Security Research Labs

Polymesh Chrome wallet review

Threat model and penetration test report

V1.0, May 12, 2021

Sina Yazdanmehr sina@srlabs.de

Genco Altan Demir genco@srlabs.de

Abstract. This study describes the results of a thorough, independent baseline security assurance audit of the Polymesh Chrome wallet by Security Research Labs. In the course of this study, Polymath provided full access to relevant documentation and supported the research team effectively.

The protection of Polymesh was independently verified to assure that existing hacking risks are understood and minimized.

The research team identified several issues ranging from low to high risk and Polymath addressed them quickly. Most of the issues are no longer present in the latest development version of Polymesh Chrome wallet.

To further improve the security of the wallet, we recommend implementing a centralized authorization control. In addition, we recommend taking advantage of best practices around Chrome extension security and web application security as well as using the most updated versions of imported libraries.

Content

1 Motivation and scope

This penetration test assesses the Polymesh Chrome wallet from an attacker perspective to discover and understand potential business logic and technical vulnerabilities that may exist in the wallet with the goal of improving the protection capabilities of the wallet.

Since the wallet possesses valuable data, an attacker might aim to compromise the target wallet and/or the web browser to steal or alter the data. During the test, different attack vectors were used on the wallet to ensure implemented security mechanisms are reliable, and the data is stored securely.

This report details the penetration test results with the aim of creating transparency in two steps:

Threat Model. The threat model is considered in terms of *hacking incentives*, i.e. the motivations to achieve the goals of breaching the integrity, confidentiality, or availability of the wallet and the data stored on it. For each hacking incentive, we postulate *hacking scenarios,* by which these goals could be reached. The threat model provides guidance for the design, implementation, and security testing of the Polymesh Chrome wallet.

Penetration test. For each hacking scenario established in the threat model, a penetration test was performed on the wallet to probe for vulnerabilities. Due diligence was applied to each hacking scenario to ensure the results were comprehensive. Polymesh Chrome wallet is built upon Polkadot{.js}, a browser extension that injects a Polkadot API signer into a page. The imported libraries and Polkadot{.js} extension were not included in this assessment's scope.

2 Methodology

To be able to effectively perform a penetration test on the wallet, a threat model driven white-box penetration testing was employed. For each identified threat, hypothetical attacks were developed and mapped to their respective threat category as outlined in Section 3.

Additionally, proof-of-concept exploit code was developed for each identified threat to help the Polymesh team with reproducing attacks while implementing mitigations.

Prioritized by risk, the codebase was assessed for present protections against the respective threats and attacks as well as the vulnerabilities that make these attacks possible. We used the *develop*[1] branch of the Polymesh Chrome wallet repository as the basis for the review. Since the Polymesh Chrome wallet codebase is entirely open source, it is realistic that a malicious actor would analyze the source code while preparing an attack.

For each threat, the penetration tester:

1. identified the relevant parts of the codebase, viable strategies for the code review, and potential applicable attack vectors.

2. identified (if applicable) the request that triggers the target function. This function was then subjected to different requests with different fuzzed and not-expected inputs, with the goal of breaching the integrity, confidentiality, or availability of the wallet.

[1] Commit: 55ca562

3. immediately reported any vulnerability that was discovered to the development team along with a proof-of-concept as well as suggestions around mitigations.

During the penetration test, we carried out a hybrid strategy utilizing a combination of code review and dynamic tests to assess the security of the Polymesh Chrome wallet codebase.

3 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in the Polymesh Chrome wallet. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, as well as the security testing. This section introduces how risk is defined and provides an overview of the identified threat scenarios. The *Hacking Value*, categorized into *low*, *medium*, and *high*, takes into account the incentive of an attacker, as well as the effort required by an attacker to successfully execute the attack. The hacking value is calculated as:

$$Hacking\ Value = \frac{Incentive}{Effort}$$

While *incentive* describes what an attacker might gain from preforming an attack successfully, *effort* estimates the complexity of this same attack. The degrees of incentive and effort are defined as follows:

Incentive:

- Low: Attacks offer the hacker little to no gain from executing the threat.
- Medium: Attacks offer the hacker considerable gains from executing the threat.
- High: Attacks offer the hacker high gains by executing this threat.

