// Browser Extension Pentest

01.14.2025 - 01.27.2025

Adena Wallet Chrome Extension

OnBloc

HALBORN

Adena Wallet Chrome Extension - OnBloc

Prepared by: HALBORN

Last Updated 03/05/2025

Date of Engagement by: January 14th, 2025 - January 27th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
13	0	2	8	3	0

TABLE OF CONTENTS

1. Introduction

`Onbloc` engaged Halborn to conduct a security assessment on the Adena Wallet browser extension, beginning on January 14th, 2025 and ending on January 27th, 2025. The security assessment was scoped to the current extension itself, version v1.14.0. The client team provided the source code to allow the security engineers to conduct testing using tools for scanning, detecting, and validating possible vulnerabilities, and report the findings during the engagement.

2. Assessment Summary

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to verify the security of the browser extension. The engineer has advanced knowledge in penetration testing, red teaming, and multiple blockchain protocols.

The goals of our security assessments are as follows:

- Improve the security of the implementation
- Identify potential security issues affecting components interacting with the extension

The wallet exposes the mnemonic phrase in memory, where the mnemonic phrase remains unencrypted in memory even after the wallet is locked, possibly allowing attackers with access to the user's machine to exfiltrate it. Additionally, the user's password was also found stored in cleartext in memory. Furthermore, the wallet uses a static encryption key to encrypt a dynamically generated UUID, which is then used to encrypt the wallet password.

The wallet has several medium-severity vulnerabilities, being two of the most important ones, sensitive data such as mnemonic phrases and private keys possibly being exposed due to the clipboard copy feature, and the wallet also lacks an auto-locking feature, leaving sessions vulnerable to unauthorized access.

Dependencies are not pinned to exact versions, allowing for potential dependency attacks, although they are set to supported ranges. GraphQL introspection queries are enabled, exposing the API schema and backend structure, which could be leveraged to craft malicious queries or access sensitive data. Additionally, the wallet configuration can be overwritten by directly accessing the registration page, compromising the wallet’s integrity and allowing unauthorized changes to the setup.

It is recommended to resolve all the security issues listed in the document to improve the security health of the application and its underlying infrastructure.

3. Test Approach And Methodology

Halborn followed both a whitebox and blackbox approach as per the scope and performed a combination of both manual and automated security testing 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.

The assessment methodology covered a range of phases of tests, and employed various tools, including but not limited to the following:

- Verify minimum necessary permissions
- Source code review
- Hardcoded credentials or API keys
- Sensitive data leakage
- Robust Content Security Policy (CSP)
- Manifest file configuration review
- Validate and sanitize user inputs
- Ensure sensitive data secure storage
- Secure communications for all network communication
- Security headers configuration
- Ensure the least privileges for scripts
- Component isolation verification
- Third-party libraries review
- Monitor extension behavior
- Event listeners and handlers review
- Data collection practices verification
- Multiple browsers verification
- Regulatory compliance
- Static code analysis
- Dynamic analysis
- Review documentation completeness
- Error handling
- Rate limiting

4. 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

5. SCOPE

FILES AND REPOSITORY

(a) Repository: adena-wallet

(b) Assessed Commit ID: 38f089a

(c) Items in scope:

Out-of-Scope:

REMEDIATION COMMIT ID:

- 8b1e068
- 24c1f57
- 5a20b39
- dfe10ab
- https://github.com/onbloc/onbloc-api-v2/pull/23/files
- SOLVED:
- a6b3f9e
- a71efaf
- 2638084
- 3afc414
- 268ea08
- 23e8771

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	2	8	3	0

IMPACT X LIKELIHOOD

		HAL-07		
		HAL-05		
	HAL-11	HAL-01 HAL-02 HAL-09 HAL-03 HAL-12 HAL-04	HAL-10	
	HAL-06 HAL-13			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
WALLET CONFIGURATION OVERWRITE VIA DIRECT ACCESS TO THE REGISTER PAGE	LOW	SOLVED - 02/04/2025
MNEMONIC PHRASE EXPOSURE IN MEMORY	HIGH	RISK ACCEPTED - 02/05/2025
REUSE OF STATIC ENCRYPTION KEY	HIGH	SOLVED - 02/04/2025
VULNERABLE THIRD-PARTY DEPENDENCIES	MEDIUM	SOLVED - 03/03/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF RATE LIMITING	MEDIUM	SOLVED - 03/03/2025
DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS	LOW	SOLVED - 02/04/2025
GRAPHQL INTROSPECTION ENABLED	LOW	SOLVED - 03/03/2025
EVENT LISTENER ACCEPTS MESSAGES FROM ANY ORIGIN	MEDIUM	SOLVED - 02/04/2025
MESSAGES SENT TO ANY ORIGIN	MEDIUM	SOLVED - 02/04/2025
LACK OF BUSINESS LOGIC VALIDATION	MEDIUM	SOLVED - 02/05/2025
POTENTIAL RISK OF SENSITIVE DATA EXPOSURE THROUGH CLIPBOARD	MEDIUM	SOLVED - 02/04/2025
LACK OF WALLET AUTO-LOCK	MEDIUM	SOLVED - 02/05/2025
UNRESTRICTIVE CONTENT-SECURITY-POLICY (CSP)	MEDIUM	RISK ACCEPTED - 02/04/2025

7. FINDINGS & TECH DETAILS

7.1 WALLET CONFIGURATION OVERWRITE VIA DIRECT ACCESS TO THE REGISTER PAGE

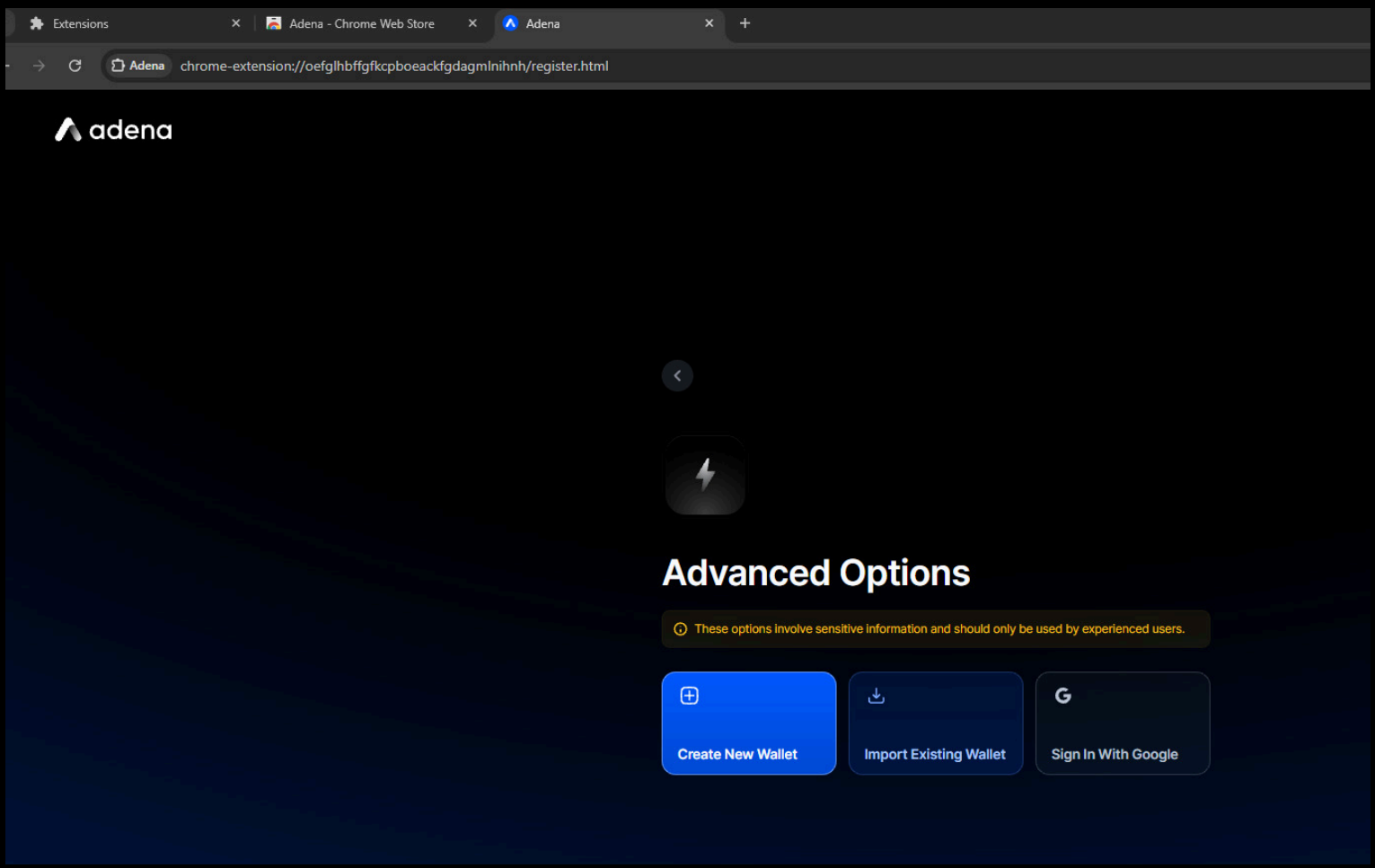
// LOW

Description

When a user directly accesses the registration page, he can overwrite the existing configuration, compromising the integrity of the wallet's setup and security.

