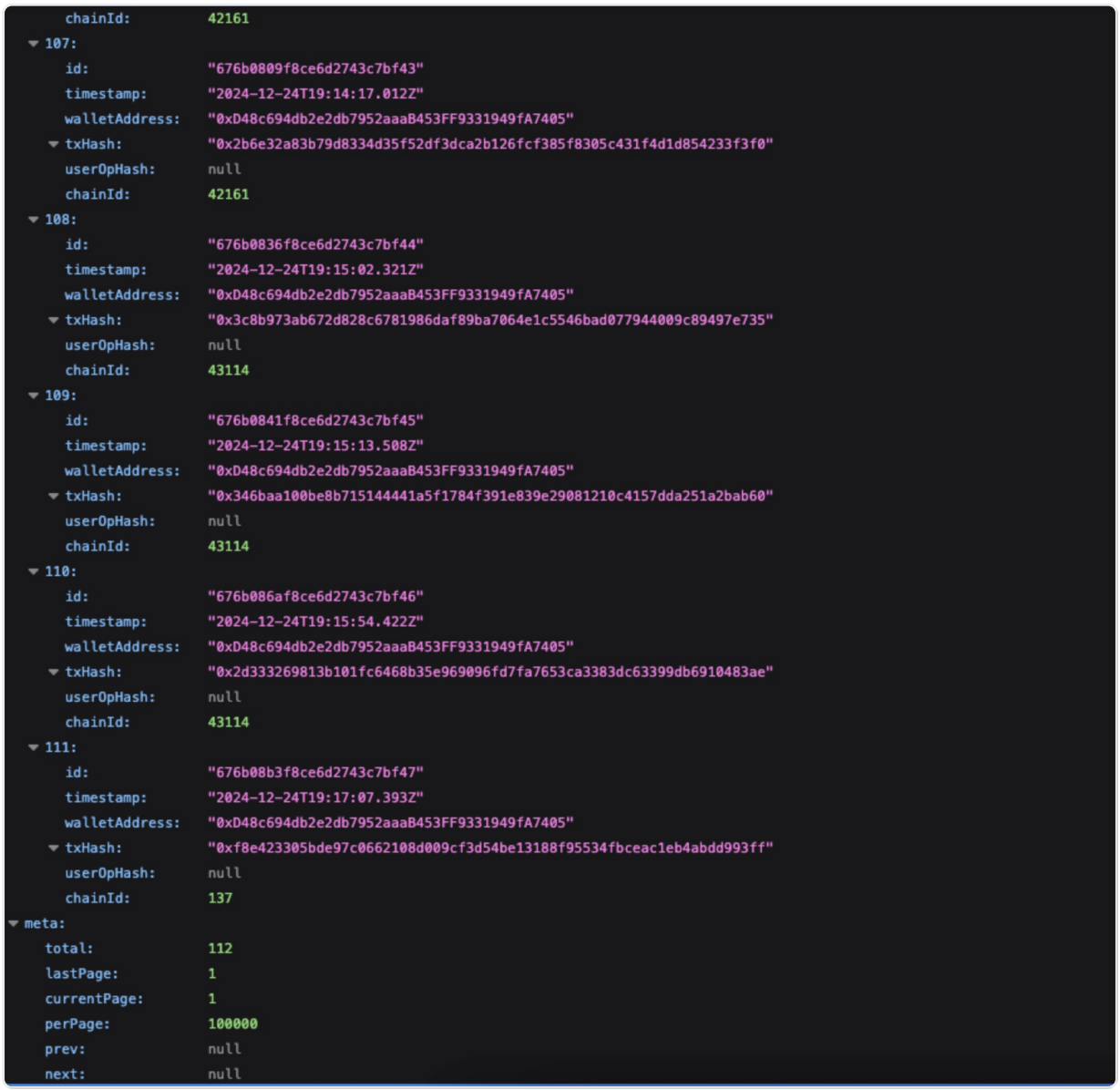
H-03 - Lack of pagination enforcement leads to DoS

Severity: High Risk

Technical Details:
While inspecting referrals/src/utils/paginator.util.ts the following line of code was discovered :
`const perPage = Number(options?.perPage || defaultOptions?.perPage) || 10;`

Since `perPage` is controlled by user input and lacks any enforced limits, an attacker can specify an excessively large value. This forces the server to generate and return an extremely large response, leading to excessive resource consumption and potentially causing a Denial of Service (DoS).

