



# Browser Extension Wallet Security Audit Report



# Table Of Contents

<b>1 Executive Summary</b>	_____
<b>2 Audit Methodology</b>	_____
<b>3 Project Overview</b>	_____
3.1 Project Introduction	_____
3.2 Vulnerability Information	_____
3.3 Vulnerability Summary	_____
<b>4 Audit Result</b>	_____
<b>5 Statement</b>	_____

# 1 Executive Summary

On 2022.02.28, the SlowMist security team received the Zecrey team's security audit application for zecrey chrome extension wallet, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "black/grey box lead, white box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.

Level	Description
Suggestion	There are better practices for coding or architecture.

## 2 Audit Methodology

The security audit process of SlowMist security team for browser extension wallet includes two steps:

The codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The browser extension wallets are manually analyzed to look for any potential issues.

The following is a list of security audit items considered during an audit:

- Transfer security
  - Signature security audit
  - Deposit/Transfer security audit
  - Transaction broadcast security audit
- Private key/Mnemonic phrase security
  - Private key/Mnemonic phrase generation security audit
  - Private key/Mnemonic phrase storage security audit
  - Private key/Mnemonic phrase usage security audit
  - Private Key/Mnemonic backup security audit
  - Private Key/Mnemonic destroy security audit
  - Random generator security audit
  - Cryptography security audit
- Web front-end security
  - Cross-Site Scripting security audit

- Third-party JS security audit
- HTTP response header security audit
- Communication security
  - Communication encryption security audit
  - Cross-domain transmission security audit
- Architecture and business logic security
  - Access control security audit
  - Wallet lock security audit
  - Business design security audit
  - Architecture design security audit
  - Denial of Service security audit

## 3 Project Overview

### 3.1 Project Introduction

#### Audit Version

<https://github.com/Zecrey-Labs/zecrey-chrome-extension/tree/audit>

commit: 97f4f99bbbf9b3e3da7072a9e22945f50a23b3d

#### Fixed Version

<https://github.com/Zecrey-Labs/zecrey-chrome-extension/tree/audit>

commit: 65672678ed92546eb350555de68a1d526fd1b1a5

### 3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Signature Reminders of Defects	Signature security audit	Low	Fixed
N2	Password verification can be bypassed	Private Key/Mnemonic backup security audit	Medium	Fixed
N3	Sensitive data is not cleared after reset	Private Key/Mnemonic destroy security audit	Low	Fixed
N4	PBKDF2 iterator value is small	Cryptography security audit	Low	Fixed
N5	Redundant configuration	Cross-Site Scripting security audit	Suggestion	Fixed
N6	Cached data is not cleared	Wallet lock security audit	Low	Fixed
N7	Paths in routes lack permission control	Access control security audit	Low	Fixed
N8	Password verification optimization	Wallet lock security audit	Suggestion	Fixed
N9	Autolock is missing	Wallet lock security audit	Suggestion	Fixed
N10	Architectural design flaws	Architecture design security audit	Medium	Fixed
N11	Error message optimization	Others	Suggestion	Confirmed

