Penetration Test on Blank Wallet for Ricitus UAB

Final Report and Management Summary

2021-11-08

CONFIDENTIAL

X41 D-SEC GmbH
Dennewartstr. 25-27
D-52068 Aachen
Amtsgericht Aachen: HRB19989

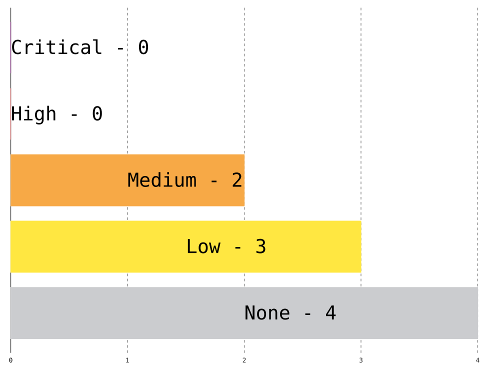
`https://x41-dsec.de/`
`info@x41-dsec.de`

Revision	*Date*	*Change*	*Author(s)*
1	2021-08-24	Initial Report	M. Orru
2	2021-08-25	Findings	M. Orru
3	2021-09-11	Findings	M. Vervier, M. Orru
4	2021-09-24	Preliminary Report	M. Vervier
5	2021-11-08	Finalization	M. Vervier

Contents

Dashboard

Target	
Customer	Ricitus UAB
Name	Blank-Wallet
Type	Chrome Extension
Version	As deployed between 2021-08-11 and 2021-11-06
Engagement	
Type	Source Code Audit
Consultants	3: Michele Orru and Markus Vervier
Engagement Effort	14.5 / 14.5 person-days, 2021-08-11 to 2021-11-06
Total issues found	5



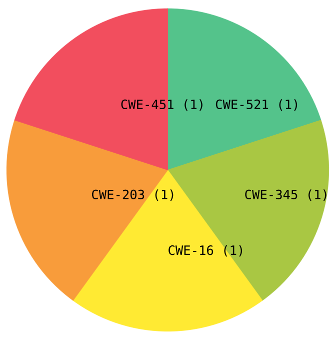


Figure 1: Issue Overview (l: Severity, r: CWE Distribution)

1 Executive Summary

In August and September 2021, X41 D-Sec GmbH performed a source code audit of the Blank-Wallet.

A total of five vulnerabilities were discovered during the test by X41. None were rated as critical, none were classified as high severity, two as medium, and three as low. Additionally, four issues without a direct security impact were identified.

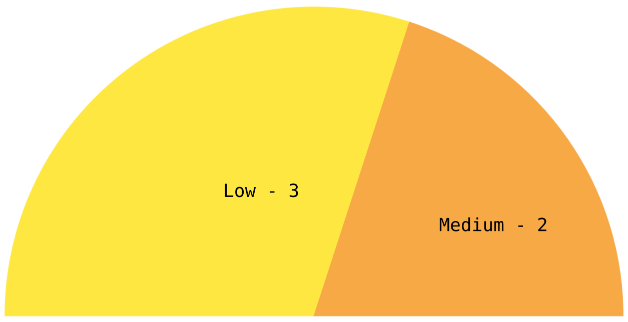


Figure 1.1: Issues and Severity

In conclusion, this security review has not uncovered severe vulnerabilities and most of the raised issues can be considered as additional defense-in-depth measures elevating the security posture of the Blank Wallet browser extension. However, it is recommended to investigate hardening measures to mitigate certain attack scenarios such as described in informational finding 5.2.1.

2 Technical Summary

Blank-Wallet is a privacy preserving Ethereum Wallet implemented as an extension for Google Chrome. Naturally, vulnerabilities affecting the Wallet could lead to loss of funds or privacy.

The security review was performed by three experienced security experts between 2021-08-11 and 2021-11-06.

The source code has been audited focusing on typical programming mistakes in JavaScript and TypeScript, and paying special attention to typical vulnerabilities and weaknesses that exist within Chrome Browser extensions. After an initial review for web vulnerabilities in August and September 2021, another targeted review was conducted against the improved code base and implementations of EIP-747 and EIP-3326 in November 2021.

