Pallad Wallet

Security Assessment

May 23rd, 2023 — Prepared by OtterSec

Caue Obici caue@osec.io

Robert Chen r@osec.io

Table of Contents

01 — Executive Summary

Overview

Palladians engaged OtterSec to assess the pallad wallet. This assessment was conducted between July 5th and July 15th, 2024. For more information on our auditing methodology, refer to chapter 07.

Key Findings

We produced 11 findings throughout this audit engagement.

In particular, we uncovered an origin spoofing vulnerability that enabled attackers to impersonate legitimate domains and zkApps (OS-PLD-ADV-00), a lack of validation bug that allowed attackers to trick users into creating session credentials with malicious purposes (OS-PLD-ADV-01) and some UI text injections that can also be used for phishing (OS-PLD-ADV-03).

We also made recommendations around the prompt window and how to improve it to prevent users from being easily scammed (OS-PLD-SUG-01, OS-PLD-SUG-02). In addition, we pointed out some non-safe data transfer behaviors that can be an issue if determined features are added to the wallet (OS-PLD-SUG-05).

02 — Scope

The source code was delivered to us in a git repository at https://github.com/palladians/pallad. This audit was performed against commit 5d17e67.

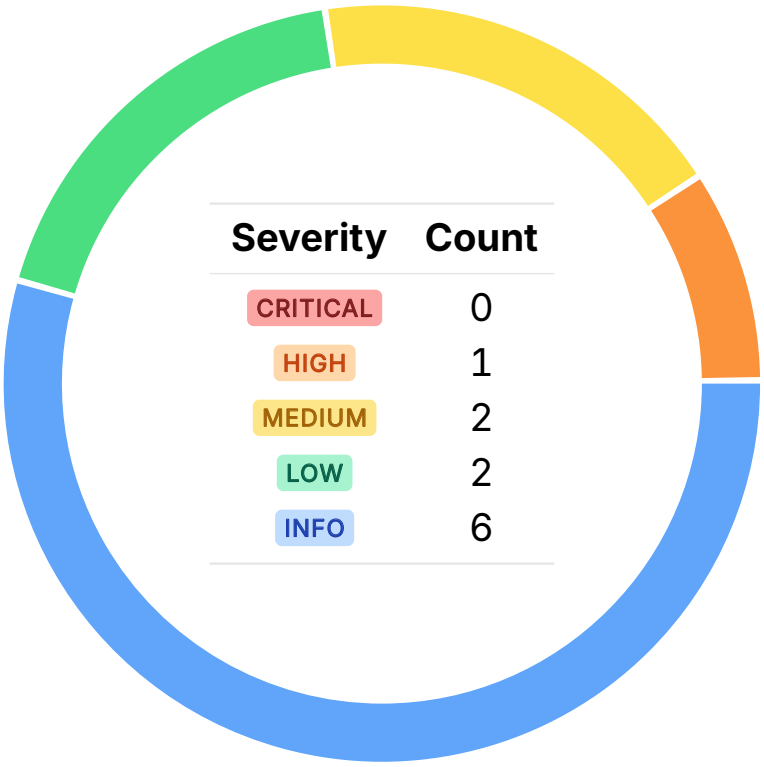
A brief description of the programs is as follows:

Name	Description
Pallad Wallet	Pallad is a MINA Network wallet that allows users to securely store their keypairs and perform actions within the MINA protocol. Using pallad it is possible to sign transactions and messages, create session credentials, create nullifiers, and see details of the account, for example, MINA balance, activities, and staking information.

03 — Findings

Overall, we reported 11 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will aid in mitigating future vulnerabilities.



Severity	Count
CRITICAL	0
HIGH	1
MEDIUM	2
LOW	2
INFO	6

04 — Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in chapter 06.

ID	Severity	Status	Description
OS-PLD-ADV-00	HIGH	RESOLVED	It is possible to spoof the origin of the request sent to the wallet.
OS-PLD-ADV-01	MEDIUM	RESOLVED	Merkle root is not validated while requesting a session using `experimental_requestSession`.
OS-PLD-ADV-02	MEDIUM	RESOLVED	It is easy to trick a user into allowing zkApps to retrieve all credentials stored.
OS-PLD-ADV-03	LOW	RESOLVED	It is possible to inject any type of message inside the prompt window.
OS-PLD-ADV-04	LOW	RESOLVED	When enabling a zkApp, the origin of the request is not shown in the prompt window.