Proof of Concept

The wallet extension allows a user to directly access the registration page. This allows users to overwrite the wallet's configuration.
By navigating to `chrome-extension://oefglhbffgfkcpboeackfgdagmlnihnh/register.html`, an attacker can overwrite the wallet's settings without any authentication or authorization:



Score

Impact: 3

Likelihood: 2

Recommendation

Implement proper access control checks to prevent access to the registration page after the wallet has been configured. Ensure that only legitimate users can modify sensitive configuration settings.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

`8b1e0681a44767263a7a20f1472eefa80554412a`

7.2 MNEMONIC PHRASE EXPOSURE IN MEMORY

// HIGH

Description

The mnemonic phrase of the wallet is kept unencrypted in memory, even the wallet was locked. As a result, an attacker with access to the user's machine could exfiltrate the mnemonic phrase. This makes the wallet web extension vulnerable to Demonic (CVE-2022-32969). The user password in cleartext was also found. It is important to recognize that the mnemonic risk extends beyond the application state; it could also be leaked into memory when the browser displays the mnemonic in clear text and as long as the process running.

Proof of Concept

Observe, for example, the following lines in https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-module/src/wallet/keyring/hd-wallet-keyring.ts:

```
15  export class HDWalletKeyring implements Keyring {
16    public readonly id: string;
17    public readonly type: KeyringType = 'HD_WALLET';
18    public readonly seed: Uint8Array;
19    public readonly mnemonic: string;
20
21    constructor({ id, mnemonic, seed }: KeyringData) {
22      if (!mnemonic || !seed) {
23        throw new Error('Invalid parameter values');
24      }
25      this.id = id || uuidv4();
26      this.mnemonic = mnemonic;
27      this.seed = Uint8Array.from(seed);
28    }
```

In this example, the mnemonic phrase may be maintained in cleart text in the memory.

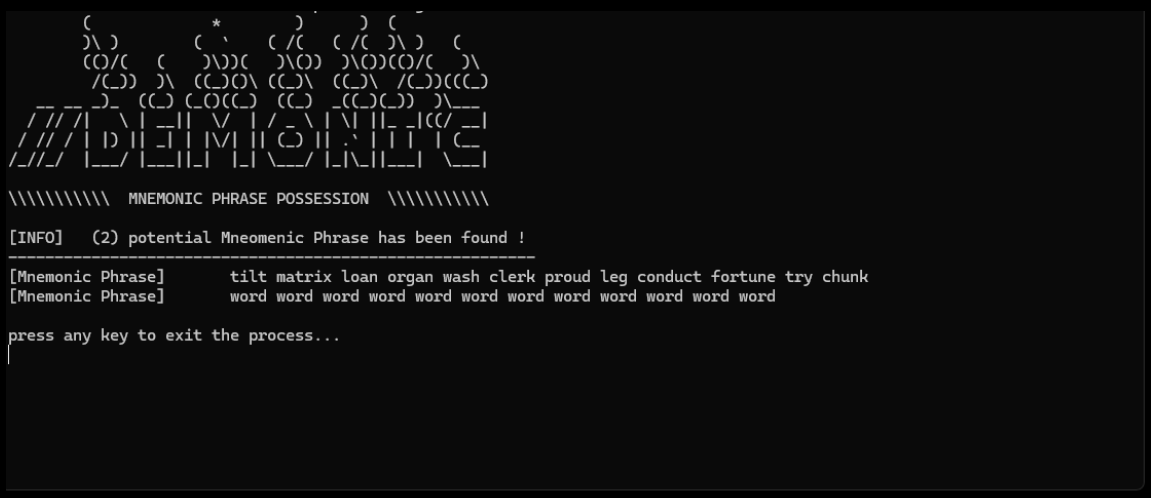
Observe, for example, the following lines in https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-extension/src/repositories/wallet/wallet.ts:

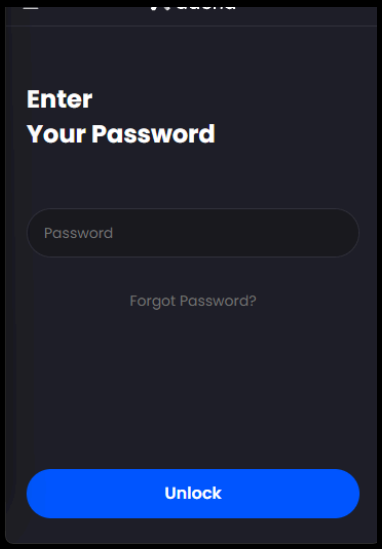
```
60    public getWalletPassword = async (): Promise<string> => {
61      const encryptedKey = await this.sessionStorage.get('ENCRYPTED_KEY');
62      const encryptedPassword = await this.sessionStorage.get('ENCRYPTED_PA
```

```
63
64      if (encryptedKey === '' || encryptedPassword === '') {
65        throw new WalletError('NOT_FOUND_PASSWORD');
66      }
67
68      try {
69        const password = decryptPassword(encryptedKey, encryptedPassword);
70        this.updateStoragePassword(password);
71        return password;
72      } catch (e) {
73        throw new WalletError('NOT_FOUND_PASSWORD');
74      }
75    };
```

In this example, the user password may be maintained in cleart text in the memory.

In the following images, evidences of this situation can be seen when the wallet is in locked state:





```
"1737634536132",
"1737634546992",
"(code for e.init)",
"1737634546950",
"1737634546918",
"1737634546949",
"1737634546903",
"1737634546904",
"1737634546917",
"Halborn-123!",
"1737634546902",
"sc-rPWID",
"sc-dChVcU",
"0.0139981465968328 64",
"9437207",
"13633559",
"chainId=",
"9a69ce76-06df-44c4-ac61-c2cb8580d92
"tilt matrix loan organ wash clerk p
"12584965",
"sc-bypJrT.",
```

```
"13633559",
"chainId=",
"9a69ce76-06df-44c4-ac61-c2cb8580d920",
"tilt matrix loan organ wash clerk proud leg conduct fortune try chunk",
"12584965",
"sc-bypJrT.",
"fcpPUT",
"3483.2000000178814",
"1737634536222",
```

Score

Impact: 5

Likelihood: 3

Recommendation

The identified vulnerability arises from the application's handling of sensitive data in plain text. To mitigate this, the team recommends the following strategies:

- Opt for storing the entropy on disk rather than the mnemonic itself. When the mnemonic is necessary in the code, consider breaking it into multiple variables. Alternatively, obfuscate the original phrase and subsequently dereference the variable holding the original phrase.
- For instances requiring mnemonic phrase handling, utilize the obfuscated variable with a function designed to reconstruct the original mnemonic phrase exactly at the point of need.
- Ensure that when the wallet is in a locked state, the mnemonic phrase is completely cleared from memory.

For the display and handling of the mnemonic phrase during wallet creation and when revealed to a logged-in user:

- Display the mnemonic phrase using an HTML5 canvas. This technique helps prevent users from copying the phrase, reducing the risk of it being unintentionally stored in memory via the clipboard.

- Limit the ability for users to copy the entire mnemonic from the extension. This approach is essential in minimizing the potential for the mnemonic to be accidentally leaked through the clipboard, thereby enhancing the security of the sensitive information.
Implementing these recommendations would significantly enhance the security of mnemonic phrase handling, reducing the risks associated with its exposure or misuse.

Remediation

RISK ACCEPTED: The **Onbloc team** applied some measures to solve this finding. However, the mnemonic phrase was found in memory when it was just revealed to the user and the password was also found in memory in cleartext when the wallet was in unlocked state.

