Metamask - EIP-7715 Permissions Snap

Date	August 2025
Auditors	Martin Ortner, Pedro Santos

1 Executive Summary

This report presents the results of our engagement with **MetaMask** to review the **Gator 7715 Permissions & Kernel Snap**.

The review was conducted over 3 calendar weeks, from **Aug 11, 2025** to **Aug 29, 2025**, with a total effort of **4.5 person-weeks**, with one reviewer for 3 person-weeks and another with 1.5 person-weeks.

The system enables users to grant time-bound, caveat-restricted permissions for various blockchain operations including ERC-20 token transfers, native token operations, and streaming payments. Through its integration with DeleGator smart accounts and the delegation toolkit, it provides a foundation for automated, user-controlled blockchain interactions while maintaining security through comprehensive validation and user consent mechanisms.

The MetaMask **Gator Permissions Snap** is an EIP-7715 permissions management system that enables granular control over blockchain permissions through MetaMask's Snap framework. This dual-snap architecture consists of the Gator Permissions Snap and the Permissions Kernel Snap, working together to implement EIP-7715 for delegated permissions and smart account interactions.

The **Permission Kernel Snap** implements a registry intended to support multiple 7715-compliant adapters in the future. At present, however, it is hardcoded to the **Gator Permission Snap Adapter**. While this reduces the kernel's direct attack surface, it simultaneously increases the system's overall exposure, since snap-to-snap communication introduces additional complexity and requires safeguards such as whitelisting, a broker model, and forwarding of the original origin.

The **Gator Permission Snap** is configured to allow communication from the Kernel Snap (the "router") and MetaMask internal wallet operations only. The **Kernel** as the request broker can ask for all offered permissions and request specific permissions requested by a DApp. The adapter also offers a getter to return all issued permissions that can only be accessed by MetaMask.

Update: October 28, 2025 - Mitigations Review

Between **October 22, 2025** and **October 28, 2025**, we conducted a review of the client's remediation measures implemented in commit 476e653 of the MetaMask/snap-7715-permissions repository.

Update: October 30, 2025 - Additional Mitigations Review

Removed `viem` dependency with 6da514a.

---

2 Scope

This review focused on the following repository and code revision:

- MetaMask/snap-7715-permissions (`2917e66247c44bafefc51514eb2eee5339b34455`)

Third party dependencies packages are out of scope for this review.

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

2.1 Objectives

Together with client, we identified the following priorities for this 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 Security Guidelines and Threat Model describing a Snap as an extension of the MetaMask Wallet Trust Module.

3 Snap Configuration Overview

As part of the assessment, the following key aspects of the Snap's configuration were identified:

3.1 EIP-7715 Permissions Kernel Snap

- The Snap serves as a permissions kernel implementing EIP-7715 for managing execution permissions in the MetaMask ecosystem.
- It acts as an intermediary that receives permission requests from DApps and routes them to appropriate permission providers (currently the Gator Permissions Provider Snap).
- The Snap is designed to support multiple permission providers in the future but currently integrates specifically with the `GATOR_PERMISSIONS_PROVIDER_SNAP_ID`.
- The Snap is state-less and only serves hard-coded adapters from the registry.

A.2.3 Timeliness of Content

Internal Architecture:

- **Registry Manager**: Manages the permission offer registry and handles finding relevant permissions to grant from available offers.
- **RPC Handler**: Processes incoming RPC requests and orchestrates the permission granting workflow.

RPC Methods:

- `wallet_requestExecutionPermissions` : The primary method exposed to DApps for requesting execution permissions. This method:
  - Parses and validates the permission request
  - Builds a permission offers registry from the Gator permissions provider
  - Finds relevant permissions to grant by filtering against registered offers
  - Invokes the permission provider snap to grant attenuated permissions
  - Returns the granted permissions to the requesting DApp

Snap Permissions:

- `snap_dialog` : Displays dialogs in the MetaMask UI ⚠️
- `endowment:rpc` :
  - `dapps: true` - Allows communication with websites/DApps
  - `snaps: true` - Enables communication with other snaps ⚠️

DApp Connectivity and Permissions:

- Connected DApp communicate with the Snap via RPC calls. Any connected DApp can communicate with the Snap. No restrictions or programmatic whitelists.
- Development origins are **not** explicitly excluded from **initial-connections** for production builds ⚠️
- The snap allows DApp-to-Snap and Snap-to-Snap. It is unclear if the latter is actually needed ⚠️

State Management:

- No state information is stored or accessed.

Dependencies:

- `@metamask/snaps-sdk:8.1.0` (marked as potential dev dependency)
- `viem:2.31.7` ⚠️

Origin Permissions

```
"initialConnections": {
  "local:http://localhost:8082": {},
  "npm:@metamask/gator-permissions-snap": {}
},
"initialPermissions": {
  "snap_dialog": {},
  "endowment:rpc": {
    "dapps": true,
    "snaps": true
  }
```

- Development origin whitelisted at **initialConnections** ⚠️
- Trusting `gator-permissions-snap` via `initialConnections` might not be needed ⚠️

Details

```
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
snap_dialog {}
endowment:rpc { dapps: true, snaps: true }
---->%---- 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/index.ts
🚜 [endowment:rpc]
    ❗ - endowment:rpc.dapps - snap can communicate with websites/dapps; check origin for internal api calls!
    ⚠️ - endowment:rpc.snaps - snap can communicate with other snaps; check origin for internal api calls!
       🔶 src/index.ts
🎄 - Package Dependencies:
      - @metamask/snaps-sdk:8.1.0              (🔺 looks like devDependency 👀)
      - viem:2.31.7
[🚀 == Metrics == ]
👉 316 (lines of code)
```

3.2 Gator Permissions Snap

- The Snap serves as a permissions provider implementing EIP-7715 for managing token spending permissions in the MetaMask ecosystem.
- It acts as the core permissions provider that receives permission requests from the EIP-7715 Permissions Kernel Snap and manages the full lifecycle of permission granting, validation, and storage.
- The Snap implements four types of token permissions: ERC-20 Token Stream, ERC-20 Token Periodic, Native Token Stream, and Native Token Periodic.
- The Snap maintains persistent state of granted permissions and provides a management interface for users to view their active permissions.

This is currently the only adapter that Kernel interfaces with.

Internal Architecture:

- **Permission Handler Factory**: Creates specialized handlers for different permission types with validation rules and UI content generation.
- **Permission Request Lifecycle Orchestrator**: Manages the complete permission request workflow from validation through user confirmation to delegation creation.
- **Confirmation Dialog System**: Handles user consent dialogs with dynamic content based on permission type and context.
- **Account Controller**: Manages EOA account operations and EIP-712 delegation signing for permission enforcement.
- **Profile Sync Manager**: Persists granted permissions across browser sessions using MetaMask's profile sync functionality.
- **Token Metadata & Pricing Services**: Fetches external token information and real-time pricing data for UI display.