Origin Spoofing HIGH OS-PLD-ADV-00

Description

The origin of the request is retrieved from unsafe data that can be attacker-controlled. That means that the origin can be modified by an attacker to trick users into believing that a malicious transaction is actually safe.

Also, this vulnerability can be used to bypass the `mina.enable()` requirement by using an already-enabled domain.

Proof of Concept

Executing this code inside any zkApp, the origin of the requests will be `https://example.com`:

```js
const channel = new BroadcastChannel("pallad")
channel.postMessage({respondAt: "aaa", isPallad: true, method: "enable", payload: {origin:
    ↪  "https://example.com"}})
channel.postMessage({respondAt: "aaa", isPallad: true, method: "mina_requestNetwork", payload:
    ↪  {origin: "https://example.com"}})
```

Remediation

As mitigation, it is recommended to forward the real origin of the request to the background service using `window.location.origin` inside the contentScript environment.

Patch

The fix was introduced in commit 9c045a9 by forwarding the actual `window.location.origin` to the background service:

>_ inject.ts TS

```ts
const origin = window.location.origin
    const responseChannel = new BroadcastChannel(data.respondAt)
    if (!data.isPallad)
      return responseChannel.postMessage({ error: "Wrong context" })
    let response
    try {
```

```
      const result: any = await sendMessage(
        data.method,
        { ...data.payload, origin },
        "background",
      )
```

Lack of merkle root validation MEDIUM OS-PLD-ADV-01

Description

When requesting a session credential using `experimental_requestSession`, the session parameters (data) and the session merkle root are sent to the wallet, as shown in this example:

```ts
const requestedSessionParams = {
  data: sessionParams,
  sessionMerkleRoot: toMerkleTree(sessionParams).getRoot(),
};

// Request wallet to sign the root
const accounts = await window.mina.request({ method: "mina_enable" });
const signedRoot = await window.mina.request({
  method: "experimental_requestSession",
  params: requestedSessionParams,
});
```

However, the session merkle root is not verified in the wallet background service, meaning that the zkApp can send a merkle root that is not even related to the session parameters sent in `data`.

Remediation

As mitigation, it is suggested to verify if the `sessionMerkleRoot` is actually the merkle root of the sent `data`.

Patch

The vulnerability was mitigated in commit ca5322e by removing `experimental_requestSession` RPC function.

>_ background/index.ts diff

```
- onMessage("experimental_requestSession", async (data) => {
-   return await provider.request({
-     method: "experimental_requestSession",
-     params: data,
-     sender: data.sender,
-   })
```

Object credentials retrieval MEDIUM OS-PLD-ADV-02

Description

Using `mina_getState`, it is possible to retrieve all credentials without the user knowing which ones are going to be accessed by the zkApp.

Proof of Concept

If a zkApp executes this code, it will request all object credentials stored in pallad vault, however, the prompt does not make it clear:

```js
await mina.request({
  method: "mina_getState",
  params: { query:
    {"@context": "https://www.w3.org/2018/credentials/v1"},
    props: ["id"]
  }
});
```

Remediation

Consider retrieving all the credential objects that the zkApp will have access to and showing them to the user, so they can make better decisions.

Patch

The commit 8f78cba mitigated the vulnerability by showing the requested credentials to the user inside the prompt window.

>_ mina_provider.ts diff

```
+         const { query, props } = params as Validation.GetStateData
+         const credentials = await this.vault.getState(query as any, props)
          const confirmation = await this.userPrompt("confirmation", {
            title: "Credential read request",
-           payload: JSON.stringify(params),
+           payload: JSON.stringify({ ...params, credentials }),
```

Message injection LOW OS-PLD-ADV-03

The prompt window uses data provided by the zkApp to show information, meaning that this information can be manipulated by attackers. In addition, the information sent by the zkApp is not parsed and every data is shown in the prompt window, making it easier for attackers to trick users into signing malicious transactions.

