# MSQ Snap



| | |
|---|---|
| **Date** | March 2024 |
| **Auditors** | Valentin Quelquejay, Martin Ortner |

# 1 Executive Summary

This report presents the results of our engagement with the **MSQ** to review the **MSQ Snap & dApp**.

The review was conducted over two and a half weeks, from **March 11, 2024** to **March 27, 2024**. A total of 2x13 person-days were spent.

The MSQ project is composed of three components: a MetaMask Snap, a properietary dApp hosted on the Internet Computer, and a client library. The Snap is specifically designed for interactions with the Internet Computer (ICP), while the proprietary dApp extends the Snap's capabilities. MSQ is crafted for seamless integration with external dApps, facilitated by the client library. It offers a range of functionalities, including authorization and payments.

## 1.1 Remarks

Users interacting with web services that are integrated with MSQ can authenticate within these services using origin-bound identities deterministically derived from their MetaMask root key. As long as users maintain possession of their seed phrase, they can recover access to any previously derived identities. This setup enables users to sign arbitrary messages from web services using these identities, thus allowing for interactions with Internet Computer canisters. Additionally, users can utilize their MSQ-managed assets to pay for goods and services.

To use their identities, users need to explicitly connect the Snap to external dApps by interacting through the management dApp. Thus, they need to give consent for the Snap to use their derived identity on a per-origin basis. Yet, note that once connected to a dApp, dApps can ***sign arbitrary data*** without requiring explicit user consent, meaning it's crucial for users to trust the dApps they authorize (see issues issue 4.1 and issue 4.4 for additional details).

The client provided a list of key risks, which we reviewed and concluded are acceptable given the current implementation. The Snap utilizes the `snap_getEntropy` function to generate entropy for different origins, ensuring that the generated entropy is safe and provides an adequate level of randomness. Additionally, Snap storage is encrypted, protecting data from unauthorized access. To mitigate supply-chain attacks, only trusted dependencies should be used. Furthermore, dependencies should be kept to a minimum, and fixed versions should be used. Finally, the Snap should protect the user at all times by ensuring that the user gives explicit consent to any privileged action (with the caveat specific to the ICP technology explained above).

## 1.2 Engagement

The collaboration between our team and the client team has demonstrated a cooperative and committed approach to security principles. The codebase shows organization and clarity, with thorough inline documentation enhancing readability and maintainability. Notably, TypeScript is utilized for compile-time type enforcement, alongside `zod`-based runtime type validation and input sanitization, aligning well with recommended practices for Snaps development.

Additionally, the client has taken proactive measures to reduce the attack surface and mitigate risks to users. The client managed to implement a good tradeoff between security and usability addressing inherent requirements to ICP protocol interaction. They have shown responsiveness by implementing changesets to mitigate identified vulnerabilities promptly, and fixing similar issues even beyond what's been shared with them initially.

# 2 Scope

The review focused on the commit hash 6f5b16dddf99624e2874876145987002ba6d4df5. The list of files in scope can be found in the Appendix.

The following supporting documents were provided:

- Functionality and architecture
- Integration
- User Stories
- Intro Video

## 2.1 Objectives

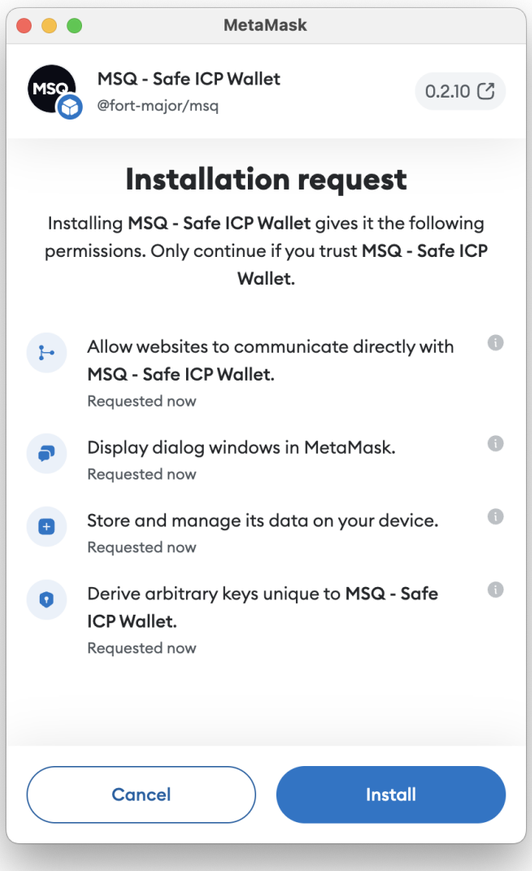
Together with the **MSQ** team, we identified the following priorities for our review:

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. Review key risks outlined in the MSQ audit docs.

# 3 Snap Outline

- The snap can access MetaMask root key entropy via `snap_getEntropy(salt)`.
- Connected dApps can communicate with the snap via RPC calls.
- RPC calls are compartmentalized. The trusted msq.tech dApp can call privileged RPC endpoints, while other dApps can only access lower-privileged endpoints:
    - Privileged endpoints (`msq.tech` dApp only):
        - protected_identity_add
        - protected_identity_login
        - protected_identity_getLoginOptions
        - protected_identity_editPseudonym
        - protected_identity_stopSession
        - protected_identity_unlinkOne
        - protected_identity_unlinkAll
        - protected_icrc1_showTransferConfirm
        - protected_icrc1_addAsset
        - protected_icrc1_addAssetAccount
        - protected_icrc1_editAssetAccount
        - protected_statistics_get
        - protected_statistics_increment
        - protected_statistics_reset
        - protected_state_getAllOriginData
        - protected_state_getAllAssetData
    - Public endpoints:
        - public_identity_sign,
        - public_identity_getPublicKey
        - public_identity_getPseudonym
        - public_identity_requestLogout
        - public_identity_requestLink
        - public_identity_requestUnlink
        - public_identity_getLinks
        - public_identity_sessionExists

## 3.1 Capabilities




### Details

```
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
endowment:rpc { dapps: true, snaps: false }
snap_dialog {}
snap_manageState {}
snap_getEntropy {}
---->%---- raw permissions
🚜 [endowment:rpc]
    ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
       🔶 src/index.ts
🚜 [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/protocols/identity.ts
       🔶 src/protocols/icrc1.ts
🚜 [snap_manageState]
    🪠 - snap_manageState - snap can store up to 100mb (isolated)
       🔶 src/state.ts
🚜 [snap_getEntropy]
    ✨ - snap_getEntropy - Gets a deterministic 256-bit entropy value, specific to the snap and the user's account. You can use this entropy to generat
       🔶 src/utils.ts
```

# 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 Public RPC Methods and Consent Management Major ✓ Fixed

### Resolution

Addressed with the following changeset: fort-major/msq@ `7f9cde2` .