RPC Methods:

- `grantPermission`: The primary method for processing permission requests. This method:
  - Validates incoming permission requests against defined schemas
  - Creates appropriate permission handlers based on request type
  - Orchestrates user confirmation dialogs showing origin, token details, and permission scope
  - Creates EIP-712 delegations for approved permissions
  - Stores granted permissions for future reference
- `getPermissionOffers`: Returns available permission types that can be granted
- `getGrantedPermissions`: Retrieves all previously granted permissions for management purposes

Snap Permissions:

- `endowment:rpc`: Enables communication with other snaps (specifically the EIP-7715 Permissions Kernel) ⚠️
- `snap_manageState`: Stores granted permissions and user preferences (up to 100MB isolated storage)
- `endowment:ethereum-provider`: Accesses Ethereum API for account operations and delegation signing ⚠️
- `endowment:network-access`: Makes HTTP requests to external APIs for token metadata and pricing ⚠️
- `snap_dialog`: Displays permission confirmation and management dialogs ⚠️
- `endowment:lifecycle-hooks`: Handles installation and homepage events
- `snap_getPreferences`: Accesses user preferences for currency and locale settings
- `endowment:page-home`: Provides homepage interface for permission management

DApp Connectivity and Permissions:

- The Snap does **not** directly communicate with DApps (`dapps: false` in RPC endowment).
- All DApp interactions are mediated through the EIP-7715 Permissions Kernel Snap for security isolation.
- Development origins are **not** explicitly restricted in production builds (manifest), however, they are excluded from the programmatic permission check ⚠️
- The snap allows Snap-to-Snap communication. `npm:@metamask/permissions-kernel-snap` is trusted by default.

State Management:

- Persistent storage of granted permissions with origin tracking and expiration management.
- Token metadata caching for improved performance and reduced external API calls.
- User preference synchronization across MetaMask installations via profile sync.

Dependencies:

- `@metamask/snaps-sdk:8.1.0` (Core Snaps functionality)
- `@metamask/delegation-core:0.2.0-rc.1` (EIP-712 delegation creation)
- `@metamask/utils:11.4.2` (Utility functions for CAIP parsing and validation)
- `viem:2.31.7` (Ethereum interactions) ⚠️ unused
- `zod:3.25.76` (Runtime validation and type safety)

Origin Permissions

```
"initialConnections": {
  "local:http://localhost:8081": {},
  "npm:@metamask/permissions-kernel-snap": {}
},
"initialPermissions": {
  "endowment:rpc": {
    "dapps": false,
    "snaps": true
  },
  "snap_manageState": {},
  "endowment:ethereum-provider": {},
  "endowment:network-access": {},
  "snap_getEntropy": {},
  "snap_dialog": {},
  "endowment:lifecycle-hooks": {},
  "snap_getPreferences": {},
  "endowment:page-home": {}
},
```

- Development origin whitelisted at **initialConnections** ⚠️
- Network access enabled for external API calls to token metadata and pricing services ⚠️
- Ethereum provider access for delegation signing and account operations ⚠️

Details

```
----%<---- raw permissions
🌐 : https://docs.metamask.io/snaps/reference/rpc-api/#wallet_requestsnaps
endowment:rpc { dapps: false, snaps: true }
snap_manageState {}
endowment:ethereum-provider {}
endowment:network-access {}
snap_getEntropy {}
snap_dialog {}
endowment:lifecycle-hooks {}
snap_getPreferences {}
endowment:page-home {}
---->%---- raw permissions
🚜 [endowment:rpc]
    ⚠️ - endowment:rpc.snaps - snap can communicate with other snaps; check origin for internal api calls!
       🔶 src/index.ts
🚜 [snap_manageState]
    🪶 - snap_manageState - snap can store up to 100mb (isolated)
       🔶 src/stateManagement.ts
🚜 [endowment:ethereum-provider]
    🔺 - endowment:ethereum-provider - snap can access ethereum API
    ⚠️ - check if the **snap code** (not site) actually accesses the global 'ethereum' object
see https://docs.metamask.io/snaps/learn/about-snaps/apis/#snap-requests
       🔶 src/clients/priceApiClient.ts
🚜 [endowment:network-access]
    🌐 - endowment:network-access - snap can access internet
     ⚠️ - this method may leak information to external api
🚜 [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
       🔺 - superfluous permission: no reference to 'snap_getEntropy'!
🚜 [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/homepage/index.tsx
       🔶 src/core/confirmation.tsx
🚜 [endowment:lifecycle-hooks]
    🔺 - endowment:lifecycle-hooks - unknown permission requested
🚜 [snap_getPreferences]
    🔺 - snap_getPreferences - unknown permission requested
🚜 [endowment:page-home]
    🔺 - endowment:page-home - unknown permission requested
🎄 - Package Dependencies:
      - @metamask/abi-utils:3.0.0               (🔺 looks like devDependency 👀)
      - @metamask/delegation-core:0.2.0-rc.1            (🔺 looks like devDependency 👀)
      - @metamask/delegation-toolkit:0.10.2             (🔺 looks like devDependency 👀)
      - @metamask/profile-sync-controller:21.0.0                (🔺 looks like devDependency 👀)
      - @metamask/snaps-sdk:8.1.0               (🔺 looks like devDependency 👀)
      - @metamask/utils:11.4.2          (🔺 looks like devDependency 👀)
      - viem:2.31.7
      - zod:3.25.76
[🚀 == Metrics == ]
...
👉 6926 (lines of code)
```

4 General Findings

Each issue has an assigned severity:

- Critical issues are directly exploitable security vulnerabilities that need to be fixed.
- 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.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- 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.
- Issues without a severity are general recommendations or optional improvements. They are not related to security or correctness and may be addressed at the discretion of the maintainer.

4.1 Remove Debug/Development Origins for Productions Builds Medium ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by programmatically excluding development hosts from production builds. `endowment:lifecycle-hooks` is now only available in `local` dev builds.

Description

It is important to verify that debug/development artefacts and settings are not active in production builds. The MetaMask KeyRing API Security Documentation emphasises the importance of removing all debug code from production snaps.

Examples

- development artefacts (localhost) in manifest

packages/gator-permissions-snap/snap.manifest.json:L20-L23

```
"initialConnections": {
  "local:http://localhost:8081": {},
  "npm:@metamask/permissions-kernel-snap": {}
},
```

packages/permissions-kernel-snap/snap.manifest.json:L20-L22

```
"initialConnections": {
  "local:http://localhost:8082": {},
  "npm:@metamask/gator-permissions-snap": {}
```

Recommendation

Ensure that debug/development configuration and artefacts are not included in production builds. This includes adding dev/debug endpoint permissions only if the environment suggests it's a debug build. The build scripts should check for the corresponding debug parameterization. This will prevent inadvertent inclusion of debug code in production builds.

4.2 Potential Sensitive Information Disclosure Through Error Logging Minor ✓ Addressed

Resolution

Most instances addressed in MetaMask/snap-7715-permissions@ `476e653` by not emitting the `error` object with the logger. The client provided additional details:

> **Summary of Finding:** Detailed logging of errors and requests was observed, posing a risk of sensitive data exposure.
> **Resolution:** Raw logging was replaced with sanitized logging, and all logs were disabled in production builds.
> **Implementation:** PR: MetaMask/snap-7715-permissions#161.
> **Verification:** Production logs were verified to exclude sensitive data (e.g., stack traces, full payloads).

Some instances are left with the logger emitting the `error` object from a tryCatch statement (mostly `logger.error` or `logger.warn` invocations). For example in `userEventDispatcher`, `AccountApiClient`, ... However, by default, the loglevel is set to disabled.

Description

Both the kernel and gator snaps log complete error objects and unvalidated request data. This might expose internal and sensitive information like stack traces, API endpoints, and user input to browser logs.

While local attack vectors may not be included in MetaMask's threat model, it is considered a good practice to avoid exposure of potential sensitive information.