Proof of Concept

This request will show a warning field within the prompt window saying the transaction is safe:

```js
mina.request({
  method: "mina_signTransaction",
  params: {
    // [malicious transaction...]
    warning: "This is not a malicious transaction. It is verified by mina protocol and 100%
        ↪  secure"
  }
})
```

Remediation

The recommended fix is to parse all data sent to the background service and show only the relevant data to the user. While parsing data, it is suggested to check data types and formats to prevent type confusion attacks and message injections on other inputs.

Patch

The mitigation was introduced in commit ca5322e by adding zod validations schemas and parsing them on each RPC function, for example in `mina_signTransaction`:

>_ background/index.ts TS

```ts
onMessage("mina_signTransaction", async ({ data }) => {
  try {
    const params = Validation.signTransactionRequestSchema.parse(data)
    return await provider.request({
      method: "mina_signTransaction",
      params,
    })
```

Origin is hidden in zkApp enable LOW OS-PLD-ADV-04

When enabling a domain using `mina.enable()` the origin that is trying to be connected is not shown to the user. This is due to the way how the `this.userPrompt` function is called in the enable flow:

>_ *mina-provider.ts* TS

```ts
const userConfirmed = await this.userPrompt("confirmation", {
  title: "Connection request.",
  payload: JSON.stringify({ origin }),
})
```

The prompt window gets the origin that will be shown using `payload.data.origin`, however, `payload.data` is undefined in this case.

Since the origin is not shown when enabling the zkApp, the user does not know what they are signing, meaning that they could allow malicious apps to retrieve information about their accounts and request malicious transactions.

Remediation

Update the payload to the correct format that the application is using:

>_ *mina-provider.ts* TS

```ts
const userConfirmed = await this.userPrompt("confirmation", {
  title: "Connection request.",
  payload: JSON.stringify({ data: { origin } }),
})
```

Patch

The fix was committed on 8f78cba by normalizing all the data being sent to the prompt window using `chrome.runtime.sendMessage`, as suggested in OS-PLD-SUG-05.

05 — General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may result in security issues in the future.

ID	Description
OS-PLD-SUG-00	All network accounts are deleted when ensuring an account at an already populated network.
OS-PLD-SUG-01	Simulate transactions to show more details to the user before they sign the transaction.
OS-PLD-SUG-02	Parse data and add more details in the prompt window to users understand what they are signing.
OS-PLD-SUG-03	There are known vulnerabilities in NPM packages and they are susceptible to supply-chain attacks.
OS-PLD-SUG-04	Make brute force attacks harder to succeed by adding more computational cost in PBKDF2.
OS-PLD-SUG-05	Use postMessage to send data instead of using query string parameters in the prompt window.

Accidental account deletion OS-PLD-SUG-00

Description

When ensuring an account on an already populated network, all accounts in this network are deleted:

>_ *accountStore.ts* TS

```ts
    if (!state.accounts[network]?.[address]) {
        state.accounts[network] = {}
        state.accounts[network][address] = {
        ...initialSingleAccountState,
        ...state.accounts[network][address],
        }
    }
```

Currently, it is not possible to have multiple addresses in the same network, so this doesn't immediately affect the extension usability, however, it is recommended to fix it to prevent future problems.

Remediation

To fix it, the network property should be overwritten only when it is `undefined`:

>_ *accountStore.ts* TS

```ts
    if (!state.accounts[network]?.[address]) {
        if (!state.accounts[network]){
            state.accounts[network] = {}
        }
        state.accounts[network][address] = {
        ...initialSingleAccountState,
        ...state.accounts[network][address],
        }
    }
```

Transaction simulation

OS-PLD-SUG-01

Description

When signing a transaction in pallad wallet, it does not show how many FTs and NFTs are being sent and also does not show how many MINA tokens are going to be received.

Remediation

As a mitigation, consider simulating transactions to show detailed information about balance changes to the user, so they can understand what they are signing.

If in the future pallad will support NFTs and FTs, the recommendation is to show balance changes of those tokens. On the other hand, if it is not something pallad will implement, simulating MINA balance changes should be enough.

