Solflare MetaMask Snaps - Solflare, Sui, Aptos

Date	August 2023
Auditors	Martin Ortner

1 Executive Summary

This report presents the results of our engagement with **Solflare Wallet** to review three **MetaMask Snaps**, the **aptos-snap** for Rise Wallet, the **solana-snap** for Solflare, and the **sui-snap** for Elli Wallet.

The review was conducted from **Aug 15, 2023** to **July 25, 2023**. A total of 9 person-days were spent.

2 Scope

The review focused on the following repositories and commit hashes:

- solflare-snap@792fcb2a253c572677271044ece061daae81ce41
- sui-snap@39740383fd11174a406a1a27e15c92f277d00779
- aptos-snap@36145fbb8f4963eb02cff26a549ac348a34d8cb3

The dapp wallets interfacing with the snap are not in scope of this review.

The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the client, we identified the following priorities for our review:

1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify vulnerabilities particular to the MetaMask Snaps SDK integration in coherence with the MetaMask Snap Threat Model describing a Snap as an extension of the MetaMask Wallet Trust Module.

3 Snap Outlines

- The three snaps share certain parts of the codebase (duplications) and expose similar functionality to dapps.
  - `ui.js` - identical
  - `privateKey.js` - identical (ex. coinId)
  - `index.js` - logically similar: diff coinSpecifics, origin checks
  - `utils.js` - aptos with additional functions: hex2bytes, bytes2hex
  - auxfiles: logically similar
- The three snaps individually request sensitive BIP32 Entropy (SLIP-44), effectively managing the following coins private root key:
  - `m/44'/501'` (Solana)
  - `m/44/784'` (Sui)
  - `m/44'/637'` (Aptos)
- Connected dapps can communicate with the snap via MetaMask snap RPC. Dapp origins are enforced as follows:
  - Solana: localhost, *solflare.com, *solfalre.dev
  - Sui: localhost, *elliwallet.dev
  - Aptos: localhost, *risewallet.dev
- Other snaps cannot directly communicate with one of the three snaps under review.
- The public key is exposed to connected dapps without additional user confirmation if the dapp chooses so.
- The private key can not be exported to the RPC origin.
- Transactions/Messages are signed within the realm of the snap.
- The snap may display custom MetaMask dialogs.

3.1 Details: Solflare Snap

```
[🚀 == FsChecks == ]
    ℹ️  undefined
    🔺 - no linter config
[🚀 == Manifest == ]
    🔺 - bundle (dist/bundle.js) does not not exist!
🚨 -- Manifest errors --
    🔺 Error: Failed to read Snap bundle file: ENOENT: no such file or directory, open '/Users/tintin/workspace/solidity/solflare-wallet-snaps-audit-26
    🔺 - package.json: metamask template package.conf repo.url not correct 'undefined'
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
endowment:rpc { dapps: true, snaps: false }
snap_dialog {}
snap_getBip32Entropy [ { path: [ 'm', "44'", "501'" ], curve: 'ed25519' } ]
---->%---- raw permissions
🚜 [endowment:rpc]
    ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
       🔶  src/index.js
🚜 [snap_dialog]
    ☝️ - snap_dialog - Displays a dialog in the MetaMask UI. There are three types of dialogs with different parameters and return types.
    ⚠️ - this method renders Markdown! check for ctrlchar/markdown/injection
       🔶  src/ui.js
🚜 [snap_getBip32Entropy]
    ✨ - snap_getBip32Entropy - Gets the SLIP-10 key for the path and curve specified by the method name.
       ⚠️ - If you call this method, you receive the user's parent key for the derivation path they request. You're managing the user's keys and assets
       🔶  src/privateKey.js
🎄 - Package Depenencies:
     - @babel/runtime:^7.21.5
     - @metamask/key-tree:^7.0.0              (🔺 looks like devDependency 👀)
     - @metamask/snaps-ui:^0.32.2             (🔺 looks like devDependency 👀)
     - bs58:^5.0.0
     - tweetnacl:^1.0.3
[🚀 == Bundle == ]
    🟢 Package.json OK
⚠️  Package.json Warnings:
    🔺 missing keywords
    🔺 missing bugs
    🔺 missing homepage
    🔺 missing repository
```

