MetaMask/Partner Snaps - Shapeshift Snap

Date	July 2023
Auditors	Valentin Quelquejay, Martin Ortner

1 Executive Summary

This report presents the results of our engagement with **ShapeShift** to review their **MetaMask Snap**.

The review was conducted from **July 24, 2023** to **July 28, 2023**. A total of 2x5 person-days were spent.

2 Scope

Our review focused on the commit hash 32d2f43808de6b107ef759b517e7df27644f4011. The list of files in scope can be found in the Appendix.

2.1 Objectives

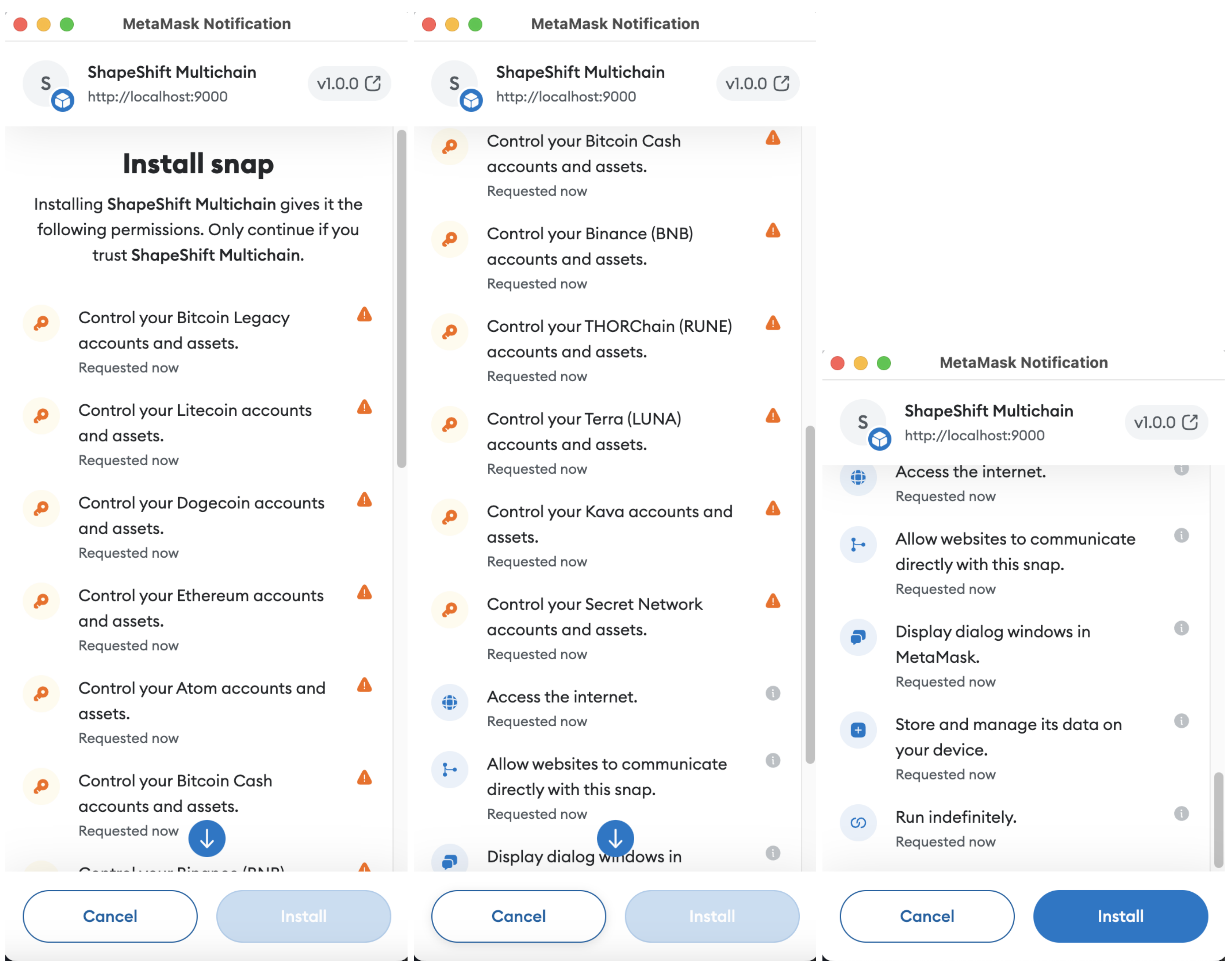
Together with the client, we identified the following priorities for our review:

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 Outline

- The snap requests sensitive BIP32 Entropy (SLIP-44), effectively managing the following coins private root key:
  - `m/44/0` (Bitcoin)
  - `m/44/2` (Litecoin)
  - `m/44/3` (Dogecoin)
  - `m/44/60` (Ethereum) ⚠️
  - `m/44/118` (Atom)
  - `m/44/145` (Bitcoin Cash)
  - `m/44/714` (BNB)
  - `m/44/931` (ThorChain/Rune)
  - `m/44/330` (Terra/Luna)
  - `m/44/459` (Kava)
  - `m/44/529` (Secret)
- ⚠️ The snap manages **the same** Ethereum private keys MetaMask manages.
- The private key can not be exported to an RPC origin.
- Transactions are signed within the realm of the snap.
- The public key is exposed to connected snaps without additional user confirmation.
- Connected dapps can communicate with the snap via MetaMask snap RPC.
- The snap may run indefinitely while processing RPC requests (`endowment:long-running`).

3.1 Capabilities


MetaMask Notification

ShapeShift Multichain
http://localhost:9000
v1.0.0

Install snap

Installing **ShapeShift Multichain** gives it the following permissions. Only continue if you trust **ShapeShift Multichain**.

