Wallet Guard

Date	July 2023

1 Executive Summary

This report presents the results of our engagement with **Wallet Guard** to review the **Wallet Guard Snap**.

The review was conducted over two person days, from **July 13, 2023** to **July 14, 2023**, by **Valentin Quelquejay**. A total of 2 person-days were spent.

2 Scope

Our review focused on the commit hash `695c7874d4ac8ffe6a454e9dd5c7fc6925189374`. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **Wallet Guard** team, we identified the following priorities for our review:

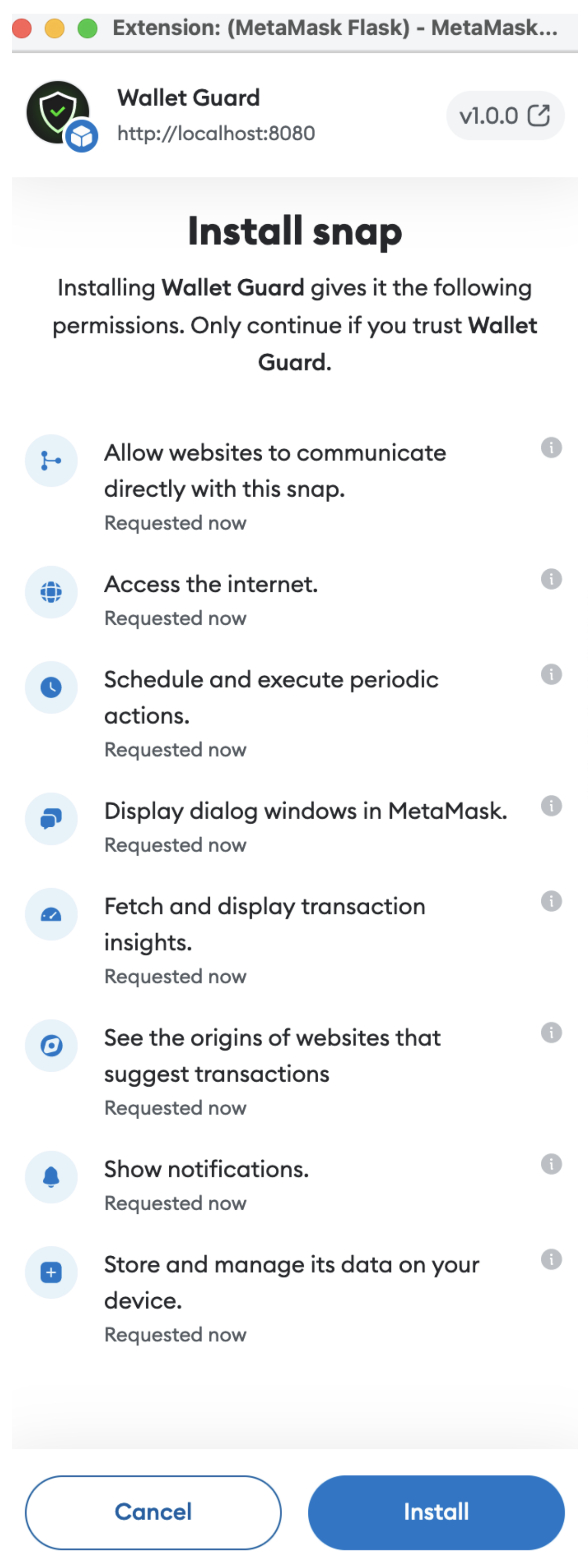
1. Correctness of the implementation, consistent with the intended functionality and without unintended edge cases.
2. Identify vulnerabilities particular to the MetaMask Snaps SDK integration in coherence with the MetaMask Snap Threat Model describing a Snap as an extension of the MetaMask Wallet Trust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

3 Snap Overview

3.1 Capabilities

- The snap stores the address of the wallet to be monitored in `snap_manageState`.
- The snap interacts with the wallet guard API (https://api.walletguard.app/) via the `fetch()` API for:
  - Simulating transactions
  - Fetching wallet approvals
- The Wallet Guard dapp (dashboard.walletguard.app) can communicate with the snap via MetaMask snaps RPC.
- The snap registers a cronjob that fires every 14 days, prompting the user to revoke its dangerous approvals.
- The snap can read every transaction, including the transaction origin, to provide transaction insights.
- The snap can display notifications to the user asking him to revoke its dangerous approvals.

3.2 Permissions


Extension: (MetaMask Flask) - MetaMask...

Wallet Guard
http://localhost:8080
v1.0.0

Install snap

Installing **Wallet Guard** gives it the following permissions. Only continue if you trust **Wallet Guard**.