- login now requires user confirmation via a MM consent message;
- the session now has 2h expiry; the next signature request after the session's expiration triggers a prompt that allows the user to either refresh the session or to log out;
- both prompts clearly state that the website, where the user is logging it, will be able to silently sign messages on behalf of one of user's identities;

Additionally, we were very concerned about the problem, we've discussed on the call. Namely, that in case of attackers being able to replace the code of the MSQ dapp (via a DNS attack, for example), they will be able to drain user's wallets empty. In this commit we've also addressed this issue the following way:

- the snap sign API now requires you to supply not the hash of the transaction, but it's body instead (we were able to find a way to make this work without changing any client-side API);
- the sign snap method then calculates the hash by itself and signs this hash;
- optionally, if the API was used from the MSQ website and we detect potentially harmful transaction (one that could move user's funds), then we prompt the user with details of this transaction and sign the message only in case the user confirms it.

This makes users' assets immutable to even such deadly attacks like ones discussed. Currently this harmful transaction detection only works at MSQ website - other websites are free to sign any transactions they want without users noticing it (but they are now semi-protected with session expiry).

### Description

User consent may not consistently be enforced. Identities are bound to their origin (URL). Third-party origins are outside the scope of this Snap and are therefore in a lower trust zone where it is uncertain what security measures are in place to protect the dApp from impersonating the user's wallet identity. dApps may be hosted on integrity-protecting endpoints (ipfs/IC), however, this is not enforced. Additionally, even when hosted on integrity-protecting endpoints there are still risks of insider and external attacks on the deployed dApp (Insider changing code, External attacker gaining access to code, Injection, Web Attacks), BGP routing related attacks (typically expensive), and DNS related attacks.

Allowing linked identities to sign with a main origin's identity extends the risk from one public origin to another.

It should be noted that identities on public RPC methods are origin bound. There is no direct way for one public origin to sign with another origin's identity unless it is linked.

### Examples

- critical functionality: acting on behalf of user
  - `SNAP_METHODS.public.identity.sign` - sign with origin bound identity
- potential identity leak
  - `SNAP_METHODS.public.identity.getPublicKey` - origin bound identity
  - `SNAP_METHODS.public.identity.getPseudonym` - origin bound pseudonym
- no concerns
  - `SNAP_METHODS.public.identity.getLinks` - origin links
  - `SNAP_METHODS.public.identity.sessionExists` - check if session for origin exists

**Example: signing**

The function `handleIdentitySign` is responsible for signing a payload with an identity. However, it has been observed that the function proceeds to sign the payload without seeking explicit user confirmation or displaying the payload in a human-readable format. This approach can significantly undermine the security and trust model of MetaMask Snaps by allowing potentially malicious operations to be executed without the user's informed consent.

**packages/snap/src/protocols/identity.ts:L268-L280**

```
export async function handleIdentitySign(bodyCBOR: string, origin: TOrigin): Promise<ArrayBuffer> {
  const body: IIdentitySignRequest = zodParse(ZIdentitySignRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();
  let session = (await manager.getOriginData(origin)).currentSession;

  if (session === undefined) {
    err(ErrorCode.UNAUTHORIZED, "Log in first");
  }

  const identity = await getSignIdentity(session.deriviationOrigin, session.identityId, body.salt);

  return await identity.sign(body.challenge);
}
```

### Recommendation

When performing critical actions on behalf of the user, always ask for consent. The user must always be notified when a dApp acts on their behalf (especially signing). For API that provides less critical information it should be considered to implement a session based consent mechanism that trades security for convenience where, e.g., linked identities or the public key can only be extracted if the user at least once confirmed this for the current origin (caching the decision).

## 4.2 Keypairs Generated by Dapps Might Be Unrecoverable Which Could Result in Loss of Funds Major

✓ Fixed

**Resolution**

The client acknowledged and mitigated this issue in commit fort-major/msq@ `59a0b88` . Note that the companion dApp can still sign data with arbitrary salts, but external dApps cannot anymore.

### Description

The current Snap implementation allows untrusted dapps to supply their own nonces for the generation of unique private keys tied to their identities. These nonces, which can be arbitrary and are not managed or stored by the Snap, introduce a risk. Specifically, if a dapp fails to securely store these nonces or ceases operation, users may irretrievably lose access to their accounts, potentially resulting in the loss of funds. This issue underscores a vulnerability in the system's design, where the reliance on external parties for the management of crucial security parameters compromises the safety and recoverability of user assets.

### Examples

From the documentation:

> MSQ is a Snap, it has no cloud storage, so all the data is self-managed by the user. **It is safe for users to lose their data (by re-installing the snap, for example) - because they can recover it from their seed phrase later.** To achieve that we use deterministic algorithms for entropy derivation.

**packages/snap/src/protocols/identity.ts:L298-L309**

```
  const body = zodParse(ZIdentityGetPublicKeyRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();
  let session = (await manager.getOriginData(origin)).currentSession;

  if (session === undefined) {
    err(ErrorCode.UNAUTHORIZED, "Log in first");
  }

  const identity = await getSignIdentity(session.deriviationOrigin, session.identityId, body.salt);

  return identity.getPublicKey().toRaw();
}
```

### Recommendation

To mitigate the risk of users losing access to their accounts and funds due to mismanaged or lost nonces by dapps, it is recommended to either:

- Enforce Deterministic Nonces: Shift to a system where the Snap generates unique identities using deterministic nonces, ensuring all accounts are recoverable, independent of dapps actions or continuity. This is already implemented to generate "root" dapps identities.
- Introduce Disclaimers: Clearly inform users and developers of the risks through disclaimers in the user interface and documentation.

## 4.3 Timestamp Logic Flaws in Snap's Caching Mechanism Medium ✓ Fixed

**Resolution**

Addressed in commit 4d6b006f2870b8b560e068ae51821d9d962d129a

### Description

The Snap employs a custom wrapper around its storage, incorporating a caching mechanism to optimize performance by updating the Snap's storage only when necessary. This caching strategy uses a timestamp-based method, maintaining records of

the last storage ( `LAST_STATE_PERSIST_TIMESTAMP` ) and state ( `STATE_UPDATE_TIMESTAMP` ) updates to decide on the need for persisting the updated state. However, two logical flaws were identified in this approach:

- First, in the `retrieveStateWrapped()` function, where the state is fetched from the Snap's storage, the state update timestamp is only refreshed if the retrieved state is null. Consequently, if an existing state is successfully retrieved, the state update timestamp remains unchanged, which is incoherent.