3.2 Details: Sui Snap

```
[🚀 == FsChecks == ]
    ℹ️  undefined
    🔺 - no linter config
[🚀 == Manifest == ]
    🔺 - bundle (dist/bundle.js) does not not exist!
🚨 -- Manifest errors --
    🔺 Error: Failed to read Snap bundle file: ENOENT: no such file or directory, open '/Users/tintin/workspace/solidity/solflare-wallet-snaps-audit-26
    🔺 - package.json: metamask template package.conf repo.url not correct 'undefined'
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
endowment:rpc { dapps: true, snaps: false }
snap_dialog {}
snap_getBip32Entropy [ { path: [ 'm', "44'", "784'" ], curve: 'ed25519' } ]
---->%---- raw permissions
🚜 [endowment:rpc]
    ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
       🔶  src/index.js
🚜 [snap_dialog]
    ☝️ - snap_dialog - Displays a dialog in the MetaMask UI. There are three types of dialogs with different parameters and return types.
    ⚠️ - this method renders Markdown! check for ctrlchar/markdown/injection
       🔶  src/ui.js
🚜 [snap_getBip32Entropy]
    ✨ - snap_getBip32Entropy - Gets the SLIP-10 key for the path and curve specified by the method name.
       ⚠️ - If you call this method, you receive the user's parent key for the derivation path they request. You're managing the user's keys and assets
       🔶  src/privateKey.js
🎄 - Package Depenencies:
     - @babel/runtime:^7.21.5
     - @metamask/key-tree:^7.0.0              (🔺 looks like devDependency 👀)
     - @metamask/snaps-ui:^0.32.2             (🔺 looks like devDependency 👀)
     - @noble/hashes:^1.3.1
     - base64-js:^1.5.1
     - tweetnacl:^1.0.3
[🚀 == Bundle == ]
    🟢 Package.json OK
⚠️  Package.json Warnings:
    🔺 missing keywords
    🔺 missing bugs
    🔺 missing homepage
    🔺 missing repository
```

3.3 Details: Aptos Snap

```
[🚀 == FsChecks == ]
   ℹ️  undefined
   🔺 - no linter config
[🚀 == Manifest == ]
   🔺 - bundle (dist/bundle.js) does not not exist!
🚨 -- Manifest errors --
   🔺 Error: Failed to read Snap bundle file: ENOENT: no such file or directory, open '/Users/tintin/workspace/solidity/solflare-wallet-snaps-audit-26
   🔺 - package.json: metamask template package.conf repo.url not correct 'undefined'
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
endowment:rpc { dapps: true, snaps: false }
snap_dialog {}
snap_getBip32Entropy [ { path: [ 'm', "44'", "637'" ], curve: 'ed25519' } ]
---->%---- raw permissions
🚜 [endowment:rpc]
   ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
      🔶  src/index.js
🚜 [snap_dialog]
   ☝️ - snap_dialog - Displays a dialog in the MetaMask UI. There are three types of dialogs with different parameters and return types.
   ⚠️ - this method renders Markdown! check for ctrlchar/markdown/injection
      🔶  src/ui.js
🚜 [snap_getBip32Entropy]
   ✨ - snap_getBip32Entropy - Gets the SLIP-10 key for the path and curve specified by the method name.
      ⚠️ - If you call this method, you receive the user's parent key for the derivation path they request. You're managing the user's keys and assets
      🔶  src/privateKey.js
🎄 - Package Depenencies:
     - @babel/runtime:^7.21.5
     - @metamask/key-tree:^7.0.0              (🔺 looks like devDependency 👀)
     - @metamask/snaps-ui:^0.32.2             (🔺 looks like devDependency 👀)
     - tweetnacl:^1.0.3
[🚀 == Bundle == ]
   🟢 Package.json OK
⚠️ Package.json Warnings:
   🔺 missing keywords
   🔺 missing bugs
   🔺 missing homepage
   🔺 missing repository
```

4 Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

4.1 Dapp May Force a Sign Approval Dialog Without Showing the Message to Be Signed Major ✓ Fixed

Resolution

Addressed by always displaying the raw message to be signed to the user. This allows them to independently verify that this is indeed what they want to sign off on. Additionally, the client provided the following statement and changesets addressing the finding:

> 4.1 - always display raw transaction payload

Changesets:

- solflare-wallet/solflare-snap@ `9537098`
- solflare-wallet/aptos-snap@ `b857c16`
- solflare-wallet/sui-snap@ `d28c4e3`

Description

With the `request.params.displayMessage` parameter in requests to `signTransaction` and `signAllTransactions` the dapp controls if the message to be signed is displayed to the user or not. Allowing the dapp to control if the data to be signed is displayed to the user is dangerous as the dapp may silently ask for a signature to sign data the user did not intend to sign. This has potential to undermine security controls and procedures implemented by MetaMask which generally enforce clarity of what data the user is requested to sign.