- Allow websites to communicate directly with this snap.
  Requested now
- Access the internet.
  Requested now
- Schedule and execute periodic actions.
  Requested now
- Display dialog windows in MetaMask.
  Requested now
- Fetch and display transaction insights.
  Requested now
- See the origins of websites that suggest transactions
  Requested now
- Show notifications.
  Requested now
- Store and manage its data on your device.
  Requested now

```
snap_dialog {}
endowment:rpc { dapps: true, snaps: false }
endowment:transaction-insight { allowTransactionOrigin: true }
endowment:network-access {}
snap_notify {}
snap_manageState {}
endowment:cronjob { jobs: [ { expression: '0 0 */14 * *', request: [Object] } ] }
```

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 Server Should Not Rely on Clients' Randomness Major ✓ Fixed

Resolution

The client acknowledged the issue, and let us know that the ID is only used for analytics purposes, to be compatible with the existing API.

Description

The snap code sends a request to the Wallet Guard API with a random UUID `crypto.randomUUID()` generated by the client. We would like to underline that the API should **never trust** clients' randomness nor assume any property about it. Relying on client-generated randomness for the API could lead to many vulnerabilities, such as replay attacks or collision issues due to the inability to ensure uniqueness. The varying algorithms used by clients may be subpar or even compromised. As this id is not used anywhere else in the snap code, we assume that it might be used on the API side. Because the API is not in scope for this review, we don't have access to the code and cannot tell whether this pseudo-random UUID is used in a safe way.

packages/snap/src/http/fetchTransaction.ts:L32-L40

```
const simulateRequest: SimulateRequestParams = {
  id: crypto.randomUUID(),
  chainID: mappedChainId,
  signer: transaction.from as string,
  origin: transactionOrigin as string,
  method: transaction.method as string,
  transaction,
  source: 'SNAP',
};
```

Recommendation

Don't rely on clients' randomness on the API. Instead, the server should assign a unique ID to every incoming request.

4.2 Missing Input Validation for `WalletAddress` Major ✓ Fixed

Resolution

The client acknowledged the issue and fixed it in PR#25 here - Snap shasum `YzN/+ty8xOTEacH19iYGw1a9+MBCgL7PUkU9d/Rf51E=`

Description

The snap prompts users to input the wallet address to be monitored. Users can set wallet addreses that do not adhere to the common Ethereum address format. The user input is not sanitized. This could lead to various injection vulnerabilities such as markdown or control character injections that could break other components. In particular, the address is sent to the API as a URL query parameter. A malicious attacker could try using that to mount URL injection attacks.

packages/snap/src/index.ts:L50-L61

```
if (
  request.method === RpcRequestMethods.UpdateAccount &&
  'walletAddress' in request.params &&
  typeof request.params.walletAddress === 'string'
) {
  const { walletAddress } = request.params;

  if (!walletAddress) {
    throw new Error('no wallet address provided');
  }

  updateWalletAddress(walletAddress);
```

Recommendation

Sanitize the address string input by the user and reject all addresses that do not adhere to the Ethereum address format.

4.3 Properties of the `transaction` Object Might Be Undefined Medium

Description

The Metamask Snaps API does not guarantee that the properties `from` and `method` of the `transaction` object are defined. Depending on the transaction type, it could happen that these properties are not defined. This would result in a runtime error when `undefined` is casted to `string`.

packages/snap/src/http/fetchTransaction.ts:L32-L40

```
const simulateRequest: SimulateRequestParams = {
  id: crypto.randomUUID(),
  chainID: mappedChainId,
  signer: transaction.from as string,
  origin: transactionOrigin as string,
  method: transaction.method as string,
  transaction,
  source: 'SNAP',
};
```

Recommendation

One should check whether properties `from`, and `method` are defined, before explicitly casting them to a string. This could be done by introducing a `hasProperty` utility function for instance.

4.4 AssetChangeComponent Displays a Change With Value 0 if `fiatValue < 0.005` Medium

Description

The `toFixed(2)` method rounds the transaction value string to 2 decimals. For transactions with `fiatValue < 0.005`, the function returns 0, meaning the component will display a transaction with zero value to the user, even if the transaction has a small yet non-zero value. This is not a good idea as it might trick the user. In that case, it would be better to default to the smallest value that can represented (i.e. 0.01) instead of 0.

packages/snap/src/components/stateChanges/AssetChangeComponent.ts:L18

```
const fiatValue = Number(stateChange.fiatValue).toFixed(2);
```

Recommendation

If `fiatValue < 0.005`, consider displaying a value of `0.01` to the user, instead of 0.

4.5 Incomplete NatSpec and General Documentation Minor

Description

The code is missing NatSpec documentation in many places. NatSpec documentation plays an important role in improving code comprehension and maintenance. Adding NatSpec documentation to functions with significant logic that provides clear explanations of behavior, inputs, and outputs enhances code readability, transparency, and maintainability of the codebase.