Effort:

- Low: Attacks are easy to execute. They require neither elaborate technical knowledge nor considerable amounts of resources.
- Medium: Attacks are somewhat difficult to execute. They might require bypassing countermeasures, the use of expensive resources or a considerable amount of technical knowledge.
- High: Attacks are difficult to execute. The attacks might require in-depth technical knowledge, vast amounts of expensive resources, bypassing countermeasures, or any combination of these factors.

After applying the framework to the Polymesh Chrome wallet, different threat scenarios were identified. Table 1 provides a high-level overview of the threat model with identified example threat scenarios and attacks, as well as their respective hacking value and effort.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confi-dentiality	High	- Unencrypted private keys should not be accessible outside of the wallet - The password used to encrypt the private key should not be disclosed - Password of encrypted exported data should not be discoverable outside of the wallet - Password of the private key encryption should not be discoverable outside of the wallet - Deanonymize the victim - An app should not be able to access the wallet without user's permission	Medium	- Bypass authorization/authentication mechanisms to access confidential information
Integrity	Medium	- Internal behavior of the wallet should not be influenced by other apps - The details of the transaction being signed by the wallet and shown to the user, must match what is actually signed - Network communication should be resistant against of the Man-In-The-Middle attack - Importing backed up data (potentially altered) should not destroy existing data.	Medium	- Fake a transaction - Compromise network connection, e.g. Man-In-The-Middle attack
Availability	Low	- An unauthorized app/user should not be able to alter or delete the wallet's data	Low	- Remove wallet's data without having permission

Table 1. Threat scenario overview. The threats for Polymesh's Chrome wallet were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

4 Findings summary

We identified eight issues - summarized in Table 2 - during our analysis of the runtime modules in scope in the Polymesh Chrome wallet codebase that enable the attacks outlined above. In summary, one high severity, one medium severity and six low severity issues were found. Most of the vulnerabilities were already mitigated by the Polymath team, as detailed in this document. Polymath decided to accept the risk posed by the weak password policy (Section 4.3).

Issue description	Severity	Reference	Remediation
Issue #1, Insufficient authorization in some methods which could result in user identity deanonymization	High	[1] [2]	polymesh-wallet#128 and polymesh-wallet#132
Issue #2, An unauthorized application can send the "*pub(authorize.tab)* request, which launches a new popup each time, without any limitation.	Medium	[3]	polymesh-wallet#139
Issue #3, Weak password policy	Low	N/A	N/A
Issue #4, Insufficient authorization/authentication on the "Delete account" feature	Low	[4]	polymesh-wallet#134
Issue #5, *extrinsicSign()* lacks an appropriate error handling mechanism	Low	[5]	polymesh-wallet#131
Issue #6, Sensitive information, e.g. the DID, is exposed in the console logs	Low	[6]	polymesh-wallet#130
Issue #7, There is no length control for the "*origin*" parameter in the notification page.	Low	[7]	polymesh-wallet#133
Issue #8, Imported third party libraries are not up to date	Low	[8]	polymesh-wallet#137

Table 2. Overview of identified issues

4.1 Insufficient authorization in some methods could expose user identity

Risk. Insufficient authorization in the following methods allows an unauthorized app to call them without user's permission:

- *poly:pub(network.get)*
- *poly:pub(network.subscribe)*
- *pub(accounts.list)*
- *pub(accounts.subscribe)*
- *pub(metadata.list)*
- *pub(metadata.provide)*
- *poly:pub(uid.requestProof)*

An unauthorized app can call the specified methods to retrieve sensitive information such as the list of the victim's accounts. An attacker may use the leaked details coupled with other information, e.g. the victim's IP address, to carry on other attacks such as deanonymizing the victim.

Additionally, unauthorized access to the *pub(accounts.list)* method may enable an attacker to link different accounts with low effort.

Mitigation. A sufficient authorization mechanism should be implemented in a similar way to other API calls. The wallet should first check the request's originator, and ensure it is a trusted application, before performing the request.

4.2 Unauthorized applications can cause DoS by launching popups

Risk. An unauthorized malicious app may send many the "pub(authorize.tab)" request in a row to cause a denial of service, since each request launches a new

popup. The app might send this request in a large volume to cause the victim's browser, or even the OS, to crash.

Mitigation. The wallet should reject a new request if there is already a pending request from the same origin. This method should be used for any request, specifically if the request makes a load.

4.3 Weak password policy

Risk. Allowing users to choose weak passwords increases the chance of a successful bruteforce attack, that might lead to a complete wallet takeover.