Remediation Hash

`24c1f575c64ba35ac9eb7d91fc08addf11e9f58d`

7.3 REUSE OF STATIC ENCRYPTION KEY

// HIGH

Description

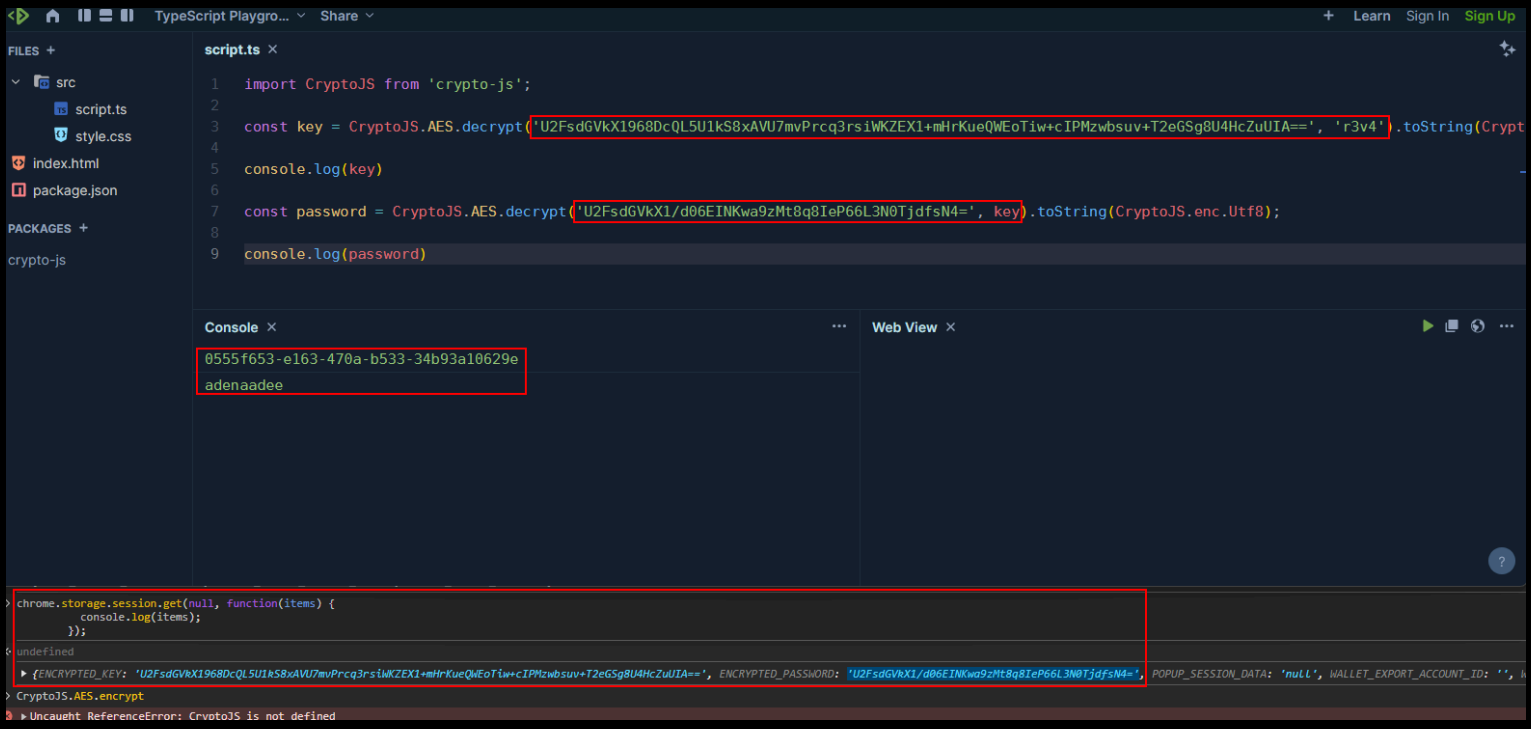
The use of a hardcoded static encryption key to encrypt a dynamically generated UUID, which is then used to encrypt the wallet password, has been detected.

Proof of Concept

```
import CryptoJS from 'crypto-js';
import { v4 as uuidv4 } from 'uuid';

// Static cipher key used for encrypting the cryptographic key
const ENCRYPT_CIPHER_KEY = 'r3v4';

export const encryptSha256Password = (password: string): string => {
  return CryptoJS.SHA256(password).toString();
};

// Encrypts a password with a dynamically generated key and returns the e
export const encryptPassword = (
  password: string,
): { encryptedKey: string; encryptedPassword: string } => {
  const cryptKey = uuidv4();
  const adenaKey = ENCRYPT_CIPHER_KEY;
  const encryptedKey = CryptoJS.AES.encrypt(cryptKey, adenaKey).toString(
  const encryptedPassword = CryptoJS.AES.encrypt(password, cryptKey).toSt
  return {
    encryptedKey,
    encryptedPassword,
  };
};

// Decrypts a password using the encrypted key and password
export const decryptPassword = (encryptedKey: string, encryptedPassword:
  const adenaKey = ENCRYPT_CIPHER_KEY;
  const key = CryptoJS.AES.decrypt(encryptedKey, adenaKey).toString(Crypt
  if (key === '') {
    throw new Error('CipherKey Decryption Failed');
  }
  const password = CryptoJS.AES.decrypt(encryptedPassword, key).toString(
```

```
33    if (password === '') {
34      throw new Error('Password Decryption Failed');
35    }
36    return password;
37  };
```

The constant key (`r3v4`) used to encrypt the dynamically generated UUID is hardcoded. Anyone with access to the source code (which is public) can obtain the cipher key that allows to decrypt the UUID. The encrypted password and encrypted key is then stored in the web extension session storage. If a malicious actor is able to access the session storage of the web extension, they can retrieve both the encrypted password and the encrypted key. With the static cipher key `r3v4`, they can decrypt the UUID and then use that to decrypt the password, ultimately gaining access to the user's wallet password in plaintext:



Additionally, using a UUID as the key for AES encryption is not cryptographically secure. While UUIDs are unique, they are not generated with cryptographic secure randomness. A better practice would be to use a secure random key generation method (e.g., using the Web Crypto API's `crypto.getRandomValues()`) for cryptographic keys.

Score

Impact: 5

Likelihood: 3

Recommendation

Avoid static keys when possible. Storing sensitive data like keys or passwords in session storage is risky, using only in-memory storage can be considered. Also consider using a stronger encryption scheme, such as AES-GCM, instead of AES-CBC (the default for `CryptoJS.AES`).

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

5a20b3900f9d5d2637e465f0006faab7f2d99171

7.4 VULNERABLE THIRD-PARTY DEPENDENCIES

// MEDIUM

Description

The scoped repository uses multiple third-party dependencies. Using vulnerable third-party libraries can result in security vulnerabilities in the project that can be exploited by attackers. This can result in data breaches, theft of sensitive information, and other security issues. However, some of them were affected by public-known vulnerabilities that may pose a risk to the global application security level.

Proof of Concept

Using the command `npm audit`, observe the public vulnerabilities of the third-party dependencies of the packages in the repository **adena-wallet**:

- https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-extension/package.json
- https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-module/package.json
- https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-torus-signin/package.json

```
        Depends on vulnerable versions of @walletconnect/socket-transport
        Depends on vulnerable versions of @walletconnect/utils
        node_modules/@walletconnect/core
        node_modules/@walletconnect/sign-client/node_modules/@walletconnect/core
          @walletconnect/client  ≤1.8.0
          Depends on vulnerable versions of @walletconnect/core
          node_modules/@walletconnect/client
        @walletconnect/sign-client  ≥2.17.1-canary-0
        Depends on vulnerable versions of @walletconnect/core
        Depends on vulnerable versions of @walletconnect/utils
        node_modules/@walletconnect/sign-client

semver  ≤5.7.1
Severity: high
Regular Expression Denial of Service in semver - https://github.com/advisories/GHSA-x6fg-f45m-jf5q
semver vulnerable to Regular Expression Denial of Service - https://github.com/advisories/GHSA-c2qf-rxjj-qqgw
fix available via `npm audit fix --force`
Will install rollup-plugin-node-builtins@2.0.0, which is a breaking change
node_modules/levelup/node_modules/semver

ws  7.0.0 - 7.5.9
Severity: high
ws affected by a DoS when handling a request with many HTTP headers - https://github.com/advisories/GHSA-3h5v-q93c-6h6q
No fix available
node_modules/@walletconnect/socket-transport/node_modules/ws
  @walletconnect/socket-transport  *
  Depends on vulnerable versions of ws
  node_modules/@walletconnect/socket-transport

18 vulnerabilities (8 low, 2 moderate, 8 high)

To address issues that do not require attention, run:
  npm audit fix

To address all issues possible (including breaking changes), run:
  npm audit fix --force

Some issues need review, and may require choosing
a different dependency.
```

Score

Impact: 4

Likelihood: 3

Recommendation

Update all affected packages to its latest version.
It is strongly 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