Note that the snap as an extension to the MetaMask trust module should not have to trust the dapp that is requesting signature.

Examples

Affects all snaps under review.

Solflare Snap

../aptos-snap/src/index.js:L39-L51

```
const { derivationPath, message, simulationResult = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(message);
assertIsString(message);
assertIsArray(simulationResult);
assertAllStrings(simulationResult);
assertIsBoolean(displayMessage);

const accepted = await renderSignTransaction(dappHost, message, simulationResult, displayMessage);
assertConfirmation(accepted);
```

../aptos-snap/src/ui.js:L28-L33

```
    text(host),
    ...(simulationResultItems.length > 0 || displayMessage ? [divider()] : []),
    ...simulationResultItems,
    ...(displayMessage ? [copyable(message)] : [])
  ])
}
```

Recommendation

In accordance with how MetaMask signing works, it is highly recommended to remove the `displayMessage` toggle and **consistently** enforce the message to be signed to be displayed. Else, there is no way for the user to verify if they are signing the correct data/transaction.

4.2 CtrlChar/Markdown Injection in `renderSignTransaction`, `renderSignAllTransactions`

Major ✓ Fixed

Resolution

The identified vulnerability has been remedied by excluding the simulation result from the dialog. It is essential to emphasize that the extension should ideally generate the simulation result as a validated source of information. The transmission of this information from a less trustworthy entity (even though it pertains to the dapp linked to the snap) to be displayed within the secure context of a snap, influencing users to endorse raw data based on this simulation result, may inherently introduce security risks. Eliminating the simulation result parameter from the RPC request effectively mitigates this specific injection opportunity. Additionally, the client has furnished the subsequent statement and alterations to address the aforementioned issue:

> 4.2 - fully remove simulation results. This removes the possibility to inject markdown or control characters in unsafe places. Given the multi-entrypoint argument and the lack of a straightforward way to parse and simulate transactions within the snap, we think raw transaction payloads provide an undeniable source of truth and let users parse those transactions with external tools. It is on our roadmap to see how this can be improved in the future

Changesets:

- solflare-wallet/solflare-snap@ `9537098`
- solflare-wallet/aptos-snap@ `b857c16`
- solflare-wallet/sui-snap@ `d28c4e3`

Description

On certain occasions, the snap may need to present a dialog to the user to request confirmation for an action or data verification. This step is crucial as dapps are not always trusted, and it's essential to prevent scenarios where they can silently sign data or perform critical operations using the user's keys without explicit permission. To create custom user-facing dialogs, MetaMask provides the Snaps UI package, equipped with style-specific components. However, some of these components have been found to have unintended side-effects.

For instance, the `text()` component can render Markdown or allow for control character injections.

In the code snippet provided below, please note that `request.params` is considered untrusted. For example, `request.params.simulationResult[]` may contain Markdown renderable strings or Control Characters that can disrupt the context of the user-displayed message.

- `request.params.origin` is validated via `(new URL(dappOrigin))?.host`; However, note, that the dapp may misrepresent the original origin!
- `requests.params.message` is safely put into a `copyable`
- `request.params.simulationResult` is **unchecked**

Please also note that the user might decide whether to sign or reject the transaction based on the simulation that is being displayed. This simulation result, however, is directly provided by the dapp which is less trusted than the metamask security module/snap. This might lead to users signing data based on potentially false information if the dapp provides malicious information. It should, therefore, be considered to generate the simulation information within the snap itself!

Examples

This affects all snaps under review.