Example

packages/permissions-kernel-snap/src/index.ts:L39-L48

```
export const onRpcRequest: OnRpcRequestHandler = async ({
  origin,
  request,
}) => {
  logger.info(
    `Custom request (origin="${origin}"):`,
    JSON.stringify(request, undefined, 2),
  );

  const handler = boundRpcHandlers[request.method];
```

packages/permissions-kernel-snap/src/registryManager.ts:L70-L122

```
async function buildPermissionOffersRegistry(
  snapId: string,
): Promise<PermissionOfferRegistry> {
  let ephemeralPermissionOfferRegistry: PermissionOfferRegistry = {};
  try {
    try {
      logger.debug(`Querying snap ${snapId} for permission offers...`);

      const response = await snapsProvider.request({
        method: 'wallet_invokeSnap',
        params: {
          snapId,
          request: {
            method: ExternalMethod.PermissionProviderGetPermissionOffers,
          },
        },
      });

      if (response) {
        const parsedOffers = parsePermissionOffersParam(response);
        const uniqueOffersToStore: RegisteredPermissionOffers =
          parsedOffers.map((offer) => {
            return {
              hostId: snapId,
              type: offer.type,
              proposedName: offer.proposedName,
            };
          });

        ephemeralPermissionOfferRegistry = {
          ...ephemeralPermissionOfferRegistry,
          [snapId]: uniqueOffersToStore,
        };

        logger.debug(
          `Snap ${snapId} supports ${ExternalMethod.PermissionProviderGetPermissionOffers}, adding to registry...`,
        );
      }
    } catch (error) {
      logger.error(
        {
          snapId,
          error,
        },
        `Snap ${snapId} does not support ${ExternalMethod.PermissionProviderGetPermissionOffers}, or returned an invalid respo
      );
    }

    return ephemeralPermissionOfferRegistry;
  } catch (error) {
    logger.error('Error building permission offer registry:', error);
    return ephemeralPermissionOfferRegistry;
  }
```

packages/gator-permissions-snap/src/index.ts:L189-L196

```
export const onRpcRequest: OnRpcRequestHandler = async ({
  origin,
  request,
}) => {
  logger.debug(
    `RPC request (origin="${origin}"):`,
    JSON.stringify(request, undefined, 2),
  );
```

packages/gator-permissions-snap/src/services/tokenPricesService.ts:L72-L107

```
async getCryptoToFiatConversion(
  tokenCaip19Type: CaipAssetType,
  balance: Hex,
  decimals: number,
): Promise<string> {
  try {
    logger.debug('TokenPricesService:getCryptoToFiatConversion()');
    const preferences = await this.#getPreferences();

    // Value in fiat=(Amount in crypto)×(Spot price)
    const tokenSpotPrice = await this.#priceApiClient.getSpotPrice(
      tokenCaip19Type,
      this.#safeParsePreferences(preferences),
    );
    const formattedBalance = Number(
      formatUnits({ value: BigInt(balance), decimals }),
    );
    const valueInFiat = formattedBalance * tokenSpotPrice;

    const humanReadableValue = formatAsCurrency(preferences, valueInFiat);
    logger.debug(
      'TokenPricesService:formatAsCurrency() - formatted balance to currency',
      humanReadableValue,
    );

    return humanReadableValue;
  } catch (error) {
    logger.error(
      'TokenPricesService:getCryptoToFiatConversion() - failed to fetch token spot price',
      error,
    );

    // TODO: Return a more user-friendly error message so failed calls to the Price API are handled gracefully
    // and do not make the entire permission request fail. We can use cached prices as a fallback to prevent this issue message i
    return '$<-->';
  }
```

Please notice similar patterns found in more locations across codebase like: `accountApiClient.ts`, `priceApiClient.ts`, `tokenMetadataService.ts`, `blockchainMetadataClient.ts`, `accountController.ts`, ...

Recommendation

Given users will likely not act or even notice console logs, consider disabling logging for production builds. Additionally, and in case you don't want or can't disable logs for prod, implement an error sanitization utility to remove any sensitive information from logs and use it to replace the raw ones.

5 Findings: Kernel

5.1 Kernel - Prototype Pollution via Method Handler Lookup Major ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by checking `Object.prototype.hasOwnProperty` on the request struct before accessing it. In `Gator` this is safeguarded by a check for `isMethodAllowedForOrigin`. The client provided additional details:

> **Summary of Finding:** request.method lacked validation, enabling prototype pollution.
> **Resolution:** We introduced explicit checks to ensure methods exist directly on the handler object (not on its prototype). Additionally, in gator-permission-snap, we enforce isMethodAllowedForOrigin, which restricts invocation so that only the kernel can call specific methods. This design prevents prototype pollution attacks in that context.
> **Implementation:** PR: MetaMask/snap-7715-permissions#147.
> **Verification:** Attempting to invoke prototype methods correctly fails.
> Reviewed call flow in Gator Snap to confirm isMethodAllowedForOrigin ensures untrusted origins cannot bypass validation.

Description

The RPC method handler lookup is vulnerable to prototype pollution attacks. When `request.method` contains prototype property names like `toString`, `valueOf`, `constructor`, or `hasOwnProperty`, the code accesses inherited methods from `Object.prototype` instead of returning `undefined`. This allows attackers to invoke unintended functions and potentially extract sensitive information or cause unexpected behavior.

Example

- `boundRpcHandlers` inherits functions from `Object.prototype`
- by accessing `boundRpcHandlers[request.method]` without whitelisting method names, the caller can call arbitrary inherited functions

packages/permissions-kernel-snap/src/index.ts:L29-L52

```ts
/**
 * Handle incoming JSON-RPC requests, sent through `wallet_invokeSnap`.
 *
 * @param args - The request handler args as object.
 * @param args.origin - The origin of the request, e.g., the website that
 * invoked the snap.
 * @param args.request - A validated JSON-RPC request object.
 * @returns The result of `snap_dialog`.
 * @throws If the request method is not valid for this snap.
 */
export const onRpcRequest: OnRpcRequestHandler = async ({
  origin,
  request,
}) => {
  logger.info(
    `Custom request (origin="${origin}"):`,
    JSON.stringify(request, undefined, 2),
  );

  const handler = boundRpcHandlers[request.method];

  if (!handler) {
    throw new Error(`Method ${request.method} not found.`);
  }
```

packages/permissions-kernel-snap/src/index.ts:L12-L27

```ts
// set up dependencies
const rpcHandler = createRpcHandler({
  permissionOfferRegistryManager: createPermissionOfferRegistryManager(snap),
  snapsProvider: snap,
});

// configure RPC methods bindings
const boundRpcHandlers: {
  [RpcMethod: string]: (options: {
    siteOrigin: string;
    params?: JsonRpcParams;
  }) => Promise<Json>;
} = {
  [RpcMethod.WalletRequestExecutionPermissions]:
    rpcHandler.requestExecutionPermissions.bind(rpcHandler),
};
```

Recommendation

Use `Object.prototype.hasOwnProperty.call()` or `Map` data structure instead of object property access, or implement explicit method validation using `Object.hasOwn()` to prevent prototype chain access.

5.2 Kernel - Superfluous Permissions in Manifest Medium ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by removing `initialConnections` altogether, changing `endowment:rpc.snaps = false`, and removing the unused `snap_dialog`.The client provided additional details:

> **Summary of Finding:** Unnecessary permissions (snap_dialog, extra connections) were declared.
> **Resolution:** Remove gatorSnap from the kernel snap's initialConnections, set the snap field to false for the endowment:rpc permission, and remove the snap_dialog permission from the kernel snap.
> **Implementation:** PRs:
> MetaMask/snap-7715-permissions#167
> MetaMask/snap-7715-permissions#153
> **Verification:** The manifest was reviewed to confirm adherence to the principle of least privilege.