The code of the Blank Wallet is well written and commented, which helps the auditors to understand what it is doing. From a local attackers perspective, Ricitus UAB could improve hardening measures that can effectively protect users funds, stored within the Blank Wallet Browser extension and the threat model explicitly excludes local attackers from the attack threat model. As a result of this, special attention was paid to the following remote attack vectors:

- An attacker controls a web site that is accessed by the extension. In that case, an attacker attempts to corrupt an extension when the extension interacts with the attacker controlled web site by exploiting vulnerabilities that arise due to insecure processing of data received through the backend services.
- An active attacker capable of intercepting, modifying, and injecting network traffic (e.g., HTTP responses).

During the course of this test, the testers were able to gain a good test coverage of the scope as described in section 3.3.1.

The most important findings made were related to the manipulation of data coming from otherwise trustworthy services such as the ones providing information about current Ethereum gas prices or other blockchain related data. Another vulnerability may allow a website to track users of the wallet extension by using unique icon image URLs provided to the extension and stored permanently there.

Other findings are bound to certain threat scenarios or are only relevant when other vulnerabilities present in the underlying technology of the browser have been successfully exploited before. Nevertheless, such vulnerabilities and scenarios are deemed relevant to the current scope and it is recommended to address them where possible.

In conclusion, this security review has not uncovered severe vulnerabilities and most of the raised issues can be considered as additional defense-in-depth measures elevating the security posture of the Blank Wallet browser extension. The code and permissions of the app seem to be well written with security in mind.

2.1 Recommended Further Tests

It is recommended to constantly audit the code base after changes are made either in the code developed by Ricitus UAB or in third party components and modules. Where possible, the mission critical third party modules and dependencies should also be subject to audits.

3 Introduction

X41 reviewed the components of a privacy preserving Ethereum Wallet called *"Blank-Wallet"*. The wallet can hide financial data by mixing transfers with a pool within Blank. This privacy is increased with increased transfers, because it should be hard to distinguish individual funds amongst them. The concept is similar and based on *Tornado*[1].

Given the nature of the protected assets and the privacy requirements that come with such transfers, security vulnerabilities in the wallet could have a high impact and lead to loss of funds or serious privacy breaches.

X41 does not give legal or business advice but recommends to verify findings where personal data disclosure is concerned against the regulations of the GDPR[2] and in particular the regulations against unauthorized disclosure of user data.

X41 exclusively reviewed the technical implementation of the Chrome extension developed by Ricitus UAB. The business model, operational security of Ricitus UAB, the security of third party services or of related smart-contracts were not in scope of this review.

3.1 Methodology

The review was mainly based on source code reviews and white-box penetration testingof the Blank-Wallet Chrome browser extension and related components.

X41 adheres to established standards for source code reviewing and penetration testing. These are in particular the *CERT Secure Coding*[3] standards and the *Study - A Penetration Testing Model*[4] of the German Federal Office for Information Security.

[1] `https://tornado.cash`
[2] `http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679`
[3] `https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards`
[4] `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Studies/Penetration/penetration_pdf.pdf?__blob=publicationFile&v=1`

3.2 Findings Overview

DESCRIPTION	SEVERITY	ID	REF
Weak Password Complexity	LOW	SHRTNM-PT-21-01	5.1.1
Backend Response Message Signing / Weak MITM Protection	MEDIUM	SHRTNM-PT-21-02	5.1.2
Unsafe-Eval Allowed via CSP	MEDIUM	SHRTNM-PT-21-03	5.1.3
Third Party Tracking via Icon URL	LOW	SHRTNM-PT-21-04	5.1.4
Untrustworthy Popup Window May be Mistaken With Extension Window	LOW	SHRTNM-PT-21-05	5.1.5
Untrustworthy Media Content Rendered Inside the Extension Context	NONE	SHRTNM-PT-21-100	5.2.1
Seed Phrase Comparison Not Timing-Safe	NONE	SHRTNM-PT-21-101	5.2.2
Verbose Error Message at goerli.infura.io	NONE	SHRTNM-PT-21-102	5.2.3
getCryptoRandom() Handles Only 32-Bit Ranges Correctly	NONE	SHRTNM-PT-21-103	5.2.4

Table 3.1: Security-Relevant Findings

3.3 Scope

The scope of this audit is the source code of a Chrome extension as provided in the GitHub repositories of the Blank-Wallet organization. In Particular the following revisions and branches as present in the GitHub repositories[5] were reviewed during this test:

- Review part 1:
  - extension repository commit id *603d6ff9de330618158a04429486d601ce2625be* on 2021-08-24
- Review part 2:
  - extension repository commit id *e479c02224769cd8c173b6862800e6f0ae66683a* on 2021-10-27
  - Branch *feature/eip3326-switchEthereumChain*
  - Branch *feature/eip-747*

After reviewing the code provided by Ricitus UAB, the components to be reviewed are estimated to have the following source code volume:

- 18.000 LoC TypeScript (excluding 4.000 LoC that appear to be tests)

[5] `https://github.com/Blank-Wallet`

- 1.400 LoC JavaScript
- JSON and Tornado configuration

Attacks resulting from 0days (security issues not yet publicly known) or other weaknesses in the Chrome browser are not considered to be in scope.

3.3.1 Coverage

A security assessment attempts to find the most important or sometimes as many of the existing problems as possible, though it is practically never possible to rule out the possibility of additional weaknesses being found in the future.

In this test, X41 found the time available to yield a good coverage of the given scope.

The code of the Chrome extension was individually reviewed including the code of the extension itself and the usage of third party packages and modules. Third party code has been audited in a targeted manner, for example to investigate potential vulnerabilities resulting from wrong API usage.

The extension code was reviewed systematically for common problems that affect browser extensions. Notably the following attack surface was investigated (among others):

3.3.1.1 Logical Vulnerabilities (focus on EIP-747 and EIP-3326)

The extension's business logic was probed for logical vulnerabilities that would allow an attacker to manipulate the extension or the user into performing unwanted actions. In particular the message signing implementation belonging to *EIP-747*[6] and the wallet switch implementation of *EIP-3326*[7] was reviewed and found to be correct.

3.3.1.2 DAPP Attacks and postMessage RPC

The extension can interact with external applications such as *DAPPS*[8]. Such apps are using specified JavaScript APIs (web3) and X41 investigated the attack surface created by this.

Events from ***postMessage()*** calls are handled correctly by the extension. From the git history it was visible that a serious vulnerability was patched 9 days before X41 could start the test:

[6] `https://eips.ethereum.org/EIPS/eip-747`
[7] `https://eips.ethereum.org/EIPS/eip-3326`
[8] `https://ethereum.org/en/dapps/`

- `https://github.com/Blank-Wallet/extension-provider/commit/c73afdb421a6e543b99728c6217af0873c802c25`

Before the bug patch, JavaScript code running on a page could have attached to message events and also sent messages via ***postMessage()*** to cause different RPC[9] actions. Since there was no check that the messages were coming from the intended origin, untrustworthy web origins could have triggered privileged actions.

The different RPC methods have been inspected for security vulnerabilities that might compromise the extension such as unchecked permissions on privileged actions. It was found that websites could only trigger the following actions via message types marked as external using ***postMessage()***:

- *EVENT_SUBSCRIPTION*
- *EXTERNAL_REQUEST*
- *SETUP_PROVIDER*
- *SET_METADATA*

The message event handler function of the content script was found to properly enforce and reject all other message types coming from the website context.

3.3.1.3 JSON RPC

Using the message type *EXTERNAL_REQUEST*, the extension exposes a set of JSON-RPC handlers that are reachable by the website. It was investigated and verified that the appropriate permission checks and / or requirements for explicit user interaction consent were working correctly.

3.3.1.4 Separation of Origins

The app was reviewed to ensure that the extension applies internal separation on actions triggered by different origins internally. This includes actions triggered by untrustworthy origins in contrast to actions triggered by trustworthy origins. An example for such a trustworthy origin would be a website that successfully applied for permissions via *web3*[10], the Ethereum JavaScript API.

[9] Remote Procedure Calls
[10] `https://web3js.readthedocs.io/en/v1.5.2/`

3.3.1.5 Extension Permissions Permissions

The permissions set in the manifest of the extension are reasonable for the intended purpose:

- “activeTab”,
- “storage”,
- “notifications”

In general, the extension does not request permissions considered as unnecessary and dangerous.