Control your Bitcoin Legacy accounts and assets.
Requested now

Control your Litecoin accounts and assets.
Requested now

Control your Dogecoin accounts and assets.
Requested now

Control your Ethereum accounts and assets.
Requested now

Control your Atom accounts and assets.
Requested now

Control your Bitcoin Cash accounts and assets.
Requested now

Control your Bitcoin Cash accounts and assets.
Requested now

Control your Binance (BNB) accounts and assets.
Requested now

Control your THORChain (RUNE) accounts and assets.
Requested now

Control your Terra (LUNA) accounts and assets.
Requested now

Control your Kava accounts and assets.
Requested now

Control your Secret Network accounts and assets.
Requested now

Access the internet.
Requested now

Allow websites to communicate directly with this snap.
Requested now

Display dialog windows in MetaMask.
Requested now

Store and manage its data on your device.
Requested now

Run indefinitely.
Requested now

Snap Permissions

Details

```
[🚀 == FsChecks == ]
    ℹ️  yarn@3.2.2
    🔺 - package-lock missing
    🔺 - yarn.lock missing
    🔺 - no linter config
[🚀 == Manifest == ]
    🔺 - bundle (../../../dist/bundle.js) does not not exist!
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
snap_dialog {}
snap_getBip32Entropy [
  { path: [ 'm', "44'", "0'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "2'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "3'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "60'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "118'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "145'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "714'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "931'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "330'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "459'" ], curve: 'secp256k1' },
  { path: [ 'm', "44'", "529'" ], curve: 'secp256k1' }
]
snap_manageState {}
endowment:network-access {}
endowment:long-running {}
endowment:rpc { dapps: true, snaps: false }
---->%---- raw permissions
🚜 [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/rpc/common/utils.ts
🚜 [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/rpc/common/utils.ts
🚜 [snap_manageState]
    📡 - snap_manageState - snap can store up to 100mb (isolated)
       🔺 - superfluous permission: no reference to 'snap_manageState'!
🚜 [endowment:network-access]
    🌐 - endowment:network-access - snap can access internet
     ⚠️ - this method may leak information to external api
       🔺 - superfluous permission: no reference to 'window.ethereum.*'!
🚜 [endowment:long-running]
    🔺 - endowment:long-running - experimental
🚜 [endowment:rpc]
    ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
       🔶  src/index.ts
[🚀 == Bundle == ]
    🟢 Package.json OK
⚠️  Package.json Warnings:
    🔺 missing keywords
    🔺 missing bugs
```

3.2 Dependencies

```
🌲 - Package Depenencies:
      - @ethersproject/providers:^5.7.0
      - @metamask/detect-provider:^2.0.0              (🔺 looks like devDependency 👀)
      - @metamask/key-tree:^7.0.0             (🔺 looks like devDependency 👀)
      - @metamask/snaps-types:0.32.2          (🔺 looks like devDependency 👀)
      - @metamask/snaps-ui:^0.32.2            (🔺 looks like devDependency 👀)
      - @shapeshiftoss/caip:^8.15.0
      - @shapeshiftoss/hdwallet-core:^1.34.0
      - @shapeshiftoss/hdwallet-native:^1.34.0
      - @shapeshiftoss/logger:^1.1.2
      - @shapeshiftoss/metamask-snaps-types:workspace:^
      - @shapeshiftoss/types:^8.3.0
      - @shapeshiftoss/unchained-client:10.1.1
```

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 RPC `eth_signMessage` Allows Linked Dapps to Sign Messages With Any Wallet Account and W/O Explicit User Consent Critical

Description

When a normal dapp requests MetaMask to sign an arbitrary message, the message is displayed to the user for confirmation before signing it. Requiring explicit user consent and displaying the message to be signed is crucial to ensure that the user has full control over what messages are signed on their behalf.

The shapeshift snap exposes an RPC endpoint for ethereum (and an undocumented one for AVAX) that allows bypassing user consent. When invoking the shapeshift snap with the `eth_signMessage` RPC method, the snap signs the message right away, silently, without requiring consent from the wallet owner or notifying them of the fact that the dapp is signing with a HD wallet account on their behalf. This severely undermines security restrictions by the MetaMask that ensure that the end-user has full control over what is being signed, giving them the option to reject signing.

For comparison, `eth_signTransaction` ask for user confirmation while `eth_signMessage` does not a.

The relevant code can be found here:

packages/snap/src/rpc/evm/common/EVMSigner.ts:L37-L47

```
async signMessage({ message }: SignMessageParamsType<T>): Promise<SignMessageResponseType<T>> {
  try {
    return (await this.signer.ethSignMessage(
      message as SignerSignMessageType<T>,
    )) as SignMessageResponseType<T>
  } catch (error) {
    this.logger.error(message, { fn: 'ethSignMessage' }, error)
    return Promise.reject(error)
  }
}
```

It also affects the `avax` implementation:

packages/snap/src/rpc/evm/avalanche/handlers.ts:L34-L45

```
export const avalancheSignMessage = async (
  params: AvalancheSignMessageParams,
): Promise<AvalancheSignMessageResponse> => {
  try {
    const avalancheSigner = new AvalancheSigner()
    await avalancheSigner.initialize()
    return await avalancheSigner.signMessage(params)
  } catch (error) {
    moduleLogger.error({ fn: 'avalancheSignMessage' }, error)
    return Promise.reject(error)
  }
}
```

Examples

A dapp can invoke `eth_signMessage` which will not require user confirmation and silently return a message that is signed with any of the users HD wallet accounts.

```
await window.ethereum.request({
    method: 'wallet_invokeSnap',
    params: {
      snapId: "local:http://localhost:9000",
      request: { method: 'eth_signMessage', params: {message: {addressNList: [0x80000000 + 44, 0x80000000 + 60, 0x80000000 + 0,
    }});
{address: '0xBaB66CfA59757200c90c79BC6e2aEe4bFBe382Be', signature: '0x5c04bcc1ca73e9f9d4bf3642150407c01c189d784dd90349…e03ca8ec0
```

Recommendation

Follow exactly the same flow MetaMask already implemented when signing arbitrary messages. Log signing requests, surface them to the user, ask for confirmation, and reject them by default (timeout). Display the origin on signing request dialogues and present the data to be signed and the account in a human-readable, understandable way while showing the original data to be signed, too.

4.2 ShapeShift Manages MetaMasks Ethereum Private Keys Major

Description

As the Snap Outline in this report mentions, the ShapeShift snap requests access to the BIP32 entropy for the Ethereum private keys. This effectively allows the ShapeShift snap to manage MetaMasks Ethereum keys directly, which comes with great responsibility. To avoid undermining established security controls put in place by the MetaMask team, the snap would have to replicate the same security functionality not to degrade the security posture of MetaMask altogether.

For reference, please take a look at issue 4.4, issue 4.1 , issue 4.9 .

Recommendation

We recommend using the Metamask provider exposed via the `endowment:ethereum-provider` RPC endpoint to perform Ethereum operations instead of managing the Ethereum keys and low-level operations directly. This avoids bypassing MetaMask security controls but falling back to proven and battle-tested user confirmation dialogs instead.

Moreover, we also asked the MM team to provide a more robust account management API that is not based on giving full low-level account access to Snaps. This would enable Snaps to perform signing operations with control over cryptographic parameters (e.g., BIP-44 derivation path) without accessing the root entropy. This will significantly decrease the risks for the end-user.