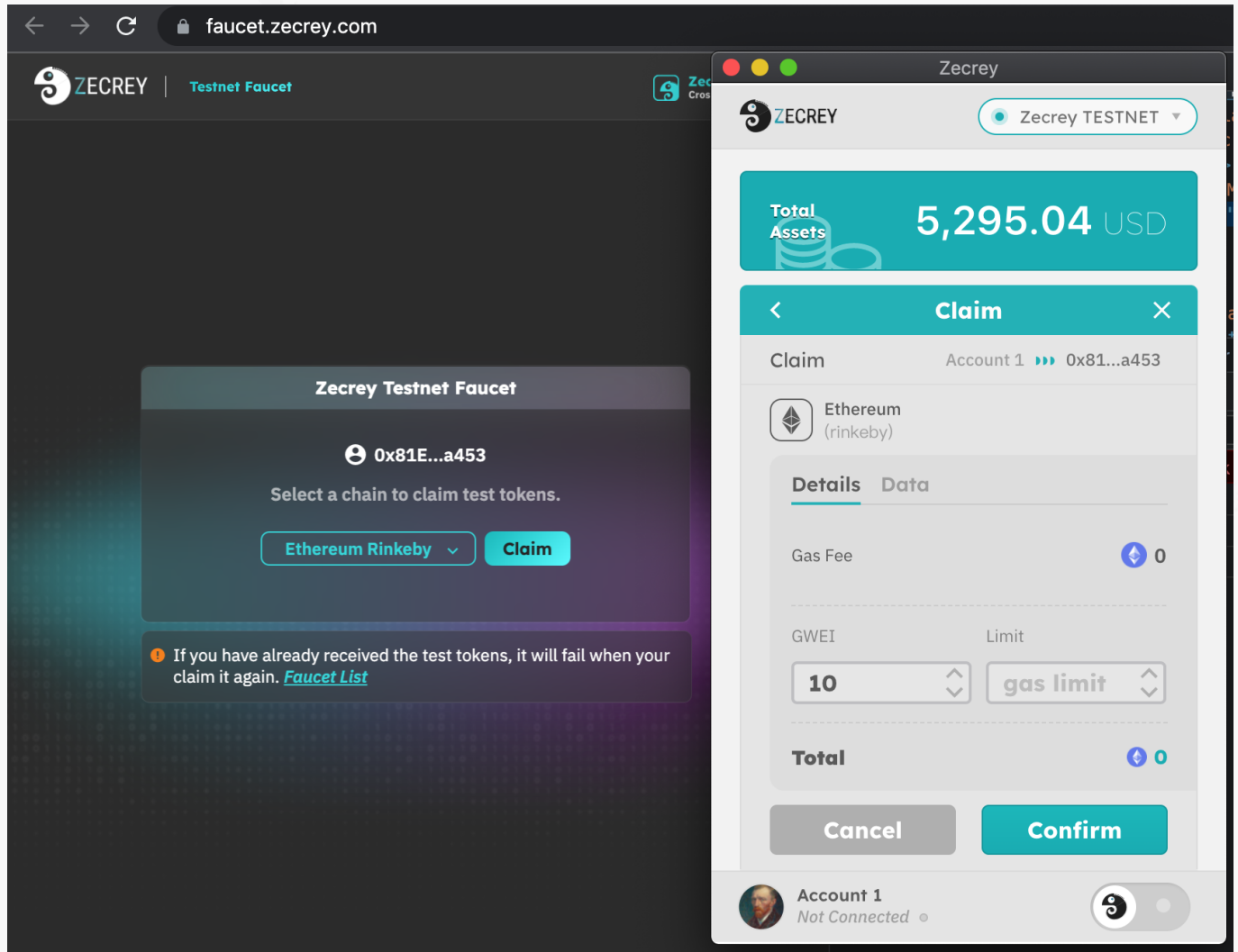
### 3.3 Vulnerability Summary

[N1] [Low] Signature Reminders of Defects

## Category: Signature security audit

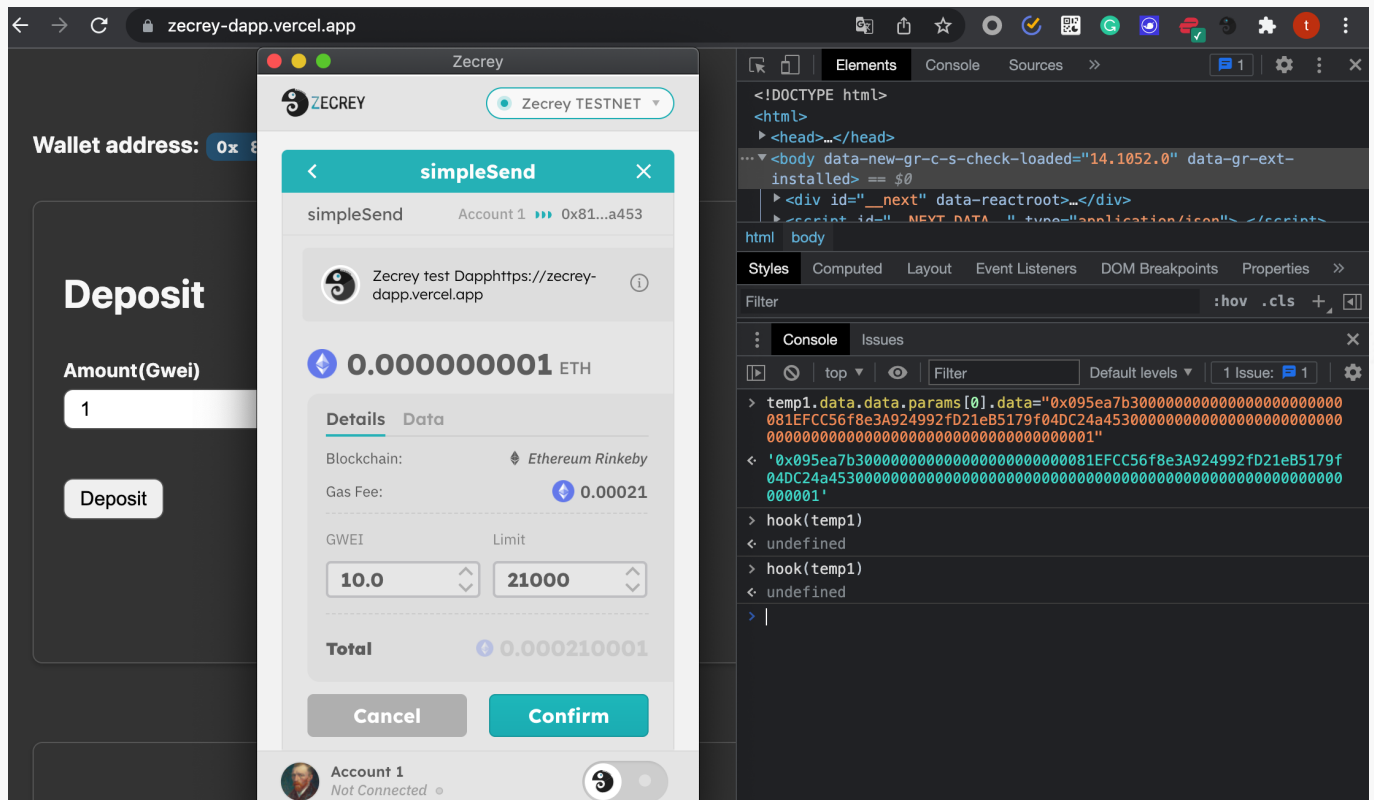
### Content

(1). The DApp domain address corresponding to the signed request is not displayed when using the zecrey testnet faucet.



(2). When signing, the type of transaction will be parsed, but the characteristics of evm are not taken into account.

When the length of the parameter is less than 64, evm will automatically complete the length of this parameter, which can bypass the check of transactions of types such as approve.



## Solution

- (1). It is recommended that the wallet should remind the DApp domain initiated by the signature when using the zecrey testnet faucet.
- (2). It is recommended to judge whether the data length of the data is consistent with the ABI format, and alert the data that cannot be parsed.

## Status

Fixed

## [N2] [Medium] Password verification can be bypassed

Category: Private Key/Mnemonic backup security audit

## Content

Password verification for exporting mnemonics and private keys is implemented in popup. The logic of judging whether the password is correct should be judged in the background. This issue can bypass the verification of the password only when it is unlocked.



For example, exporting the mnemonic and private key can bypass the verification.

