SALUS SECURITY

MAY 2025

CODE SECURITY ASSESSMENT

SIP

Overview

Project Name	SIP
File Name	SIP_audit_report_2025-06-06
Creator	Salus Security
Create Date	15 May 2025
Total days	10 Days
Receive Date	28 May 2025; 6 June 2025

Modify History

Version	Modify Date	Modifier	Modify Type	Modified Chapter	Modified Content
1	6 June 2025	SIP	M		

*Modified Type are categorized by **A** - ADDED **M** - MODIFIED **D** – DELETED

Copyright Statement

Table of Contents

Introduction

1 About Salus Security

At Salus Security, we are in the business of trust. We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve. In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

2 Assessment Scope

The penetration test coverage is assessed based on the scope of assets that need to be tested. This penetration test only covers the browser plug-in under the name of sip. Therefore, the sip plug-in has a high importance and becomes the main penetration assessment target. This report reflects the detailed security summary report related to the security of this browser plug-in.

3 Risk Summary Description

Overall risk level: Low

Description: Through actual security penetration testing, we found 6 security risks. These include unfiltered parameters, sensitive information memory leakage, and test account information leakage in the source code. Combining these vulnerabilities, it can be judged that these vulnerabilities may be exploited by hackers or criminals, and there are certain security risks.

4 Disclaimer

This report is considered by Salus Security to be private information; it is licensed to four under the terms of the project statement of work. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Salus Security.

Test Coverage Disclaimer
All activities undertaken by Salus Security in association with this project were performed in accordance with a statement of work and mutually agreed upon project plan. Security Penetration Testing projects are time-boxed and often reliant on the information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Salus Security uses automated testing techniques to test the controls and security properties of software rapidly. These techniques augment our penetration testing work, but each has its limitations. Their use is also limited by the time and resource constraints of a project.

Salus Security makes all effort but holds no responsibility for the findings of this penetration testing. Salus Security makes no judgments on the underlying business model or the individuals involved in the project.