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

dfe10ab2d24cd277a1de75ec054ec27b940e54b1

7.5 LACK OF RATE LIMITING

// MEDIUM

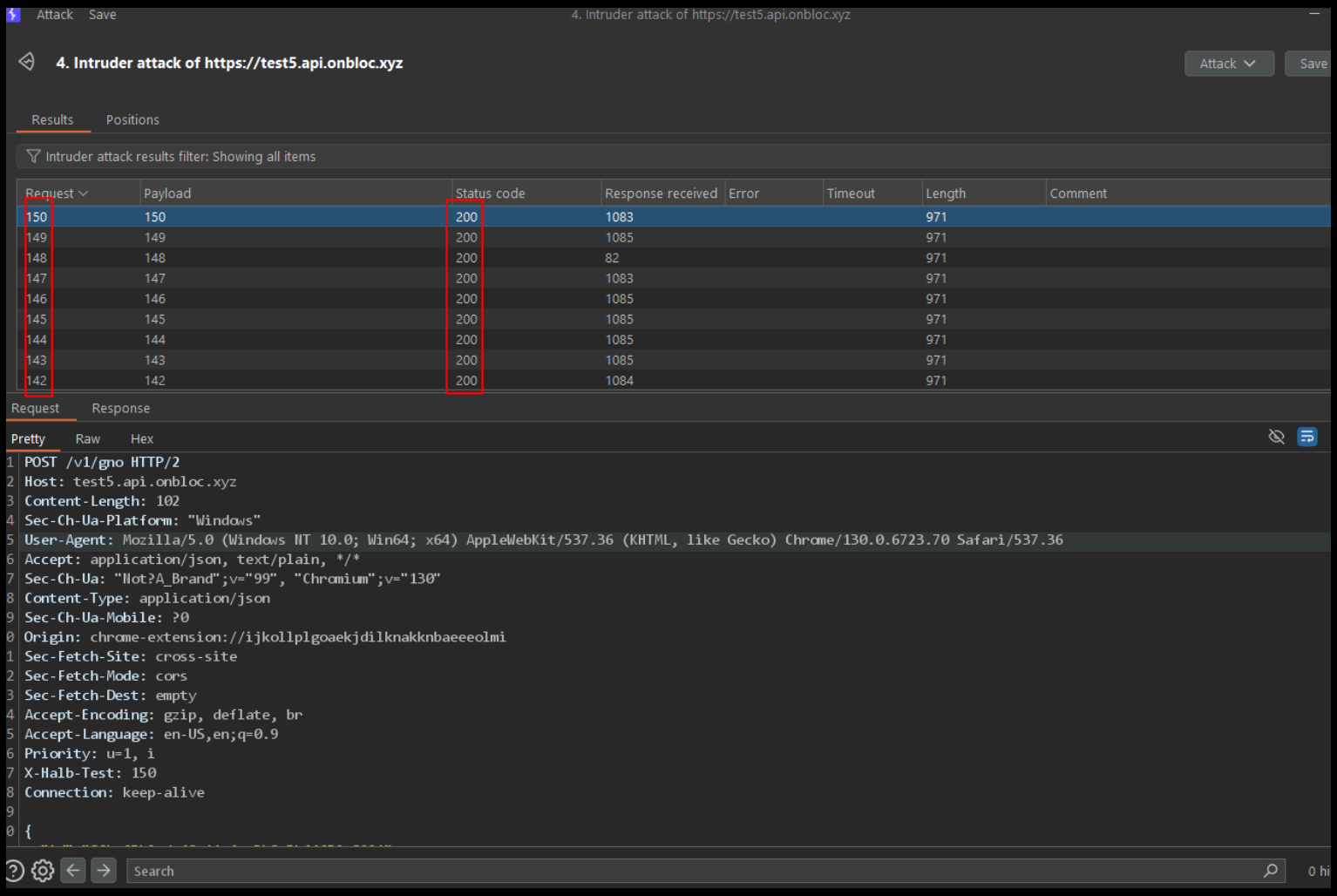
Description

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 policy 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, or resources of bandwidth and CPU usage could be abused as well.

Proof of Concept

During the assessment, some example interesting endpoints where found allowing multiple parallel requests (**POST** HTTP request to **https://test5.api.onbloc.xyz/v1/gno**):



Score

Impact: 3

Likelihood: 4

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

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

https://github.com/onbloc/onbloc-api-v2/pull/23/files

7.6 DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS

// LOW

Description

The application contained multiple dependencies that were not pinned to an exact version, but they were set to a supported version (x̂.x.x). This could potentially allow dependency attacks.

Proof of Concept

Observe the versioning of the third-party dependencies of the packages in the repository **adena-wallet**:

- https://github.com/onbloc/adena-wallet/blob/38f089a9003110c773a3c6c6ae5807ef6db2bebb/packages/adena-extension/package.json
- https://github.com/onbloc/adena-wallet/blob/38f089a9003110c773a3c6c6ae5807ef6db2bebb/packages/adena-module/package.json
- https://github.com/onbloc/adena-wallet/blob/38f089a9003110c773a3c6c6ae5807ef6db2bebb/packages/adena-torus-signin/package.json

Dependencies of the **adena-extension** package:

```
77    "dependencies": {
78      "@adena-wallet/sdk": "^0.0.2",
79      "@bufbuild/protobuf": "^2.2.3",
80      "@gnolang/gno-js-client": "1.3.1",
81      "@gnolang/tm2-js-client": "1.2.3",
82      "@tanstack/react-query": "^4.36.1",
83      "@vespaiach/axios-fetch-adapter": "^0.3.1",
84      "adena-module": "*",
85      "adena-torus-signin": "*",
86      "axios": "0.27.2",
87      "bignumber.js": "^9.1.2",
88      "crypto-js": "^4.2.0",
89      "dayjs": "^1.11.10",
90      "html-loader": "^5.0.0",
91      "lottie-web": "5.10.2",
92      "node-polyfill-webpack-plugin": "^4.0.0",
93      "qrcode.react": "^3.1.0",
94      "react": "^18.2.0",
95      "react-dom": "^18.2.0",
96      "react-router-dom": "^6.22.1",
97      "recoil": "^0.7.7",
98      "styled-components": "^5.3.11",
```

```
    "uuidv4": "^6.2.13",
    "zxcvbn": "^4.4.2"
  }
```

Score

Impact: 2

Likelihood: 2

Recommendation

The repository dependencies in the `package.json` files should be pinned to exact versions to prevent dependency attacks.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

SOLVED: the Onbloc team solved this vulnerability.

7.7 GRAPHQL INTROSPECTION ENABLED

// LOW

Description

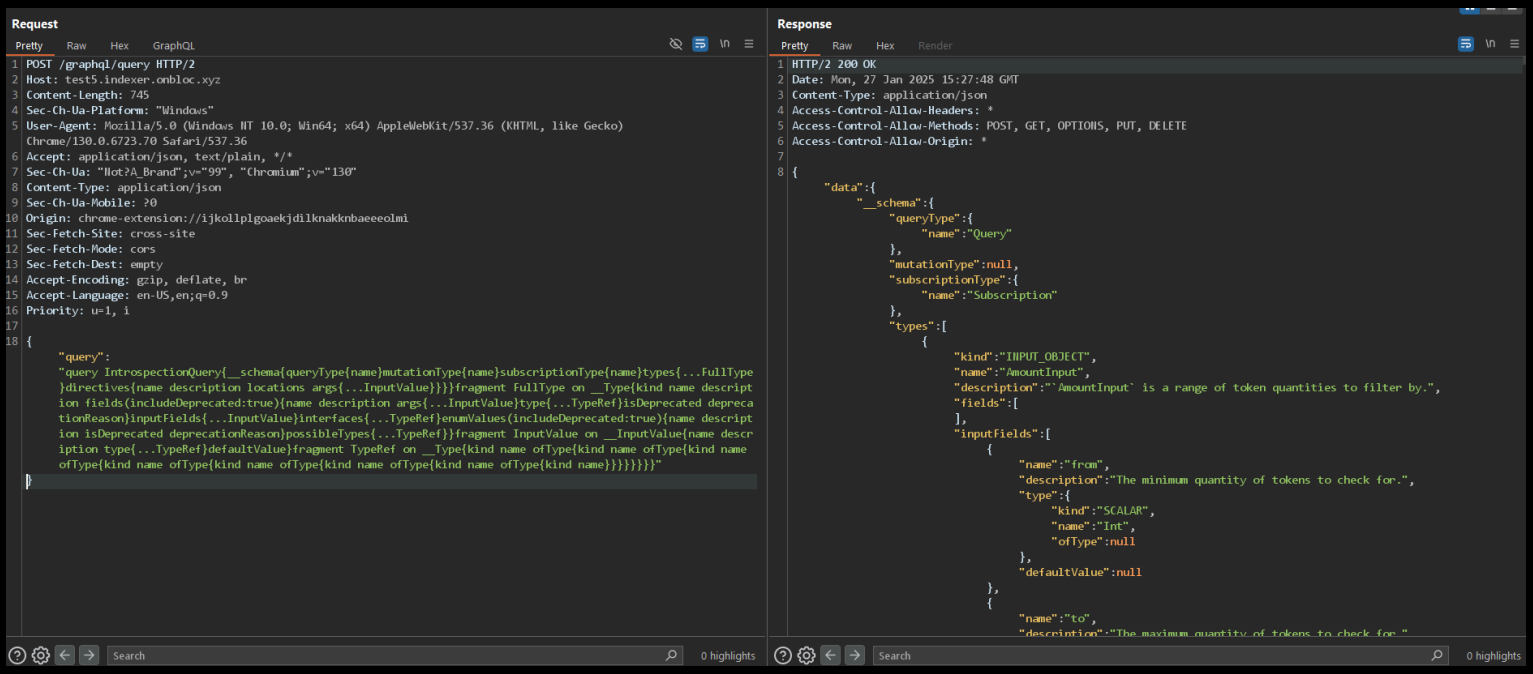
During the assessment, we discovered that GraphQL introspection was enabled, which exposes the API schema and structure. This information could be exploited by attackers to gain insights into your backend systems, craft malicious queries or mutations, and potentially access sensitive data or perform unintended actions.

An attacker has complete insight of the backend queries and can further exploit these weaknesses, allowing them to run queries and mutations on the database without requiring administrative privileges. This chained attack significantly increases the potential impact and risk of unauthorized data access, data manipulation, and further exploitation of your application and infrastructure.

Allowing introspection queries also possibly permits an attacker to launch several introspection queries in parallel possibly causing the server to consume a lot of resources and maybe leading into a Denial-of-Service.

Proof of Concept

In the following image, the GraphQL introspection query can be seen:



Score

Impact: 2

Likelihood: 2

Recommendation

It is recommended to make sure that there is no risk to the application and to mitigate the risks associated with GraphQL introspection in production environments, it is recommended the following actions:

- Disable GraphQL introspection in the production environment by configuring your GraphQL server appropriately.
- Implement proper access controls to ensure that only authorized users and developers can access the API schema.

Remediation

SOLVED: The **Onbloc team** solved this finding.

7.8 EVENT LISTENER ACCEPTS MESSAGES FROM ANY ORIGIN

// MEDIUM

Description

Accepting events from any origin in an event listener can lead to security vulnerabilities such as unauthorized actions, event spoofing, and exposure to malicious websites, possibly compromising the integrity of sensitive operations.