For example browsing to:
http://stg.unhosted.com/transaction/0xD48c694db2e2db7952aaaB453FF9331949fA7405?perPage=100000 reveals that the server accepts any value provided to the perPage parameter.



Impact:
High, this vulnerability could be exploited to overload the backend referral services, potentially causing downtime.

Recommendation:
Implement a limit to the number of results being returned by page.

Developer Response:

Fixed at commit `401b5f1`.

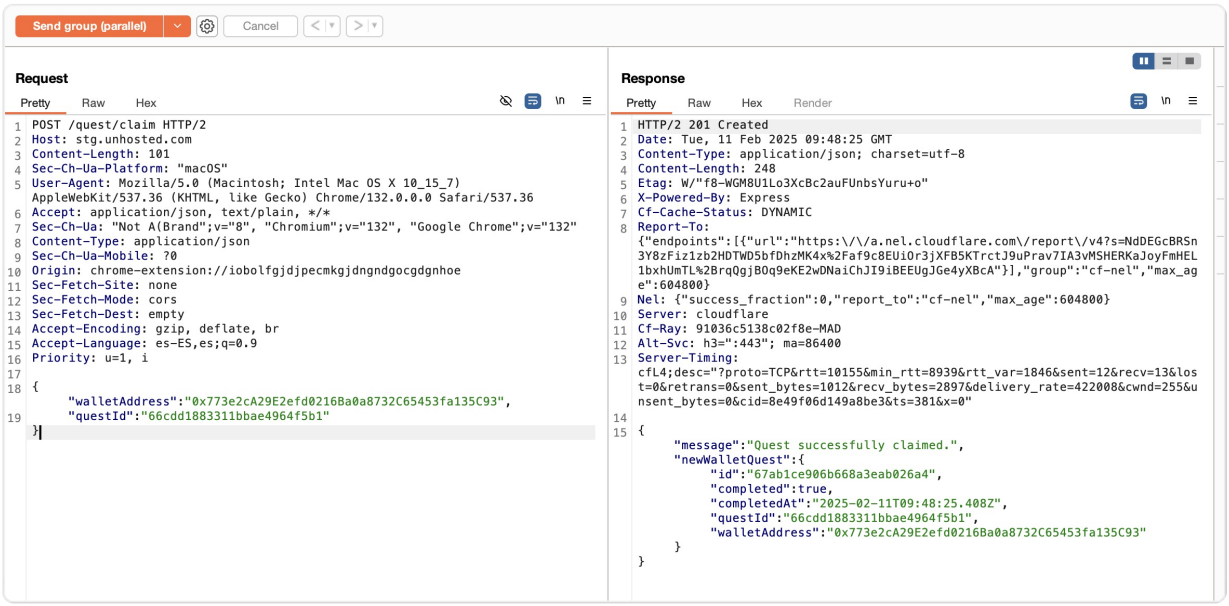
Medium Risk

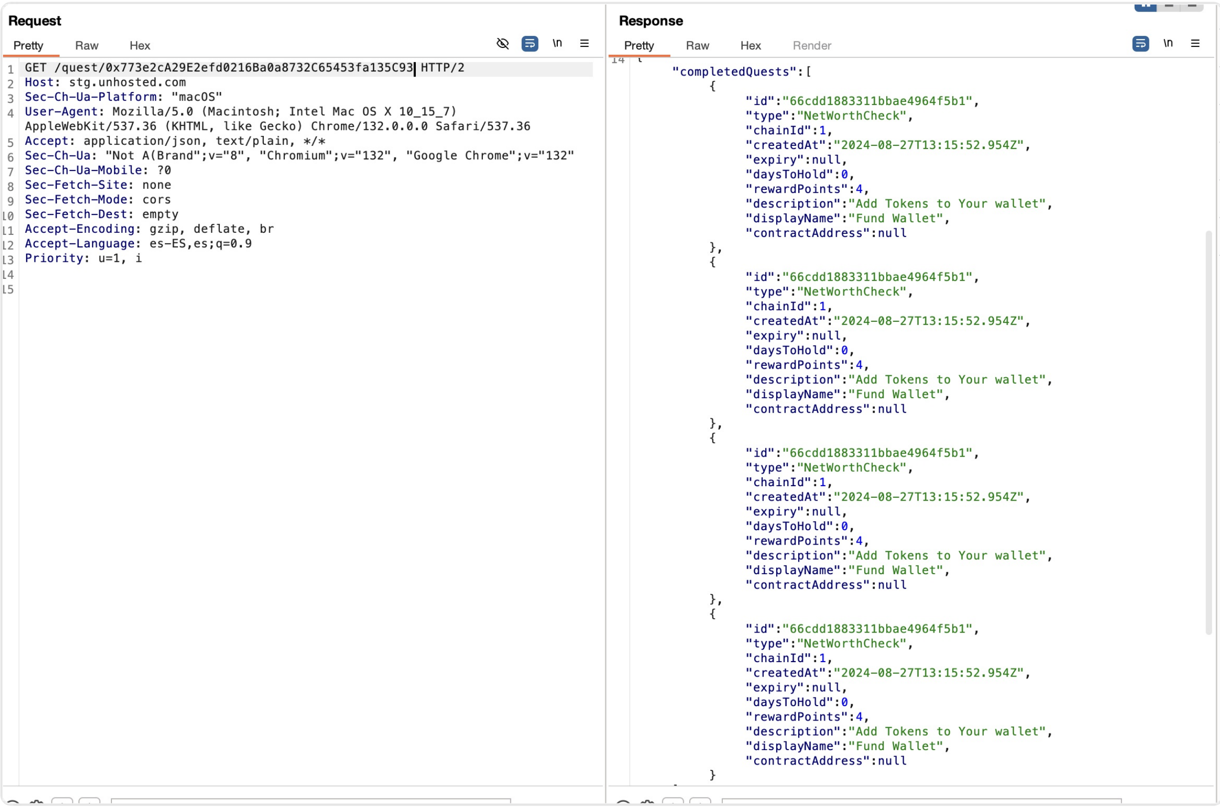
M-01 - Race condition allows to repeatedly claim quests

Severity: Medium Risk

Technical Details:
It was discovered that there is a race condition in `/quest/claim` endpoint allowing to claim a quest more than one time.

In order to test this vulnerability, five claim requests were sent in parallel. As a result, the quest was successfully claimed four out of the five attempts.





Impact:
Medium, Increase points by claiming quests several times.