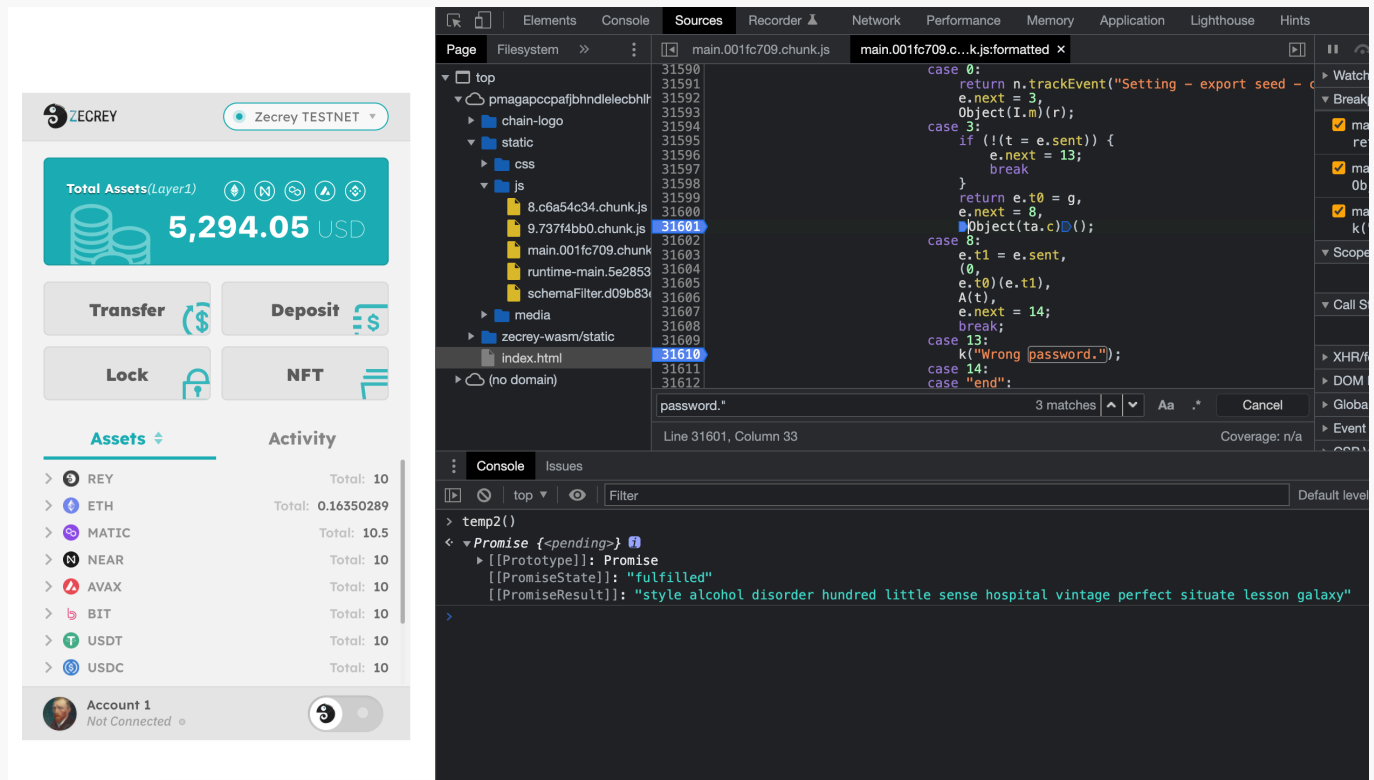
- [src/components/account-menu/ExportMnemonic.tsx#L241](#)

```
const submit = async () => {
  segment.trackEvent("Setting - export seed - confirm");
  let val = await isPasswordCorrect(pwd);
  if (val) {
    setMnemonic(await exportMnemonic());
    setMatched(val);
  } else {
    setError("Wrong password.");
  }
};
```

- [src/components/account-menu/ExportPrivateKey.tsx#L185](#)

```
const submit = async () => {
  segment.trackEvent("Setting - Export Secret key - Confirm");
  let val = await isPasswordCorrect(pwd);
  if (val) {
    setKey(await exportPrivateKey());
    setMatched(val);
  } else {
    setError("Wrong password.");
  }
};
```

The wallet can bypass the password verification when exporting the mnemonic or private key.



## Solution

It is recommended to return the mnemonic or private key after implementing password verification in the background.