Description

The permissions kernel snap declares multiple unnecessary permissions in its manifest that violate the principle of least privilege and unnecessarily expand the attack surface. The manifest includes three categories of superfluous permissions:

- The `initialConnections` section includes `npm:@metamask/gator-permissions-snap` despite the kernel snap never initiating communication with the gator snap. The kernel only processes responses when invoking other snaps.
- The `endowment:rpc` permission grants `"snaps": true` access, but no other snaps initiate communication with the kernel snap. The kernel only handles request-response communications in a unidirectional flow (DApp → Kernel → Gator).
- The `snap_dialog` permission is declared but never used in the codebase. All dialog operations are handled by the gator permissions snap.

Example

packages/permissions-kernel-snap/snap.manifest.json:L20-L30

```
"initialConnections": {
  "local:http://localhost:8082": {},
  "npm:@metamask/gator-permissions-snap": {}
},
"initialPermissions": {
  "snap_dialog": {},
  "endowment:rpc": {
    "dapps": true,
    "snaps": true
  }
},
```

Recommendation

Remove unused `npm:@metamask/gator-permissions-snap` from `initialConnections`, set `"snaps": false` in the RPC endowment configuration and remove `snap_dialog` to eliminate unnecessary permissions and reduce the potential attack surface.

Also see: Remove Development Artefacts issue 4.1 .

5.3 Kernel - Unused Dependency 'Viem' Minor ✓ Fixed

Resolution

Update Oct 30, 2025: The client has removed the import with MetaMask/snap-7715-permissions@ `6da514a`.

Still present in MetaMask/snap-7715-permissions@ `476e653`. `7715-permissions-shared`, which is used by both snaps, imports `viem` but seems to be unused. `viem` is also imported by `site` but `site` references it in code.

The client originally provided the following additional details:

> **Summary of Finding:** The viem dependency was found to be unused.
> **Resolution:** The unused dependency was removed.
> **Implementation:** PR: MetaMask/snap-7715-permissions#154.
> **Verification:** The dependency no longer appears in package.json, and builds successfully complete.

Description

The project declares `viem@2.31.7` as a dependency but never imports or utilizes this package anywhere in the codebase. This creates unnecessary supply chain exposure and violates security best practices for dependency management.

Example

packages/permissions-kernel-snap/package.json:L49-L52

```
"dependencies": {
  "@metamask/snaps-sdk": "8.1.0",
  "viem": "2.31.7"
},
```

No import statements found in any TypeScript files (`src/**/*.ts`) or test configurations.

Recommendation

Remove the unused `viem` dependency from package.json.

5.4 Kernel - `onRpcRequest` Misleading Function Documentation Minor ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by providing an accurate `@returns` documentation. The client provided additional details:

> **Resolution:** JSDoc was updated to accurately describe the return type.
> **Implementation:** PR: MetaMask/snap-7715-permissions#160
> **Verification:** Documentation was reviewed, revealing no discrepancies.

Description

The `onRpcRequest` function documentation contains incorrect information about the return value. The JSDoc comment states that the function "The result of `snap_dialog`" while the actual implementation returns the result of the 7715 permissions adapter response from `boundRpcHandlers.requestExecutionPermissions()`.

Example

packages/permissions-kernel-snap/src/index.ts:L29-L38

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

Recommendation

Update the JSDoc documentation to accurately reflect that the function returns the 7715 permissions adapter response, not a snap dialog result.

5.5 Kernel - Recommendation: Enforce Strict Runtime Type & Input Validation Early ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by enforcing input validation on RPC requests early. The client provided additional details:

> **Resolution:** Zod schemas were introduced for validating incoming RPC requests prior to processing.
> **Implementation:** PR: MetaMask/snap-7715-permissions#160.
> **Verification:** Malformed requests are now rejected at an early stage.

Description

The `onRpcRequest` handler does not validate incoming requests before processing them. The JSDoc mentions it expects a "validated JSON-RPC request object", but the code should verify if that's actually happening, rather than directly accessing `request.method` and `request.params` without any checks.

The missing validation could enable prototype pollution attacks (see issue 5.1) since it does not check `request.method` in any way before using it as an object key.

Please note `request.params` gets validated later, but by then it might be too late to prevent method lookup issues.

Example

packages/permissions-kernel-snap/src/index.ts:L29-L60

```
/**
 * Handle incoming JSON-RPC requests, sent through `wallet_invokeSnap`.
 *
 * @param args - The request handler args as object.
 * @param args.origin - The origin of the request, e.g., the website that
 * invoked the snap.
 * @param args.request - A validated JSON-RPC request object.
 * @returns The result of `snap_dialog`.
 * @throws If the request method is not valid for this snap.
 */
export const onRpcRequest: OnRpcRequestHandler = async ({
  origin,
  request,
}) => {
  logger.info(
    `Custom request (origin="${origin}"):`,
    JSON.stringify(request, undefined, 2),
  );

  const handler = boundRpcHandlers[request.method];

  if (!handler) {
    throw new Error(`Method ${request.method} not found.`);
  }

  const result = await handler({
    siteOrigin: origin,
    params: request.params as JsonRpcParams,
  });

  return result;
};
```

The code assumes `request.method` is a valid string and `request.params` conforms to `JsonRpcParams` without any runtime verification.

Recommendation

Implement comprehensive Zod schema validation for all incoming RPC requests before processing.

5.6 Kernel - Inefficient Array Concatenation ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by using `array.flat()`. The client provided additional details:

> **Resolution:** The reduce method was replaced with Array.prototype.flat().
> **Implementation:** PR: MetaMask/snap-7715-permissions#159.
> **Verification:** Code review confirmed array operations are now efficient and safe.

Description

The `getRegisteredPermissionOffers` function uses an inefficient O(n²) array concatenation pattern. It is using spread operators inside a reduce, which recreates the entire array on every iteration. While the current registry is hardcoded to the gator snap, and even considering numerous permissions offers, the risk of causing significant performance degradation or denial of service is reduced, however, it still should be addressed.

Example

packages/permissions-kernel-snap/src/registryManager.ts:L125-L142

```
/**
 * Flattens the permission offer registry into a single array of registered permission offers.
 *
 * @param permissionOfferRegistry - The permission offer registry to flatten.
 * @returns The registered permission offers.
 */
function getRegisteredPermissionOffers(
  permissionOfferRegistry: PermissionOfferRegistry,
): RegisteredPermissionOffers {
  if (Object.keys(permissionOfferRegistry).length === 0) {
    return [];
  }

  return Object.values(permissionOfferRegistry).reduce(
    (acc, offers) => [...acc, ...offers],
    [],
  );
}
```

Recommendation

Replace the inefficient reduce pattern with `Array.prototype.flat()` which operates in O(n) time complexity, or use `Array.prototype.concat()` for better performance and reduced memory allocation.

6 Findings: Gator

6.1 Gator - Currency Mismatch for Non-English Users Major ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by updating the logic to invoke the Price API with the user's preferred currency and updating the UI element when falling back to USD. The client provided additional details:

> **Resolution:** Logic was updated to invoke the Price API with the user's preferred currency, incorporating a fallback mechanism.
> **Implementation:** PR: MetaMask/snap-7715-permissions#152
> **Verification:** Testing across EN, FR, and DE locales confirmed correct currency handling.

Description

A currency mismatch exists in the `TokenPricesService` implementation that causes non-English users to receive USD price data while displaying it with their local currency formatting. This can potentially create a financial misinformation during permission grant decisions.

The issue occurs in the `#safeParsePreferences()` method which forces USD currency for all non-English locales when fetching price data, while the `formatAsCurrency()` function uses the user's original currency preference for display formatting. This mismatch results in users seeing USD price values formatted with their local currency symbols (EUR, GBP, etc.).