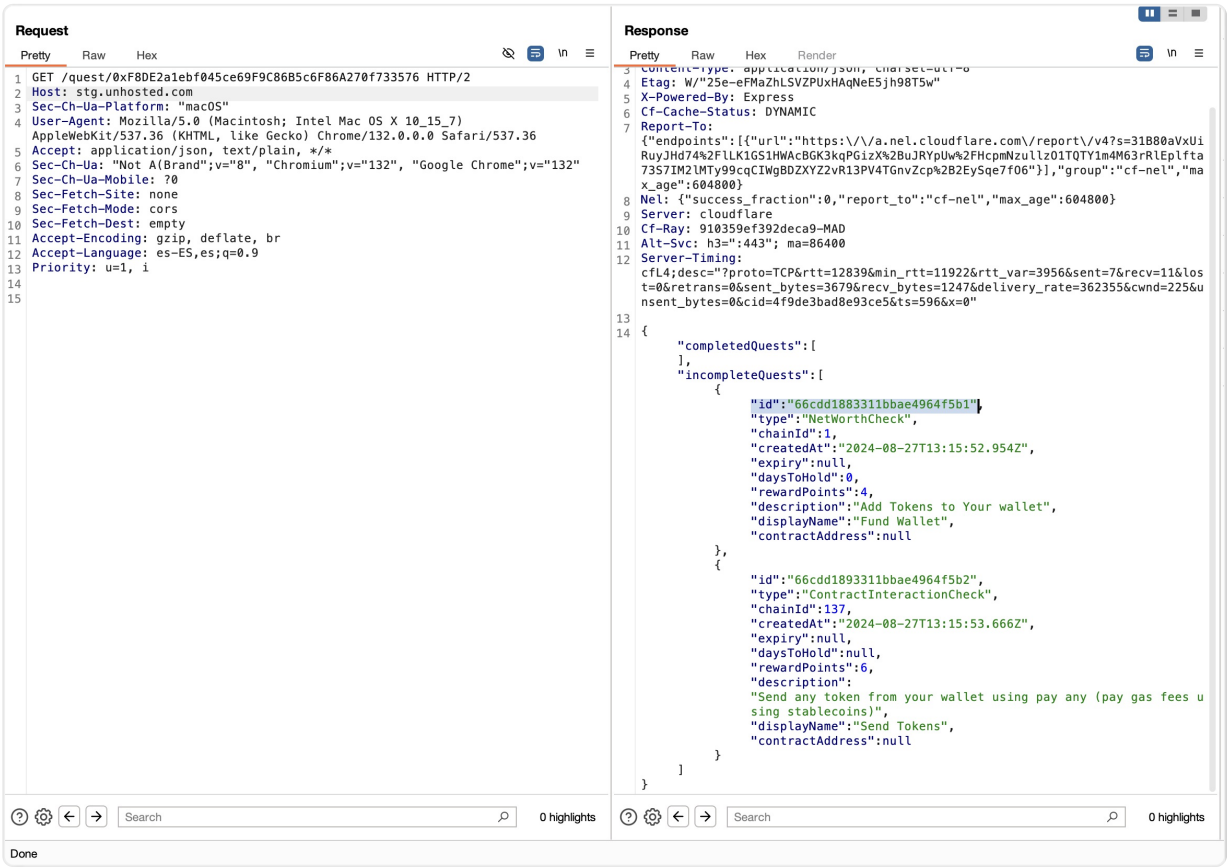
Developer Response:
Fixed. The quests have been removed for the current version.