## Status

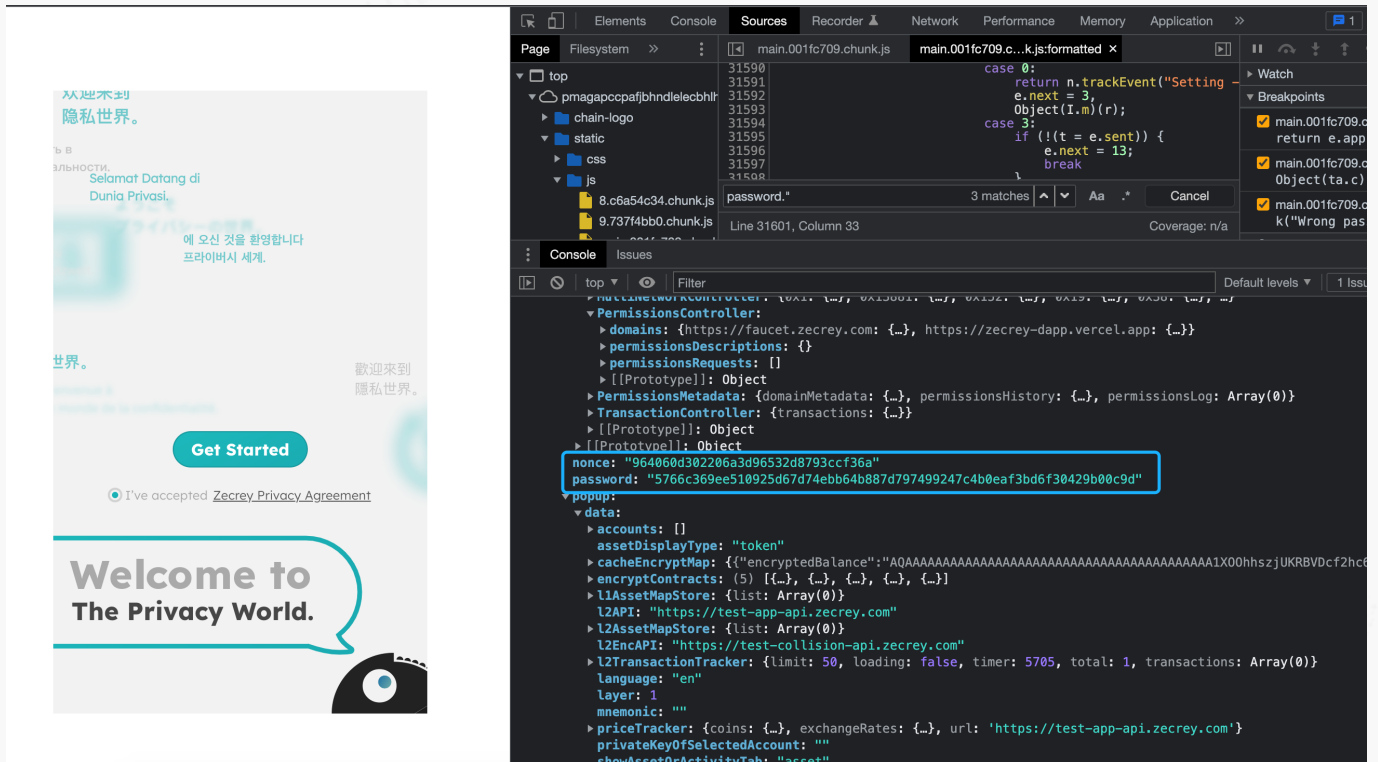
Fixed

**[N3] [Low] Sensitive data is not cleared after reset**

**Category: Private Key/Mnemonic destroy security audit**

## Content

After the wallet is reset, the nonce, password, and authorized domains are not cleared.



## Solution

It is recommended to delete sensitive data such as nonce, password, and authorized domains after resetting the wallet.

## Status

Fixed

## [N4] [Low] PBKDF2 iterator value is small

Category: Cryptography security audit

## Content

The value of the iterator is small and may be easily cracked by brute force.

- src/lib/utlis.ts#L99

```
const kdf = async (password: string): Promise<string> => {
  const salt = await getNonce();
  return Crypto.PBKDF2(password, salt, {
    keySize: 8,
    iterations: 1000,
```

```
    hasher: Crypto.algo.SHA256,  
  }).toString();  
};
```

## Solution

The number of iterator settings needs to be strengthened, the recommendation is PBKDF2-HMAC-SHA256: 310,000 iterations.

But the implementation of metamask uses 10000 iterations

Reference:

<https://github.com/MetaMask/browser-passworder/blob/main/src/index.ts#L121>

[https://cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html#pbkdf2](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#pbkdf2)

## Status

Fixed; The iterations has been modified to 10000.

## [N5] [Suggestion] Redundant configuration

**Category: Cross-Site Scripting security audit**

## Content

The manifest uses `file:///*/`, `http://localhost/`, `unsafe-eval`, and some configuration is not necessary.

- public/manifest.json

```
"content_scripts": [  
  {  
    "matches": ["file:///*/", "http:///*/", "https:///*/"],  
    "js": ["contentscript.js"],  
    "run_at": "document_start",  
    "all_frames": true  
  }  
],
```

```
"web_accessible_resources": ["inpage.js"],
"content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",
"permissions": [
  "tabs",
  "storage",
  "notifications",
  "https://min-api.cryptocompare.com/",
  "https://*.infura.io/",
  "https://*.etherscan.io/",
  "https://*.arbitrum.io/",
  "https://*.aurora.dev/",
  "https://*.maticvigil.com/",
  "https://*.chainstacklabs.com/",
  "https://data-seed-prebsc-1-s1.binance.org:8545/",
  "https://api.avax-test.network/",
  "https://*.zecrey.com/",
  "https://*.polygonscan.com/",
  "http://localhost/*",
  "https://sentry.zecrey.com/*"
]
```

## Solution

It is recommended to delete unnecessary configuration.