3.3.1.6 Cryptographic Weaknesses

The extension has been investigated for cryptographic weaknesses such as the usage of weak cryptographic algorithms or wrong usage of APIs. While from a development point of view, algorithms could be upgraded and modernized, no practical impact could be seen from this at this point.

3.3.1.7 Injection Attacks

The extension and its content has been inspected for injection attacks that aim to inject active JavaScript, HTML[11], or other content into the extension’s privileged pages and script. No exploitable vulnerability was identified so far, but it was found that the extension accepts icon images passed on by untrustworthy websites that are rendered in the HTML context of the extension origin. In the time given and according to current knowledge, XSS[12] attacks are not possible, but the attack surface should be further investigated. More information is available in finding 5.1.4.

[11] HyperText Markup Language
[12] Cross-site Scripting

4 Rating Methodology for Security Vulnerabilities

Security vulnerabilities are given a purely technical rating by the testers as they are discovered during the test. Business factors and financial risks for Ricitus UAB are beyond the scope of a penetration test which focuses entirely on technical factors. Yet technical results from a penetration test may be an integral part of a general risk assessment. A penetration test is based on a limited time frame and only covers vulnerabilities and security issues which have been found in the given time, there is no claim for full coverage.

In total, five different ratings exist, which are as follows:

Severity Rating
None
Low
Medium
High
Critical

A low rating indicates that the vulnerability is either very hard for an attacker to exploit due to special circumstances, or that the impact of exploitation is limited, whereas findings with a medium rating are more likely to be exploited or have a higher impact. High and critical ratings are assigned when the testers deem the prerequisites realistic or trivial and the impact significant or very significant.

Findings with the rating 'none' are called side findings and are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

4.1 Common Weakness Enumeration

The CWE[1] is a set of software weaknesses that allows the categorization of vulnerabilities and weaknesses in software. If applicable, X41 provides the CWE-ID for each vulnerability that is discovered during a test.

CWE is a very powerful method to categorize a vulnerability and to give general descriptions and solution advice on recurring vulnerability types. CWE is developed by *MITRE*[2]. More information can be found on the CWE website at `https://cwe.mitre.org/`.

[1] Common Weakness Enumeration
[2] `https://www.mitre.org`

5 Results

This chapter describes the results of this test. The security-relevant findings are documented in Section 5.1. Additionally, findings without a direct security impact are documented in Section 5.2.

5.1 Findings

The following subsections describe findings with a direct security impact that were discovered during the test.

5.1.1 SHRTNM-PT-21-01: Weak Password Complexity

Severity:	LOW
CWE:	521 – Weak Password Requirements
Affected Component:	packages/ui/src/routes/setup/PasswordSetupPage.tsx:20

5.1.1.1 Description

During a dynamic test and source code audit of the Blank Wallet Chrome Browser extension, it was noticed that the Blank Wallet does not enforce a sufficiently strong password complexity policy and allows users to set weak wallet passwords.

A local attacker, having access to a computer of the victim running the Blank Wallet Browser extension, could leverage this and mount automated brute force or dictionary attacks.

The code snippet in listing 5.1 shows that the password complexity only requires a password of length 8, having one lowercase letter and one digit set.

```
const schema = yup.object().shape({
  password: yup
    .string()
    .required('No password provided.')
    .min(8, 'Password should be at least 8 characters long.')
    .matches(
      /(?=.*\d)(?=.*[a-z])/,
      'Password must contain at least one lowercase character and one digit.'
    ),
    [...]
```

Listing 5.1: Password Setup Schema

While users can set a sufficiently secure password, real world applications show that many users will resort to weaker and predictable passphrases unless indicators of password strength are present.

5.1.1.2 Solution Advice

X41 recommends to implement a strong password complexity policy to prevent users to set low-complexity passwords, which can be brute-forced or guessed easily, at least fulfilling the following criteria[1]:

- Usage of:
  - Lower and uppercase characters
  - At least one digit
  - Special characters
- Prohibition of using dictionary words

[1] `https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html`

5.1.2 SHRTNM-PT-21-02: Backend Response Message Signing / Weak MITM Protection

Severity:	MEDIUM
CWE:	345 – Insufficient Verification of Data Authenticity
Affected Component:	RPC / Third Party Services

5.1.2.1 Description

While testing the Blank Wallet Chrome Browser extension, it was observed that the Blank Wallet interacts with numerous backend API[2] services, for example `https://goerli.infura.io`, among others. The Blank Wallet extension processes the JSON[3] responses that are received from the backend without any additional security measures ensuring integrity and authenticity of the actual response body, apart from standard TLS[4].