M-02 - Lack of authorization allows to claim arbitrary quests

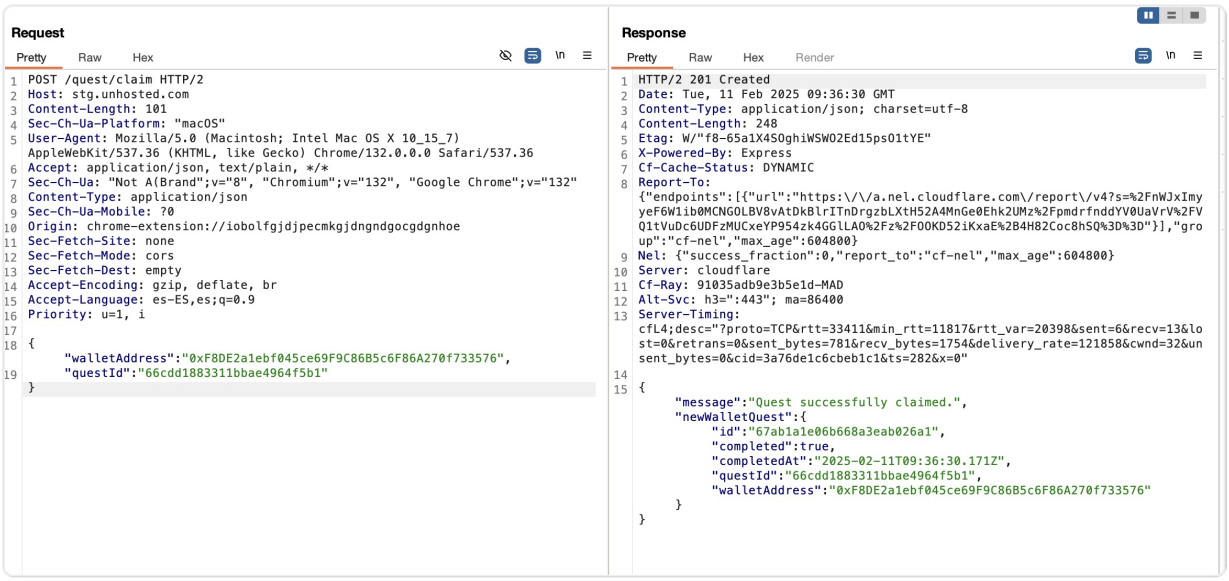
Severity: Medium Risk

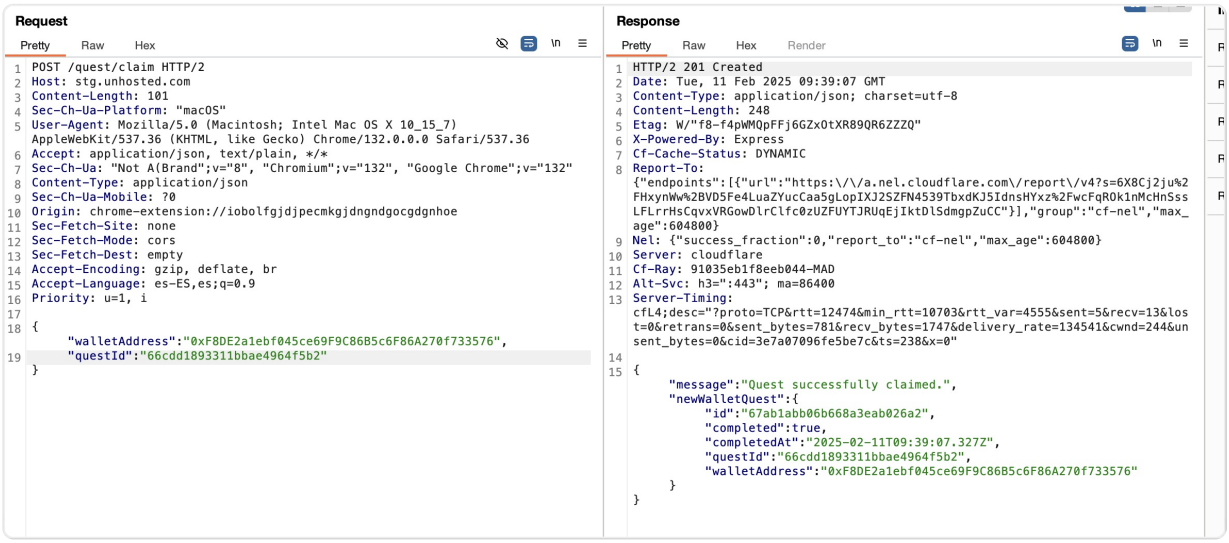
Technical Details:
It was discovered that there are not any checks in place to verify a user has successfully completed a quest before awarding points. Since endpoint `/wallet/walletAddress` returns the IDs of uncompleted quests, a malicious user can just send a POST request to `/quest/claim` passing their walletAddress and questId to successfully claim any quest.