Recommendation

We recommend adding NatSpec documentation to every function that contains significant logic. Especially all the Snaps handlers. This will improve the readability, transparency, and maintainability of the codebase. We also recommend adding a detailed high-level documentation about the Snaps features, components, and permissions in the README.

4.6 `formatFiatValue()` Can Be Simplified Minor

Description

The function `formatFiatValue` formats a number to a string that is displayed to the user. The function formats numbers with at most 2 decimal digits, removes the trailing zeros, and adds commas as thousands separators.

The function first converts the number to a string representing the number in fixed-point notation. Then, it uses regex to remove the trailing zeros if they exist. Finally, it adds the thousands separators.

packages/snap/src/utils/helpers.ts:L16-L26

```
export const formatFiatValue = (
  fiatValue: string,
  maxDecimals: number,
): string => {
  const fiatWithRoundedDecimals = Number(fiatValue)
    .toFixed(maxDecimals) // round to maxDecimals
    .replace(/\.00$/u, ''); // removes 00 if it exists

  const fiatWithCommas = numberWithCommas(fiatWithRoundedDecimals); // add commas
  return `$${fiatWithCommas}`;
};
```

The design of the function is unnecessarily complex. The whole design could be simplified using the native `toLocaleString()` function with appropriate parameters.

Recommendation

Simplify the design by using the native `toLocaleString` function. For instance, the function could be used as follows `toLocaleString('en-US',{minimumFractionDigits: 0, maximumFractionDigits: 2})`

4.7 No Way to Disable Approvals Checking, and Transaction Analytics Minor

Description

Currently, there is no easy way to disable wallet approval monitoring and/or transaction simulation apart from uninstalling the snap. Users might want to opt out of wallet monitoring or disable transaction simulation selectively e.g., for privacy concerns.

Recommendation

We would recommend implementing a mechanism that allows users to selectively disable the snap features.

4.8 `devDependencies` Erroneously Listed as `dependencies` Minor

Description

The following dependencies are only used for development purpose and should therefore be listed as “devDependencies” instead of “dependencies” in the package.json file. Indeed, the TypeScript code is compiled into a bundle, which is released. Meaning the snap “production” code should not contain any external dependency.

packages/snap/package.json:L28-L31

```
"dependencies": {
  "@metamask/snaps-types": "^0.32.2",
  "@metamask/snaps-ui": "^0.32.2"
},
```

Recommendation

List the dependencies as “devDependencies”.

4.9 `package.json` - Missing Author Minor

Description

The `package.json` file is missing the author name, the link to the project homepage, and to the bug tracker.

Recommendation

According to package publishing best practices, we recommend adding those elements to the `package.json` file.

4.10 Extra ‘If’ Statement

Description

The `onRpcRequest()` handler returns early if `walletAddress` is not defined.

packages/snap/src/index.ts:L57-L59

```
if (!walletAddress) {
  throw new Error('no wallet address provided');
}
```

Thus, the extra ‘if’ check before calling `snap.request()` is superfluous and can be removed.

packages/snap/src/index.ts:L57-L64

```
if (!walletAddress) {
  throw new Error('no wallet address provided');
}

updateWalletAddress(walletAddress);

if (walletAddress) {
  await snap.request({
```

Recommendation

Remove the extra 'if' check.

4.11 Misleading Comment

Description

The NatSpec comment indicates that `onRpcRequest()` returns "the result of `snap_dialog`" while the method either does not return anything, or returns the Ethereum address of the monitored wallet.

packages/snap/src/index.ts:L24-L34

```
/**
 * Handle incoming JSON-RPC requests, sent through `wallet_invokeSnap`.
 *
 * @param args - The request handler args as object.
 * @param args.origin - The origin of the request, e.g., the website that
 * invoked the snap.
 * @param args.request - A validated JSON-RPC request object.
 * @returns The result of `snap_dialog`.
 * @throws If the request method is not valid for this snap.
 */
```

Recommendation

Fix the comment.

4.12 Wallet Monitoring Improvements

Description

The snap allows the user to set an arbitrary wallet address to be monitored for dangerous approvals. This feature is only of limited use and could be improved by:

- Allowing to specify multiple addresses to monitor (a wallet typically consists of many accounts that are managed under the wallet key)
- Allowing users to fetch connected addresses via the `ethereum` API directly instead of requiring the user to input valid accounts
- For privacy reasons, allowing users to opt out of transaction analytics on a per-account basis (Currently, every transaction and transaction origin is sent to the API, even if no monitored wallet address is set).