5 Audit Risk details

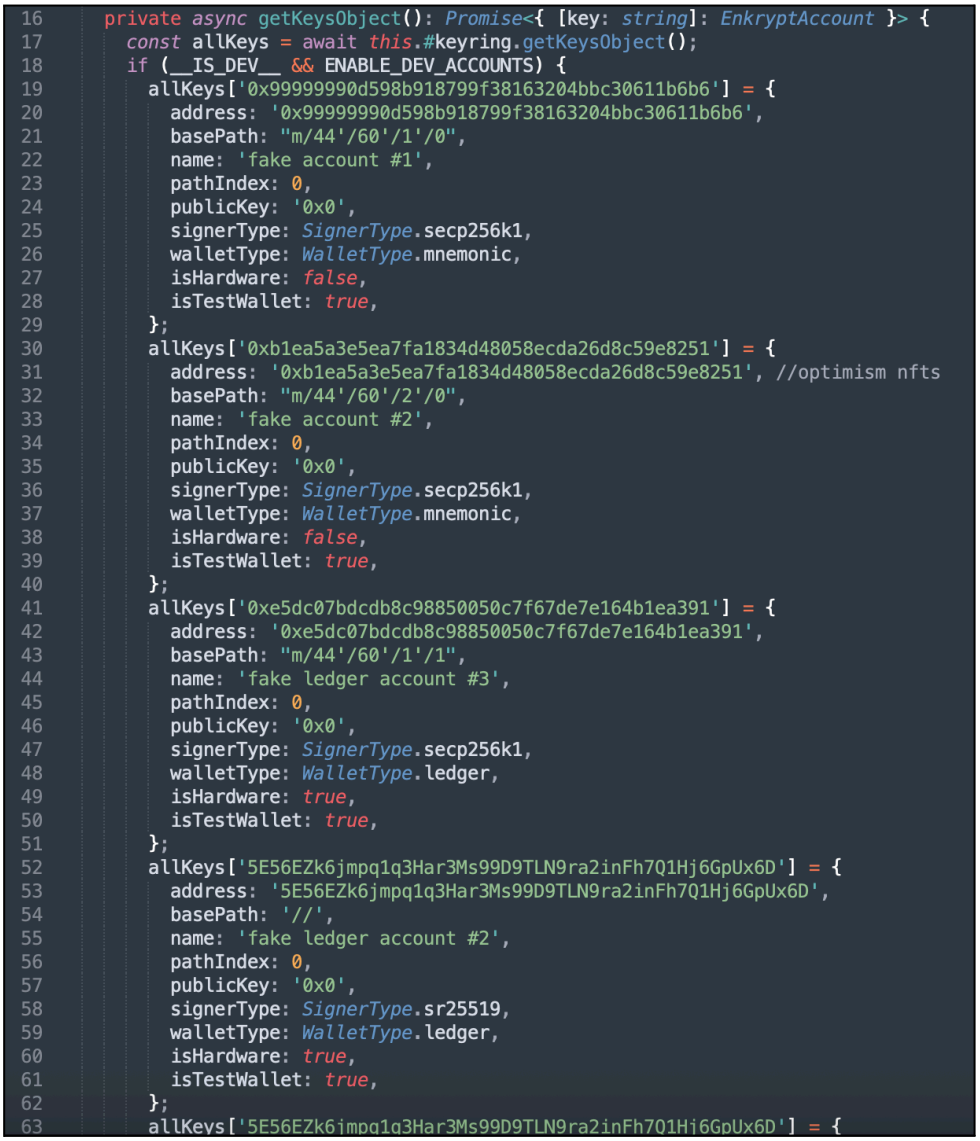
5.1 Missing chrome version detection

Vulnerability Name:	Missing chrome version detection
Risk Level:	Low
Vulnerability URL:	chrome-extension-main/packages/extension/src/ui/action/utils/browser.ts
Vulnerability Description:	The Sip wallet, relying on Chrome's security measures, is built on top of the Chrome browser. However, the lack of Chrome version testing means that the wallet might be operating in an environment with known security vulnerabilities. These vulnerabilities could be exploited by malicious attackers, potentially compromising the security of users.
Vulnerability Detail:	The current code's detectBrowser() function identifies the browser type (Chrome, Firefox, etc.) by string matching navigator.userAgent, but does not extract the version number at all.

	browser.ts
	1 import Browser from 'webextension-polyfill';
	2 import { Updates } from '@/ui/action/types/updates';
	3 export const BROWSER_NAMES = {
	4 chrome: 'chrome',
	5 firefox: 'firefox',
	6 brave: 'brave',
	7 edge: 'edge',
	8 opera: 'opera',
	9 safari: 'safari',
	10 };
	11
	12 export const detectOS = (): Promise<{ os: string; arch: string }> => {
	13 return Browser.runtime.getPlatformInfo().then(info => {
	14 return {
	15 os: info.os,
	16 arch: info.arch,
	17 };
	18 });
	19 };
	20
	21 export const detectBrowser = (): string => {
	22 const { userAgent } = navigator;
	23
	24 if (userAgent.match(/^((?!chrome|android).)*safari/i)) {
	25 return BROWSER_NAMES.safari;
	26 }
	27 if (userAgent.match(/Opera|OPR/i)) {
	28 return BROWSER_NAMES.opera;
	29 }
	30 if (userAgent.match(/edg/i)) {
	31 return BROWSER_NAMES.edge;
	32 }
	33 if (userAgent.match(/chrome|chromium|crios/i)) {
	34 return BROWSER_NAMES.chrome;
	35 }
	36 if (userAgent.match(/firefox|fxios/i)) {
	37 return BROWSER_NAMES.firefox;
	38 }
	39 return '';
	40 };
	41
	42 export const openLink = (url: string) => {
	43 if (detectBrowser() === BROWSER_NAMES.firefox) {
	44 Browser.windows.create({ url, focused: true });
	45 } else {
	46 window.open(url, '_blank', 'noopener');
	47 }
	48 };
	49
	50 export const getLatestEnkryptVersion = (): Promise<string> => {
	51 return fetch(

Fix Suggestion:	To mitigate this risk, it is strongly recommended to implement Chrome browser version checking within the Sip wallet. By incorporating version detection, the wallet can ensure that it operates in a secure environment and take appropriate measures if an outdated or vulnerable version of Chrome is detected.
Repair Status:	Resolved