Solflare Snap

- `signTransaction`

../solflare-snap/src/index.js:L41-L53

```
const { derivationPath, message, simulationResult = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(message);
assertIsString(message);
assertIsArray(simulationResult);
assertAllStrings(simulationResult);
assertIsBoolean(displayMessage);

const accepted = await renderSignTransaction(dappHost, message, simulationResult, displayMessage);
assertConfirmation(accepted);
```

../solflare-snap/src/ui.js:L19-L35

```
export function renderSignTransaction(host, message, simulationResult, displayMessage = true) {
  const simulationResultItems = simulationResult.map((item) => text(item));

  return snap.request({
    method: 'snap_dialog',
    params: {
      type: 'confirmation',
      content: panel([
        heading('Sign transaction'),
        text(host),
        ...(simulationResultItems.length > 0 || displayMessage ? [divider()] : []),
        ...simulationResultItems,
        ...(displayMessage ? [copyable(message)] : [])
      ])
    }
  });
}
```

- `renderSignAllTransactions`

../solflare-snap/src/ui.js:L51-L55

```

simulationResults[i].forEach((item) => uiElements.push(text(item)));
if (displayMessage) {
  uiElements.push(copyable(messages[i]));
}
```

Sui Snap

../sui-snap/src/index.js:L41-L52

```
const { derivationPath, message, simulationResult = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(message);
assertIsString(message);
assertIsArray(simulationResult);
assertAllStrings(simulationResult);
assertIsBoolean(displayMessage);

const accepted = await renderSignTransaction(dappHost, message, simulationResult, displayMessage);
assertConfirmation(accepted);
```

../sui-snap/src/index.js:L64-L77

```
const { derivationPath, messages, simulationResults = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(messages);
assertIsArray(messages);
assertInput(messages.length);
assertAllStrings(messages);
assertIsArray(simulationResults);
assertInput(messages.length === simulationResults.length);
assertIsBoolean(displayMessage);

const accepted = await renderSignAllTransactions(dappHost, messages, simulationResults, displayMessage);
assertConfirmation(accepted);
```

Aptos Snap

../aptos-snap/src/index.js:L39-L50

```
const { derivationPath, message, simulationResult = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(message);
assertIsString(message);
assertIsArray(simulationResult);
assertAllStrings(simulationResult);
assertIsBoolean(displayMessage);

const accepted = await renderSignTransaction(dappHost, message, simulationResult, displayMessage);
assertConfirmation(accepted);
```

../aptos-snap/src/index.js:L62-L75

```
const { derivationPath, messages, simulationResults = [], displayMessage = true } = request.params || {};

assertInput(derivationPath);
assertIsString(derivationPath);
assertInput(messages);
assertIsArray(messages);
assertInput(messages.length);
assertAllStrings(messages);
assertIsArray(simulationResults);
assertInput(messages.length === simulationResults.length);
assertIsBoolean(displayMessage);

const accepted = await renderSignAllTransactions(dappHost, messages, simulationResults, displayMessage);
assertConfirmation(accepted);
```

Recommendation

Validate inputs. Encode data in a safe way to be displayed to the user. Show the original data provided within a pre-text or code block (copyable). Show derived or decoded information (token recipient) as additional information to the user. If possible, generate a trusted simulation result within the snap.

4.3 Insufficient Input Validation `deriveKeyPair()` Medium

Resolution

The client has issued the following statement:

> Solflare Comment/Background
>
> We have inspected the @metamask/key-tree library and made sure that all incorrect inputs throw errors. This is the reason why we didn't duplicate the checks. The reason why we supply full paths is because the APIs of our Wallet forward full paths and to avoid sensitive splitting of paths in various places.
>
> Solflare Resolution
>
> We respect the recommendation, but due to architectural decisions up the stack, we would like to keep the deriveKeyPair() API as-is.

Statement from the Assessment Team:

The principle of defense in depth necessitates comprehensive scrutiny across all layers of the system. It is unwise to rely solely on a third-party library to handle error propagation, as you lack control over the underlying codebase. We strongly advocate for the pervasive implementation of input validation mechanisms in adherence to secure coding practices, thereby fortifying the defense in depth strategy. Given that this module pertains to wallet trust, it is imperative that coding standards adhere to the highest levels of security, even if it results in redundant checks. Prioritizing safety over convenience is paramount.

Regarding the API design, we acknowledge the intent for code reuse. Nevertheless, we propose the adoption of a well-defined API structure that employs structured data instead of parseable strings. This approach enhances security and simplifies maintenance, mitigating potential complications.

Description

`path` is checked for correct type: `string`

../solflare-snap/src/index.js:L23-L30

```
case 'getPublicKey': {
  const { derivationPath, confirm = false } = request.params || {};

  assertInput(derivationPath);
  assertIsString(derivationPath);
  assertIsBoolean(confirm);

  const keyPair = await deriveKeyPair(derivationPath);
```

but is not checked for valid key derivation path format which may lead to unexpected outcomes or unhandled exceptions.

