Zeal Wallet Security Review

Executive Summary

Engagement Overview

Zeal Wallet engaged Limit Break Labs to review the security of their wallet application. Specifically the chrome extension, mobile applications, smart contracts, and web platform backend. From May 6 to May 24, 2024 one consultant conducted a review of the client-provided source code, with two person-weeks of effort.

Project Scope

Our testing efforts were focused on flaws that could impact the wallet's confidentiality, integrity, or availability. We conducted this audit with full knowledge of the codebase, including access to documentation and the team's development staff.

Summary of Findings

The audit uncovered flaws in the code that impact confidentiality, integrity, or availability. However, they are all difficult to exploit.

Project Goals

The engagement was scoped to provide a review of the Zeal Wallet application. Specifically we sought to answer the following non-exhaustive list of questions:

- Can an unauthorized user send transactions from the Zeal 4337 wallet?
- Is it possible to "brick" a Zeal smart wallet contract?
- Are the passkeys generated by Zeal Wallet stored safely and confidentially from other applications on a user's device?
- Is it possible to compromise the backend Zeal web service, either by denial of service or leak of sensitive information?
- Does the wallet safely store private key accounts, in addition to passkey accounts?
- Do the target smart contracts integrate with Safe correctly?

Project Targets

The engagement involved a review and testing of the Zeal monorepo as detailed below:

Repository: https://github.com/zealwallet/monorepo
Version: 1d8c9f24f73cab20ad73f2083aeb303a7551d024

Code Evaluation Matrix

Category	Mark	Description
Access Control	Moderate	Issues relating to access control were identified, but with moderate to considerate difficulty to exploit. See issues M-1 and L-1 for details.
Mathematics	Good	The codebase does not employ complex arithmetic.

Complexity	Moderate	The codebase would benefit from additional documentation to manage complexity. For example, state transition diagrams for the React Native applications.
Decentralization	Good	The project does not have critical dependencies on any proprietary codebase or party.
Documentation	Low	The project is not well documented.
Testing and verification	Good	The project is well tested across all components.

Critical Findings

None.

High Findings

None.

Medium Findings

M-1: Malicious Chrome Extensions May Phish Zeal Wallet Passkeys

Description: The Zeal Wallet Chrome extension stores passkeys with a "relay provider ID" of `passkey.zealwallet.com`. Using domains as a relay provider ID should only be done when the origin of the client relay is the domain itself, or when no other option is available.

Any 3rd party Chrome Extension that specifies host permissions including the Zeal Wallet domain (for example, the popular `<all_urls>`) will be able to create passkeys or request passkey signatures for the `passkey.zealwallet.com` relay provider id.

Using the all URLs permission is a very common practice among common Chrome extensions, used by cryptocurrency wallets, ad blockers, language learning apps, and more. If any of these extensions were to be malicious or compromised, they can use this permission to trick a user into signing a request with their Zeal Wallet passkey. Additionally they would be able to overwrite the user's passkey with another.

The rules for what relay provider IDs may be specified by what extensions are detailed here: https://lists.w3.org/Archives/Public/public-webauthn/2023Dec/0078.html.

Recommendation: For an extension with id `id1`, store passkeys under the relay provider ID `chrome-extension://id1`. This way the credential cannot be accessed by any other extension or website.

Exploit Scenario:

Alice installs the Speechify extension, currently the top Google recommended extension on the Chrome Web Store. This extension requires the `<all_urls>` permission.

The extension is either compromised or purchased by a malicious actor who ships an update to the extension. The next time Alice opens her browser, the extension redirects her to its own html page that presents as a Zeal Wallet update.

To the user, they are forced to distinguish Speechify's ID (e.g.: "knmhgknapieeibpenaeofgffhgokikod") from Zeal's (e.g.: "gjgkofgpcmpfpggbgjgdfaaifcmoklbl"). This is quite difficult, so they accept the page as Zeal's. When the malicious application requests a signature for the user's Zeal passkey, it is granted and their wallet is compromised.

POC: We have provided an example extension that demonstrates the basic attack vector in the appendices

Dev. response: Acknowledged and fixed in #4059. Even though it's difficult to execute this type of attack, we decided to add two additional checks for the Client data passed with the passkey signature:

- Challenge offset
- Origin

Challenge offset is a hardcoded check to ensure the correctness of the layout of the client data in the signature, and origin checks against a set of whitelisted origins controlled by Zeal. This will prevent any malicious actor from abusing this attack vector, as the origin field cannot be manipulated.

Low Findings

L-1: Mobile wallet private key accounts only offer low-security protections.

Description: The Zeal Wallet application provides functionality for both private-key and passkey accounts. In the mobile application, private-key accounts are protected only by a 4 digit PIN with no option for more complex passcodes. If a user's device or or device backups were to be compromised, an attacker could easily brute force access to these accounts.

Recommendation: Allow, or even enforce, users to use more complex passcodes to protect their private key accounts.

Dev. response: Acknowledged, won't fix. It is common practice for apps to primarily rely on device security, something that can be seen with other wallet apps which have a similar level of protection, or even examples like the Uniswap wallet, which completely omits a passcode. The application will rely on a combination of the 4-digit PIN code and device-level security provided by Google / Apple devices (e.g., biometrics), cloud backups using PIN for encryption are disabled. It is assumed that the user has a secure mobile device setup.

Appendices

Below we include sample code for a chrome extension that abuses the `passkey.zealwallet.com` rpID to access and create webauthn credentials.

`manifest.json`:

```
{
  "name": "Hello World",
  "version": "0.1",
  "key":
```

```
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr1Cssen3kE1Kzw8x2vhc0neVV3Im5HhQ28cl2zptYPz

  "manifest_version": 3,
  "description": "Basic Hello World Extension",
  "host_permissions": ["<all_urls>"]
}
```

`page.html`

```
<html>
  <head>
    <title>Test Page</title>
  </head>
  <body>
    <script src="page.js"></script>
  </body>
</html>
```

`page.js`

```
async function run() {
  const publicKeyCredentialCreationOptions = {
    challenge: new ArrayBuffer(32),
    rp: {
      name: "Example",
      id: "passkey.zealwallet.com",
    },
    user: {
      id: new ArrayBuffer(32),
      name: "username",
      displayName: "displayname",
    },
    excludeCredentials: [{
      id: new ArrayBuffer(32),
      type: 'public-key',
      transports: ['internal'],
    }],
    pubKeyCredParams: [{ type: 'public-key', alg: -7 }],
    authenticatorSelection: {
      requireResidentKey: true,
      residentKey: 'required',
    }
  };

  await navigator.credentials.create({
    publicKey: publicKeyCredentialCreationOptions
  });

  const publicKeyCredentialRequestOptions = {
    challenge: new ArrayBuffer(32),
    rpId: 'passkey.zealwallet.com',
    allowCredentials: [],
    userVerification: "discouraged"
```

```
  };

  const credential2 = await navigator.credentials.get({
    publicKey: publicKeyCredentialRequestOptions
  });
  console.log(credential2);
}

run();
```