**packages/snap/src/state.ts:L464-L472**

```
  if (state == null) {
    const s = makeDefaultState();
    STATE = s;

    STATE_UPDATE_TIMESTAMP = Date.now();
  } else {
    STATE = zodParse(ZState, fromCBOR(state.data as string));
  }
}
```

- Second, the `persistStateLocal()` function, responsible for persisting the state, updates the `LAST_STATE_PERSIST_TIMESTAMP` before the actual state update is executed. This can lead to an invalid timestamp if the subsequent state update operation (e.g., due to CBOR encoding errors or failures in the internal Snap's storage mechanism) fails, risking state and storage inconsistencies.

**packages/snap/src/state.ts:L527-L538**

```
async function persistStateLocal(): Promise<void> {
  if (LAST_STATE_PERSIST_TIMESTAMP >= STATE_UPDATE_TIMESTAMP) return;

  zodParse(ZState, STATE);

  LAST_STATE_PERSIST_TIMESTAMP = Date.now();

  await snap.request({
    method: "snap_manageState",
    params: {
      operation: "update",
      newState: { data: toCBOR(STATE) },
```

### Recommendation

Adjust the timestamp update logic as follows:

- In `retrieveStateWrapped()`, ensure the `STATE_UPDATE_TIMESTAMP` is refreshed whenever a state (not null or empty) is retrieved, not just when the state is null.
- In `persistStateLocal()`, update the `LAST_STATE_PERSIST_TIMESTAMP` only after the state has been successfully persisted, preventing inaccuracies in the event of a failed state update.

## 4.4 Protected (Administrative Origin) RPC Methods and Consent Management Medium ✓ Fixed

### Resolution

Addressed with the following changesets: fort-major/msq@ `7f9cde2` and fort-major/msq@ `0b9f8d1` (removing whitelisted method names, only allowing `icrc1_transfer` )

The client provided the following statement:

- login now requires user confirmation via a MM consent message;
- the session now has 2h expiry; the next signature request after the session's expiration triggers a prompt that allows the user to either refresh the session or to log out;
- both prompts clearly state that the website, where the user is logging it, will be able to silently sign messages on behalf of one of user's identities;

Additionally, we were very concerned about the problem, we've discussed on the call. Namely, that in case of attackers being able to replace the code of the MSQ dapp (via a DNS attack, for example), they will be able to drain user's wallets empty. In this commit we've also addressed this issue the following way:

- the snap sign API now requires you to supply not the hash of the transaction, but it's body instead (we were able to find a way to make this work without changing any client-side API);
- the sign snap method then calculates the hash by itself and signs this hash;
- optionally, if the API was used from the MSQ website and we detect potentially harmful transaction (one that could move user's funds), then we prompt the user with details of this transaction and sign the message only in case the user confirms it.

This makes users' assets immutable to even such deadly attacks like ones discussed. Currently this harmful transaction detection only works at MSQ website - other websites are free to sign any transactions they want without users noticing it (but they are now semi-protected with session expiry).

### Description

Identities are bound to their origin (URL). Third-party origins are outside the scope of this Snap and are therefore in a lower trust zone where it is unsure what security measures are in place to protect the dApp from impersonating the users' wallet identity. dApps may be hosted on integrity protecting endpoints (ipfs/IC), however, this is not enforced.

Protected RPC functions can only be invoked by the MSQ administrative origin. User consent may not consistently be enforced on the administrative origin.

The administrative origin is identified by the origin URL. According to the client the dApp is hosted on an integrity protecting endpoint (IC). This already protects from direct manipulation of the deployed code, however, it may still be problematic as the Snap and Management dApp are in different trust zones with the dApp being exposed to many external factors that make it more prone to web related attacks. That said, even when hosted on integrity protecting endpoins there are still risks of insider and external attacks on the deployed dApp (Insider changing code, External attacker gaining access to code, Injection, Web Attacks), BGP routing related attacks (typically expensive), and DNS related attacks. In the worst case, an insider/external attacker gaining control of the trusted origin may be able to perform actions on many users behalf's without them knowing (given that the user accesses the management origin).

### Examples

- critical
  - `SNAP_METHODS.protected.identity.login` - log into any origin
  - `SNAP_METHODS.protected.identity.editPseudonym`
  - `SNAP_METHODS.protected.icrc1.editAssetAccount`
- unclear
  - `SNAP_METHODS.protected.statistics.get`
  - `SNAP_METHODS.protected.statistics.increment`
  - `SNAP_METHODS.protected.statistics.reset`
- potential privacy leak
  - `SNAP_METHODS.protected.state.getAllOriginData` - subset of origin data (no keys)
  - `SNAP_METHODS.protected.state.getAllAssetData` - subset of asset data
  - `SNAP_METHODS.protected.identity.getLoginOptions`

### Recommendation

When performing critical actions on behalf of the user, always ask for consent. The user must always be notified when a dApp acts on their behalf (especially signing). For API that provides less critical information it should be considered to implement a lazy session based consent mechanism that trades security for convenience where, i.e., data can only be extracted from the snap if the user at least once confirmed this for the current session.

## 4.5 Protected_handleIdentityLogin - Unchecked `withIdentityId` Medium ✓ Fixed

| Resolution |
| --- |
| Addressed with the following changeset enforcing that a valid `identityId` was supplied: fort-major/msq@`59a0b88` |

### Description

`protected_handleIdentityLogin` is used to create a new session and logs in to a particular origin (e.g., a website). At a certain point, a new session object is created, where `identityId: body.withIdentityId` is set. The `withIdentityId` is an unchecked request parameter, which could potentially lead to an inconsistency where the origin set's an invalid ID.