../solflare-snap/src/privateKey.js:L20-L20

```
const segments = path.split('/').slice(3).filter(Boolean);
```

For example, the function allows non alpha-num `0-9+'`, `slip:x` prefixes, or empty elements (`"m/44'/784'".split("/").slice(3).filter(Boolean) => []`).

Affects all snaps under review.

Recommendation

In general, the API design should be re-designed with the RPC functions receiving and validating only the last part of the key part, enforcing the format to be valid for the use case.

4.4 Dapp May Suppress User Confirmation on Request to Extract Pubkey; May Extract Any Net-Key

Medium

Resolution

The client has issued the following statement:

> Solflare Resolution
>
> Given that arbitrary dApps can not suppress user confirmation (facilitated by the Wallet), that essential wallet flows are enabled by the feature and the comparison with the security model of a hardware wallet, Solflare would like to retain the functionality.

Statement from the Assessment Team:

As discussed, it's crucial to recognize that the snap represents the trusted component in this configuration, while the dapp's trustworthiness may not always be guaranteed. It's imperative that your dapp does not assume control over the user's trust solely by displaying a confirmation message. The ideal approach should align with the security controls employed by the MetaMask (MM) implementation, which consists of (1) establishing a connection between MM and the website and (2) granting explicit approval for all accounts accessible by the website. This is in stark contrast to assuming that connecting the snap to a dapp implicitly authorizes the dapp to enumerate all accounts managed by the snap. Such a deviation from the MM design not only strays from user expectations but also introduces security concerns. It's worth noting that we have also reported this issue to the MM Development Team, as it necessitates a more comprehensive solution.

Recommendation: The dapp should not exert control over the presentation of dialogs to the user. Users must always remain cognizant of every action occurring within the trust module boundary (the snap). There should be no provision for "silent" execution of trust module actions, thereby ensuring transparency and user awareness at all times.

Description

With the `request.params.confirm` parameter in requests to signTransaction and signAllTransactions the dapp controls if the user is requested confirmation to return the public key. If the dapp sets `confirm=false` the user will not be informed that the dapp accessed their pubkey information (any account). Allowing the dapp to control if the user is asked to extract certain (derived) information from the snap is intransparent and may leak sensitive information. Especially in a setting where the snap is gatekeeping access to user specific information.

Examples

Affects all snaps under review.

../solflare-snap/src/index.js:L23-L36

```
case 'getPublicKey': {
  const { derivationPath, confirm = false } = request.params || {};

  assertInput(derivationPath);
  assertIsString(derivationPath);
  assertIsBoolean(confirm);

  const keyPair = await deriveKeyPair(derivationPath);
  const pubkey = bs58.encode(keyPair.publicKey);

  if (confirm) {
    const accepted = await renderGetPublicKey(dappHost, pubkey);
    assertConfirmation(accepted);
  }
```

Recommendation

The snap should strictly enforce user confirmation on the first time the pubkey is requested from an origin. A potentially untrusted dapp (even though origin restricted; a dapp might turn malicious and should therefore be treated as untrusted) should never be able to silently dictate what security measures be enabled with a snap request.

4.5 Production Builds Allow Development and Localhost Origins; Snap Does Not Enforce Transport Security

Medium Partially Addressed

Resolution

The client has issued the following statement:

Solflare Comment/Background

When implementing RPC access restrictions, our assumption was that MetaMask will always forward valid domains as the origin thus making `http□//..solflare.com` not possible.

Solflare Resolution

Solflare will implement enforcing Transport Security by removing the option to proceed with the `http` protocol. Solflare will be deploying a development build snap under a different package name that will allow the localhost origin. This development snap does not need to be in the allow list and we can consume it purely through Flask. We would need to keep both *solflare.com and *solflare.dev origins as these represent our production and staging environments and we would like to retain the possibility of using the production snap with our staging environment. The reason that subdomains are wildcarded is that we are going to make dApp specific changes and deploy them on different subdomains (for example - a swap dApp would not want to have a widget with swap inside of it). We don't consider this a major risk as our processes around DNS management are very strict and limited

Changesets:

- solflare-wallet/solflare-snap@ `749d2b0`
- solflare-wallet/aptos-snap@ `eef10b5`
- solflare-wallet/sui-snap@ `898295f`

Statement from the Assessment Team:

The core tenet here is a robust defense-in-depth strategy that operates on the premise of making no assumptions. We must underscore the criticality of this principle in the context of trust module code. The same level of scrutiny should apply to wildcard origins. The rationale is rooted in the potential vulnerabilities associated with subdomain takeover scenarios, such as Azure instance compromise or DNS hijacking, among others. If an attacker can manipulate a subdomain, leading users to visit a site like malicious.solflare.com, there is a substantial risk of successfully conducting phishing attacks, potentially tricking users into permitting dapp actions on the snap. This not only jeopardizes user funds but also poses a severe threat to your organization's reputation. It falls squarely on your shoulders to ensure that such scenarios do not materialize through robust security controls.