Reference:

[https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src#unsafe\\_eval\\_expressions](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src#unsafe_eval_expressions)

## Status

Fixed; The project team response:

Since wasm needs to use unsafe-eval, it will not be fixed for this time.

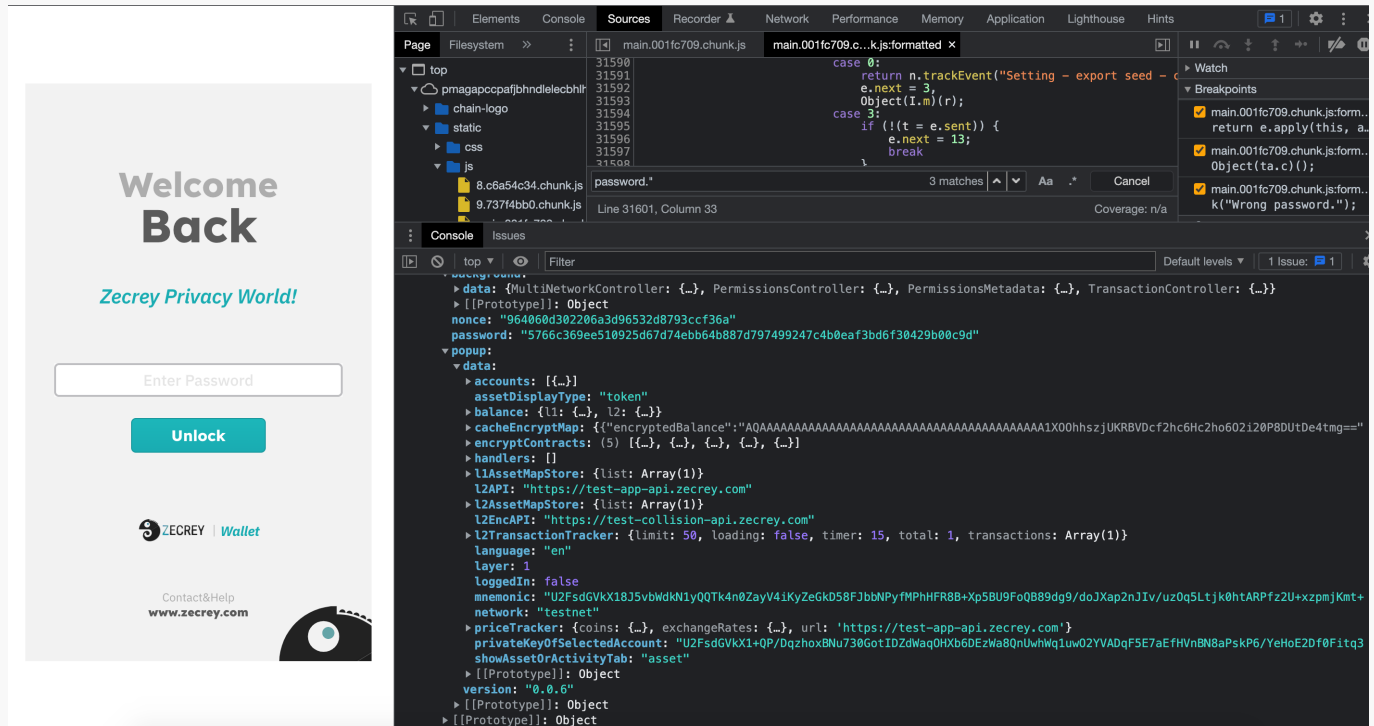
## [N6] [Low] Cached data is not cleared

**Category: Wallet lock security audit**

## Content

Encrypted password is cached in local storage and can be retrieved in locked state. If password and nonce are

obtained, which may be brute-forced



- src/lib/utlis.ts#L82-L93

```
export const sha256WithNonce = async (message: string): Promise<string> => {
  const nonce = await getNonce();
  const messageParsed = Crypto.enc.Utf8.parse(message);
  return Crypto.SHA256(nonce + messageParsed).toString();
};

export const savePasswordToLocalStorage = async (password: string) => {
  const sha256 = await sha256WithNonce(password);
  await ExtensionLocalStorage.setWithKey({
    password: sha256,
  });
};
```

## Solution

It is recommended to clear the password-related cached data after the wallet is locked.

## Status

Fixed

## [N7] [Low] Paths in routes lack permission control

**Category:** Access control security audit

### Content

When the wallet is locked, the path in the url can be modified to bypass the locked page, there is an issue of lack of permission control, However, the signature function will also pop up the unlock page. Since "this.password" has been cleared, it needs to be unlocked before the transfer.