### Examples

#### packages/snap/src/protocols/identity.ts:L76-L101

```
export async function protected_handleIdentityLogin(bodyCBOR: string): Promise<true> {
  const body: IIdentityLoginRequest = zodParse(ZIdentityLoginRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();

  if (body.withLinkedOrigin !== undefined && body.withLinkedOrigin !== body.toOrigin) {
    if (!manager.linkExists(body.withLinkedOrigin, body.toOrigin))
      err(ErrorCode.UNAUTHORIZED, "Unable to login without a link");
  }

  const originData = await manager.getOriginData(body.toOrigin);
  if (Object.keys(originData.masks).length === 0) {
    unreacheable("login - no origin data found");
  }

  const timestamp = new Date().getTime();
  originData.currentSession = {
    deriviationOrigin: body.withLinkedOrigin ?? body.toOrigin,
    identityId: body.withIdentityId,
    timestampMs: timestamp,
  };

  manager.setOriginData(body.toOrigin, originData);
  manager.incrementStats({ login: 1 });

  return true;
}
```

### Recommendation

`withIdentityId` is an id relative to the origins masks, hence, it should be validated against the number of existing masks.

The client validated the issue providing more context:

> Moreover, when withLinkedOrigin is provided, it should validate against masks of linked origin data. And if not, then it should validate with the originData.masks.

## 4.6 Protected_handleAddAssetAccount - Should Verify `name`, `symbol` Matches `assetId` Medium

✓ Fixed

### Resolution

Addressed with the following changeset, escaping and validating asset data more strictly, verifying that the assetId is valid and displaying symbol and name from it's internal data: fort-major/msq@ `59a0b88`

### Description

The function `protected_handleAddAssetAccount` adds an account to an existing asset. It takes the asset name/symbol and assetId as inputs and then adds an account to the `assetId` if the user approves.

The dialog shown to the user displays the target assets `symbol` and `name`. However, this information comes from the dApp and it only used within the dialog. There is no check if the `assetId` matches the `name` and `symbol` which might allow the dApp to mislead the user into accepting the addition of an account for an asset that does not match the displayed `name` and `symbol`.

### Examples

**packages/snap/src/protocols/icrc1.ts:L90-L113**

```
export async function protected_handleAddAssetAccount(bodyCBOR: string): Promise<string | null> {
  const body = zodParse(ZICRC1AddAssetAccountRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();

  const agreed = await snap.request({
    method: "snap_dialog",
    params: {
      type: "confirmation",
      content: panel([
        heading(`🔒 Confirm New ${body.symbol} Account 🔒`),
        text(`Are you sure you want to create a new **${body.name}** (**${body.symbol}**) token account?`),
        text(`This will allow you to send and receive **${body.symbol}** tokens.`),
        divider(),
        text("**Confirm?** 🚀"),
      ]),
    },
  });

  if (!agreed) return null;

  const accountName = manager.addAssetAccount(body.assetId);

  return accountName;
}
```

### Recommendation

The function should take an `assetId` as input parameter only. Then check if the `assetId` has accounts registered. If that's the case, display the `name`, `symbol` that corresponds to the `assetId` and add an account upon user confirmation.

## 4.7 Entropy / Signature Handling & Hardening Medium ✓ Fixed

### Resolution

The following changeset is addressing this issue by being more strict on type and input validation: fort-major/msq@ `26a0eec`

### Description

The functions `getBaseEntropy` and `getSignIdentity` lack validation for the correct type of arguments or the presence of control characters that may allow context breaks (newline).

For example, in the `SNAP_METHODS.public.identity.requestLink` method handler, the `body.withOrigin` is unsanitized and concatenated directly for the resulting salt. `withOrigin` is a user provided value and may include `\n` which breaks the context of the salt structure.

### Examples

**packages/snap/src/utils.ts:L75-L89**

```
export async function getSignIdentity(
  origin: TOrigin,
  identityId: TIdentityId,
  salt: Uint8Array,
): Promise<Secp256k1KeyIdentity> {
  // the MSQ site has constant origin
  // this will allow us to change the domain name without users losing their funds and accounts
  const orig = isMsq(origin) ? "https://msq.tech" : origin;

  // shared prefix may be used in following updates
  const entropy = await getEntropy(orig, identityId, "identity-sign\nshared", salt);

  return Secp256k1KeyIdentity.fromSecretKey(entropy);
}
```

**packages/snap/src/utils.ts:L105-L115**

```
async function getBaseEntropy(origin: TOrigin, identityId: TIdentityId, internalSalt: string): Promise<Uint8Array> {
  const generated: string = await snap.request({
    method: "snap_getEntropy",
    params: {
      version: 1,
      salt: `\x0amsq-snap\n${origin}\n${identityId}\n${internalSalt}`,
    },
  });

  return hexToBytes(generated.slice(2));
}
```

### Recommendation

To address this, enforcing that `identityId` is a positive number, valid id and `origin` is a valid URL (free from control characters) would mitigate misuse of this functionality.

## 4.8 makeAvatarSvgCustom - Potential SVG HTML Injection, React `innerHTML` Medium ✓ Fixed

**Resolution**

Addressed with fort-major/msq@`26a0eec` by adding `dompurify` to the manually generated SVG-XML, still using `innerHTML` but sanitized for XSS, escaping `bgcolor` (incomplete: see below) and turning `makeAvatarSvgCustom` into an internal function with `makeAvatarSvg` being the external interface where most params cannot be directly controlled.

Additional fix addressing shortcomings of the previous solution as `escapeHtml` cannot be used for HTML-Attrib sanitization, adding regex checks for html color arguments for `makeAvatarSvgCustom`: fort-major/msq@`59a0b88`

### Description

The function `makeAvatarSvgCustom` inserts the given arguments directly into a string that represents an XML SVG image. Since the arguments are not sanitized, there is a potential risk for XML-SVG injection, which could include malicious scripts.

Please note that this affects all arguments provided to the function, especially `bgColor`, but also the ones that are calculating `cx,cy` because the addition turns into a string concatenation if `face[xy]` is a string.

The severity rating is based on the current exploitability which is comparable low with the demo implementations of the front-end.