Proof of Concept

Observe the following lines in https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-extension/src/content.ts, where an event listener is initialized without any checks to the origin of the events:

```
23 const initListener = (): void => {
24   window.addEventListener(
25     'message',
26     (event) => {
27       try {
28         if (event.data?.status === 'request') {
29           sendMessage(event);
30         } else {
31           return event.data;
32         }
33       } catch (e) {
34         console.error(e);
35       }
36     },
37     false,
38   );
39 };
```

This event listener accepts events from any source, as the Content script is injected on every URL as the Manifest establish:

```
17   "content_scripts": [
18     {
19       "matches": ["<all_urls>"],
20       "js": ["content.js"]
21
22
```

```
    }
  ],
```

Score

Impact: 3

Likelihood: 3

Recommendation

Restrict the sources that can send events to the event listener in the Content script, if possible. Note that this behaviour may be intended.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

a6b3f9e75df0ababe426e20f5f72571f71ecddde

7.9 MESSAGES SENT TO ANY ORIGIN

// MEDIUM

Description

Using `window.postMessage(this.eventMessage, '*')` in a crypto wallet extension can be a security risk, as it allows messages to be sent to any origin, potentially exposing sensitive data to malicious actors or untrusted domains.

Proof of Concept

Observe the following lines in https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-extension/src/inject/executor/executor.ts, the messages are sent to any origin:

```
162   private sendEventMessage = <T = unknown>(
163     eventMessage: InjectionMessage,
164   ): Promise<WalletResponse<T>> => {
165     this.listen();
166     this.eventMessage = {
167       ...eventMessage,
168       protocol: window.location.protocol,
169       hostname: window.location.hostname,
170       key: this.eventKey,
171     };
172     window.postMessage(this.eventMessage, '*');
173     this.messages[this.eventKey] = {
174       request: this.eventMessage,
175       response: undefined,
176     };
```

Score

Impact: 3

Likelihood: 3

Recommendation

To prevent sensitive data from being exposed to untrusted sources, trusted origins can be specified in `window.postMessage` instead of using `'*'`. Note that this behavior may be intended.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

`a71efaff7a6b3b7594b9f7db7aee7a1a9beb131a`

7.10 LACK OF BUSINESS LOGIC VALIDATION

// MEDIUM

Description

It was possible to add bogus or malicious tokens to the user's balance by manipulating the server response. The wallet allowed users to add tokens either manually or by searching through a list of supported tokens. Although manual addition of malicious tokens was prevented due to input validation checks, the vulnerability lies in the handling of server responses for the supported token list.
By intercepting and modifying the server response to a POST request sent to `https://test5.api.onbloc.xyz/v1/gno` (triggered during the "Search" functionality under "Add Custom Token" menus), it was possible to inject and display unsupported or malicious tokens in the token search results. These tokens could then be selected and added by the user, bypassing the application's intended validation mechanisms and business logic.

Impact

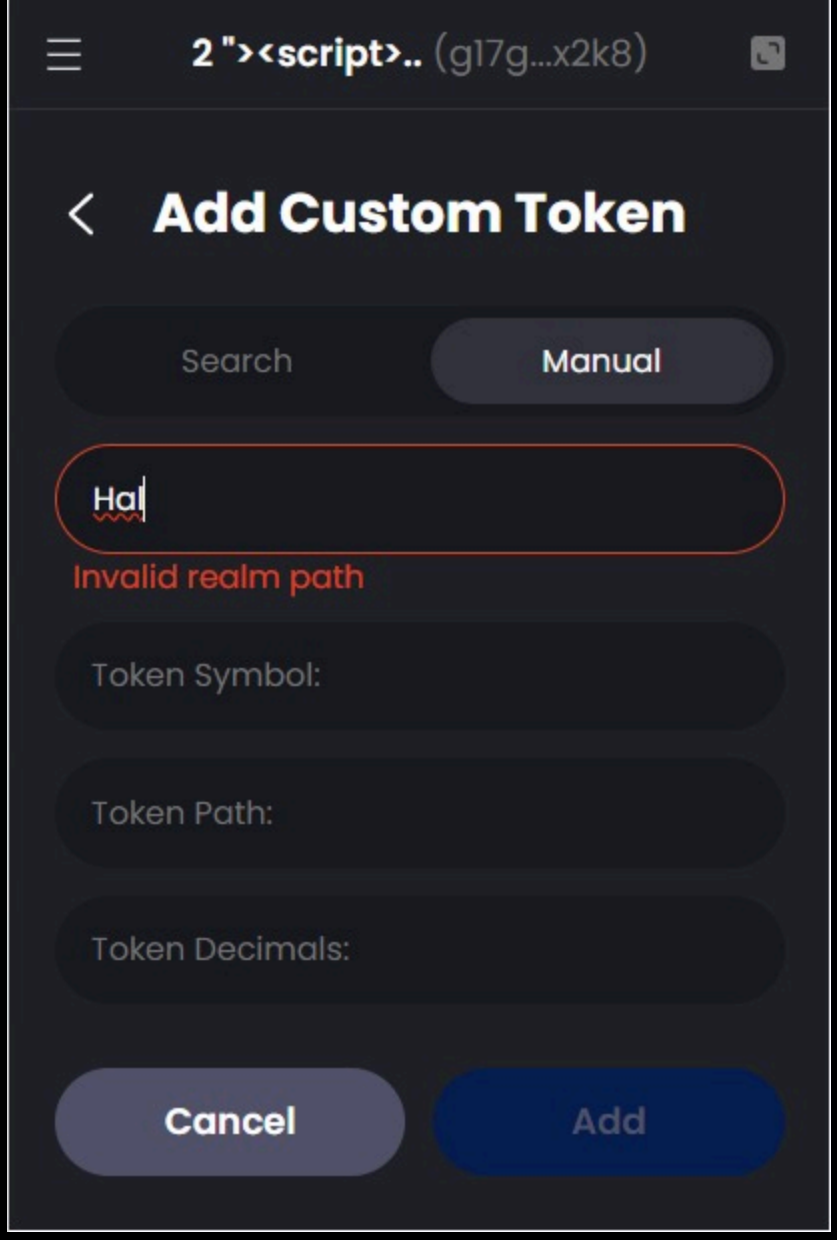
This vulnerability breaks the integrity of the business logic and execution workflows within the wallet, leading to unintended or unexpected behaviors. Specifically:

- **Workflow Disruption**: The wallet's logic is designed to prevent the addition of unsupported or malicious tokens. Manipulating the server response allows bypassing this workflow, rendering the wallet's security checks ineffective.
- **Integrity Violations**: The application operates on the assumption that only validated tokens can be added by users. By injecting malicious or bogus tokens, attackers can compromise this assumption, potentially destabilizing dependent operations like balance calculations or token transfers.
- **Application Misbehavior**: Fraudulent tokens with improper metadata (e.g., invalid contract addresses or incompatible formats) may lead to application crashes, errors, or undefined behaviors during transactions or token management processes.
- **Unintended Data States**: The wallet may process data in unintended ways, such as incorrect balance updates or failed transactions, due to the introduction of unsupported tokens. This breaks the expected state of the application and creates potential confusion or mistrust among users.
- **Exploitation of Ecosystem Logic**: These unintended behaviors could be further leveraged by attackers to exploit other components of the wallet or related systems (e.g., exploiting token-related vulnerabilities or breaking synchronization between the wallet and external services).

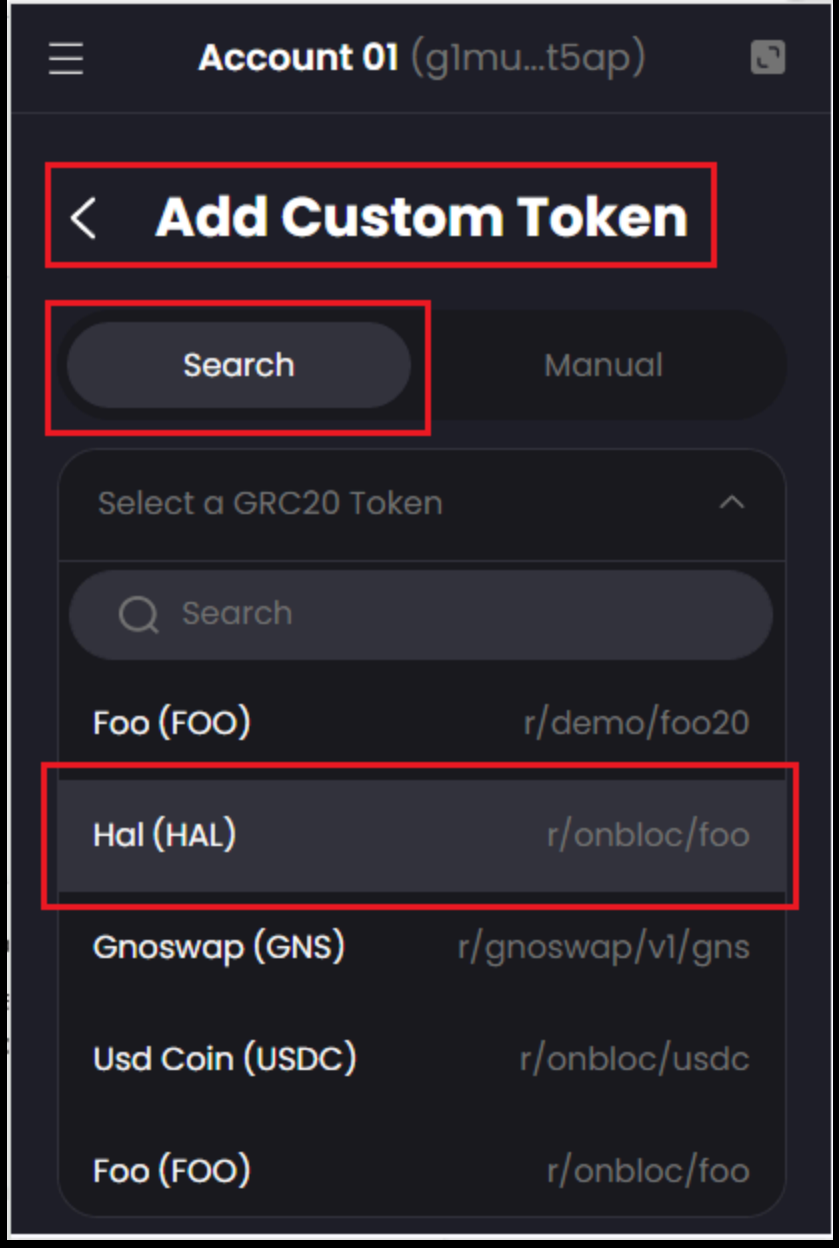
By undermining the wallet's business logic, this vulnerability introduces systemic risks that compromise both the integrity and reliability of the application's core workflows.