- src/constant/routes.ts#L1-L12

```
export const DEFAULT_ROUTE = "/";
export const LOGIN_ROUTE = "/login";
export const ENCRYPT_ROUTE = "/encrypt";
export const L1_LOCK_ROUTE = "/lock";
export const L1_NFT_ROUTE = "/nft";
export const L2_UNLOCK_ROUTE = "/unlock";
export const L1_TRANSFER_ROUTE = "/l1/transfer";
export const L2_TRANSFER_ROUTE = "/l2/transfer";
export const L2_WITHDRAW_ROUTE = "/l2/withdraw";
export const L2_ATOM_SWAP_ROUTE = "/l2/atomswap";
export const L2_LIQUIDITY_ROUTE = "/l2/liquidity";
export const WEB3_CONNECT_ROUTE = "/web3/connect";
export const WEB3_SIGN_MESSAGE = "/web3/eth_sign";
export const WEB3_PERSONAL_SIGN_MESSAGE = "/web3/personal_sign";
export const WEB3_L1_Transaction = "/web3/transaction";
export const L1_CLAIM = "/l1/claim";
```

### Solution

It is recommended to judge the status of the wallet in each paths in routes, and only after the wallet is unlocked can the relevant pages be accessed.

### Status

Fixed

## [N8] [Suggestion] Password verification optimization

**Category: Wallet lock security audit****Content**

There is a optimization, the password should be sent to bg for verification, and the login status will be changed after the verification has passed.

- src/components/password-match/index.tsx#L30

```
const submit = async () => {
  const matched = await isPasswordCorrect(pwd);
  if (matched) {
    const chrome = getChrome();
    chrome.runtime.sendMessage(
      { type: "set_password", payload: pwd },
      (res: { payload: boolean }) => {
        if (res && res.payload) {
          store.updateLoggedIn(true);
        } else {
          setError("Failed to login.");
        }
      }
    );
  } else {
    setError("Incorrect password.");
  }
};
```

**Solution**

It is recommended to implement password verification logic in the background.

**Status**

Fixed

**[N9] [Suggestion] Autolock is missing****Category: Wallet lock security audit****Content**



No mechanism for automatic locking was found.

## Solution

It is recommended to add an automatic locking mechanism.