### Examples

**packages/shared/src/avatar.ts:L23-L89**

```ts
/**
 * Generates a custom avatar SVG string based on provided parameters including body color, body angle,
 * face expression, and optional background and eye colors. This function allows for the creation of a
 * personalized avatar with specific characteristics defined by the input parameters. The SVG is constructed
 * with various elements such as circles for the body and eyes, and a custom path for the face expression.
 * Additional details like eye pupils and mouth are also included, with positions adjusted based on the body angle.
 *
 * @param {string} id - A unique identifier used to generate clip paths for the eyes, ensuring they are unique within the SVG.
 * @param {string} bodyColor - The fill color for the avatar's body.
 * @param {IAngle} bodyAngle - An object containing the center coordinates for the body and face, used to position elements.
 * @param {string} faceExpression - A string representing the SVG path for the face expression.
 * @param {string} [bgColor="#1E1F28"] - Optional background color of the SVG. Defaults to a dark gray if not specified.
 * @param {string} [eyeWhiteColor="white"] - Optional color for the whites of the eyes. Defaults to white if not specified.
 * @returns {string} A string representation of the SVG for the custom avatar.
 */
export function makeAvatarSvgCustom(
  id: string,
  bodyColor: string,
  bodyAngle: IAngle,
  faceExpression: string,
  bgColor: string = "#1E1F28",
  eyeWhiteColor: string = "white",
): string {
  const { bodyCx, bodyCy, faceX, faceY } = bodyAngle;

  const eyeWhite1Cx = faceX + EYE_WHITE_1_CX;
  const eyeWhite1Cy = faceY + EYE_WHITE_1_CY;
  const eyePupil1Cx = faceX + EYE_PUPIL_1_CX;
  const eyePupil1Cy = faceY + EYE_PUPIL_1_CY;

  const eyeWhite2Cx = faceX + EYE_WHITE_2_CX;
  const eyeWhite2Cy = faceY + EYE_WHITE_2_CY;
  const eyePupil2Cx = faceX + EYE_PUPIL_2_CX;
  const eyePupil2Cy = faceY + EYE_PUPIL_2_CY;

  const mouthX = faceX + MOUTH_X;
  const mouthY = faceY + MOUTH_Y;

  return `
    <svg xmlns="http://www.w3.org/2000/svg" width="100" height="100" style="position:relative;width:100%;height:100%;" viewBox
      <defs>
        <clipPath id="clip-eye-1-${id}">
          <circle cx="${eyeWhite1Cx}" cy="${eyeWhite1Cy}" r="6" />
        </clipPath>
        <clipPath id="clip-eye-2-${id}">
          <circle cx="${eyeWhite2Cx}" cy="${eyeWhite2Cy}" r="6" />
        </clipPath>
      </defs>

      <rect id="bg" x="0" y="0" width="100" height="100" fill="${bgColor}"/>

        <g id="body-group">
          <circle id="body" cx="${bodyCx}" cy="${bodyCy}" r="50" fill="${bodyColor}" />

          <circle id="eye-white-1" cx="${eyeWhite1Cx}" cy="${eyeWhite1Cy}" r="6" fill="${eyeWhiteColor}" />
          <circle id="eye-pupil-1" cx="${eyePupil1Cx}" cy="${eyePupil1Cy}" r="5" fill="#0A0B15" clip-path="url(#clip-eye-1-${id}

          <circle id="eye-white-2" cx="${eyeWhite2Cx}" cy="${eyeWhite2Cy}" r="6" fill="${eyeWhiteColor}" />
          <circle id="eye-pupil-1" cx="${eyePupil2Cx}" cy="${eyePupil2Cy}" r="5" fill="#0A0B15" clip-path="url(#clip-eye-2-${id}

          <g transform="translate(${mouthX}, ${mouthY})" id="mouth">
            ${faceExpression}
          </g>
        </g>
      </svg>
    `;
}
```

used in:

**apps/site/src/frontend/components/boop-avatar/index.tsx:L37-L54**

```ts
export function CustomBoopAvatar(props: ICustomBoopAvatarProps) {
  return (
    <BoopAvatarWrapper
      classList={props.classList}
      size={props.size}
      ref={(r) => {
        r.innerHTML = makeAvatarSvgCustom(
          props.id,
          props.bodyColor,
          props.angle,
          FACE_EXPRESSIONS[props.faceExpression - 1],
          props.bgColor,
          props.eyeWhiteColor,
        );
      }}
    />
  );
}
```

**packages/client/src/identity.ts:L69-L83**

```
/**
 * ## Returns user's avatar for current MSQ identity
 *
 * This avatar is an auto-generated SVG image
 * and should be treated as an easy way to render avatars for users without profiles.
 *
 * @param {string | undefined} bgColor
 * @returns {Promise<string>} avatar SVG src string as "data:image/svg+xml..."
 */
getAvatarSrc(bgColor?: string): Promise<string> {
  const principal = this.getPrincipal();
  const svg = btoa(makeAvatarSvg(principal, bgColor));

  return Promise.resolve(`${svg}`);
}
```

**apps/demo/src/frontend/pages/index/index.tsx:L73-L76**

```
const profile: IProfile = {
  pseudonym: await identity.getPseudonym(),
  avatarSrc: await identity.getAvatarSrc(),
};
```

used with `innerHTML`

**apps/site/src/frontend/components/boop-avatar/index.tsx:L15-L24**

```
export function BoopAvatar(props: IBoopAvatarProps) {
  return (
    <BoopAvatarWrapper
      size={props.size}
      ref={(r) => {
        r.innerHTML = makeAvatarSvg(props.principal);
      }}
    />
  );
}
```

### Recommendation

Runtime typecheck provided values (numbers vs. strings). Sanitize and validate arguments before embedding then with HTML or use a templating language to build the SVG (recommended)

## 4.9 CtrlChar/Markdown Injection Medium Partially Addressed

**Resolution**

Addressed with the following changeset, wrapping the Snap native UI Text element (accepts Markdown), escaping control characters. Note that the client chose to allow Markdown `*_` style elements which is not ideal, as it gives some control over the presentation of data inside the Snap context to the calling dApp.

Changeset: fort-major/msq@ `2704c68`

### 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 potentially unintended side-effects.

For instance, the text() component can render Markdown or allow for control character injections. Specifically this poses a concern because users trust information displayed by the Snap.

In the code snippet provided below, please note that the variable `body` is provided by the dApp. It may contain Markdown renderable strings or Control Characters that can disrupt the context of the user-displayed message. It appears that only protected methods (admin origin) are affected by this, which is reflected in the severity rating of this finding.

### Examples

- `protected_handleAddAssetAccount` - decodes `ZICRC1AddAssetAccountRequest` from `CBOR` with `body` potentially containing markdown or control chars.