Mitigation. A stronger password policy should be implemented. Ensure users choose a longer password, minimum 12 chars, and optionally add uppercase, numbers and punctuation signs requirements.

4.4 Insufficient authentication on "Delete account" feature

Risk. The wallet does not authenticate the user (e.g. by requiring password entry) before allowing the user to delete an account.

An unauthorized attacker who has physical access to a victim's device would be able to delete the victim's accounts without knowing the accounts' passwords. The victim might lose her account if she does not have a backup of the account or the recovery phrase.

Mitigation. The wallet should re-authenticate the user before authorizing account deletion. To accomplish this, the wallet may ask the account's password, verify it, and then delete the account if the password was correct.

4.5 *extrinsicSign()* lacks an appropriate error handling mechanism

Risk. Error messages shown on the notification popup and on the main page persist on the main page and cause the wallet not to function until the user restarts the browser. This can be caused by a request that contains malformed *era*, *version* or *nonce*, or a combination of these parameters.

An authorized app may cause a temporary denial of service by sending a malformed *extrinsic.sign* request, and force the victim to restart her browser to able to use the wallet again.

Mitigation. The wallet should eliminate the error situation once it is shown to the user, or when the user clicks on the "reload".

4.6 Sensitive information, e.g. the DID, is exposed in the console logs

Risk. An attacker who has physical access to a victim's device, would be able to access the information exposed in the logs via the Chrome inspector.

Mitigation. Sensitive information should not be exposed in logs, error messages, etc.

4.7 Missing length control for the "*origin*" parameter in notification page

Risk. An application can set the origin to any value with arbitrary length.

Since the notification page has a fixed size, a malicious app might set a long crafted text as the origin to push the original text. An attacker might trick a victim to authorize a request by exploiting this issue.

Mitigation. Check length of the "origin" parameter, e.g. ensure it is not longer than 30 characters, before rendering it on the notification page.

4.8 Imported third party libraries are not up to date

Risk. An attacker might exploit a known vulnerability in one of the imported third party libraries.

Mitigation. Ensure the most up-to-date version of third-party libraries are imported.

5 Evolution suggestion

We recommend taking the following measures to harden the wallet against potential security vulnerabilities introduced in future development.

Use a centralized authorization control. During the review, it turned out that there is no centralized authorization control implemented to authorize requests before executing the request. This weakness was the root cause of Issue #1 (Section 4.1). Although the Polymesh team has addressed Issue #1, we recommend implementing a centralized authorization control to authorize all requests before executing them.
This control can mitigate unauthorized access issues in the future if the Polymesh team adds a new method to the wallet.

Follow a web application security guideline (e.g., OWASP) in future developments. Many client-side web application attacks, e.g. cross-site scripting (XSS), might be applicable to the wallet. To protect against such potential security issues we recommend to follow both an updated web application security guideline [9] as well as Chrome extension developer guidelines issued by Google [10].

Maintain up-to-date versions of third-party libraries. Since the wallet has been built upon different libraries and the Polkadot{.js} extension, we recommend implementing regular updates to ensure the most up-to-date versions of those libraries are in use. A known vulnerability in an older version of an imported library might enable an attacker to carry on an attack against of the wallet, or its data.

6 References

1. **Polymesh Chrome wallet, issue #1,**
   https://github.com/PolymathNetwork/polymesh-wallet/pull/128
2. **Polymesh Chrome wallet, issue #1,**
   https://github.com/PolymathNetwork/polymesh-wallet/pull/132
3. **Polymesh Chrome wallet, issue #2,**
   https://github.com/PolymathNetwork/polymesh-wallet/pull/139

4. **Polymesh Chrome wallet, issue #4,**
https://github.com/PolymathNetwork/polymesh-wallet/pull/134

5. **Polymesh Chrome wallet, issue #5,**
https://github.com/PolymathNetwork/polymesh-wallet/pull/131

6. **Polymesh Chrome wallet, issue #6,**
https://github.com/PolymathNetwork/polymesh-wallet/pull/130

7. **Polymesh Chrome wallet, issue #7,**
https://github.com/PolymathNetwork/polymesh-wallet/pull/133

8. **Polymesh Chrome wallet, issue #8,**
https://github.com/PolymathNetwork/polymesh-wallet/pull/137

9. **OWASP Application Security Verification Standard,**
https://owasp.org/www-project-application-security-verification-standard/

10. **Google Chrome extension development guideline,**
https://developer.chrome.com/docs/extensions/mv3/security/