A sufficiently well-positioned attacker that is able to mount a MITM[5] attack and therefore eavesdrop and edit HTTP[6] requests and responses, could abuse this lack of a signature of the HTTP responses and tamper with the responses without the user noticing, and potentially cause further unspecified harm to the user, for example an attacker could try to show fake information to the user or exploit vulnerabilities that may exist when the Blank Wallet Chrome extension processes JSON RPC responses in an insecure manner.

Since TLS is relying on a number of CA[7] recognized by the Chrome browser as root of trust, an attacker that has access to any of these CAcould issue an unauthorized rogue certificate and intercept the connections.

5.1.2.2 Solution Advice

X41 proposes adding a signature to all HTTP responses, in particular JSON RPC response messages, that are signed by the service that produced the response message. The Blank Wallet Chrome Browser extension should verify the signature of the received response in order to ensure integrity and authenticity. Additionally, a regular security assessment of the external services is recommended.

[2] Application Programming Interface
[3] JavaScript Object Notation
[4] Transport Layer Security
[5] Man-in-the-middle Attack
[6] HyperText Transfer Protocol
[7] Certificate Authority

5.1.3 SHRTNM-PT-21-03: Unsafe-Eval Allowed via CSP

Severity:	MEDIUM
CWE:	16 – Configuration
Affected Component:	packages/ui/src/routes/setup/PasswordSetupPage.tsx:20

5.1.3.1 Description

It was discovered during the test that *unsafe-eval* is allowed in the CSP[8] as seen in listing 5.2.

```
    "content_security_policy": "script-src 'self' 'unsafe-eval'; object-src 'self'",
```

Listing 5.2: CSP unsafe-eval

The CSP allows a website to constrain certain features such as execution of JavaScript from sources that are not intended by the developer. In this particular case, the wallet is explicitly allowing unsafe and dynamic evaluation of JavaScript.

Attackers that found a way to inject content into the extension components could therefore also inject dynamic code that is evaluated at runtime, potentially compromising the security of the extension.

5.1.3.2 Solution Advice

X41 recommends to implement a strong CSP policy without `unsafe-eval`.

[8] Content Security Policy

5.1.4 SHRTNM-PT-21-04: Third Party Tracking via Icon URL

Severity:	LOW
CWE:	203 – Information Exposure Through Discrepancy
Affected Component:	packages/ui/src/routes/connect/ConnectPage.tsx

5.1.4.1 Description

During the review X41 discovered that using the RPC method *SET_METADATA*, any website can register a unique URL[9] used to load an icon for this website and use it to track when a particular user sees the permission dialogue or opens the extension settings. This might impact the wallet user's privacy since the loading of the icon URL can reveal both the fact that a user is using the extension at a current point in time and also leak the IP[10] address and user agent of that user unless special security and privacy precautions are taken.

An example of such a call is shown in listing 5.3 where the origin `www.x41-dsec.de` registers the metadata including an unique icon URL.

```
window.postMessage({"id": '1635969884521.0', "origin": 'external', 'message': "SET_METADATA",
↪ 'request': {"siteMetadata": {"name": "www.x41-dsec.de", "iconURL":
↪ "https://www.x41-dsec.de/unique/canary-unique-identifier-4711" }}}, "*")
```

Listing 5.3: Set Metadata Call

The icon will be rendered as image in the following places of the extension popup HTML:

- Permission Request Dialog
- Connected Sites Overview
- Connected Site Details

The HTML markup used to render the image is shown in listing 5.4

```
<img alt="icon" src="https://www.x41-dsec.de/canary2" class="max-h-11">
```

[9] Uniform Resource Locator
[10] Internet Protocol

Listing 5.4: Set Metadata Call

When the icon is loaded by the extension, the browser will make a request to the website serving the registered URL. Even if a website is trusted by the user at a certain point in time, this website might be compromised in the future. A user does not have the option to remove such a compromised site without triggering a load of the icon on the settings page. Furthermore, it was observed that a permission request dialogue in the extension popup does not timeout and can trigger the icon load operation at a later point in time, when the user might not expect it.