Recommendation: To mitigate these risks, it is imperative to maintain a curated list of official production hosts and validate requests against these origins. For development builds of the extension, you may consider allowing dev and wildcard hosts. However, when introducing new subdomains, it is crucial to manage them meticulously and refrain from merely deploying new subdomain websites, as this approach can become challenging to control. Ultimately, the decision rests with your organization regarding the level of risk tolerance. Based on experience, we would strongly advise against taking unnecessary risks in this regard.

Description

The snaps RPC access is restricted to certain origins only. However, there is no logic that disables development/test domains from origin checks in production builds.

This means, that, any localhost app is allowed to connect to snap (any port, not hardcoded to snap id; should not allow dev domain). The origin enforcing regex allows non-transport security-enabled connections, i.e. `http://wallet.solflare.com` is allowed while it should be enforced as `https://wallet.solflare.com`. Furthermore, the origin check allows potentially insecure subdomains, i.e. `https://beta.test.solflare.com`. Additionally, invalid domains are allowed as well, i.e. `http://..solflare.com`

Examples

Solflare Snap

../solflare-snap/src/index.js:L7-L17

```
module.exports.onRpcRequest = async ({ origin, request }) => {
  if (
    !origin ||
    (
      !origin.match(/^https?:\/\/localhost:[0-9]{1,4}$/) &&
      !origin.match(/^https?:\/\/(?:\S+\.)?solflare\.com$/) &&
      !origin.match(/^https?:\/\/(?:\S+\.)?solflare\.dev$/)
    )
  ) {
    throw new Error('Invalid origin');
  }
```

Aptos Snap

../aptos-snap/src/index.js:L6-L15

```
module.exports.onRpcRequest = async ({ origin, request }) => {
  if (
    !origin ||
    (
      !origin.match(/^https?:\/\/localhost:[0-9]{1,4}$/) &&
      !origin.match(/^https?:\/\/(?:\S+\.)?risewallet\.dev$/)
    )
  ) {
    throw new Error('Invalid origin');
  }
```

Sui Snap

../sui-snap/src/index.js:L8-L17

```
module.exports.onRpcRequest = async ({ origin, request }) => {
  if (
    !origin ||
    (
      !origin.match(/^https?:\/\/localhost:[0-9]{1,4}$/) &&
      !origin.match(/^https?:\/\/(?:\S+\.)?elliwallet\.dev$/)
    )
  ) {
    throw new Error('Invalid origin');
  }
```

Recommendation

Implement logic that removes development/localhost origin from the allow list for production builds. Employ strict checks on the format of provided origin. Do not by default allow all subdomains.

4.6 Misleading HTML Entity and Function Name `getPrivkey` Minor ✓ Fixed

Resolution

Addressed by following renaming traces of “private key”. Additionally, the client has issued the following statement:

> Solflare Comment/Background
>
> The index.html file is used for testing and naming hasn’t been updated since the snap was initially developed. This file doesn’t compile into the bundle.
>
> Solflare Resolution
>
> Solflare will be renaming the function

Changesets:

- solflare-wallet/solflare-snap@ `aa629cb`
- solflare-wallet/aptos-snap@ `e603b9f`
- solflare-wallet/sui-snap@ `5cbd097`

Description

Several lines in the code refer to a private key while the functionality always returns a public key. I.e. `getPrivkey` actually calls `getPublicKey`.

../solflare-snap/index.html:L17-L21

```
const getPrivkeyButton = document.querySelector('button.getPrivkey')

connectButton.addEventListener('click', connect)
getPrivkeyButton.addEventListener('click', getPrivkey)
```

../solflare-snap/index.html:L30-L50

```
async function getPrivkey () {
  try {
    const response = await ethereum.request({
      method: 'wallet_invokeSnap',
      params: {
        snapId,
        request: {
          method: 'getPublicKey',
          params: {
            derivationPath: `m/44'/501'/0'/0'`,
            confirm: true
          }
        }
      }
    })
    console.log(response);
  } catch (err) {
    console.error(err)
    alert('Problem happened: ' + err.message || err)
  }
}
```

Affects all snaps under review.