## Status

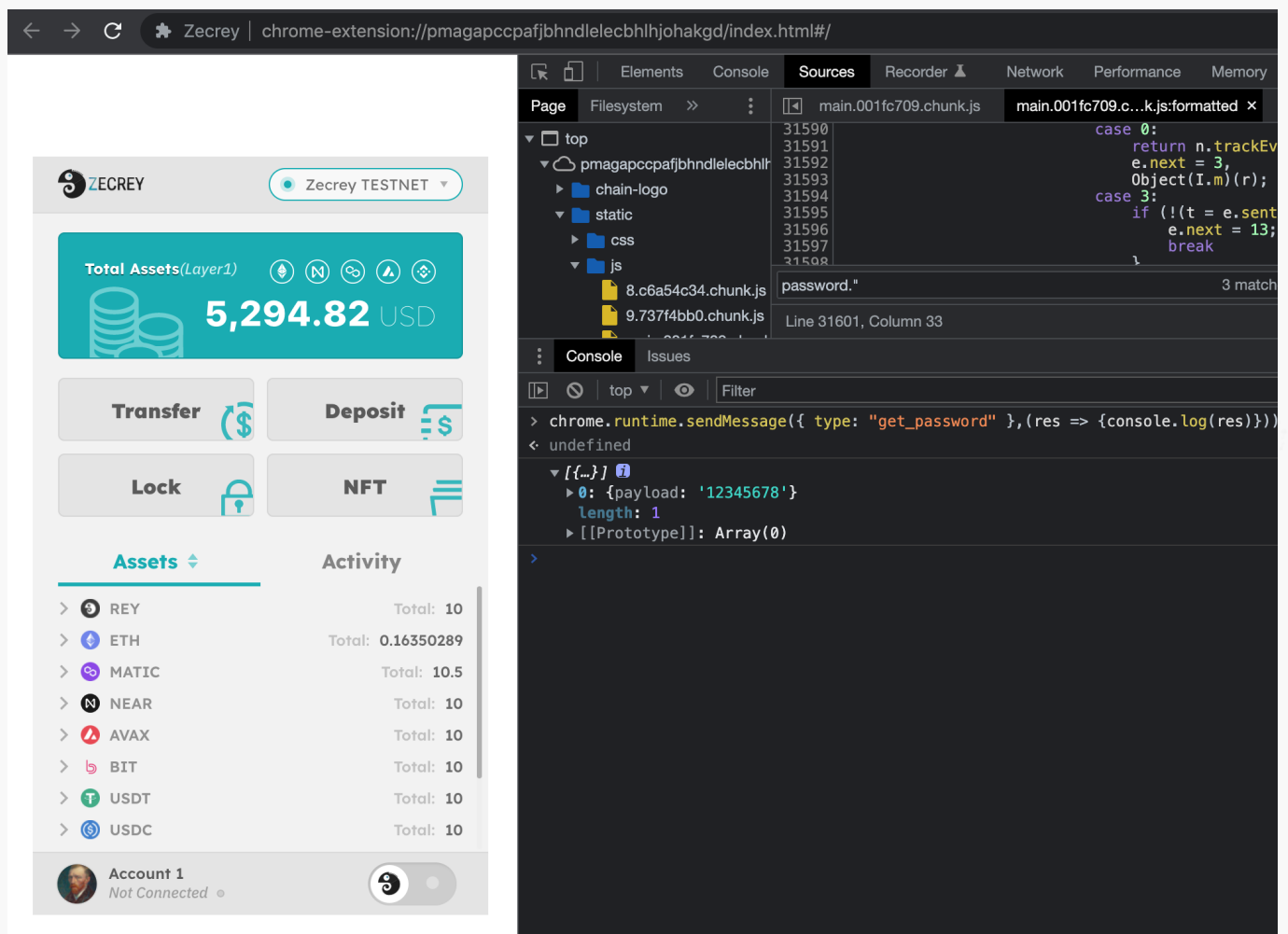
Fixed

## [N10] [Medium] Architectural design flaws

Category: Architecture design security audit

## Content

At the popup layer, users can use get\_password to get the plaintext password.



The screenshot displays the Zecrey web application interface on the left and the Chrome DevTools developer console on the right. The application shows a user's total assets as 5,294.82 USD and a list of assets including REY, ETH, MATIC, NEAR, AVAX, BIT, USDT, and USDC. The developer console shows a message from the browser's runtime, indicating that a 'get\_password' message was received from the extension. The message payload is '12345678', which is the plaintext password. The console also shows the message's length as 1 and its prototype as Array(0).

## Solution

It is recommended to store the password in the background variable, and then verify and obtain the password should be implemented in the background, not in the popup layer, and then judge.

## Status

Fixed

## [N11] [Suggestion] Error message optimization

Category: Others

## Content

When an error is reported, the prompt is MetaMask, which needs to be optimized.

The screenshot displays a web application interface on the left and a browser developer console on the right. The web application has a dark theme and includes sections for 'Deposit', 'Transfer token', 'Sign Message', and 'Sign results'. The 'Deposit' section shows a 'Wallet address' as '0x 81EF ... a453' and an 'Amount(Gwei)' input field with the value '1'. The 'Transfer token' section has a 'To' input field and an 'Amount(Gwei)' input field with the value '1'. The 'Sign Message' and 'Sign results' sections are currently empty. The browser developer console on the right shows the 'Elements' panel at the top, displaying the HTML structure of the page. Below it, the 'Console' panel shows several error messages. A red box highlights a specific error: 'MetaMask: Received invalid network parameters. Please report this bug. {chainId: '', networkVersion: ''}'. Other messages include 'MetaMask - RPC Error: Request of type 'wallet\_requestPermissions' already pending for origin https://zecrey-dapp.vercel.app. Please wait.' and an 'Uncaught (in promise)' error with a detailed message and stack trace.

## Solution

It is recommended to deal with the error message.

## Status

Confirmed

## 4 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
OX002203160003	SlowMist Security Team	2022.02.28 - 2022.03.16	Passed

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 2 medium risk, 5 low risk, 5 suggestion vulnerabilities. and a suggestion was confirmed; All the other issues were fixed.

## 5 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.



**Official Website**  
[www.slowmist.com](http://www.slowmist.com)



**E-mail**  
[team@slowmist.com](mailto:team@slowmist.com)



**Twitter**  
[@SlowMist\\_Team](https://twitter.com/SlowMist_Team)



**Github**  
<https://github.com/slowmist>