4.13 Consider Submitting Snap Version With Backend API Requests

Description

Consider adding the snap package version to the API requests in order to get insights about what snap versions are used in the field. This could be useful for future debugging and forensics when multiple snap versions will coexist.

packages/snap/src/http/fetchTransaction.ts:L32-L40

```
const simulateRequest: SimulateRequestParams = {
  id: crypto.randomUUID(),
  chainID: mappedChainId,
  signer: transaction.from as string,
  origin: transactionOrigin as string,
  method: transaction.method as string,
  transaction,
  source: 'SNAP',
};
```

packages/snap/src/types/simulateApi.ts:L25-L35

```
export type SimulateRequestParams = {
  id: string;
  chainID: string;
  signer: string;
  origin: string;
  method: string;
  transaction: {
    [key: string]: Json;
  };
  source: 'SNAP';
};
```

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash

File	SHA-1 hash
../packages/snap/snap.config.js	`09ea1d61eb7c441435218bf5844c40b570b1fc0f`
../packages/snap/src/components/OnboardingReminderComponent.ts	`b8d3fad340a6b805df9d10bf9891f5f3413aa6cd`
../packages/snap/src/components/RiskFactorsComponent.ts	`b27e86fd63721a43da1e35812cbc7b7172aa8526`
../packages/snap/src/components/SimulationOverviewComponent.ts	`02077aab340a8a915a184376f0214c9087b9dbf8`
../packages/snap/src/components/StateChangesComponent.ts	`51b4ca57158edf5217c75c357365cb1db2bd3857`
../packages/snap/src/components/assetChanges/AssetChangeComponent.ts	`15778c73bafa117ec14e5a3b142fd1e5137a3285`
../packages/snap/src/components/assetChanges/GasComponent.ts	`f9d0bd451d46bf24271f8804622f1c765344f48a`
../packages/snap/src/components/assetChanges/NoChangesComponent.ts	`81e71eaa666021c3d2b7bdf90fca0df7be10af59`
../packages/snap/src/components/assetChanges/index.ts	`8490fe8d8c3184bd9e7245766a0c7c9ece25cd04`
../packages/snap/src/components/errors/ErrorComponent.ts	`134004a5962002945b2f80ec5aa05340fc9b9b5a`
../packages/snap/src/components/errors/InsufficientFundsComponent.ts	`5a6ab45939ac75a2ff453acee11dfadfcd438602`
../packages/snap/src/components/errors/RevertComponent.ts	`5bf65f6c2b1572f592e53d1ed7339bd8f1a18826`
../packages/snap/src/components/errors/TooManyRequestsComponent.ts	`19c0a31091089c90ebf3f6a112d708cc22b90405`
../packages/snap/src/components/errors/UnauthorizedComponent.ts	`ff469b19832e4eb92f21aee326a088ea343b5b30`
../packages/snap/src/components/errors/UnsupportedChainComponent.ts	`bcbecb1fc2be669386f491f5f5ca4ee79ac71867`
../packages/snap/src/components/errors/index.ts	`e2b2e66ee47d846a566b3981f6db3237a4fba9f3`
../packages/snap/src/components/errors/mapper.ts	`991c63f37ad822f62b69154a65aeeda43aaf2614`
../packages/snap/src/components/index.ts	`c2f9362ed8d5a82ae761734af7aa326485d35d8a`
../packages/snap/src/http/fetchApprovals.ts	`026a4d625c48d2827f736aecddbe3b7d84daee8c`
../packages/snap/src/http/fetchTransaction.ts	`6e93224cf1c281ff99a23b45739fe2546e854419`
../packages/snap/src/index.ts	`0dc598b61a696879bb45e8d118bacb5d61a4659d`
../packages/snap/src/types/approvalsApi.ts	`dd143f3ab7400fc5053a2de0dd4c5ccec3035bcf`
../packages/snap/src/types/chains.ts	`c53a0eba39d2f21f1b7ad50eec94b55e122a3b5e`
../packages/snap/src/types/simulateApi.ts	`75dda9659e890e28f65ffa954a47b9fd06651fad`
../packages/snap/src/utils/account.ts	`04cd1040973b9d86343125d4a373e439b1ed88bc`
../packages/snap/src/utils/config.ts	`d500d900e8c1980d36aa5e2a5ba9f93060faa8af`
../packages/snap/src/utils/environment.ts	`d690ac5172e767a63f86119a746dc3c99c626319`
../packages/snap/src/utils/helpers.ts	`14f29385952ea65acc989cace66a68a0bbc097fd`
../packages/snap/src/package.json	`bf4ea0f52c4653e6409b2a4051be617002761b71`

Appendix 2 - Disclosure

A.2.1 Purpose of Reports

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

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.