5.2 Test account leakage risk

Vulnerability Name:	Test account leakage risk
Risk Level:	Low
Vulnerability URL:	chrome-extension-main/packages/extension/src/libs/keyring/public-keyring.ts
Vulnerability Description:	If ENABLE_DEV_ACCOUNTS is mistakenly enabled in a production environment, or if development code is not properly stripped during the build, the test account may be exposed. The fixed address and path of the test account may be exploited by an attacker.
Vulnerability Detail:	

```
16    private async getKeysObject(): Promise<{ [key: string]: EnkryptAccount }> {
17      const allKeys = await this.#keyring.getKeysObject();
18      if (__IS_DEV__ && ENABLE_DEV_ACCOUNTS) {
19        allKeys['0x99999990d598b918799f38163204bbc30611b6b6'] = {
20          address: '0x99999990d598b918799f38163204bbc30611b6b6',
21          basePath: "m/44'/60'/1'/0",
22          name: 'fake account #1',
23          pathIndex: 0,
24          publicKey: '0x0',
25          signerType: SignerType.secp256k1,
26          walletType: WalletType.mnemonic,
27          isHardware: false,
28          isTestWallet: true,
29        };
30        allKeys['0xb1ea5a3e5ea7fa1834d48058ecda26d8c59e8251'] = {
31          address: '0xb1ea5a3e5ea7fa1834d48058ecda26d8c59e8251', //optimism nfts
32          basePath: "m/44'/60'/2'/0",
33          name: 'fake account #2',
34          pathIndex: 0,
35          publicKey: '0x0',
36          signerType: SignerType.secp256k1,
37          walletType: WalletType.mnemonic,
38          isHardware: false,
39          isTestWallet: true,
40        };
41        allKeys['0xe5dc07bdcdb8c98850050c7f67de7e164b1ea391'] = {
42          address: '0xe5dc07bdcdb8c98850050c7f67de7e164b1ea391',
43          basePath: "m/44'/60'/1'/1",
44          name: 'fake ledger account #3',
45          pathIndex: 0,
46          publicKey: '0x0',
47          signerType: SignerType.secp256k1,
48          walletType: WalletType.ledger,
49          isHardware: true,
50          isTestWallet: true,
51        };
52        allKeys['5E56EZk6jmpq1q3Har3Ms99D9TLN9ra2inFh7Q1Hj6GpUx6D'] = {
53          address: '5E56EZk6jmpq1q3Har3Ms99D9TLN9ra2inFh7Q1Hj6GpUx6D',
54          basePath: '//',
55          name: 'fake ledger account #2',
56          pathIndex: 0,
57          publicKey: '0x0',
58          signerType: SignerType.sr25519,
59          walletType: WalletType.ledger,
60          isHardware: true,
61          isTestWallet: true,
62        };
63        allKeys['5E56EZk6jmpq1q3Har3Ms99D9TLN9ra2inFh7Q1Hj6GpUx6D'] = {
```

Fix Suggestion:	Make sure ENABLE_DEV_ACCOUNTS is set to false during production builds, and remove unnecessary test information.
Repair Status:	Resolved

5.3 Password plaintext transmission and memory residual risk

Vulnerability Name:	Password plaintext transmission and memory residual risk
Risk Level:	Low
Vulnerability URL:	chrome-extension-main/packages/extension/src/libs/background/internal/unlock.ts
Vulnerability Description:	The password is passed in plain text via message.params. If the source of the message is not encrypted (such as through an insecure communication channel), it may be intercepted by a man-in-the-middle attack. The password exists in the memory as a string, and no secure buffer (such as Buffer) is used, and the memory is not cleared in time.
Vulnerability Detail:	Risky code snippet (unlock.ts) const password = message.params[0] as string;

	unlock.ts
	1 import { getCustomError } from '@/libs/error';
	2 import KeyRingBase from '@/libs/keyring/keyring';
	3 import { InternalOnMessageResponse } from '@/types/messenger';
	4 import { RPCRequestType } from '@enkryptcom/types';
	5 import { initAccounts } from '@/libs/utils/initialize-wallet';
	6
	7 const unlock = (
	8 keyring: KeyRingBase,
	9 message: RPCRequestType,
	10): Promise<InternalOnMessageResponse> => {
	11 if (!message.params || message.params.length < 1)
	12 return Promise.resolve({
	13 error: getCustomError('background: invalid params for unlocking'),
	14 });
	15 const password = message.params[0] as string;
	16 const initNewAccounts = (message.params[1] as boolean) ?? false;
	17 return keyring
	18 .unlock(password)
	19 .then(async () => {
	20 if (initNewAccounts) {
	21 await initAccounts(keyring);
	22 }
	23 return {
	24 result: JSON.stringify(true),
	25 };
	26 })
	27 .catch(e => {
	28 return {
	29 error: getCustomError(e.message),
	30 };
	31 });
	32 };
	33
	34 export default unlock;
	35

Fix Suggestion:	1.Encrypted transmission: Make sure the message channel uses TLS/HTTPS or end-to-end encryption.
	
	2.Secure memory management:
	// Use Buffer and clear the memory
	const passwordBuffer = Buffer.from(message.params[0] as string, 'utf8');
	// Clear immediately after using the password
	passwordBuffer.fill(0);

Repair Status:	Resolved

5.4 Unverified network switching

Vulnerability Name:	Unverified network switching
Risk Level:	Low

Vulnerability URL:	chrome-extension-main/packages/extension/src/libs/background/internal/change-network.ts
Vulnerability Description:	Directly use the unverified networkName parameter to switch networks without checking whether the network is legitimate or belongs to the current provider. An attacker can construct malicious requests to switch to any network (such as a private network controlled by the attacker) to hijack transactions or steal assets.