Additionally, the rendering of untrustworthy media content provides an attack vector for attacks against the browser itself with the goal to subvert the sandboxing and security measures that protect the wallet extension of Ricitus UAB. Since the browser code is out of scope for this project, this is not seen as a direct security vulnerability. More information is given in side finding 5.2.1. No injection attacks in current browsers have been identified via the icon image.

5.1.4.2 Solution Advice

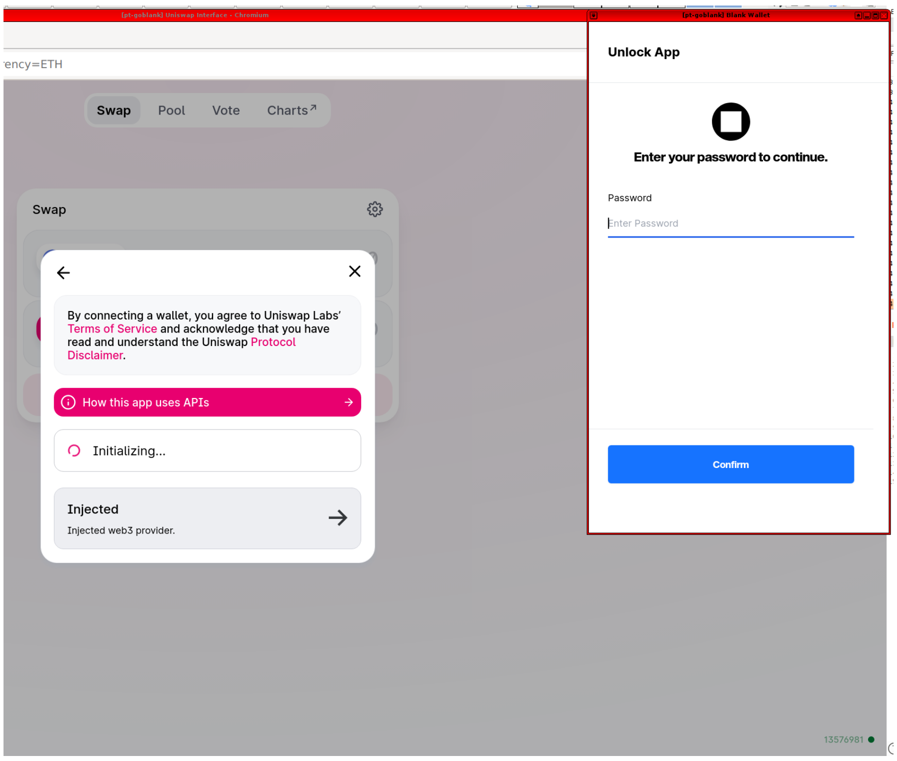
It is recommended to cache the icon image files for all connected web origins. This will ensure that the icon file load does not leak information to external websites when the extension settings are used by the user. Furthermore, it is recommended to set timeout on permission request dialogues and clear the state of unfinished dialogues when the browser is restarted.

5.1.5 SHRTNM-PT-21-05: Untrustworthy Popup Window May be Mistaken With Extension Window

Severity:	LOW
CWE:	451 – User Interface (UI) Misrepresentation of Critical Information
Affected Component:	Extension Popup

5.1.5.1 Description

It was found that a website can trigger the extension popup window to be shown in a separate browser window as shown in figure 5.1.



By connecting a wallet, you agree to Uniswap Labs' Terms of Service and acknowledge that you have read and understand the Uniswap Protocol Disclaimer.

Enter your password to continue.

Figure 5.1: Extension Popup Window

This popup window does not have a location bar, but otherwise it looks exactly like any other popup window. An malicious website could spawn an identical looking popup to trick a user to enter sensitive data such as the wallet password into the fake window.

5.1.5.2 Solution Advice

It is recommended to provide a unique visual cue to the user that the window belongs to the extension. For example by using the browserAction popup feature[11], the created window would be visible as belong to an extension window. Alternatively, showing a unique fingerprint in the popup that cannot be obtained by a remote website could be a workaround.