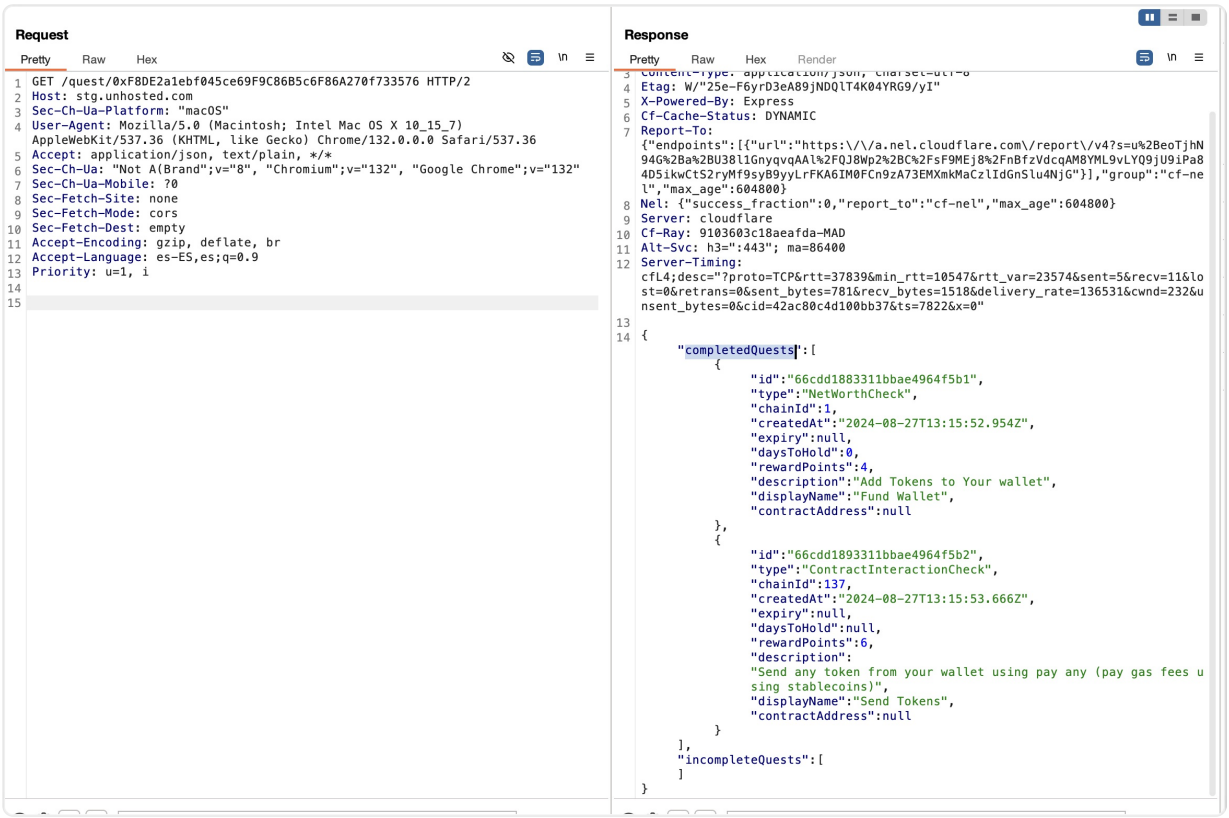
This allows for the enumeration of quest IDs:



Additionally, quests can be claimed without completing the required tasks:







Impact:
Malicious users can claim any quest and receive points without completing the required tasks.

Recommendation:
Implement authorization checks to verify a quest has been completed before allowing to claim it.

Developer Response:
Fixed. The quests have been removed in the current version.