Examples

packages/gator-permissions-snap/src/utils/locale.ts:L11-L14

```
export const FALLBACK_PREFERENCE: Preferences = {
  locale: 'en',
  currency: 'USD',
};
```

packages/gator-permissions-snap/src/services/tokenPricesService.ts:L35-L41

```
#safeParsePreferences(preferences: Preferences): VsCurrencyParam {
  const { currency, locale } = preferences;
  return locale === 'en'
    ? (currency.toLowerCase() as VsCurrencyParam)
    : (FALLBACK_PREFERENCE.currency.toLowerCase() as VsCurrencyParam);
}
```

packages/gator-permissions-snap/src/services/tokenPricesService.ts:L79-L96

```
const preferences = await this.#getPreferences();

// Value in fiat=(Amount in crypto)×(Spot price)
const tokenSpotPrice = await this.#priceApiClient.getSpotPrice(
  tokenCaip19Type,
  this.#safeParsePreferences(preferences),
);
const formattedBalance = Number(
  formatUnits({ value: BigInt(balance), decimals }),
);
const valueInFiat = formattedBalance * tokenSpotPrice;

const humanReadableValue = formatAsCurrency(preferences, valueInFiat);
logger.debug(
  'TokenPricesService:formatAsCurrency() - formatted balance to currency',
  humanReadableValue,
);
```

Recommendation

Ensure the Price API is called with the user's preferred currency when supported, add validation to ensure Price API supports it, and/or add currency conversion logic as a fallback.

Additionally, if the preferred currency is unavailable, it should be clear to the user that the price shown is using a different currency to avoid misinformation.

6.2 Gator - Superfluous Permissions in Manifest Medium ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by removing `snap_getEntropy`. The client provided additional details:

> **Resolution:** The unused snap_getEntropy permission was removed.
> **Implementation:** PR: MetaMask/snap-7715-permissions#153.
> **Verification:** The manifest was confirmed to align with the principle of least privilege.

Description

The snap manifest declares the `snap_getEntropy` permission in its `initialPermissions` section, but analysis of the codebase reveals no usage of entropy generation functionality. The `snap_getEntropy` permission grants access to cryptographically secure random number generation, which is a sensitive capability that should only be requested when actually needed.

This permission allows the snap to:

- Generate cryptographically secure random values
- Access entropy for cryptographic operations
- Potentially use randomness for security-critical functions

Since the gator-permissions-snap does not utilize entropy generation in its implementation, this permission violates the principle of least privilege by granting unnecessary capabilities that could be exploited if the snap were compromised.

Example

packages/gator-permissions-snap/snap.manifest.json:L32

```
"snap_getEntropy": {},
```

Recommendation

Remove the unused `snap_getEntropy` permission from the `initialPermissions` configuration to eliminate unnecessary capabilities and reduce the potential attack surface.

Also see: Remove Development Artefacts issue 4.1 .

6.3 Gator - Missing Runtime Verification for API Responses Medium ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by enforcing manual and zod runtime validation via `makeValidatedRequestWithRetry` for API responses. The client provided additional details:

Resolution: Schema validation using Zod was added for API responses, and request timeouts were enforced.
Implementation: PR: MetaMask/snap-7715-permissions#158
Verification: Invalid or malformed responses are now safely rejected.

Description

The `AccountApiClient.getTokenBalanceAndMetadata()` method uses a compile time type assertion `(await response.json()) as TokenBalanceResponse` without validating the actual structure or content of every field in the response at runtime, trusting external API data blindly. Additionally, the method lacks timeout controls, response size limits, and field-level validation.

For example, fields like `iconUrl`, `symbol`, and `decimals` are returned to the caller without sanitization. Additionally, there is no protection against hanging requests that could cause disruptions in MetaMask user experience.

Examples

getTokenBalanceAndMetadata

- decimals: untrusted numeric value
- symbol: untrusted string
- iconUrl: untrusted URL (pot. extension-csrf, tracking) - later used to fetch icon and converted to b64-data-uri
- account: is verified against internal accounts but not ensured to be an array at runtime.
- type: is manually checked
- balance: implicitly via `BigInt` conversion

packages/gator-permissions-snap/src/clients/accountApiClient.ts:L102-L119

```
const response = await this.#fetch(
  `${this.#baseUrl}/tokens/${tokenAddress}?accountAddresses=${account}&chainId=${chainId}`,
);

if (!response.ok) {
  const message = `HTTP error. Failed to fetch token balance for account(${account}) and token(${tokenAddress}) on chain(${cha
  logger.error(message);

  throw new Error(message);
}

const { accounts, type, iconUrl, symbol, decimals } =
  (await response.json()) as TokenBalanceResponse;

const accountLowercase = account.toLowerCase();
const accountData = accounts.find(
  (acc) => acc.accountAddress.toLowerCase() === accountLowercase,
);
```

getSpotPrice - missing runtime type enforcement

packages/gator-permissions-snap/src/clients/priceApiClient.ts:L40-L76

```
const response = await this.#fetch(
  `${
    this.#baseUrl
  }/v3/spot-prices?includeMarketData=false&vsCurrency=${vsCurrency}&assetIds=${caipAssetType}`,
);

if (!response.ok) {
  logger.error(
    `HTTP error! Failed to fetch spot price for caipAssetType(${caipAssetType}) and vsCurrency(${vsCurrency}): ${response.stat
  );
  throw new Error(
    `HTTP error! Failed to fetch spot price for caipAssetType(${caipAssetType}) and vsCurrency(${vsCurrency}): ${response.stat
  );
}

const spotPricesRes = (await response.json()) as SpotPricesRes;

const assetTypeData = spotPricesRes[caipAssetType];
if (!assetTypeData) {
  logger.error(
    `No spot price found in result for the token CAIP-19 asset type: ${caipAssetType}`,
  );
  throw new Error(
    `No spot price found in result for the token CAIP-19 asset type: ${caipAssetType}`,
  );
}

const vsCurrencyData = assetTypeData[vsCurrency];
if (!vsCurrencyData) {
  logger.error(
    `No spot price found in result for the currency: ${vsCurrency}`,
  );
  throw new Error(
    `No spot price found in result for the currency: ${vsCurrency}`,
  );
}
return vsCurrencyData;
```

Recommendation

- Add request timeout controls to prevent hanging requests.
- Implement runtime JSON schema validation via zod instead of relying on type assertions.

6.4 Gator - Missing Runtime Schema Verification for Profile Sync Store/Retrieve Medium ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by enforcing zod validation on profile sync read and writes. The client provided additional details:

> **Resolution:** Runtime schema validation and size limits were implemented for stored permission data.
> **Implementation:** PR: MetaMask/snap-7715-permissions#157
> **Verification:** Invalid JSON now fails validation and is rejected.

Description

The Profile Sync performs unsafe deserialization of permission data without runtime schema validation.

- `JSON.parse` only validates JSON syntax, not data structure
- The type assertion `as StoredGrantedPermission` is a compile-time hint only
- At runtime, there's no guarantee the parsed object matches the expected structure

The code comments explicitly state that "it is up to the SDK consumer to enforce proper schema management" (lines 147–149). However, no such validation is implemented. Because profile sync data is shared across devices, erroneous or malicious data could propagate and impact all connected devices.