4.3 Superfluous Permission `endowment:network-access` Major

Description

The snap requests permission `endowment:network-access` to interact with external entities over HTTP/fetch. While the sandbox/demo dapp may use the fetch API in its context as a web app, the requested permission is only relevant for the snap and the snap never calls the `fetch()` API. Hence, the permission is requested but never used.

Requesting more permissions than necessary should always be avoided following the principle of least privilege.

Examples

packages/snap/snap.manifest.json:L69

```
"endowment:network-access": {},
```

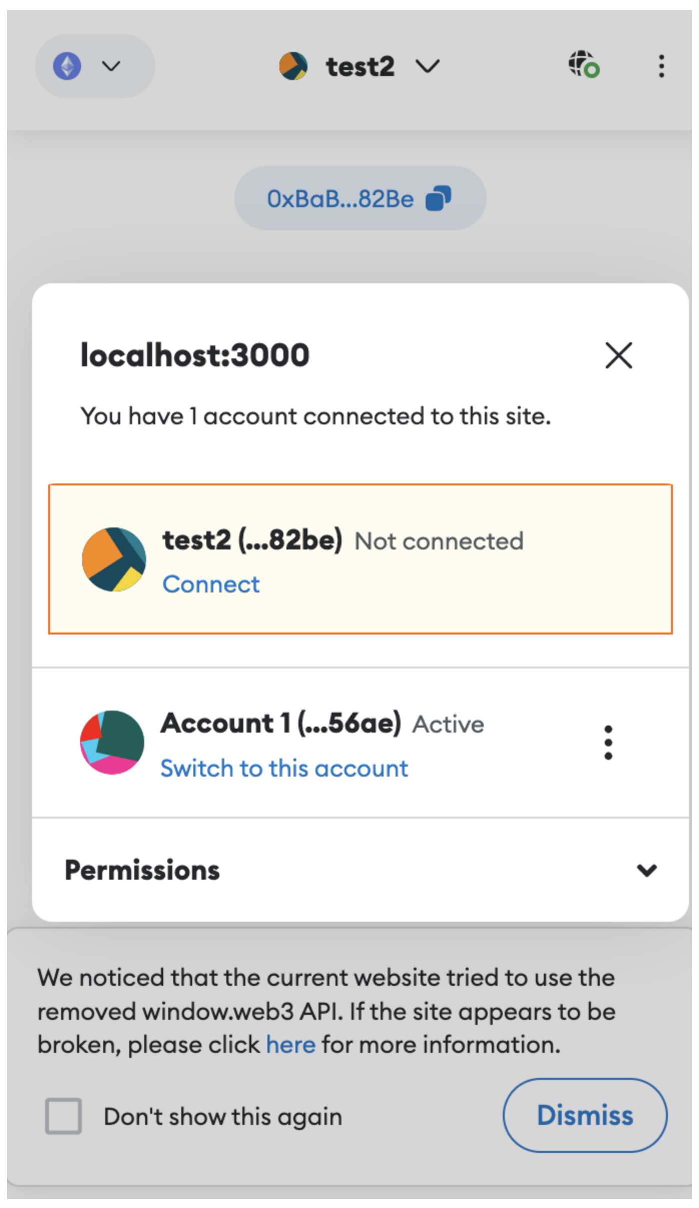
Recommendation

Remove superfluous permissions.

4.4 RPC `eth_getAddress` Undermines MetaMask Security Features by Exposing All Accounts W/O Explicit User Consent Major

Description

Metamask by default protects wallet addresses from being exposed to connected websites. A user wishing to expose a wallet address to a dapp must explicitly connect that address with the dapp website. Here is an example of MetaMask Flask with a random test wallet managing 2 accounts. `Account 1` is connected to metamask, `test2` is not.



test2

0xBaB...82Be

localhost:3000

You have 1 account connected to this site.

test2 (...82be) Not connected
Connect

Account 1 (...56ae) Active
Switch to this account

Permissions

We noticed that the current website tried to use the removed window.web3 API. If the site appears to be broken, please click here for more information.

Don't show this again

Dismiss

The dapp can query for connected addresses via the MetaMask injected provider RPC method `eth_requestAccounts`. Other non-connected addresses will not be returned:

```
await window.ethereum.request({ method: 'eth_requestAccounts' })
['0x3d0c4e58b3ff2516455f79c1147eb95f125d56ae']
```

In contrast, the shapeshift snap requests low-level access for the ethereum root key. The snap exposes a similar RPC endpoint named `eth_getAddress` that returns all ethereum addresses. Any connected dapp can query the snap to retrieve ethereum addresses. The dapp - not necessarily trusted - can even silently interact with the snap to enumerate all ethereum addresses, whether they're 'connected' to the dapp or not. This effectively bypasses MetaMask security measures where the user defines the addresses to expose.

Via the ShapeShift snap `eth_getAddress` RPC endpoint, the dapp can effectively enumerate all addresses even though they're not connected via the main wallet. This circumvents MetaMask security measures undermining established security principles of the wallet.

Note that all `*_getAddress` RPC endpoints exhibit this problem.

Examples

- MetaMask APO only exposes connected addresses

```
await window.ethereum.request({ method: 'eth_requestAccounts' })
['0x3d0c4e58b3ff2516455f79c1147eb95f125d56ae']
```

- the same address can be retrieved via the ShapeShift snap RPC API

```
await window.ethereum.request({
    method: 'wallet_invokeSnap',
    params: {
      snapId: "local:http://localhost:9000",
      request: { method: 'eth_getAddress', params: {addressParams:{addressNList: [0x80000000 + 44, 0x80000000 + 60, 0x80000000 +
    }});
'0x3D0C4e58b3fF2516455f79c1147eB95F125d56aE'
```

- MetaMask does not expose other addresses. However, the snap RPC API allows to enumerate non-connected addresses too, undermining the security of the main wallet.

```
await window.ethereum.request({
    method: 'wallet_invokeSnap',
    params: {
      snapId: "local:http://localhost:9000",
      request: { method: 'eth_getAddress', params: {addressParams:{addressNList: [0x80000000 + 44, 0x80000000 + 60, 0x80000000 +
    }});
'0xBaB66CfA59757200c90c79BC6e2aEe4bFBe382Be'
```

Recommendation

Ensure that the implemented security measures match those of the main wallet. Allow users to choose which addresses they want to expose to the dapp. Do not give low-level access to all wallet addresses without user consent.