**packages/snap/src/protocols/icrc1.ts:L90-L107**

```
export async function protected_handleAddAssetAccount(bodyCBOR: string): Promise<string | null> {
  const body = zodParse(ZICRC1AddAssetAccountRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();

  const agreed = await snap.request({
    method: "snap_dialog",
    params: {
      type: "confirmation",
      content: panel([
        heading(`🔒 Confirm New ${body.symbol} Account 🔒`),
        text(`Are you sure you want to create a new **${body.name}** (**${body.symbol}**) token account?`),
        text(`This will allow you to send and receive **${body.symbol}** tokens.`),
        divider(),
        text("**Confirm?** 🚀"),
      ]),
    },
  });
```

- `protected_handleAddAsset` - `body`

### packages/snap/src/protocols/icrc1.ts:L59-L78

```
export async function protected_handleAddAsset(bodyCBOR: string): Promise<IAssetDataExternal[] | null> {
  const body = zodParse(ZICRC1AddAssetRequest, fromCBOR(bodyCBOR));
  const manager = await StateManager.make();

  const assetNames = body.assets.filter((it) => it.name && it.symbol).map((it) => `${it.name} (${it.symbol})`);

  if (assetNames.length > 0) {
    const agreed = await snap.request({
      method: "snap_dialog",
      params: {
        type: "confirmation",
        content: panel([
          heading(`🔒 Confirm New Assets 🔒`),
          text(`Are you sure you want to add the following tokens to your managed assets list?`),
          ...assetNames.map((it) => text(` - **${it}**`)),
          divider(),
          text("**Confirm?** 🚀"),
        ]),
      },
    });
```

- `protected_handleShowICRC1TransferConfirm` - `body`

### packages/snap/src/protocols/icrc1.ts:L26-L57

```
export async function protected_handleShowICRC1TransferConfirm(bodyCBOR: string): Promise<boolean> {
  const body = zodParse(ZShowICRC1TransferConfirmRequest, fromCBOR(bodyCBOR));

  const agreed = await snap.request({
    method: "snap_dialog",
    params: {
      type: "confirmation",
      content: panel([
        heading(`💳 Confirm ${body.ticker} Transfer 💳`),
        text("**Protocol:**"),
        text("ICRC-1"),
        text("**Initiator:**"),
        text(`🌐 ${originToHostname(body.requestOrigin)}`),
        text("**From:**"),
        text(body.from),
        text("**To principal ID:**"),
        text(body.to.owner),
        text("**To subaccount ID:**"),
        text(body.to.subaccount !== undefined ? bytesToHex(body.to.subaccount) : "Default subaccount ID"),
        text("**Total amount:**"),
        heading(`${body.totalAmountStr} ${body.ticker}`),
        divider(),
        heading("🚨 BE CAREFUL! 🚨"),
        text("This action is irreversible. You won't be able to recover your funds!"),
        divider(),
        text("**Confirm?** 🚀"),
      ]),
    },
  });

  return Boolean(agreed);
}
```

## Recommendation

Validate inputs. Encode data in a safe way to be displayed to the user (markdown, control chars). Show the original data provided within a pre-text or code block (copyable). Consider setting `markdown: false` for text ui components that do not render text.

### ../packages/snaps-sdk/src/ui/components/text.ts:L34-L53

```ts
/**
 * Create a {@link Text} node.
 *
 * @param args - The node arguments. This can be either a string
 * and a boolean, or an object with a `value` property
 * and an optional `markdown` property.
 * @param args.value - The text content of the node.
 * @param args.markdown - An optional flag to enable or disable markdown. This
 * is enabled by default.
 * @returns The text node as object.
 * @example
 * const node = text({ value: 'Hello, world!' });
 * const node = text('Hello, world!');
 * const node = text({ value: 'Hello, world!', markdown: false });
 * const node = text('Hello, world!', false);
 */
export const text = createBuilder(NodeType.Text, TextStruct, [
  'value',
  'markdown',
]);
```

```ts
const node = text({ value: 'Hello, world!', markdown: false });
```

## 4.10 Shared/hexToBytes - Incorrect Hex String Handling Minor ✓ Fixed

**Resolution**

Addressed this with the following changeset, enforcing the input string to be of hex-chars with a correct length: fort-major/msq@ `2704c68`

### Description

The function `hexToBytes` aims to convert a hex string into a `Uint8Array`. It splits the string into parts of two characters each and then tries to parse each part into an integer.

However, the function fails to validate that the provided `hexString` is of valid hex characters only. This may lead to the function interpreting non-hex characters incorrectly as zero bytes while it should throw an exception/report an error condition instead.

In the case where `hexString` contains non-hex characters, the `parseInt` will return `NaN` which in turn gets mapped to `[00, ..]` elements in the resulting `Uint8Array`.

### Examples

**packages/shared/src/encoding.ts:L42-L58**

```ts
/**
 * ## Decodes {@link Uint8Array} from hex-string
 *
 * @see {@link bytesToHex}
 *
 * @param hexString
 * @returns
 */
export const hexToBytes = (hexString: string): Uint8Array => {
  const matches = hexString.match(/.{1,2}/g);

  if (matches == null) {
    throw new Error("Invalid hexstring");
  }

  return Uint8Array.from(matches.map((byte) => parseInt(byte, 16)));
};
```

### Recommendation

Before passing the hex string into parseInt function, add validation checks to verify if the given string is a valid hex string of correct length.

```ts
export const hexToBytes = (hexString: string): Uint8Array => {
    if (!/^([0-9A-Fa-f]{2})+$/.test(hexString)) {
        throw new Error("Invalid hexstring");
    }

    const matches = hexString.match(/.{1,2}/g);
    return Uint8Array.from(matches.map((byte) => {
        const parsed = parseInt(byte, 16);
        if (isNaN(parsed)) {
            throw new Error('Invalid byte found')
        }
        return parsed;
    }));
};
```

## 4.11 Unused Imports ✓ Fixed

**Resolution**

Addressed with the following changeset, removing the unused imports in `encodings.ts`: fort-major/msq@`26a0eec` and fort-major/msq@`4d6b006` addressing the remaining unused imports.

### Description

While reviewing the codebase, we identified multiple instances of unused imports across the project's files. The presence of these unused imports may affect its maintainability and readability.

### Examples

(This list is not exhaustive)

- `IStatisticsData`, `ZStatisticsData`

**packages/snap/src/protocols/statistics.ts:L1-L8**

```
import {
  IStatisticsData,
  type IStatistics,
  ZStatisticsData,
  zodParse,
  fromCBOR,
  ZStatisticsIncrementRequest,
} from "@fort-major/msq-shared";
```

- `jsSHA`, `Crc32`, duplicate import `Principal`

**packages/shared/src/encoding.ts:L1-L5**

```
import { Principal } from "@dfinity/principal";
import { Encoder } from "cbor-x";
import { Crc32 } from "@aws-crypto/crc32";
import jsSHA from "jssha";
```

- `IMetaMaskEthereumProvider`

**packages/client/src/client.ts:L1**

```
import { IGetSnapsResponse, IMetaMaskEthereumProvider, ISnapRequest } from "./types";
```

### Recommendation

Remove unused imports. Implementing a linter in the development workflow can help automate the detection and removal of such imports, ensuring a cleaner codebase and promoting best coding practices.