Examples

packages/gator-permissions-snap/src/profileSync/profileSync.ts:L99-L118

```
/**
 * Retrieve all granted permission items under the "7715_permissions" feature will result in GET /api/v1/userstorage/7715_permiss
 * VALUES: decrypted("JSONstringifyPermission1", storage_key), decrypted("JSONstringifyPermission2", storage_key).
 * @returns All granted permissions.
 */
async function getAllGrantedPermissions(): Promise<
  StoredGrantedPermission[]
> {
  try {
    await authenticate();

    const items = await userStorage.getAllFeatureItems(FEATURE);
    return (
      items?.map((item) => JSON.parse(item) as StoredGrantedPermission) ?? []
    );
  } catch (error) {
    logger.error('Error fetching all granted permissions:', error);
    throw error;
  }
}
```

packages/gator-permissions-snap/src/profileSync/profileSync.ts:L126-L141

```
async function getGrantedPermission(
  permissionContext: Hex,
): Promise<StoredGrantedPermission | null> {
  try {
    await authenticate();

    const path: UserStorageGenericPathWithFeatureAndKey = `${FEATURE}.${generateObjectKey(permissionContext)}`;
    const permission = await userStorage.getItem(path);

    return permission
      ? (JSON.parse(permission) as StoredGrantedPermission)
      : null;
  } catch (error) {
    logger.error('Error fetching granted permissions:', error);
    throw error;
  }
```

setter should prevent the storage of non conformant structures

packages/gator-permissions-snap/src/profileSync/profileSync.ts:L144-L166

```ts
/**
 * Store the granted permission in profile sync.
 *
 * Persisting "<permissionContext>" key under "gator_7715_permissions" feature
 * value has to be serialized to string and does not exceed 400kb
 * it is up to the SDK consumer to enforce proper schema management
 * will result in PUT /api/v1/userstorage/gator_7715_permissions/Hash(<storage_key+<permissionContext>">)
 * VALUE: encrypted("JSONstringifyPermission", storage_key).
 * @param storedGrantedPermission - The permission response to store.
 */
async function storeGrantedPermission(
  storedGrantedPermission: StoredGrantedPermission,
): Promise<void> {
  try {
    await authenticate();

    const path: UserStorageGenericPathWithFeatureAndKey = `${FEATURE}.${generateObjectKey(storedGrantedPermission.permissionRe
    await userStorage.setItem(path, JSON.stringify(storedGrantedPermission));
  } catch (error) {
    logger.error('Error storing granted permission:', error);
    throw error;
  }
}
```

Recommendation

Implement runtime schema validation using Zod for all stored permission data.

Add size validation before storage to enforce the documented 400kb limit and handle validation errors gracefully to prevent crashes from corrupted data.

6.5 Gator - Inconsistent Atomicity Expectation in Batch Permission Grants Minor ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by first checking all permissions for approvals and only storing them, when the complete batch is approved. The client provided additional details:

> **Resolution:** Updated the implementation to ensure permission requests are processed sequentially. Storage is only updated once all permissions have been successfully processed, guaranteeing atomicity.
> **Implementation:** PR: MetaMask/snap-7715-permissions#150
> **Verification:** Partial failures were confirmed to no longer result in orphaned permissions.

Description

The batch permission handling in `grantPermission` introduces a critical atomicity flaw. When processing multiple permissions, successful grants are written to ProfileSync but are never reported back to the requesting dApp if any subsequent grant fails. Because `Promise.all` aborts on the first rejection, earlier successful operations remain committed while the overall request is returned as a failure.

This results in orphaned permissions: permissions that exist in the system but are unknown to the client. Such partial success weakens transactional integrity, leading to semi-consistent state, potential permission confusion, and corrupted audit trails. Since partial permissions are not returned the severity was estimated as Minor.

Example

A batch request `[A, B, C]`:

- `A` and `B` succeed and are stored.
- `C` fails.
- The dApp receives a total failure response, but `A` and `B` remain silently stored in ProfileSync.

```ts
// Batch request: [Permission A, Permission B, Permission C]
// Execution flow:
// 1. Permission A: ✅ Signed → ✅ Stored → ✅ Success
// 2. Permission B: ✅ Signed → ✅ Stored → ✅ Success
// 3. Permission C: ✅ Signed → ❌ Storage Fails → 💥 Promise.all REJECTS

// Result: Permissions A & B are stored but NEVER returned to dApp
// Impact: Orphaned permissions in storage, inconsistent dApp state
```

packages/gator-permissions-snap/src/rpc/rpcHandler.ts:L50-L84

```ts
/**
 * Handles grant permission requests.
 *
 * @param params - The parameters for the grant permission request.
 * @returns The result of the grant permission request.
 */
const grantPermission = async (params?: Json): Promise<Json> => {
  logger.debug('grantPermissions()', params);
  const { permissionsRequest, siteOrigin } =
    validatePermissionRequestParam(params);

  const responses = await Promise.all(
    permissionsRequest.map(async (request) => {
      const handler =
        permissionHandlerFactory.createPermissionHandler(request);

      const permissionResponse =
        await handler.handlePermissionRequest(siteOrigin);

      if (!permissionResponse.approved) {
        throw new Error(permissionResponse.reason);
      }

      if (permissionResponse.response) {
        await profileSyncManager.storeGrantedPermission({
          permissionResponse: permissionResponse.response,
          siteOrigin,
        });
      }

      return permissionResponse.response;
    }),
  );

  return responses as Json[];
```

Recommendation

Adopt a transaction-safe approach:

- Use a two-phase commit with pending and confirmed states, or
- Replace `Promise.all` with `Promise.allSettled` and introduce cleanup logic to roll back partial grants, or
- have ProfileSync provide a transaction wrapper.

6.6 Gator - Lifecycle Hooks Permission Used Primarily for Development Features ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by only including `endowment:lifecycle-hooks` when `snapEnv === 'local'`. The client provided additional details:

> **Resolution:** endowment:lifecycle-hooks was removed from production builds.
> **Implementation:** PR: MetaMask/snap-7715-permissions#164.
> **Verification:** The manifest was reviewed to confirm its removal.

Description

The `endowment:lifecycle-hooks` permission is requested but the `onInstall` handler primarily serves development purposes, with minimal production functionality (welcome screen). This expands the attack surface for limited production benefit and violates the principle of least privilege.

1. **Development-only logic** (lines 243-254): Conditional installation of the message signing snap only in local development mode
2. **Minimal production functionality** (line 258): Only shows a welcome screen

The development logic is gated behind environment conditions (`snapEnv === 'local' && isStorePermissionsFeatureEnabled`), meaning that in production builds, the lifecycle hook primarily serves to display a welcome screen - functionality that could be handled through other means.

Example

packages/gator-permissions-snap/src/index.ts:L232-L259

```ts
export const onInstall: OnInstallHandler = async () => {
  /**
   * Local Development Only
   *
   * The message signing snap must be installed and the gator permissions snap must
   * have permission to communicate with the message signing snap, or the request is rejected.
   *
   * Since the message signing snap is preinstalled in production, and has
   * initialConnections configured to automatically connect to the gator snap, this is not needed in production.
   */
  // eslint-disable-next-line no-restricted-globals
  if (snapEnv === 'local' && isStorePermissionsFeatureEnabled) {
    const installedSnaps = (await snap.request({
      method: 'wallet_getSnaps',
    })) as unknown as GetSnapsResponse;
    if (!installedSnaps[MESSAGE_SIGNING_SNAP_ID]) {
      logger.debug('Installing local message signing snap');
      await snap.request({
        method: 'wallet_requestSnaps',
        params: {
          [MESSAGE_SIGNING_SNAP_ID]: {},
        },
      });
    }

  await homepage.showWelcomeScreen();
};
```

Recommendation

Consider removing the `endowment:lifecycle-hooks` permission for production builds since:

6.7 Gator - Misleading Debug Message ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by changing the debug message text. The client provided additional details:

> **Resolution:** The debug log message in getNonce() was corrected.
> **Implementation:** PR: MetaMask/snap-7715-permissions#156.
> **Verification:** Logs now accurately reflect the function name.