4.5 Signing Request Fails to Display Origin and User Account on Confirmation Message Major

Description

The signing request message does not display the user account used to sign the message. A malicious dapp may pretend to sign a message with one account while issuing an RPC call for a different account.

ShapeShift snap signing requests should implement similar security measures to how MetaMask signing requests work. Being fully transparent on "who signs what", and displaying the origin of the request. This is especially important on multi-dapp snaps to avoid users being tricked into signing transactions they did not intend to sign (wrong signer; dapp race condition).

Please note that we have also reported to the MM Snaps team that dialogs do not, by default, hint at the origin of the action. We hope this will be addressed commonly for all snaps in the future.

Recommendation

Display the signing account in a human-readable and expected format on the signing request. Also, display the origin of the RPC call.

4.6 Control Character and Markdown Injection in `snap_dialog` Major

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.

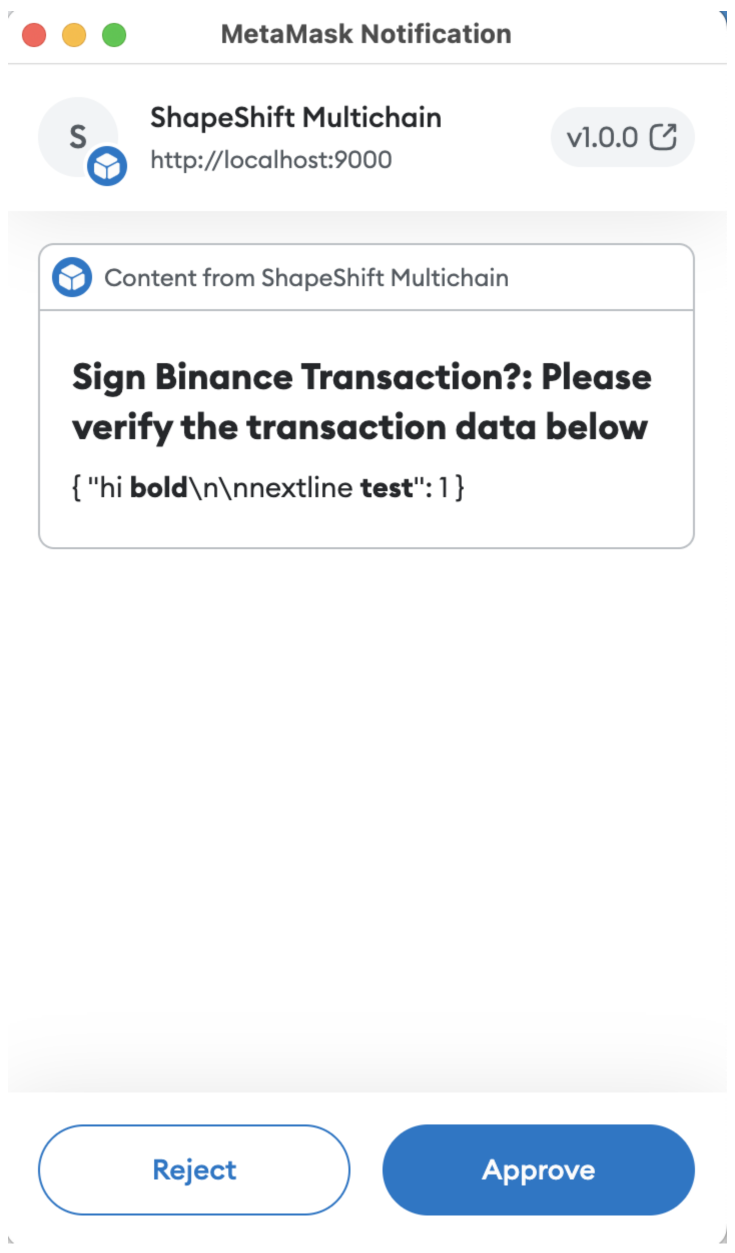
For instance, the text() component can render Markdown or allow for control character injections. Specifically for the ShapeShift snap, this poses a concern because when the snap asks the user to sign structured data, that data might be mistakenly interpreted as Markdown. As a result, the user could inadvertently sign something they did not intend to sign. This means that if the message-to-be-signed contains Markdown renderable text, the displayed message for user approval will be inaccurate.

In the code snippet provided below, please note that the variable `params` is considered potentially untrusted. It may contain Markdown renderable strings or Control Characters that can disrupt the context of the user-displayed message.

Examples

- `UserConfirm` does not sanitize params

```
await window.ethereum.request({
    method: 'wallet_invokeSnap',
    params: {
      snapId: "local:http://localhost:9000",
      request: { method: 'binance_signTransaction', params: {transaction:{"hi **bold**\n\nnextline **test**":1}}},
    }});
```


MetaMask Notification

ShapeShift Multichain
http://localhost:9000
v1.0.0

Content from ShapeShift Multichain

Sign Binance Transaction?: Please verify the transaction data below

{"hi **bold**\n\nnextline **test**":1}

packages/snap/src/rpc/common/utils.ts:L89-L110

```
export const userConfirm = async (params: userConfirmParam): Promise<boolean> => {
  try {
    /* eslint-disable-next-line no-undef */
    const ret = await snap.request({
      method: 'snap_dialog',
      params: {
        type: 'confirmation',
        content: panel([
          heading(`${params.prompt}: ${params.description}`),
          text(params.textAreaContent),
        ]),
      },
    })
    if (!ret) {
      return false
    }
  } catch (error) {
    moduleLogger.error(error, { fn: 'userConfirm' }, 'Could not display confirmation dialog')
    return false
  }
  return true
}
```

packages/snap/src/rpc/common/BaseSigner.ts:L54-L60

```
protected async confirmTransaction(transaction: any): Promise<boolean> {
  return await userConfirm({
    prompt: `Sign ${this.coin} Transaction?`,
    description: 'Please verify the transaction data below',
    textAreaContent: JSON.stringify(transaction, null, 2),
  })
}
```

- `SnapDialog` (not referenced anywhere but potentially vulnerable)

packages/adapter/src/metamask/metamask.ts:L114-L140

```
/**
 * TODO: This is a snap-native call - a handler must be added to the snap onRpcRequest() method to support this.
 */
export const snapDialog = async ({
  prompt,
  description,
  textAreaContent,
}: {
  prompt: string
  description: string
  textAreaContent: string
}): Promise<boolean> => {
  const provider = await getMetaMaskProvider()
  if (provider === undefined) {
    throw new Error('Could not get MetaMask provider')
  }
  if (provider.request === undefined) {
    throw new Error('MetaMask provider does not define a .request() method')
  }
  try {
    const ret = await provider.request({
      method: 'snap_dialog',
      params: {
        type: 'confirmation',
        content: panel([heading(`${prompt}: ${description}`), text(textAreaContent)]),
      },
    })
```

Please note that we have also reported the need for plaintext UI elements to the MM Snaps team. We hope this will be addressed commonly for all snaps in the future.