Vulnerability Detail:	Risky code snippet (change-network.ts)
	
	const networkName = message.params[0] as string;
	const network = (await getNetworkByName(networkName)) as BaseNetwork;
	change-network.ts
	1 import { getCustomError } from '@/libs/error';
	2 import { getNetworkByName } from '@/libs/utils/networks';
	3 import { BaseNetwork } from '@/types/base-network';
	4 import {
	5 ActionSendMessage,
	6 InternalOnMessageResponse,
	7 Message,
	8 } from '@/types/messenger';
	9 import { RPCRequestType } from '@enkryptcom/types';
	10 import { TabProviderType } from '../types';
	11
	12 const changeNetwork = async (
	13 msg: Message,
	14 tabProviders: TabProviderType,
	15): Promise<InternalOnMessageResponse> => {
	16 const message = JSON.parse(msg.message) as RPCRequestType;
	17 if (!message.params || message.params.length < 1)
	18 return Promise.resolve({
	19 error: getCustomError('background: invalid params for change network'),
	20 });
	21 const networkName = message.params[0] as string;
	22 const network = (await getNetworkByName(networkName)) as BaseNetwork;
	23 const actionMsg = msg as any as ActionSendMessage;
	24 if (
	25 actionMsg.provider &&
	26 actionMsg.tabId &&
	27 tabProviders[actionMsg.provider][actionMsg.tabId]
	28) {
	29 tabProviders[actionMsg.provider][actionMsg.tabId].setRequestProvider(
	30 network,
	31);
	32 }
	33 return Promise.resolve({
	34 result: JSON.stringify(true),
	35 });
	36 };
	37
	38 export default changeNetwork;
	39

Fix Suggestion:	Strictly limit the scope of network switching, only allow switching to networks supported by the current Provider (including custom networks), implement security verification through getProviderNetworkByName, and completely block the risk of malicious network switching.

Repair Status:	Resolved

5.5 Message injection vulnerability

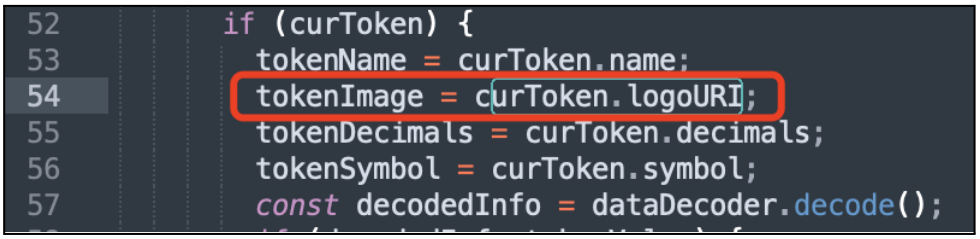
Vulnerability Name:	Message injection vulnerability
Risk Level:	Low
Vulnerability URL:	chrome-extension-main/packages/extension/src/libs/background/index.ts
Vulnerability Description:	JSON.parse(msg.message) is executed directly without verification, and maliciously constructed messages may lead to RCE
Vulnerability Detail:	Risky code snippet (change-network.ts)
Fix Suggestion:	Use a safe JSON parser, add try-catch, and handle exceptions
Repair Status:	Resolved

```
74        }
75      }
76      async externalHandler(
77        msg: Message,
78        options: ExternalMessageOptions = { savePersistentEvents: true },
79      ): Promise<OnMessageResponse> {
80        const { method, params } = JSON.parse(msg.message);
81        const _provider = msg.provider;
82        const _tabid = msg.sender.tabId;
83        if (_provider === ProviderName.enkrypt) {
84          if (method === InternalMethods.buyCA) {
85            // @ts-ignore
86            return buyCA(this.#keyring, msg);
87          }
88          if (
89            method === InternalMethods.newWindowInit ||
90            method === InternalMethods.newWindowUnload
91          ) {
92            this.#persistentEvents.deleteEvents(_tabid);
93            return {
94              result: JSON.stringify(true),
```

5.6 Token metadata injection

Vulnerability Name:	Token metadata injection
Risk Level:	Low
Vulnerability URL:	chrome-extension-main/packages/extension/src/providers/ethereum/libs/transaction/decoder.ts

Vulnerability Description:	JFile: decoder.ts
	tokenImage = curToken.logoURI; // Use external URL directly
	A malicious token contract can return a dangerous URL
	For example
	"logoURI": "javascript:alert('XSS')"
	SVG images may contain malicious scripts

Vulnerability Detail:	

Fix Suggestion:	Implement strict URL filtering
	const sanitizeURI = (uri: string) => {
	if (!uri.startsWith('https://')) return '';
	if (uri.endsWith('.svg')) return ''; // Disable SVG
	return uri;
	};

Repair Status:	Resolved

Security Summary

Security Recommendations

1) In response to the vulnerabilities found in the penetration test, Sip should take immediate measures to fix several key security issues. First, improve the memory leak problem, and also delete the residual test information in the extension. Regular security audits and vulnerability assessments can help to timely discover and fix potential risks and ensure overall security.