Description

The function is `getNonce` and the debug log prints `BlockchainTokenMetadataClient:getTokenBalanceAndMetadata()`.

packages/gator-permissions-snap/src/clients/nonceCaveatClient.ts:L33-L46

```ts
public async getNonce({
  chainId,
  account,
}: {
  chainId: number;
  account: Hex;
}): Promise<bigint> {
  logger.debug('BlockchainTokenMetadataClient:getTokenBalanceAndMetadata()');

  if (!chainId) {
    const message = 'No chainId provided to fetch nonce';
    logger.error(message);
    throw new Error(message);
  }
```

Recommendation

Fix debug message.

6.8 Gator - Recommendation: TokenIcon SVG Runtime Type Enforcement ✓ Fixed

Resolution

Fixed in MetaMask/snap-7715-permissions@ `476e653` by enforcing zod schema input validation on the caller provided parameters. The client provided additional details:

> **Resolution:** Runtime type checks were added for SVG inputs (base64 data URI, width, height).
> **Implementation:** PR: MetaMask/snap-7715-permissions#155
> **Verification:** Malformed inputs are rejected at runtime.

Description

The `TokenIcon` SVG image is generated through string concatenation, which poses a potential risk of HTML injection. The image's width and height are not enforced at runtime and may deviate from their compile-time type declarations. Although we did not identify any instances where this issue is directly exploitable, we strongly recommend implementing runtime type checks and format validations to mitigate potential risks.

- base64 data uri is always in data uri format when passed by callers. however the function does not enforce it.
- width/height are never passed to TokenIcon and defaults to numeric 24 (px).

Also, the SVG is rendered within an `<img>` tag, which prevents JavaScript or interaction events from being supported. Nevertheless, the inputs should be strictly validated to reduce attack surface.

Examples

packages/gator-permissions-snap/src/ui/components/TokenIcon.tsx:L11-L23

```
export const TokenIcon: SnapComponent<TokenIconParams> = ({
  imageDataBase64,
  altText,
  width = 24,
  height = 24,
}) => {
  if (!imageDataBase64) {
    return <Text> </Text>;
  }

  const imageSvg = `<svg xmlns="http://www.w3.org/2000/svg" width="24" height="24" viewBox="0 0 24 24">
    <image href="${imageDataBase64}" width="${width}" height="${height}" />
  </svg>`;
```

7 Document Change Log

Version	Date	Description
1.0	2025-08-20	Initial report
1.1	2025-10-28	Mitigations review: MetaMask/snap-7715-permissions@476e653
1.2	2025-10-30	Mitigations review: MetaMask/snap-7715-permissions@6da514a

Appendix 1 - Files in Scope

This review covered the following files:

File	SHA-1 hash
LICENSE.APACHE2	`7c8c3d0d057417da25b92823c1075d9b4297af55`
LICENSE.MIT0	`8ac914386a5bf48bbf57f0289773e24ba5c36086`
packages/gator-permissions-snap/images/icon.svg	`747c68b49c159aa4e320b76cfe1d086f798da24f`
packages/gator-permissions-snap/images/toggle_disabled.svg	`08661c273c3fb0d1065b4a5cd1f3623c1f9c0e15`
packages/gator-permissions-snap/images/toggle_enabled.svg	`701275f64fa8951268584704f787c7d0e17aca83`
packages/gator-permissions-snap/jest.config.js	`a39ea3b638dfc3ee88ffd03da53523107c325a30`
packages/gator-permissions-snap/jest.setup.ts	`bbee55256035e8e23710822bd96921cb52eeb3d1`
packages/gator-permissions-snap/scripts/build-preinstalled-snap.js	`c7486810f94aefbf0c8cdf08d8c9ce1711ff157b`
packages/gator-permissions-snap/snap.config.ts	`a8dc0f2e03e43cfe6690118c98618456789f71f8`
packages/gator-permissions-snap/src/clients/accountApiClient.ts	`267479513f5a3b52ff8cba1b7ca71d08d3fec6ce`
packages/gator-permissions-snap/src/clients/blockchainMetadataClient.ts	`15c616a9da116bfaea627f77a1bad64e96596b4a`
packages/gator-permissions-snap/src/clients/nonceCaveatClient.ts	`b938be986102b867d2ae1778285d92f683ee283c`
packages/gator-permissions-snap/src/clients/priceApiClient.ts	`614c23b307679e6d877a83b2c4bec561ffb93fe2`
packages/gator-permissions-snap/src/clients/types.ts	`ffaaefcf6ed0938bca198e59d5037b7dab088363`
packages/gator-permissions-snap/src/constants.ts	`9f6143eefc5a922d96e846602b95879908789610`
packages/gator-permissions-snap/src/core/accountController.ts	`000c53d85ee1621c24fbbc6da1e51be898df732a`
packages/gator-permissions-snap/src/core/chainMetadata.ts	`3154803c5f6a8b76bf172bb6e738c844c683d106`
packages/gator-permissions-snap/src/core/confirmation.tsx	`b3912e984d125e5698c1a1aa9b21b236d41696ce`
packages/gator-permissions-snap/src/core/confirmationFactory.ts	`219271b36986f9e4cc15285e3565d1664f352357`
packages/gator-permissions-snap/src/core/permissionHandler.ts	`22c1ceb774766fa539cfb0c3a2fa7c5b7c507298`
packages/gator-permissions-snap/src/core/permissionHandlerContent.tsx	`12b97c1fe502ee276748a09308cd36e24f9ff3e3`
packages/gator-permissions-snap/src/core/permissionHandlerFactory.ts	`27328f3f8e7a9543ad783174cb8dcf8b014df3a2`
packages/gator-permissions-snap/src/core/permissionRequestLifecycleOrchestrator.ts	`f868563d74e78c78afd205e0a16b108a9118ce74`
packages/gator-permissions-snap/src/core/rules.tsx	`49598b8eb8f1a1b757a5072bf2737cdc14407e9a`
packages/gator-permissions-snap/src/core/types.ts	`ee73f9016f324e86f0ccf94840ffd6ef1a4174f4`
packages/gator-permissions-snap/src/homepage/index.tsx	`a4ea60d11f4ef406b71f72786d4389c59b9ce49b`