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. Show derived or decoded information (token recipient) as additional information to the user.

4.7 Lack of High-Level and Inline Documentation Medium

Description

The codebase currently lacks inline documentation, and the repository is missing high-level documentation explaining the Snap capabilities and features. This absence of documentation poses several concerns for future maintenance and transparency. Without inline documentation, as the codebase grows, understanding the code's logic and functionality can be more challenging for developers, making maintenance and bug fixes more time-consuming and error-prone. Additionally, the absence of high-level documentation makes grasping the Snap's intended functionality and capabilities hard for end-users.

Recommendation

We recommend adding inline documentation throughout the codebase to facilitate comprehension of the code's behavior and contribute to its maintainability. We also recommend adding comprehensive high-level documentation in the repository, detailing the Snap's capabilities, features, and intended usage. This will offer insights to developers and end-users, promoting transparency for all parties.

4.8 Notify on Chain Switches and Allow Users to Restrict Access to Chain Specific Functionality and Data Medium

Description

MetaMask core is set to Ethereum as the default network. When switching to BNB or other Networks, Metamask asks the user to confirm the switch. This ensures that, at any point, the user is fully aware of the network they are currently operating on.

ShapeShift Snap exports multi-chain functionality, making it available to connected dapps via the MetaMask RPC. Connected dapps can request operations on various chains without requiring the users to confirm a chain switch. This deviates from the MetaMask security principles of always keeping the user informed about chain switches. Furthermore, the user does not have fine-grained control over what chain functionality is exposed to the dapp.

For example, since there is no origin check in the RPC handler `onRpcRequest()`, any connected dapp may access ShapeShift snap functionality. Some dapps may only require access to Avalanche or Thorchain-related functionality, while others may request access to functionality for several chains. Following the principle of least privilege, the user should be able to choose the chains dapps can access instead of granting access to every chain as soon as the dapp is connected to the snap. Indeed, this behavior poses a substantial phishing risk.

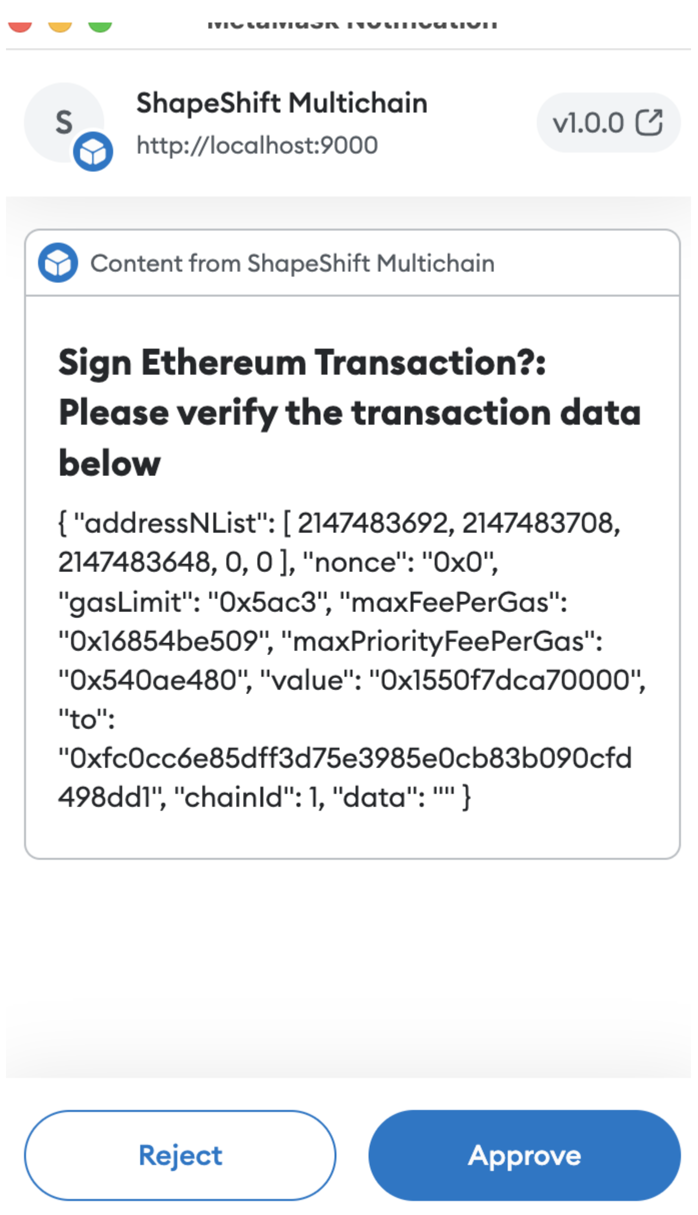
Recommendation

We recommend keeping an internal state of the last chain used. When a dapp requests to access functionality for a different chain, ask the user to confirm the chain switch. Give users control over what chains they want to expose to the dapp and keep a record of their choice. For example, the first time a dapp access Avalanche-specific features, the user should be able to accept or reject the dapp from accessing the network. Incorporate MetaMask's security measures without compromising or weakening them in any way.

4.9 RPC `*_signTransaction` Endpoints Should Display Human Readable Transaction Data Medium

Description

The transaction signing process lacks essential information to make sense of the transaction data object. The addressNList is assumed to be a BIP-32 path without proper explanation, and the contained information is presented in a non-human-readable format. As a result, the user cannot easily identify critical information, such as the signer's address. This leads to a non-user-friendly experience, which also poses security concerns.



ShapeShift Multichain
http://localhost:9000

Content from ShapeShift Multichain

Sign Ethereum Transaction?: Please verify the transaction data below

{ "addressNList": [2147483692, 2147483708, 2147483648, 0, 0], "nonce": "0x0", "gasLimit": "0x5ac3", "maxFeePerGas": "0x16854be509", "maxPriorityFeePerGas": "0x540ae480", "value": "0x1550f7dca70000", "to": "0xfc0cc6e85dff3d75e3985e0cb83b090cfd498dd1", "chainId": 1, "data": "" }

Recommendation

Provide some means for the user to understand what they are signing. Display the signing request origin (multi-dapp usage). Additionally, show the raw data they're actually signing. Decode the BIP-32 key path to a user-readable address.