[11] `https://developer.chrome.com/docs/extensions/reference/browserAction/#popups`

5.2 Side Findings

The following observations do not have a direct security impact, but are related to security hardening, affect functionality, or other topics that are not directly related to security. X41 recommends to mitigate these issues as well, because they often become exploitable in the future. Doing so will strengthen the security of the system and is recommended for defense in depth.

5.2.1 SHRTNM-PT-21-100: Untrustworthy Media Content Rendered Inside the Extension Context

Affected Component:	packages/ui/src/routes/connect/ConnectPage.tsx

5.2.1.1 Description

As described in section 5.1.4, rendering of untrustworthy media files such as images inside the extension may pose a security threat. Attackers might try to exploit vulnerabilities present in the image parsing code of the Chrome web browser to compromise the extension process and the wallet private keys stored within.

This threat is not just theoretical as demonstrated by the multitude of vulnerabilities identified in the Chrome browser each month. Examples for vulnerabilities that could be used to compromise the wallet are:

- *Issue 116162: Heap-buffer-overflow in wk_png_inflate:* `https://bugs.chromium.org/p/chromium/issues/detail?id=116162`
- *Issue 702934: Heap-use-after-free in cr_png_set_longjmp_fn:* `https://bugs.chromium.org/p/chromium/issues/detail?id=702934`
- *Issue 1023843: CVE-2019-2201: libjpeg-turbo: code execution:* `https://bugs.chromium.org/p/chromium/issues/detail?id=1023843`

Due to the sandboxing and site isolation features of modern browsers, a vulnerability exploited in a certain web origin is isolated from other origins and especially extensions that run in a separate sandbox. If the vulnerability is exploited in the same sandbox, for example by rendering a malicious image inside this sandbox, the security isolation provided by the browser sandboxes is compromised.

5.2.1.2 Solution Advice

To mitigate the described technical threat, it is recommended to convert the icon media file in the content script to a simple and standardized file format with a lower probability of having unknown vulnerabilities such as *BMP*. It is important to do this conversion in the content script and not in the extension since the content script runs inside the same sandbox as the website origin sending the icon file. Equally important is that the extension should verify the converted icon file to be of expected type and to force rending of the file only as that type.

5.2.2 SHRTNM-PT-21-101: Seed Phrase Comparison Not Timing-Safe

Affected Component:	packages/background/src/controllers/BlankController.ts:1136

5.2.2.1 Description

It was found that the validation of the seed phrase within ***verifySP()*** makes use of a time-unsafe comparison when verifying the provided seed of the user against the locally stored seed value using ===. The string comparison in JavaScript is an algorithm that is linearly time-variant to the equivalence of the input strings. In this specific case, the more the user-input matches the seed phrase, the greater the runtime of the comparison. This could be abused by attackers during brute-force attacks of the seed phrase, rendering the minimal time differences measurable.

The code snippet in listing 5.5 shows how the user provided seed phrase is compared against the vault seed phrase using the === equality JavaScript operator.

```
/**
 * Method to verify if the user has correctly completed the seed phrase challenge
 *
 * @param seedPhrase
 */
private async verifySP({
  seedPhrase,
}: RequestVerifySeedPhrase): Promise<boolean> {
  const vaultSeedPhrase = await this.verifySeedPhrase();
  if (seedPhrase === vaultSeedPhrase) {
    this.onboardingController.isSeedPhraseBackedUp = true;
    return true;
  } else {
    throw new Error('Seed Phrase is not valid');
  }
}
```

Listing 5.5: Seed Phrase Verification

5.2.2.2 Solution Advice

For all string comparisons that contain sensitive information, X41 recommends to use time-safe comparisons, such as those implemented by the crypto package and the function ***crypto.timingSafeEqual(a, b)***[12]. By doing so, the string comparison runs with a time that is constant for strings of the same length. This will prevent attackers from using time as a side-channel, thus mitigating the risk of sensitive information being extracted.