M-03 - Password, seed phrases or private key might be extracted from the browser memory

Severity: Medium Risk

Technical Details:
Sensitive data, including the mnemonic, private key, and user password, remain in the extension's process memory after being used or displayed. These values persist until the extension is fully closed, posing a significant security risk.

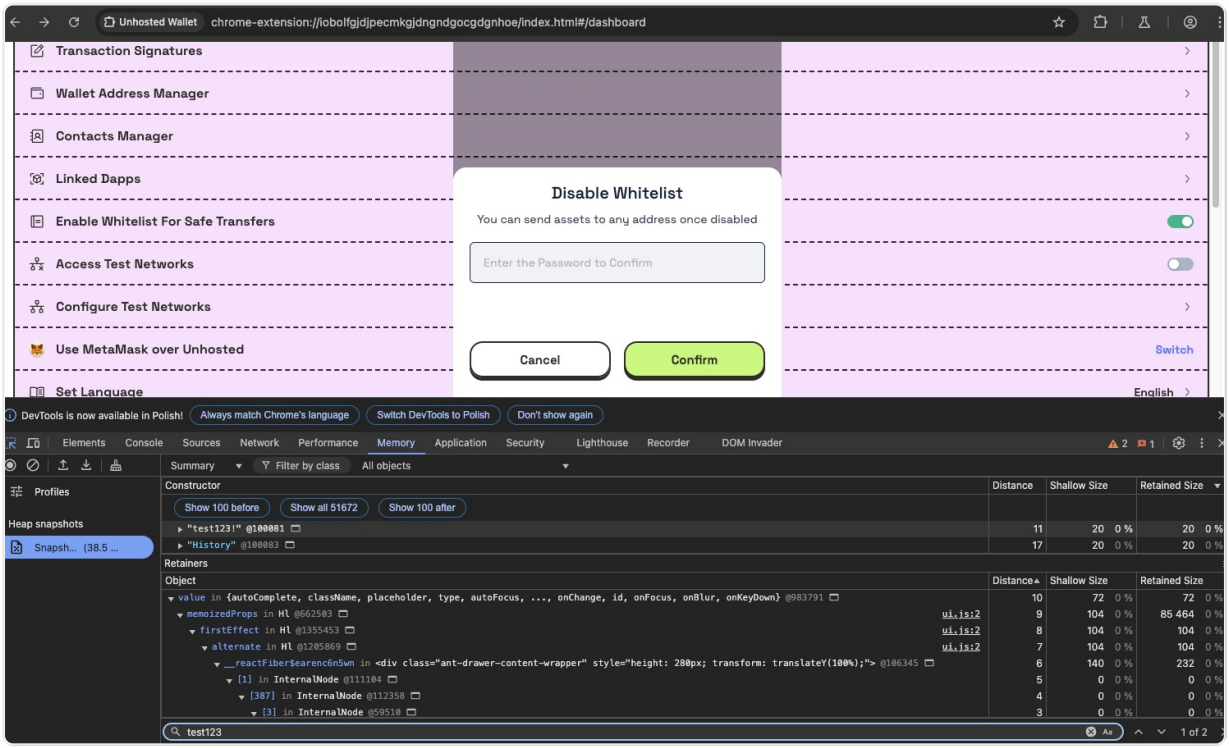
This is particularly problematic because:

- If an attacker gains physical access to the device or if the device is infected with malware, the browser's memory could be dumped.
- This would allow an attacker to extract these confidential values without requiring the user's password, bypassing standard authentication flows.

Additionally, these sensitive strings remain in memory even after `setLocked` is called, meaning that even when the extension is locked, requiring a password to unlock, the data remains accessible in the subprocess memory.

Proof of Concept:

Password being extracted from the extension memory:



Impact:
Extraction of mnemonic, private key or user password from the memory of the extension process, leading to the funds being stolen.

Recommendation:
Make sure that after every critical operation, like password usage, private key displaying or other, the memory part storing these strings are properly overwritten.

Developer Response:

Fixed at commit `61a1c96`.

We've implemented an improved input component to securely handle passwords and private keys, preventing leaks by intercepting input via onKeyDown and encoding values into Uint8Array. Additionally, values received from the wallet controller are encoded, and they are only decoded when being sent back.

For display and copying, we've adjusted how seed phrases and private keys are passed to the respective components—exposing them to the DOM only when necessary and ensuring they are removed from memory during unmount.

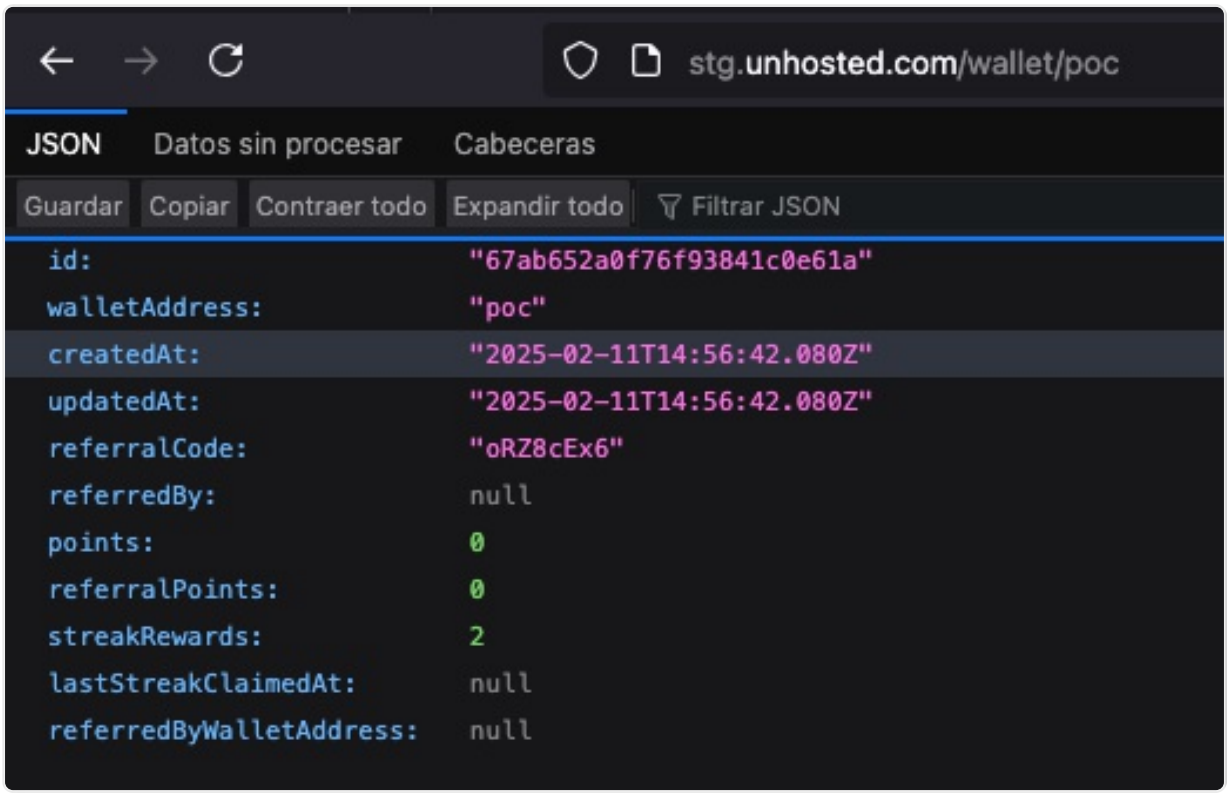
Low Risk

L-01 - Improper address check in wallet creation

Severity: Low Risk

Technical Details:
The address check being performed in `referrals/src/wallet/wallet.controller.ts` is incorrect, allowing a user to create and insert new wallets in the DB with arbitrary data.

As an example, browse to: http://stg.unhosted.com/wallet/poc



Impact:
Filling the DB with invalid addresses.

Recommendation:
Review how the check is implemented.

Developer Response:
Fixed at commit `589d031`.

The validation was being bypassed due to how decorators interact with object destructuring in JavaScript. We resolved this by passing the entire object instead of destructuring it, ensuring that validation executes correctly.

L-02 - Missing upper cache limit

Severity: Low Risk

Technical Details:
It was found that proper expiration times were set for cache storing. While this prevents usage of outdated data or storing unnecessary information, there is no upper limit defined for how much of such information might be stored in the user browser.