4.10 Asynchronicity Might Lead to an Undefined Ethereum Provider Minor

Description

The current logic to get the Ethereum provider implemented in `getMetaMaskProvider` does not have any timeout to wait for provider initialization. If this logic is used in the Snap initialization code, because of the asynchronous nature of the initialization code, the function might return an "undefined" provider because the provider is not yet initialized. This would throw an error and prevent the detection of the installed Metamask version. It would be better to provide a safe timeout and wait before deciding whether the provider is undefined.
Alternatively, one could rely on the `@metamask/detect-provider` package (which is already part of the project dependencies) to get the provider.

packages/snap/src/rpc/common/utils.ts:L112-L119

```
const getMetaMaskProvider = async (): Promise<ExternalProvider> => {
  try {
    // eslint-disable-next-line no-undef
    const provider = (window as any).ethereum
    assert(provider, 'Could not detect Ethereum provider')
    return provider
  } catch (error) {
    moduleLogger.error(
```

Recommendation

Rely on the `@metamask/detect-provider` package to get the Ethereum provider or implement a timeout before deciding whether the provider is undefined.

4.11 Avalanche RPC Endpoints `avax_*` Are Enabled in the Snap but Disabled in the Sandbox App Minor

Description

The AVAX functionality is enabled in the snap but disabled in the sandbox dapp. This is problematic as the functionality is present in the snap but not documented or surfaced anywhere.

packages/sandbox/src/components/AssetCardList/AssetCardListConfig.ts:L65-L69

```
name: 'Avalanche',
icon: 'avax.png',
symbol: 'AVAX',
enabled: false,
actions: {
```

packages/snap/src/index.ts:L88-L100

```
export const onRpcRequest: OnRpcRequestHandler = async ({ request }) => {
  const { method, params } = request
  switch (method) {
    case 'avax_getAddress':
      return await avalancheGetAddress(params)
    case 'avax_signMessage':
      return await avalancheSignMessage(params)
    case 'avax_signTransaction':
      return await avalancheSignTransaction(params)
    case 'avax_verifyMessage':
      return await avalancheVerifyMessage(params)
    case 'avax_broadcastTransaction':
      return await ethereumBroadcastTransaction(params)
```

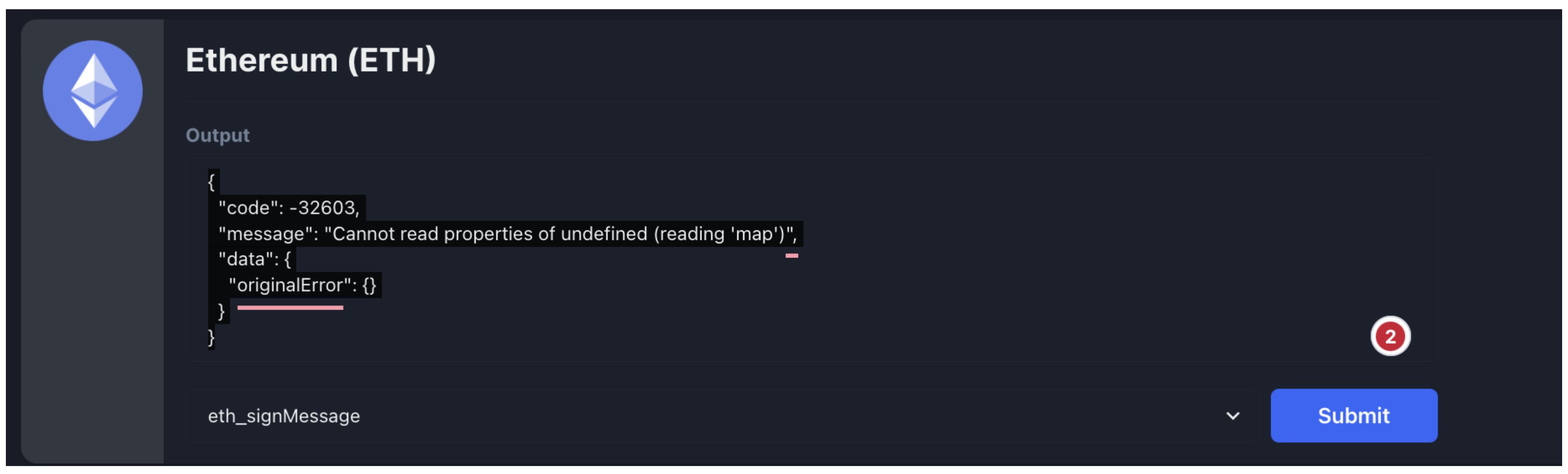
Recommentation

Similarly to other networks, surface the AVAX functionality in the sandbox dapp.

4.12 RPC `[eth|avax]_signMessage` Endpoints Return Errors Minor

Description

The sandbox dapp returns an error when calling `eth_signMessage` :


Ethereum (ETH)

Output

```
{
  "code": -32603,
  "message": "Cannot read properties of undefined (reading 'map')",
  "data": {
    "originalError": {}
  }
}
```

```
{
  "code": -32603,
  "message": "Cannot read properties of undefined (reading 'map')",
  "data": {
    "originalError": {}
  }
}
```

Recommendation

Fix the default message to be signed. Include the from address/HD-path in the signing parameters.

4.13 Unused Interface Declaration Minor

Description

The interface type `RPCRequest` is declared but not referenced in the codebase.

Examples

packages/snap/src/index.ts:L83-L94

```
interface RPCRequest {
  origin: string
  request: ShapeShiftSnapRPCRequest
}

export const onRpcRequest: OnRpcRequestHandler = async ({ request }) => {
  const { method, params } = request
  switch (method) {
    case 'avax_getAddress':
      return await avalancheGetAddress(params)
    case 'avax_signMessage':
      return await avalancheSignMessage(params)
```

Recommendation

```
async ({ request } : RPCRequest ) => {
```

4.14 Every RPC Request Leads to the Creation of a New Signer Object

Description

Every RPC request to the Snap follows the same pattern: it goes through a "handler" that creates a new signer for the proper chain, initializes it, and returns the result of the signer's operation. As an example, here is the handler for the `cosmos_getAddress` RPC request:

packages/snap/src/rpc/cosmossdk/cosmos/handlers.ts:L17-L28

```
export const cosmosGetAddress = async (
  params: CosmosGetAddressParams,
): Promise<CosmosGetAddressResponse> => {
  try {
    const cosmosSigner = new CosmosSigner()
    await cosmosSigner.initialize()
    return await cosmosSigner.getAddress(params)
  } catch (error) {
    moduleLogger.error({ fn: 'cosmosGetAddress' }, error)
    return Promise.reject(error)
  }
}
```