# Appendix 1 - Files in Scope

The client provided the following files in scope:

- `/packages/shared/`
  - `src/**/*.ts`
  - `package.json`
- `/packages/snap/`
  - `src/**/*.ts`
  - `snap.manifest.json`
  - `snap.config.js`
  - `package.json`
- `/packages/client/`
  - `src/**/*.ts`
  - `inline-env-vars.js`
  - `package.json`
- `/apps/site/`
  - `src/frontend/**/*` (except for styles - `**/*/style.ts`)
  - `index.html`
  - `/public/.ic-assets.json5`
  - `vite.config.ts`
  - `package.json`
- `/apps/demo` (for reference)

This audit covered the following files:

| File | SHA-1 hash |
| --- | --- |
| apps/site/index.html | `8b9be4102fb72bcc21f6252ee56a3d445e59811b` |
| apps/site/src/frontend/backend.ts | `b6c3868837c9b932f2810f72d0b7ca4ae2822447` |
| apps/site/src/frontend/components/account-card/index.tsx | `d76ba120b044009f3749dd1b7b815a72acb1fb3c` |
| apps/site/src/frontend/components/account-card/style.ts | `34891584734c65489487b1d62376bdaeefa1a27e` |

| File | SHA-1 hash |
| --- | --- |
| apps/site/src/frontend/components/add-account-btn/index.tsx | `53969e696804c8271c90b2f444721d1f7918851f` |
| apps/site/src/frontend/components/add-new-mask-btn/index.tsx | `211a5667f300528653c984c7d79ef1e9926d9783` |
| apps/site/src/frontend/components/add-new-mask-btn/style.ts | `3a8faa6c2e6a07fff67fe32c005acd24d42b4dff` |
| apps/site/src/frontend/components/boop-avatar/index.tsx | `62a56160629e2c2aa7d7d945ddc476d70e0b1790` |
| apps/site/src/frontend/components/boop-avatar/style.tsx | `547a3fdadcf8356dc3180c9d6dd04e70c26f0d00` |
| apps/site/src/frontend/components/cabinet-nav/index.tsx | `b5625ec63c61897b1125617d40d749f874fc9484` |
| apps/site/src/frontend/components/cabinet-nav/styles.ts | `58ad8db0349d9b7b9b3325d4cce23ca44bf13050` |
| apps/site/src/frontend/components/contact-us-btn/index.tsx | `2a5211ddbb12b6019e2e0472be1f486d2259292b` |
| apps/site/src/frontend/components/contact-us-btn/style.ts | `0809d1e4f4bb7c2904260c3c6cf9682eaaaaaf55` |
| apps/site/src/frontend/components/divider/style.ts | `b1ad2985dd999f7b32780d3dd5288f04dc0a9c34` |
| apps/site/src/frontend/components/error-spoiler/index.tsx | `44c3c558e0a144ebf51d250ba2f2b509ac9927e4` |
| apps/site/src/frontend/components/header/index.tsx | `37baea6762fb461ea3b3455415004d2411aa8e4a` |
| apps/site/src/frontend/components/loader/index.tsx | `0e794fe08608ad68504a906e682bedbe4cb48997` |
| apps/site/src/frontend/components/login-option/index.tsx | `7799a6ecd24a297c6acdc5523d21888f3fd76fd2` |
| apps/site/src/frontend/components/login-option/style.ts | `2e4a2f036dc1eb7779433e09f55938274e0e9b6b` |
| apps/site/src/frontend/components/modal/index.tsx | `118b1b63dbec528b0c488cd743ce3194d1c01e04` |
| apps/site/src/frontend/components/notification-bar/index.tsx | `83596e6211d998962724ad418c8d9bd46c6c2c1c` |
| apps/site/src/frontend/components/notification-bar/style.ts | `9619aab409f7b213d3e1a64811cc57195f4a5bb5` |
| apps/site/src/frontend/components/spoiler/index.tsx | `49475e0eb501aa07063948a67b92a8ce5ccaa19c` |
| apps/site/src/frontend/components/spoiler/style.ts | `1d395dd18623b612d14c61f964259a1bc6da0a59` |
| apps/site/src/frontend/components/toggle/index.tsx | `e6da652eed80769e68ba839908380d1286e8bdc3` |
| apps/site/src/frontend/components/txn-history-entry/index.tsx | `9245ad3ce511150df1a24696ea8dd8c9b441028f` |
| apps/site/src/frontend/components/txn-history-entry/style.ts | `ce100793f18d8f0ec29a99cd0d4cc46506c033ad` |
| apps/site/src/frontend/components/txn-history-modal/index.tsx | `7ca507a764d089a1fefbcad46502c6020b3d437e` |
| apps/site/src/frontend/components/txn-history-modal/style.ts | `549efe7b3b3fff6a6f1858154a0913c2a8524c02` |
| apps/site/src/frontend/index.tsx | `0505e0951580a96d5a894f9147a3e05c4ff13e1f` |
| apps/site/src/frontend/pages/cabinet/my-assets/index.tsx | `8122f35b14fd58829640e3e7b02b0d39206865da` |
| apps/site/src/frontend/pages/cabinet/my-assets/receive/index.tsx | `ba9f7e4f7253e8e36f8f472007930189078906d8` |
| apps/site/src/frontend/pages/cabinet/my-assets/send/index.tsx | `4dabbd793f4da8792097490ef44e0fa48f7f61bf` |
| apps/site/src/frontend/pages/cabinet/my-assets/send/style.ts | `2e8460791d23cc676d2be2cb6f434068a8ed93ce` |
| apps/site/src/frontend/pages/cabinet/my-assets/style.ts | `909c9819fdcf11be9edb8e7d2bba53ff9175d028` |
| apps/site/src/frontend/pages/cabinet/my-assets/txn-history/index.tsx | `1f2bda3b8d409a151c79e645e1b9e41c140237ce` |
| apps/site/src/frontend/pages/cabinet/my-assets/txn-history/style.ts | `3008b0ff3ccb0136b1cba69982c155948525f138` |
| apps/site/src/frontend/pages/cabinet/my-links/index.tsx | `3b974a91838b2067ce5a2b90ebe0804f6912399f` |
| apps/site/src/frontend/pages/cabinet/my-links/style.ts | `fdff9b76861611ef084bb224546e6da94f633a1a` |
| apps/site/src/frontend/pages/cabinet/my-masks/index.tsx | `a41ef2620a14d4fe0041d0daf9cf10fba962773c` |
| apps/site/src/frontend/pages/cabinet/my-masks/style.ts | `c2b5aa37fd471b99ea30c4920599af0a08f04625` |
| apps/site/src/frontend/pages/cabinet/my-sessions/index.tsx | `c3ad1ce3baa36c86e53b7014e81729f846f2f511` |
| apps/site/src/frontend/pages/cabinet/my-sessions/style.ts | `6e0ba119360b2c1b1a70ab95cee4275078b7dae1` |
| apps/site/src/frontend/pages/error/index.tsx | `b5f2d341d72e4294a1039894b3fba1b5e47a29cb` |
| apps/site/src/frontend/pages/icrc35/index.ts | `1133575ea2099500bbf17c681dc9c74e2d73a593` |
| apps/site/src/frontend/pages/index/index.tsx | `5d9b95ad7ae49d0e5629844b45ef2968f0aa24b5` |
| apps/site/src/frontend/pages/integration/login/index.tsx | `4845703fd6310342084c05700a2f1ffcda1b059a` |
| apps/site/src/frontend/pages/integration/login/style.ts | `04690601978e209a4041b9f71f68cb2a3aafbefb` |
| apps/site/src/frontend/pages/integration/payment/checkout/index.tsx | `f84672b5f4f5c6250a1541b2cefb14907bf49974` |
| apps/site/src/frontend/pages/integration/payment/checkout/style.ts | `5d4d5c6c90ef3f1c3fe5f9dd2cc696c8704757e0` |
| apps/site/src/frontend/pages/integration/payment/index.tsx | `814ec2bf86443050c9b62a21e0d585b37ccbaee0` |
| apps/site/src/frontend/pages/integration/payment/style.ts | `6cb16a0b6b0c48b9cf844f4d852ae06d6c474d23` |
| apps/site/src/frontend/pages/integration/payment/url-payment.tsx | `88f8b70be2e61d9b9bf7e74e231b0e6c52e9e1da` |
| apps/site/src/frontend/pages/statistics/index.tsx | `eb7055e4939ef18dfa1fc69e2d2a0e22c2c9bc21` |
| apps/site/src/frontend/pages/statistics/style.ts | `d49633a063ad52d9bc8598d046310ea1ce7012c3` |
| apps/site/src/frontend/pages/txn/fail.tsx | `babdb5d001cd5b92f49ee1a42a70f3d2fd2887e3` |
| apps/site/src/frontend/pages/txn/style.ts | `7cde98ae9d39606e202cd14c233b20f4ca2191a5` |