Recommendation

Never return or extract a private key from the snap. Rename the variables in code to accurately reflect what type of data is being handled.

4.7 Use of Outdated `snap.config.json` Instead of `snap.config.js` Minor ✓ Fixed

Resolution

Addressed by switching to the `snap.config.js`. Additionally, the client has issued the following statement:

> Solflare Resolution
>
> Solflare will be moving to snap.config.js

Changesets:

- solflare-wallet/solflare-snap@ `5228b37`
- solflare-wallet/aptos-snap@ `62ec7b4`
- solflare-wallet/sui-snap@ `3420723`

Description

According to the MetaMask-CLI Documentation the use of `snap.config.json` is discouraged and projects should switch to `snap.config.js` instead.

4.8 Inconsistent or Blank Fields in `package.json` Minor ✓ Fixed

Resolution

Addressed by switching to the `snap.config.js`. Additionally, the client has issued the following statement:

> Solflare Resolution
>
> Solflare will be adding missing information

Changesets:

- solflare-wallet/solflare-snap@ `f91a27d`
- solflare-wallet/aptos-snap@ `cb61e41`
- solflare-wallet/sui-snap@ `9819463`

Description

The `package.json` file serves as a critical source of information for users, developers, and security analysts. Neglecting to provide complete and accurate metadata in this file can lead to misunderstandings, vulnerabilities, and potential exploitation. By meticulously filling out fields like `bugs`, `homepage`, `repository`, `author`, and `description`, developers can enhance transparency, facilitate collaboration, and minimize security risks associated with incomplete or misleading project information.

../solflare-snap/package.json:L2-L13

```
"name": "@solflare-wallet/solana-snap",
"version": "1.0.0",
"description": "",
"main": "src/index.js",
"scripts": {
  "start": "mm-snap build && mm-snap serve",
  "build": "mm-snap build",
  "deploy": "npm run build && npm publish"
},
"author": "",
"license": "ISC",
"files": [
```

Affects all snaps under review.

Recommendation

Provide meta information for bugs, homepage, repository, author (non-blank), description (non-blank).

4.9 Consider Using `@metamask/detect-provider`

Resolution

The client has issued the following statement:

> Solflare Comment/Background
>
> The index.html file is used for testing and naming hasn't been updated since the snap was initially developed. This file doesn't compile into the bundle.
>
> Solflare Resolution
>
> Since this is just a testing file and we use proper detection mechanisms in the Wallet, Solflare will not implement a fix in the helper file.

Description

Consider using the Metamask provided library `@metamask/detect-provider` for inpage metamask/flask detection.

../solflare-snap/index.html:L23-L28

```
async function connect () {
  await ethereum.request({
    method: 'wallet_requestSnaps',
    params: { [snapId]: {} }
  })
}
```

4.10 Consider Prefixing RPC Calls With `solana_*`, `sui_*`, `aptos_*`

Resolution

The client has issued the following statement:

> Solflare Comment/Background
>
> The naming convention we have adopted was based on the structure of snap method invocations. Namely, when a snap invocation is happening, the wallet_invokeSnap method is called with a strictly specified snapId (which we consider scope) and the request method. Our thinking is that prefixing the methods in addition to explicitly stating the snapId might lead to unnecessary duplication.
>
> Solflare Resolution
>
> Solflare will not implement prefixes to both avoid naming duplication and because our wallets share a common architecture/codebase and are scoped by the snapId, we would like to avoid needing to call different methods for different wallets

Description

APIs (Application Programming Interfaces) play a crucial role in modern software development, enabling interoperability and communication between different software components. One often overlooked aspect of API design is the naming of API functions. Poorly named API functions can lead to security vulnerabilities and confusion, compromising user information integrity and security.

Consider prefixing the RPC method handlers with the respective protocols they are serving.

4.11 Consider Moving to TypeScript

Resolution

The client has issued the following statement:

> Solflare Comment/Background
>
> Only comment is that we began developing snaps before there was a way to write them in TypeScript. Also, the initial audit was done on a JavaScript codebase and in the benefit of reducing the gap audit and saving time we opted to stay in JavaScript. We definitely recognize all the TypeScript benefits, especially as most of our stack is written in TypeScript.
>
> Solflare Resolution
>
> It is definitely on our roadmap to migrate to TypeScript.