The signer is initialized by deriving the keys from the Metamask BIP32 entropy (in `this.initializeSigner()`) and creating a new unchained HTTP client for the appropriate RPC endpoint.

packages/snap/src/rpc/cosmossdk/cosmos/CosmosSigner.ts:L31-L46

```
async initialize(
  { broadcastUrl }: SignerInitializeArgs = {
    broadcastUrl: broadcastUrls.DEFAULT_UNCHAINED_COSMOS_HTTP_URL,
  },
) {
  const httpProviderConfiguration = new unchained.cosmos.Configuration({
    basePath: broadcastUrl,
  })
  try {
    this.signer = await this.initializeSigner()
    this.httpProvider = new unchained.cosmos.V1Api(httpProviderConfiguration)
    this._initialized = true
  } catch (error) {
    this.logger.error(error, { fn: 'getSigner' }, `Failed to initialize ${this.coin}Signer`)
  }
}
```

While this pattern works, it is sub-optimal performance-wise as the Snap re-creates new signer objects for every RPC request. Assuming RPC requests are generally sequential, e.g., get_address -> sign_tx -> broadcast_tx, this might lead to the creation/garbage collection of many objects with potentially expensive operations (key derivation, for instance). In that case, using the Singleton pattern would likely help as it would prevent creating a new signer to process every RPC request. The presence of the initialization pattern with the "initialized" flag in the signer classes also indicates that the initial engineer's intentions were likely to use such a pattern. Yet, this flag is not used in the current codebase.

Recommendation

While the current code does not pose a problem from a security standpoint, we would recommend leveraging the Singleton pattern for the signer classes. This would prevent creating new signer objects for every RPC request but only a single instance of the signer class for the first request. It would positively impact the performance of the Snap when dealing with many RPC requests.

4.15 Misleading Error Message (Copy-Paste)

Description

`snapDialog` logs a misleading error message if confirmation times out. This seems to be a copy-past error from the `walletSnap` method.

packages/adapter/src/metamask/metamask.ts:L142-L147

```
} catch (error) {
  /** User did not confirm the action or an error was encountered */
  moduleLogger.error(error, { fn: 'walletSnap' }, `wallet_snap_* RPC call failed.`)

  return Promise.reject(error)
}
```

packages/adapter/src/metamask/metamask.ts:L107-L111

```
  return ret
} catch (error) {
  moduleLogger.error(error, { fn: 'walletSnap' }, `wallet_snap_* RPC call failed.`)
  return Promise.reject(error)
}
```

Recommendation

Log an accurate error message.

Appendix 1 - Files in Scope

#	Total	Code	Comment	ToDo	Name	Sha1
1	180	163	12		src/index.ts	ac9282eb466b8c77c80c7a79eecb36957367e00a
2	221	79	130		src/lib/logger.ts	9990887bd33d55eb9d7b347cd55d14be10067a88
3	6	2	4		src/rpc/common/BaseSigner.test.ts	18f499fefea0edb920281fcdd8b9aeb7dc208692
4	75	60			src/rpc/common/BaseSigner.ts	03f9996278fa83fcaae5240a071c660f66982b1d
5	12	12	9		src/rpc/common/constants.ts	d1edfeafb6fd6819b8bf9b935dc2415b5948ce82
6	2	2			src/rpc/common/index.ts	36ff19e2b8a598e7425ba94512bfc8910160ca20
7	221	79	130		src/rpc/common/lib/logger.ts	9990887bd33d55eb9d7b347cd55d14be10067a88
8	177	154	9		src/rpc/common/utils.ts	be5de8a46bd8e312d9cf49d80c20cbce0be549f6
9	11	9	1	1	src/rpc/cosmossdk/binance/BinanceSigner.test.ts	3a8612e2d1a1dc5e13a6e634142761b92b41de96
10	82	76			src/rpc/cosmossdk/binance/BinanceSigner.ts	8de57883e37e4609a5c456cf24fd6650646b2c6e
11	54	49	1		src/rpc/cosmossdk/binance/handlers.ts	cdc02d6a7d970c9957bdc538a91618cff3aba3b9
12	2	2			src/rpc/cosmossdk/binance/index.ts	5f81407ba609cf9f7cf2d90dc941166cf05c675e
13	1	1			src/rpc/cosmossdk/common/CosmosSDKSigner.test.ts	196f479661dbd60636a9ba75e2ec68610bbed93e
14	25	21			src/rpc/cosmossdk/common/CosmosSDKSigner.ts	07b6fa2961928a442f552c85fb039c77081c0943
15	1	1			src/rpc/cosmossdk/common/index.ts	5b8741e76484dafd14c9a7c315159e6106fb0099