| File | SHA-1 hash |
| --- | --- |
| apps/site/src/frontend/pages/txn/success.tsx | `bdb735ead3ec5640e9f7ba0c81de9ffbe748bddb` |
| apps/site/src/frontend/store/assets.tsx | `e7159e8232cb3084c7b1925e545a26db527135e7` |
| apps/site/src/frontend/store/global.tsx | `94385ea230675b1d14d6a61cbcb4fa0c16d3bc73` |
| apps/site/src/frontend/store/origins.tsx | `637cae9834e366cb1ef96d8c2a3e0144faad8167` |
| apps/site/src/frontend/ui-kit/button.tsx | `fe2e6a56c82934acb9b5638fd3a522091f3772b6` |
| apps/site/src/frontend/ui-kit/icon.tsx | `94a68331c079dbd2a0c135873cb33896ac3e78e2` |
| apps/site/src/frontend/ui-kit/index.ts | `d2f36288203bf07228062926796f54746c6676e3` |
| apps/site/src/frontend/ui-kit/input.tsx | `c68ca6c2e3e964f8c9ed0cd861427481190c64f4` |
| apps/site/src/frontend/ui-kit/typography.ts | `72dada658b8371134c88186c82bb7e2f822d2394` |
| apps/site/src/frontend/utils/index.ts | `d7ee8b71495ed76eb00ab041e4fe5aa8fafc97fe` |
| apps/site/vite.config.ts | `14ae454602a17b4c3c9b8b8fd11d87146fe80017` |
| example.env | `f7a317343cee45951eb900898b36ee068ffde49a` |
| packages/client/inline-env-vars.js | `bcf60bfbb7f4bf40b154aecd23a3141dc12d39b9` |
| packages/client/src/client.ts | `0f9839c32b56081a2e1ce4c12d6dbf41f41e2a4f` |
| packages/client/src/der.ts | `349096fd2124b147520754f8e5a6cff7d1e7facc` |
| packages/client/src/icrc35-client.ts | `9157436a6b8c008070a1d1d1686d59c4c2c78e55` |
| packages/client/src/identity.ts | `2ebda390d8af85f7c4f9be9854cda9d3e7791919` |
| packages/client/src/index.ts | `947732909d8240cb5bf8ea3ada377bd6f8950408` |
| packages/client/src/internal.ts | `a613131644cde88f9b97f518e42552f60e13babe` |
| packages/client/src/types.ts | `babb8553a72a5af627092866aad9ff30ac653554` |
| packages/shared/src/avatar.ts | `8eae636bc7362422bd833400b2fc93bf5949278c` |
| packages/shared/src/encoding.ts | `d3e426f0c7710f1b2a9019bc6eb9a917e3ad1e62` |
| packages/shared/src/index.ts | `ab14eed2b74c85746524226c5c6aa082b7640478` |
| packages/shared/src/types.ts | `74e5227fc1729e977a2e38e133a3e451d7bf0722` |
| packages/snap/jest.config.js | `d1c409efee5958ce5346655d3912b644894e098d` |
| packages/snap/snap.config.js | `f9325fcd7f0fa159af4d56edf5015116ce834238` |
| packages/snap/src/index.ts | `190e84d22c2e6daec6f62df904c37a5780e9be48` |
| packages/snap/src/protocols/icrc1.ts | `210daa9f4ecd7ede579f7c338a81bd1b822daefa` |
| packages/snap/src/protocols/identity.ts | `89879b78884e70b81714da1356b5abb46dcc8c9d` |
| packages/snap/src/protocols/state.ts | `02726e414dd7bc80ec321d0c6197c18cf8735a91` |
| packages/snap/src/protocols/statistics.ts | `d930d52b7376258bb0460666b10c7871ec3ecc51` |
| packages/snap/src/state.ts | `36c0c8980d43091bec59a7497e29587e5f019f0c` |
| packages/snap/src/utils.ts | `6520426ebaff305a51bf7f758667917e1b4c00c4` |

# 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.