[12] `https://nodejs.org/api/crypto.html#crypto_crypto_timingsafeequal_a_b`

5.2.3 SHRTNM-PT-21-102: Verbose Error Message at goerli.infura.io

Affected Component:	goerli.infura.io

When removing a few zeroes from the data parameter value in the HTTP request parameters, a verbose error message is triggered as shown in listing 5.6.

```
Request:
    POST /v3/529b28a50dd446f19258621f4474a02f HTTP/2
    Host: goerli.infura.io
    Content-Length: 904
    Cache-Control: max-age=0
    Sec-Ch-Ua: " Not A;Brand";v="99", "Chromium";v="90"
    Sec-Ch-Ua-Mobile: ?0
    User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
    ↪  Chrome/90.0.4430.212 Safari/537.36
    Content-Type: application/json
    Accept: */*
    Origin: chrome-extension://cejjgibodjcgoanbnmaafijlpochegef

    {"method":"eth_call","params":[{"to":"0x906f63676923374a7b9781bcc1b1532488d45a7a","data":
    ↪  "0xf0002ea90000000000000..."},"latest"],"id":3135,"jsonrpc":"2.0"}

Response:
    {"jsonrpc":"2.0","id":3135,"error":
    {"code":-32602,"message":"invalid argument 0: json: cannot unmarshal hex
     string of odd length into Go struct field TransactionArgs.data of type hexutil.Bytes"}}
```

Listing 5.6: Verbose Error

While the error messages are displayed on a third party site, they might give attackers valuable information about the system that could be useful in further attacks.

5.2.3.1 Solution Advice

Error messages should not be displayed with full details. If full details are required, X41 to replace the error message with a generic error message, but log the detailed error internally for debugging purposes.

5.2.4 SHRTNM-PT-21-103: getCryptoRandom() Handles Only 32-Bit Ranges Correctly

Affected Component:	extension/packages/background/src/controllers/blank-deposit/utils/getCryptoRandom.ts

5.2.4.1 Description

As seen in listing 5.7, the random number range is not expected to be larger than 32 bit. If it is larger, the distribution of random numbers is not uniform across the full randomness range anymore. While no direct security impact in this context is visible, uniformness is usually an expected property of a PRNG[13].

```
    if (typeof window !== 'undefined') {
        if (!window.crypto) {
          throw new Error('Browser does not support Crypto lib');
        }
        randomInRange =
          window.crypto.getRandomValues(new Uint32Array(1))[0] / 0x100000000;
      } else {
        const { randomBytes } = require('crypto');
        randomInRange = randomBytes(4).readUInt32LE() / 0x100000000;
      }
      return Math.floor(randomInRange * maxValue);
```

Listing 5.7: Seed Phrase Verification

5.2.4.2 Solution Advice

It is recommended to raise an error if the randomness range cannot be treated as 32-bit number. If larger ranges are required, it is recommended to extend the range to 64- or 128-bit width.

[13] Pseudo Random Number Generator

6 About X41 D-Sec GmbH

X41 D-Sec GmbH is an expert provider for application security and penetration testing services. Having extensive industry experience and expertise in the area of information security, a strong core security team of world-class security experts enables X41 D-Sec GmbH to perform premium security services.

X41 has the following references that show their experience in the field:

- Review of the Mozilla Firefox updater[1]
- X41 Browser Security White Paper[2]
- Review of Cryptographic Protocols (Wire)[3]
- Identification of flaws in Fax Machines[4,5]
- Smartcard Stack Fuzzing[6]

The testers at X41 have extensive experience with penetration testing and red teaming exercises in complex environments. This includes enterprise environments with thousands of users and vendor infrastructures such as the Mozilla Firefox Updater (Balrog).

Fields of expertise in the area of application security encompass security-centered code reviews, binary reverse-engineering and vulnerability-discovery. Custom research and IT security consulting, as well as support services, are the core competencies of X41. The team has a strong technical background and performs security reviews of complex and high-profile applications such as Google Chrome and Microsoft Edge web browsers.

X41 D-Sec GmbH can be reached via `https://x41-dsec.de` or `mailto:info@x41-dsec.de`.

[1] `https://blog.mozilla.org/security/2018/10/09/trusting-the-delivery-of-firefox-updates/`
[2] `https://browser-security.x41-dsec.de/X41-Browser-Security-White-Paper.pdf`
[3] `https://www.x41-dsec.de/reports/Kudelski-X41-Wire-Report-phase1-20170208.pdf`
[4] `https://www.x41-dsec.de/lab/blog/fax/`
[5] `https://2018.zeronights.ru/en/reports/zero-fax-given/`
[6] `https://www.x41-dsec.de/lab/blog/smartcards/`

Acronyms