#	Total	Code	Comment	ToDo	Name	Sha1
16	1	1			src/rpc/cosmossdk/cosmos/CosmosSigner.test.ts	d4bcab8a350db525be180adfda765e149609e90f
17	100	94			src/rpc/cosmossdk/cosmos/CosmosSigner.ts	53e92f51a8c1aa21eadaae37b6ee2bdfe41e584b
18	54	49			src/rpc/cosmossdk/cosmos/handlers.ts	5c8cd4e5d0736a4bea49f75ab869bdc72fc02fbd
19	2	2			src/rpc/cosmossdk/cosmos/index.ts	a182f06741a3fdefeb521a1cca8a759e63e4ca7b
20	1	1			src/rpc/cosmossdk/kava/KavaSigner.test.ts	dbe897900c33cbe6475735f62f2559ead6fbc2f4
21	80	74			src/rpc/cosmossdk/kava/KavaSigner.ts	42ce2c01e73a5ffe86d776f16d0e0f66c205e83b
22	54	49	1		src/rpc/cosmossdk/kava/handlers.ts	86edcd95554fe8e4e87751d235faefae3b48895d
23	2	2			src/rpc/cosmossdk/kava/index.ts	a0051d4d94158c3c7267dc42dfcc53e4d9e904bc
24	1	1			src/rpc/cosmossdk/osmosis/OsmosisSigner.test.ts	18098e189de77d5629fa85debf86a40123f37497
25	100	94			src/rpc/cosmossdk/osmosis/OsmosisSigner.ts	fda102dfce034a5e120ddfe3edde63aef53988d4
26	54	49			src/rpc/cosmossdk/osmosis/handlers.ts	b178d17ad6b4a73b8449c08242c40f8c6b747427
27	2	2			src/rpc/cosmossdk/osmosis/index.ts	c0bd1cc0d33821d6e48238a0de3b0e00c4c737d3
28	1	1			src/rpc/cosmossdk/secret/SecretSigner.test.ts	fc2761333e0b6dd106bca4425dabc384cb138804
29	82	76			src/rpc/cosmossdk/secret/SecretSigner.ts	24a55e442a6cef8b8578fc36c8f9031311fe4b37
30	54	49	1		src/rpc/cosmossdk/secret/handlers.ts	6d02ca6c6657b1581d6618fafb4004b4d5c27d98
31	2	2			src/rpc/cosmossdk/secret/index.ts	8e99418cfa3eef173e86305daecedf6b3d067671
32	1	1			src/rpc/cosmossdk/terra/TerraSigner.test.ts	6120fa23eb1c3bd2a838c205dac24a1fe567e4e5
33	80	74			src/rpc/cosmossdk/terra/TerraSigner.ts	8ccfb3cfb0defa8500813ed5ffbe2b2bdcda54f4
34	54	49	1		src/rpc/cosmossdk/terra/handlers.ts	7c15638d262d1976b23cdf29f7290fa02823098e
35	2	2			src/rpc/cosmossdk/terra/index.ts	4487180ae3a451232d0776385bb199ce89dc9a47
36	1	1			src/rpc/cosmossdk/thorchain/ThorchainSigner.test.ts	afd701c57fcdafaaa1a27478a95ea96af51b7a24
37	100	94			src/rpc/cosmossdk/thorchain/ThorchainSigner.ts	f163a9d4949bdc6a195d9751f28e00a4cf323891
38	54	49			src/rpc/cosmossdk/thorchain/handlers.ts	fb06d91192a3f90dea822c081a983b9636991f4b
39	2	2			src/rpc/cosmossdk/thorchain/index.ts	2106036a925ccdcf8fa281631bb6b8b22475b7ed
40	1	1			src/rpc/evm/avalanche/AvalancheSigner.test.ts	2f232733f8d076e0bd10404bda1ef385d27db947
41	36	33			src/rpc/evm/avalanche/AvalancheSigner.ts	a6128fc3ed74b453a26832648f232c91e6ee04a5
42	84	77			src/rpc/evm/avalanche/handlers.ts	7d962713a11dbe8fa090a8a37bc59a2f17cc56b9
43	2	2			src/rpc/evm/avalanche/index.ts	562bdf2b769b2347a514d767c9c2796f63511a33

#	Total	Code	Comment	ToDo	Name	Sha1
44	91	85			src/rpc/evm/common/EVMSigner.ts	5092c0dfb4e5f5235d8ba0be10e7c76b045bd870
45	1	1			src/rpc/evm/common/index.ts	6433895fcdc01d752b66a6a2a44f0aee9ae3deaa
46	1	1			src/rpc/evm/ethereum/EthereumSigner.test.ts	fa40839f6703a3069be49bd93041ae6aa68293dd
47	34	31			src/rpc/evm/ethereum/EthereumSigner.ts	49a5c0ce31a33b7c50ad0596c895686375cbd83c
48	84	77			src/rpc/evm/ethereum/handlers.ts	019e57b1112bdcb7d5be1e7800c034e7d255900f
49	2	2			src/rpc/evm/ethereum/index.ts	fd90ef10dfddae80515989f1e5c232e499fa6db7
50	1	1			src/rpc/utxo/bitcoin/BitcoinSigner.test.ts	a39f3bf4e46521090d25a7daae33ee9e2058242f
51	36	32			src/rpc/utxo/bitcoin/BitcoinSigner.ts	bde2c49f5a732b8528c71e7230a05c8386fa50af
52	54	49			src/rpc/utxo/bitcoin/handlers.ts	f30140e47c395e0adeaec1dcd6c2aab49f69e952
53	2	2			src/rpc/utxo/bitcoin/index.ts	9dd92ccb55e6208d452f4e2870b831c8c0f05402
54	1	1			src/rpc/utxo/bitcoincash/BitcoinCashSigner.test.ts	580add5459e4ec691b5d1e3e709cffa4cd136365
55	34	31			src/rpc/utxo/bitcoincash/BitcoinCashSigner.ts	75170d17d530265769541c32908289f7bf3e3cb1
56	54	49			src/rpc/utxo/bitcoincash/handlers.ts	8c5163a724ab576f855b4c892f065f909b7c4668
57	2	2			src/rpc/utxo/bitcoincash/index.ts	dc422227cb5ef5f5b1c30ad409d42fa4bf14e901
58	1	1			src/rpc/utxo/common/UTXOSigner.test.ts	4052c183d21d3c2cd7787a5d2282cbe6260f916d
59	63	59			src/rpc/utxo/common/UTXOSigner.ts	9fd6c43fdb613c8331a4007f284e5a98870963ab
60	1				src/rpc/utxo/common/index.ts	da39a3ee5e6b4b0d3255bfef95601890afd80709
61	1	1			src/rpc/utxo/dogecoin/DogecoinSigner.test.ts	ea7457e69b0dc876f66b08e267b7b91ce17690a1
62	34	31			src/rpc/utxo/dogecoin/DogecoinSigner.ts	c5a24c40b39833ae1e4d8c32a4305db090651363
63	54	49			src/rpc/utxo/dogecoin/handlers.ts	790e7bf3efce3e5ebf7ba65b7f82beb298952f3e
64	2	2			src/rpc/utxo/dogecoin/index.ts	5e8d5578e57a244d9994d7b0b3b68ce06ab5a36e
65	1	1			src/rpc/utxo/litecoin/LitecoinSigner.test.ts	5bfeaf80764217dca5db0a5a2c1aca3a65b2a466
66	34	31			src/rpc/utxo/litecoin/LitecoinSigner.ts	0be56dac69efd2cc25e147841ce43d996a6c2464
67	54	49			src/rpc/utxo/litecoin/handlers.ts	d816f61ce7ecffb257d7513a7e2217924cb83fb0
68	2	2			src/rpc/utxo/litecoin/index.ts	cc160f14f2efc5ab742ff5ac7ac468c145f60888
69	15	10	2		src/types/vendor/hook-shell-script-webpack-plugin.d.ts	5f5c09a92888e67343baa941f183e9170c15e7b6
=====	=====	=====	=====	=====		
Σ	2736	2241	301	1		

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.