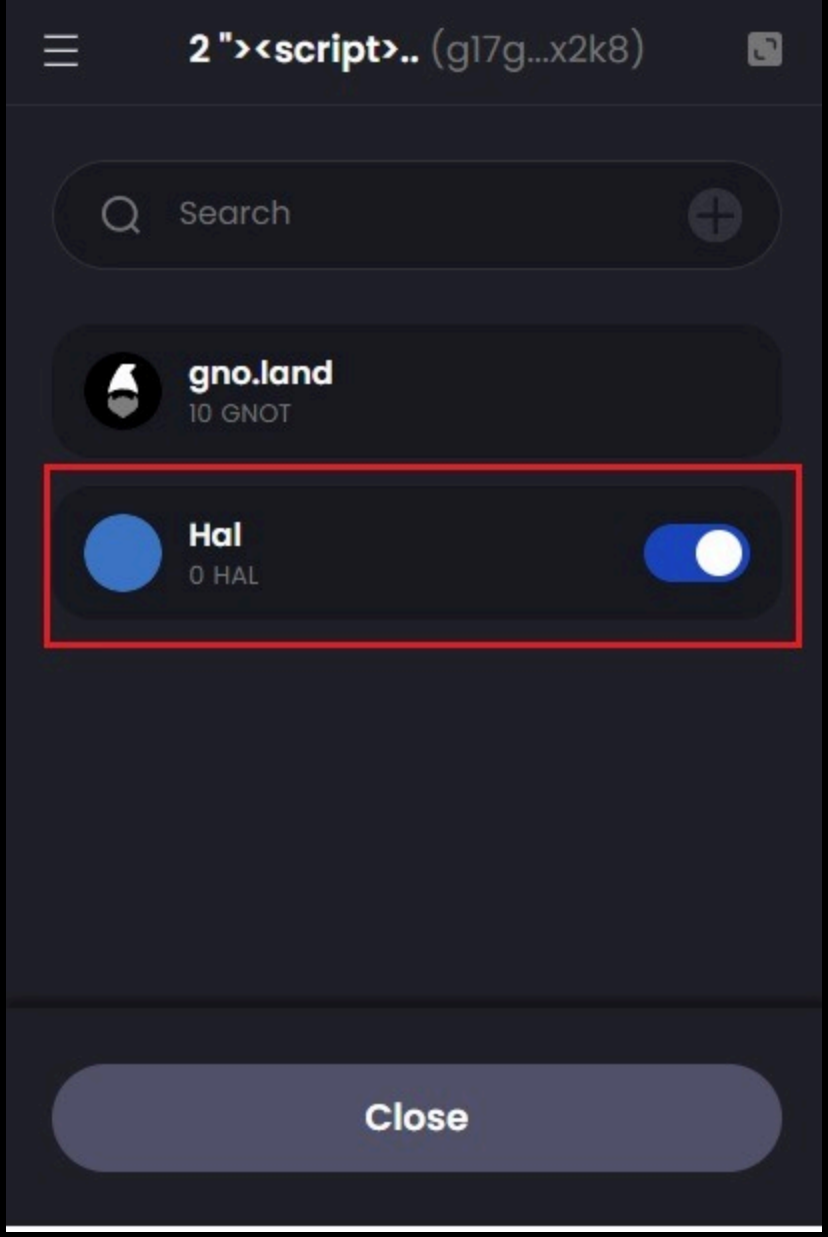
Proof of Concept

From the "Manual" section of "Add Custom Token" menu was not possible to add a custom token that was not existing.



However, if the server response was manipulated, it was possible to add custom tokens that were not actually existing, as the following picture shows.


Account 01 (g1mu...t5ap)
Add Custom Token
Search
Manual
Select a GRC20 Token
Search
Foo (FOO) r/demo/foo20
Hal (HAL) r/onbloc/foo
Gnoswap (GNS) r/gnoswap/v1/gns
Usd Coin (USDC) r/onbloc/usdc
Foo (FOO) r/onbloc/foo



As an example, it was possible to manipulate with Match & Replace The proxy tool was configured with the next Match & Replace rules:

- Match text:

```
{"name":"Foo","owner":"g1u7y667z64x2h7vc6fmpcprgey4ck233jaww9zq","symbol":"FOO","packagePath":"gno.land/r/demo/foo20","decimals":4}
```

- Replace:

```
{"name":"Foo","owner":"g1u7y667z64x2h7vc6fmpcprgey4ck233jaww9zq","symbol":"FOO","packagePath":"gno.land/r/demo/foo20","decimals":4},
{"name":"Hal","owner":"g17gprk6rymnnlayhfhx437xq7z48p26dxtux2k8","symbol":"HAL","packagePath":"bydp841zm2en4rcayczmsj063x9oxfl4.oastify.com","decimals":6,"packagePath":"gno.land/r/onbloc/foo"}
```

The following request and responses were affected by the rules above.
Request:

```
POST /v1/gno HTTP/2
Host: test5.api.onbloc.xyz
Content-Length: 99
Sec-Ch-Ua-Platform: "Windows"
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Accept: application/json, text/plain, */*
Sec-Ch-Ua: "Not A(Brand";v="8", "Chromium";v="132", "Google Chrome";v="13
Content-Type: application/json
Sec-Ch-Ua-Mobile: ?0
Origin:chrome-extension://pilbblmedmdhgjjomdopdlealdjminno
Sec-Fetch-Site: cross-site
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Accept-Encoding: gzip, deflate, br
Accept-Language: es-ES,es;q=0.9
Priority: u=1, i

{"id":"5b146960-d9bf-11ef-94d7-8b187d58a657","jsonrpc":"2.0","method":"ge
```

Response:

- Original Response:

```
HTTP/2 200 OK
Date: Thu, 23 Jan 2025 19:22:18 GMT
Content-Type: application/json
Content-Length: 1476
Access-Control-Allow-Headers: Accept, Content-Type, Content-Length, Accep
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: X-AUTH-KEY, X-AUTH-DATA

{"result":[{"name":"Foo","owner":"g1u7y667z64x2h7vc6fmpcprgey4ck233jaww9z
```