Description

JavaScript, as a dynamically typed language, lacks the stringent type checks and static analysis that can catch a wide range of programming errors during compile time. While JavaScript is widely adopted due to its simplicity and ubiquity, it poses certain security challenges that could be mitigated by transitioning to TypeScript.

1. Type-Related Vulnerabilities: JavaScript's loose typing allows developers to perform operations on variables without explicit type declarations. This freedom can inadvertently lead to vulnerabilities such as type coercion attacks, where an attacker manipulates input data to trigger unintended behaviors. TypeScript enforces strong typing, which can catch type-related issues early in the development process, reducing the risk of such vulnerabilities.

2. Null and Undefined Errors: JavaScript's permissiveness with null and undefined values can lead to runtime errors and crashes. Insecure access to null or undefined properties can open doors to injection attacks, privilege escalation, or application crashes. TypeScript's strict null checks force developers to handle these cases explicitly, minimizing the chances of overlooking critical input validation

3. Misconfigured Objects and Inheritance: JavaScript's prototype-based inheritance can lead to unforeseen object interactions and security breaches when not carefully managed. TypeScript's class-based object-oriented approach enhances code organization and provides better control over inheritance, reducing the likelihood of unexpected security vulnerabilities.

4. Reduced Maintenance Complexity: Maintaining a JavaScript codebase can become increasingly complex as a project grows. With TypeScript, the inclusion of type annotations and stricter rules aids in code comprehension, refactoring, and identifying security vulnerabilities. This streamlined development process indirectly contributes to a more secure software ecosystem.

Remember, this project is implementing an extension to the MetaMask Trust Module. As such, we recommend implementing compile time security controls by moving the codebase to TypeScript with strict linting rules enabled.

Appendix 1 - Files in Scope

This review covered the following files:

File	SHA-1 hash
aptos-snap/index.html	`5e3a44cb5b21a1d500aff78c66918137502f5952`
aptos-snap/logo.svg	`7e9ca06eddd17aef8b58abf8f77bf631e1e1fbf2`
aptos-snap/src/index.js	`cee7e6cb64a8df7e9739d79f05de5a9a9d70adb6`
aptos-snap/src/privateKey.js	`b31c752080e5a8ab20a7bba74845a52fde37aa7b`
aptos-snap/src/ui.js	`1242056e76afe38d9d086a82d92de0f939672682`
aptos-snap/src/utils.js	`3bba0e810a3ff1f28e7a3d75a2a2bcbb10adb112`
sui-snap/index.html	`e23de59c3a1baf10b1293ff37a751336d17a7901`
sui-snap/logo.svg	`2ad0ad0e48b5371ccbb2f87db5850c5f379c9164`
sui-snap/src/index.js	`956e011716598fb3f27254f3eec87f79ce662823`
sui-snap/src/privateKey.js	`38d4814835c1c70e7f6a664e5054d69002576e21`
sui-snap/src/ui.js	`1242056e76afe38d9d086a82d92de0f939672682`
sui-snap/src/utils.js	`3bba0e810a3ff1f28e7a3d75a2a2bcbb10adb112`
solflare-snap/index.html	`9128e7eb26165bb90d7deb50f94e39b68f3ff32d`
solflare-snap/logo.svg	`6041863b433142dc5f14d9477c2555e0ed383c8f`
solflare-snap/src/index.js	`4497a4b0b8a53929aae134964ad148904cc77b67`
solflare-snap/src/privateKey.js	`42818b4ff1e78e9fc83e3202cc15a35b7a17e78d`
solflare-snap/src/ui.js	`1242056e76afe38d9d086a82d92de0f939672682`
solflare-snap/src/utils.js	`41e9e87cdd5ae877e2d579e51e2620e4dde0160a`

Appendix 2 - Disclosure

Consensys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via Consensys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any third party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any third party by virtue of publishing these Reports.

A.2.1 Purpose of Reports

The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

A.2.2 Links to Other Web Sites from This Web Site

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Consensys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that Consensys and CD are not responsible for the content or operation of such Web sites, and that Consensys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that Consensys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. Consensys and CD assumes no responsibility for the use of

third-party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

A.2.3 Timeliness of Content

The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice unless indicated otherwise, by Consensys and CD.