Improve experimental_requestSession UX OS-PLD-SUG-02

Description

When a zkApp sends a `experimental_requestSession` RPC call, the prompt shown to the user does not contain much information about what the session is and why it is dangerous to sign it. In addition, the session data is not checked and validated, this includes the expiration time and the merkle root.

The expiration time should be validated to see if it is not before the current time and if the expiration window is not too long. If that is the case, it should warn the user against this time.

The session data is not validated against the `sessionMerkleRoot`, meaning that the zkApp can provide legitimate data with a malicious merkle root to be signed. This would trick the user into signing a malicious session credential (detailed in OS-PLD-ADV-01).

Remediation

As mitigation, consider adding more details and warnings to the prompt window and validating the expiration time and the merkle root.

The prompt window should warn the users about the risks of signing a credential to a malicious zkApp and also warn about the expiration time. Since the session credential is not secured by a password, if it leaks somehow in the future and is still valid, it can lead to loss of funds.

In addition, make sure to parse all the information received and show the details to the user.

Update and pin NPM packages OS-PLD-SUG-03

Description

The result of `pnpm audit` shows that vulnerable packages are being used in the application:

```bash
| high | ws affected by a DoS when handling a request with many HTTP headers |
| --- | --- |
| Package | ws |
| Vulnerable versions | >=8.0.0 <8.17.1 |
| Patched versions | >=8.17.1 |
| Paths | |
| More info | https://github.com/advisories/GHSA-3h5v-q93c-6h6q |

...
8 vulnerabilities found
Severity: 3 moderate | 5 high
```

In addition, all the used packages are not pinned, meaning that a malicious update in a package could compromise the entire wallet.

Remediation

It is recommended to update all vulnerable packages to a version in which the vulnerabilities are already fixed and pin all the dependencies to lower the likelihood of a supply-chain attack.

Add more PBKDF2 iterations OS-PLD-SUG-04

Description

OWASP recommends the use of 210k PBKDF2 iterations to provide more security against brute force attacks, however, pallad is using 19k iterations only:

>_ *emip3.ts* TS

```ts
const PBKDF2_ITERATIONS = 19_162
const SALT_LENGTH = 32

export const createPbkdf2Key = async (
  passphrase: Uint8Array,
  salt: Uint8Array | Uint16Array,
) => {
  const saltAsUint8Array = new Uint8Array(salt.buffer)
  const derivedKey = await pbkdf2Async(sha512, passphrase, saltAsUint8Array, {
    c: PBKDF2_ITERATIONS,
    dkLen: KEY_LENGTH,
  })
  return derivedKey
}
```

Remediation

It is recommended to increase the number of iterations by the maximum value possible before starting to decrease performance and UX.

Remove data from query string

OS-PLD-SUG-05

Description

The prompt window uses a query string to retrieve information about the title, payload, and `windowId`:

>_ prompts.ts TS

```ts
const fullUrl = `prompt.html?title=${encodeURIComponent(
    metadata.title,
)}&payload=${encodeURIComponent(
    metadata.payload ?? "",
)}&inputType=${inputType}&windowId=${newWindow.id}`

await chrome.tabs.update(tabId, { url: fullUrl })
```

There are some tricks to open a `chrome-extension://` page in chrome using javascript but it has some requirements. It was not found any way of doing this in the pallad extension, however, it is recommended to remove data from the query string since it is potentially attacker-controllable.

Remediation

Consider using `chrome.runtime.postMessage` instead of using the query string to pass information.

06 — Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings may be found in the General Findings.

CRITICAL

Vulnerabilities that immediately result in a loss of user funds with minimal preconditions.

Examples:

- Misconfigured authority or access control validation.
- Improperly designed economic incentives leading to loss of funds.

HIGH

Vulnerabilities that may result in a loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions.
- Exploitation involving high capital requirement with respect to payout.

MEDIUM

Vulnerabilities that may result in denial of service scenarios or degraded usability.

Examples:

- Computational limit exhaustion through malicious input.
- Forced exceptions in the normal user flow.

LOW

Low probability vulnerabilities, which are still exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions.

INFO

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants.
- Improved input validation.

07 — Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the program's implementation requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of thumb, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to comprehensively understand the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the others may have missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.