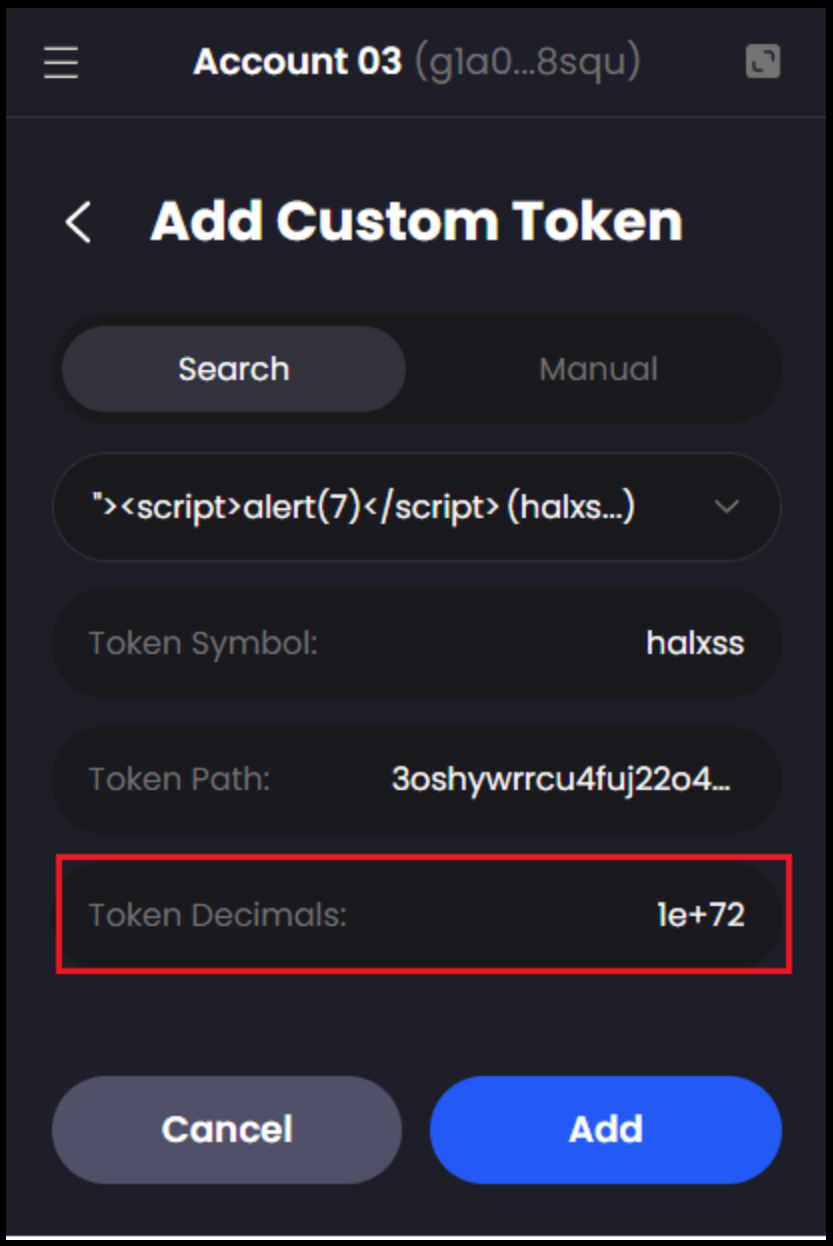
- Auto-modified Response:

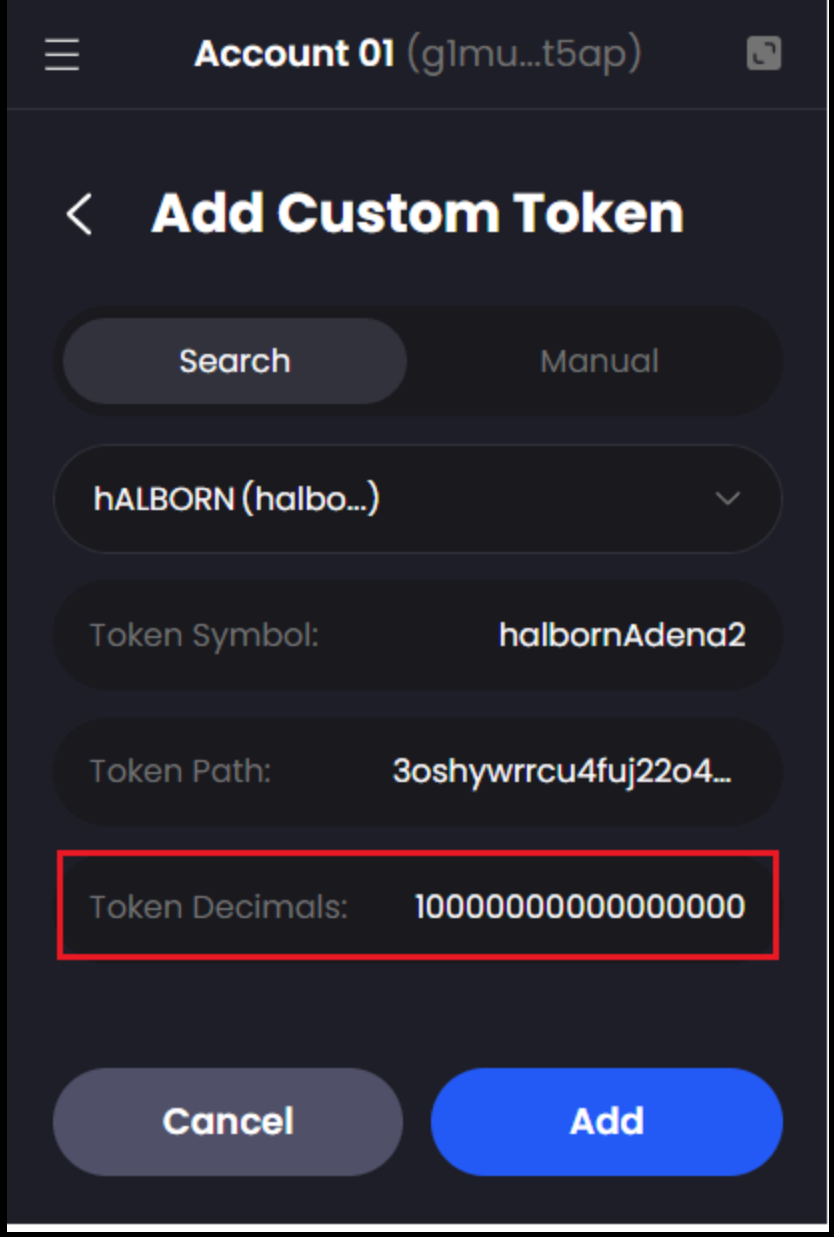
```
HTTP/2 200 OK
Date: Thu, 23 Jan 2025 19:22:18 GMT
Content-Type: application/json
Content-Length: 1669
Access-Control-Allow-Headers: Accept, Content-Type, Content-Length, Accep
Access-Control-Allow-Methods: POST, GET, OPTIONS, PUT, DELETE
```

```
7  Access-Control-Allow-Origin: *
8  Access-Control-Expose-Headers: X-AUTH-KEY, X-AUTH-DATA
9
10 {"result":[{"name":"Foo","owner":"g1u7y667z64x2h7vc6fmpcprgey4ck233jaww9z
```

That way, the Front-End of the wallet application started showing new bogus tokens to be added from the

Other than that, it was possible to add tokens to the User's wallet with malicious `Token Names` and weird values for the `Token Decimals`, as shown in the following pictures:





Score

Impact: 3

Likelihood: 3

Recommendation

- **Server-Side Validation**: Ensure the server-side API validates all tokens that are about to be added to the user wallet, warrantying that the token features match the expected standards, like Token Decimals, Token Symbol length and characters allowed, etc...
- **Response Tamper Detection**: Utilize cryptographic methods (e.g., digital signatures) to validate the integrity of server responses.
- **Log and Monitor**: Log attempts to manipulate server responses and monitor unusual behavior for early detection of potential exploitation.
- **User Alerts**: Warn users about the risks of adding tokens not verified by the wallet, and provide clear visual cues for unverified tokens.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