Given that multiple `eth-*` methods are supported by `canCache`, a malicious site could utilize it to fill the user's memory with dummy data, leading to decreased user experience and potential browser crash.

Impact:
Decreased user experience and potential browser crash.

Recommendation:
We recommend defining a maximum cap for cache that should be allowed by the extension.

Developer Response:
Fixed in PR 219.

Informational

I-01 - Backend implements never used services

Severity: Informational

Technical Details:
It was found that the Moralis service in `apps/metadata/src/moralis/moralis-metadata.service.ts` defines the `getTransactionVerbose` and `getWalletNetWorth` functions, which are not called by any of the current functionalities.

While not a security issue itself, such functions might be related to not-implemented, forgotten functionalities, which need to be created to fulfill functional requirements.

Impact:
Potential missing functionality or service implementation.

Recommendation:
We recommend removing the unused functions or defining proper handlers to utilize them.

Developer Response:
Fixed at commit e4e7a4b.

I-02 - Unused WalletController handlers

Severity: Informational

Technical Details:
In the `WalletController` handlers list, defined in `src/background/controller/wallet.ts`, multiple of them are not used anywhere in the logic, or called internally. While not directly a security issue, such leftovers might be problematic if some of these functionalities were planned to be implemented, but during the development phase they were forgotten.

Samples: `clearRabbyPointsSignature`, `getLastGetAddress`, `clearWatchMode`, `checkHasMnemonic`, `deriveNewAccountFromMnemonic`, `getAccountsCount`, `checkLedgerHasHIDPermission`, `completedTransaction`, `updateInitAlianNameStatus`, `getCustomTestnetTxReceipt`.

Impact:
Potentially not implemented functionalities, leading to broken logic or missing business assumptions coverage.

Recommendation:
We recommend removing these handlers if they are not needed in the current logic, or implementing proper functionalities to utilize them in the current codebase.

Developer Response:
Fixed in PR 220

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.
This security engagement does not guarantee against a hack. It is a review of the codebase during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the system agree to use the code at their own risk. Any modifications to the code will require a new security review.