2638084db90297b23067ce5e02805f1692459a3d

7.11 POTENTIAL RISK OF SENSITIVE DATA EXPOSURE THROUGH CLIPBOARD

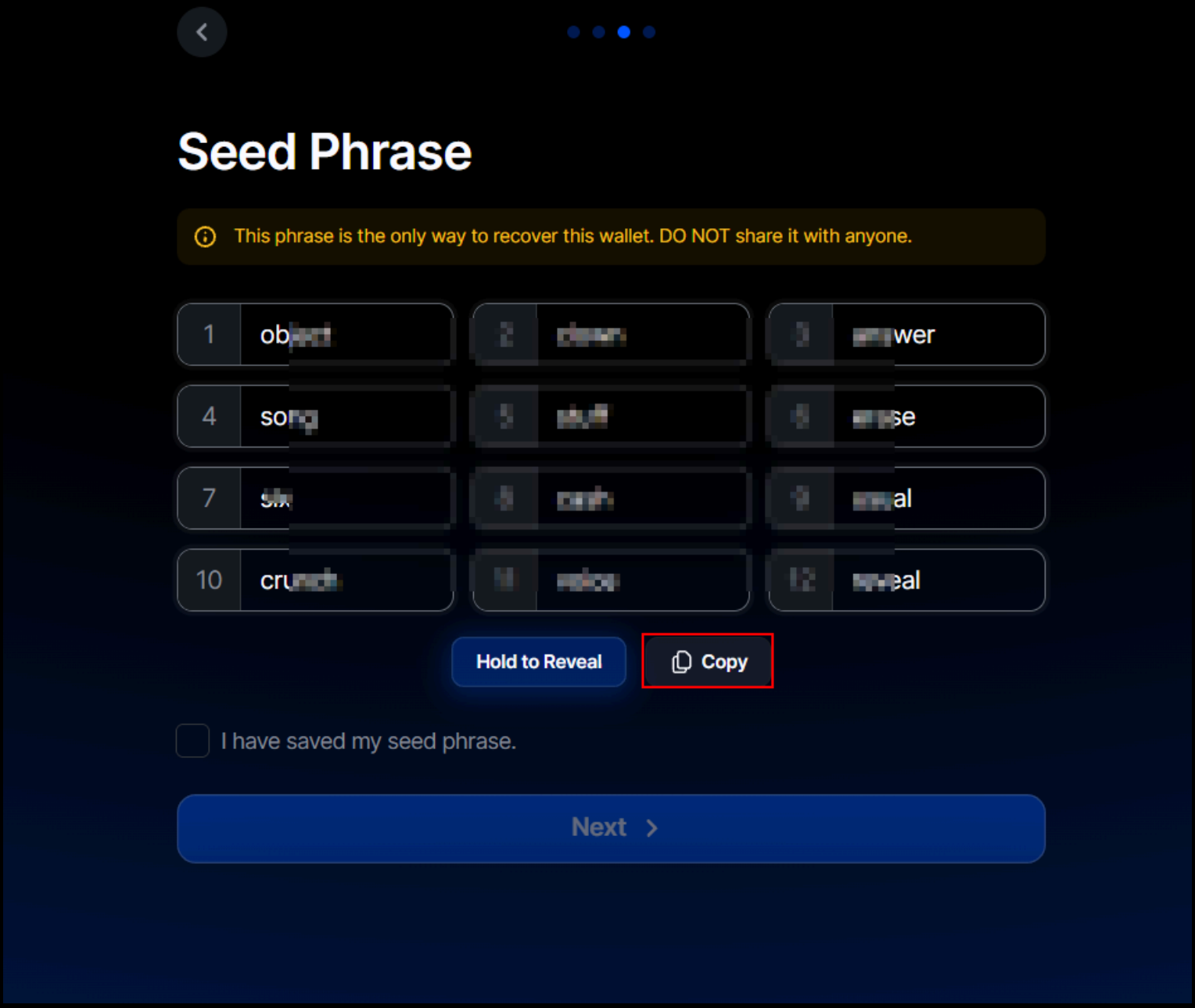
// MEDIUM

Description

The wallet application facilitates copying sensitive data, specifically mnemonic and private key passphrases, to the clipboard. This functionality presents a significant security risk, as clipboard data could be accessed both locally and remotely by unauthorized processes or malicious web pages. Attackers can exploit this vulnerability by leveraging scripts or pages designed to capture clipboard content, thereby compromising the confidentiality of critical information.
Furthermore, in multi-device environments where clipboard sharing is enabled (such as between smartphones, tablets, and laptops), sensitive data copied to the clipboard can be inadvertently exposed across multiple devices. This significantly increases the attack surface, as an attacker gaining access to any linked device can retrieve the copied mnemonic or private key.

Proof of Concept

Look at the following example, in which the mnemonic phrase can be copied to the clipboard through the wallet:



Score

Impact: 3

Likelihood: 3

Recommendation

It is recommended to avoid providing a direct copy-to-clipboard feature for sensitive information, such as mnemonic and private key passphrases. Instead, consider implementing a secure display mechanism that requires users to manually input or write down the passphrase. If clipboard functionality is essential for user experience, implement automatic clipboard clearing after a short period to reduce exposure time. Additionally, display a warning when sensitive information is copied, advising users to clear their clipboard and avoid copying data in shared or multi-device environments where clipboard data can be synchronized across devices. Another option is to allow users to copy almost all the information necessary but not the

complete secret, this way, the secret is not copied entirely to the clipboard. These measures will significantly reduce the risk of unauthorized access to sensitive information.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

3afc414b2a63ce9ddc72f53b5360fc6b31467b7c

7.12 LACK OF WALLET AUTO-LOCK

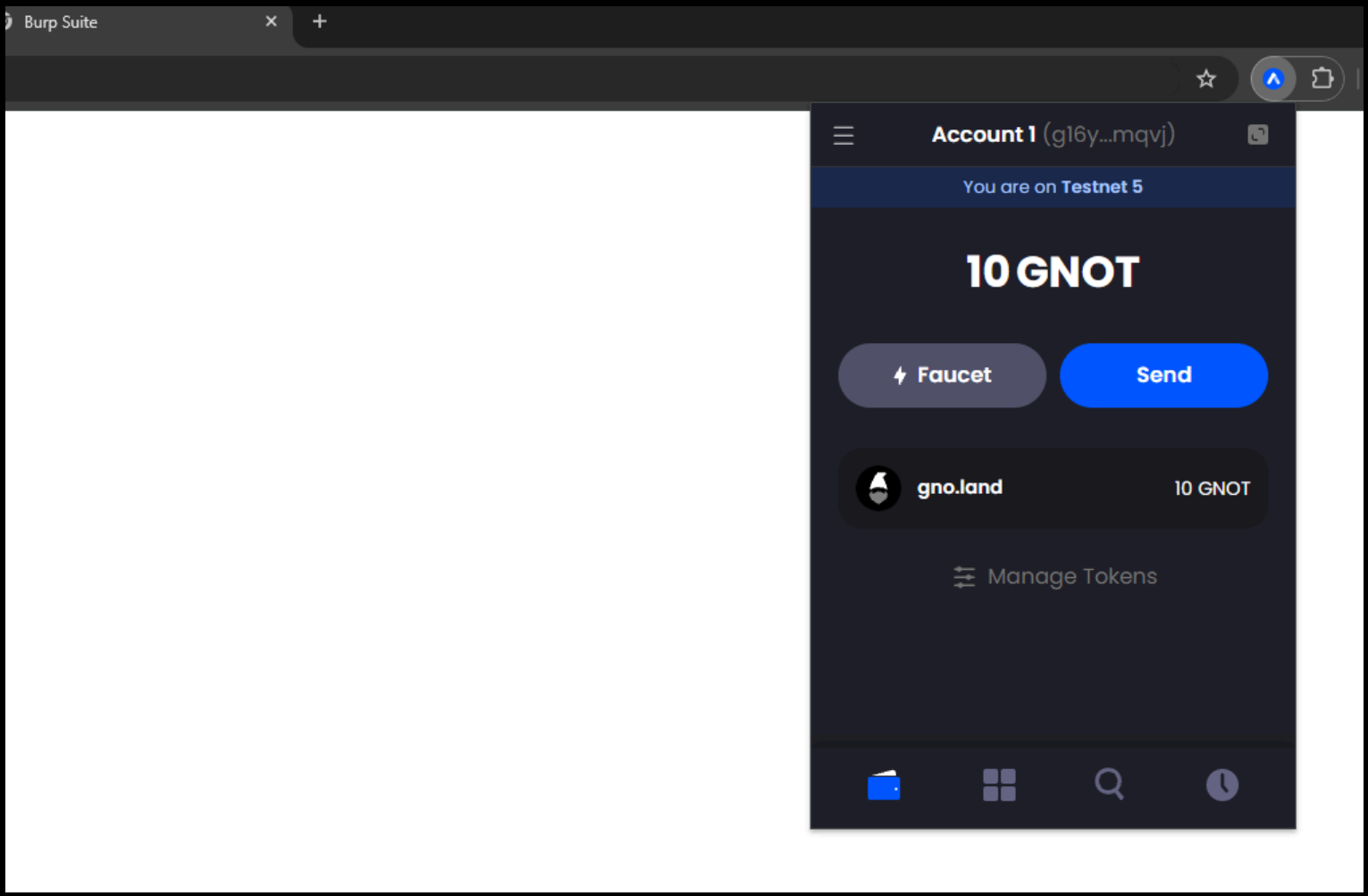
// MEDIUM

Description

The wallet does not implement an auto-lock feature, allowing long access to the wallet without reauthentication, which can lead to unauthorized usage if the user leaves their session unattended.

Proof of Concept

The wallet does not seem to automatically lock itself after a period of inactivity, leaving it accessible indefinitely. This increases the risk of unauthorized access. After logging into the wallet, leaving the session idle does not trigger any auto-lock mechanism, allowing anyone with access to the device to interact with the wallet without re-authentication:



Score

Impact: 3

Likelihood: 3

Recommendation

Implement an auto-lock feature that locks the wallet after a predefined period of inactivity, requiring the user to re-authenticate before regaining access. This will prevent unauthorized access during unattended sessions.

Remediation

SOLVED: The **Onbloc team** solved this finding.

Remediation Hash

268ea08065206fae6808a995ca80f27ce23c04b2

7.13 UNRESTRICTIVE CONTENT-SECURITY-POLICY (CSP)

// MEDIUM

Description

The wallet browser extension has a partially restrictive declared Content-Security-Policy (CSP) that could be improved to enhance security.

Proof of Concept

Observe the CSP declaration of the Adena Wallet web extension in https://github.com/onbloc/adena-wallet/blob/38f089a900311ec773a3c6c6ae5807ef6db2bebb/packages/adena-extension/public/manifest.json:

```
33    "content_security_policy": {
34      "extension_pages": "script-src 'self' 'wasm-unsafe-eval'; img-src 'se
35    }
```

Score

Impact: 3

Likelihood: 3

Recommendation

The wallet browser extension fails to define a `default-src` directive, which acts as a fallback policy ensuring that no malicious resources are loaded from untrusted sources that have not been explicitly declared. The absence of this fallback increases the risk of loading unintended content from unauthorized origins.
Additionally:

- Remove `unsafe-*` keywords from the CSP declaration, unless absolutely necessary:
  - `unsafe-inline`
  - `wasm-unsafe-eval`
- Limit `http://127.0.0.1:*` access in `connect-src` to specific trusted local ports or remove `localhost` access unless absolutely necessary.
- Avoid `data:` URLs for `img-src` and `font-src`.
- Restrict `frame-src` to trusted, known domains instead of broad `https:`.

Remediation

RISK ACCEPTED: The **Onbloc team** applied some measures to solve this finding. However, some `unsafe-*` keywords are still necessary due to the `libsodium` dependency.

Remediation Hash

23e877147700b53f4c8542b930eb2533b91593e1

References

https://mikewest.org/2011/10/secure-chrome-extensions-content-security-policy/

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.