File	SHA-1 hash
packages/gator-permissions-snap/src/index.ts	`0f20db943b3f498602a9289b919bbf4d3159d6e1`
packages/gator-permissions-snap/src/permissions/contextValidation.ts	`da927c5d8417245d9e0a7f9f9f3824a57ac1214a`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/caveats.ts	`3e580ebc593126b9a0c7a95d1eec6297e28319b4`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/content.tsx	`27b427d995101c6d564e2a5b581b29779ba6355a`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/context.ts	`a11a4adccdc7ca42cf341661cb051447d8ec3309`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/index.ts	`5d3f5087d7f8da91cc24a4de2fb7fccb71abe8f3`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/rules.ts	`c7c863a7c98a6ec1518980c614df92a1c227e451`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/types.ts	`20418570e8fcee337d114ec9018ca9e9ab38c0af`
packages/gator-permissions-snap/src/permissions/erc20TokenPeriodic/validation.ts	`444f35ef86b13eb3936351bf974e66837e505851`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/caveats.ts	`a033d141d9c640a49524bd095dd0770a81272ada`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/content.tsx	`f6c1e33326fca2399c06a9421413fa52ebf1b74b`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/context.ts	`78265977f7421505993971399a6e7bd4b905e92e`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/index.ts	`4fae5c9c01e8b67b795be12c398742ad51b43ddd`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/rules.ts	`dcdb0284f24dcc714f547d099f0c5787d8bb2058`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/types.ts	`0d4f331525cbc4620fd342289830bdc4e39fb586`
packages/gator-permissions-snap/src/permissions/erc20TokenStream/validation.ts	`a46c889cef8def3d0c38a81632f3bb0c01a8e769`
packages/gator-permissions-snap/src/permissions/iconUtil.ts	`779b305201bf0508da53d586d48066539e797156`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/caveats.ts	`0a99b0219571fc689351692abeff69e388a08eaf`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/content.tsx	`47cf7be8e020da538327640f6636bfac7ab05b92`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/context.ts	`4570cbe0f93d87ee97dfdd4bd8b6c5018e22d7b5`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/index.ts	`5e29e4203477a0d73a53809d2fe4edbed8c1b3fd`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/rules.ts	`587cec294c818bbb439fb193224e6b19aaf6cdbc`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/types.ts	`98b2678cd0ce5fdd515cc212b02739115bbf36f0`
packages/gator-permissions-snap/src/permissions/nativeTokenPeriodic/validation.ts	`2d2525d19116559f318fab21d4689ccbaae8355f`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/caveats.ts	`46051eed07543ed15cd1f2602628dadf471e453a`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/content.tsx	`f37d2726beeeac75966081a54423b590f2d3ba50`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/context.ts	`ac43e0db5c57c09861dd42e85db16760a6fd1a15`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/index.ts	`2120ed1ac42791fa02118f4be8e3a65932924267`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/rules.ts	`e3dd06fc4d205be1e3b691bc85ca4c9ffb661ab5`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/types.ts	`b710b7ddca846675c379291ef44ac320703fa173`
packages/gator-permissions-snap/src/permissions/nativeTokenStream/validation.ts	`5fd7bf937abffd9ce3eb0e4f60e116c5f6f8439a`
packages/gator-permissions-snap/src/permissions/permissionOffers.ts	`aec3b051220d3c9a75721b9460adaf7182dab39e`
packages/gator-permissions-snap/src/permissions/validation.ts	`dac746182cf8d86d0ded2af2ff1d9e65c2c2dcc1`
packages/gator-permissions-snap/src/profileSync/config.ts	`9b64c57db0b347ff0196df5b28320dc3c830759c`
packages/gator-permissions-snap/src/profileSync/index.ts	`7d95e45f69db1ccdacd6243960db385147e4f2df`
packages/gator-permissions-snap/src/profileSync/options.ts	`29505f2c36567d0aaeb6208e486f7406c2e1f960`
packages/gator-permissions-snap/src/profileSync/profileSync.ts	`f946be64cb28cf86f098e3be4cf0bf942180c539`
packages/gator-permissions-snap/src/rpc/permissions.ts	`359f9762a300dd59b69a15ccc37dcbd233c8c128`
packages/gator-permissions-snap/src/rpc/rpcHandler.ts	`d7a96554058e1e4a77276f78e899add341040bb6`
packages/gator-permissions-snap/src/rpc/rpcMethod.ts	`c009720b95652254bae6dd44d76e147753e12f61`
packages/gator-permissions-snap/src/services/nonceCaveatService.ts	`3cb3498b1a1d16924c11335e0c49e7b0eecd30a4`
packages/gator-permissions-snap/src/services/tokenMetadataService.ts	`79383d014f03b0627608697f090a8fdfde7c05b7`
packages/gator-permissions-snap/src/services/tokenPricesService.ts	`f813297fb008c5da179ca4e8d7e2b97ee7dface2`
packages/gator-permissions-snap/src/stateManagement.ts	`79007573da3258ef15acf5259f4a939c5995efee`
packages/gator-permissions-snap/src/ui/components/DateTimeField.tsx	`b8076baf0b113f39657ea512a2b4745ada3f651a`
packages/gator-permissions-snap/src/ui/components/DropdownField.tsx	`b7fd79cfe93c00c833b41208aed12421755533c0`
packages/gator-permissions-snap/src/ui/components/Field.tsx	`9c2d2a76f4d8b5a8b690780c0bcba18898426fc2`
packages/gator-permissions-snap/src/ui/components/InputField.tsx	`25837e2cebcf90a53cc0f5f7a344c12bc0af1743`
packages/gator-permissions-snap/src/ui/components/ShowMoreText.tsx	`f81a639e1f3d7c2e46e85fe45f32170730a03d08`
packages/gator-permissions-snap/src/ui/components/SkeletonField.tsx	`f79b9f2c62bb93995ca06f73da73c60a09e64ac7`
packages/gator-permissions-snap/src/ui/components/TextField.tsx	`3f7d62f5f85b85f92795195a7d1fda0619551304`
packages/gator-permissions-snap/src/ui/components/TokenField.tsx	`76d72636ecb2fd163fc708e83f9cc10608143c09`
packages/gator-permissions-snap/src/ui/components/TokenIcon.tsx	`231be99e9d0cd83b8ec4a295a7836813e7cf5d3f`

File	SHA-1 hash
packages/gator-permissions-snap/src/ui/components/TooltipIcon.tsx	`0edbe761a8aebbf2fff3b3e1805203bba1b7c569`
packages/gator-permissions-snap/src/ui/components/index.ts	`1e7f4089b501a604c811ed49bce0504f0723c90d`
packages/gator-permissions-snap/src/userEventDispatcher.ts	`b2ea29722fecda25916749a37bf605e6d6d13b6b`
packages/gator-permissions-snap/src/utils/locale.ts	`867998bcfcbaa5eb79d9586bddf8971023a6c3ee`
packages/gator-permissions-snap/src/utils/string.ts	`2f91bb16e9414c11ff128a69139fe5b9045b1b5b`
packages/gator-permissions-snap/src/utils/time.ts	`57587b36ee311a48ec5c9e1a8fd2a26dbba33ef3`
packages/gator-permissions-snap/src/utils/validate.ts	`2d1858c0d61160369ec61d71c632430e53b16ec6`
packages/gator-permissions-snap/src/utils/value.ts	`86317060f35d107b0d635834bcd85ff7e5618ce7`
packages/gator-permissions-snap/svg-transformer.js	`36626851f55d008bf32a6a56697d89e566e0762e`
packages/permissions-kernel-snap/images/icon.svg	`747c68b49c159aa4e320b76cfe1d086f798da24f`
packages/permissions-kernel-snap/jest.config.js	`d76b4c37b6a3673a45a454ba99f8e4731ee5adfd`
packages/permissions-kernel-snap/jest.setup.ts	`bbee55256035e8e23710822bd96921cb52eeb3d1`
packages/permissions-kernel-snap/scripts/build-preinstalled-snap.js	`1d09b4da1d58fc1ab1cacb0f9e5e7bd7c3c01733`
packages/permissions-kernel-snap/snap.config.ts	`1a05bb79e1bcba834a3fac4cb1901ab591f80542`
packages/permissions-kernel-snap/src/index.ts	`237446a92c4fc9854d1254da6dc176bd94c75455`
packages/permissions-kernel-snap/src/registryManager.ts	`50f5ab94e15db40b63ee716b3186b00b3c7ae345`
packages/permissions-kernel-snap/src/rpc/rpcHandler.ts	`2826b12d6f4ee93c14fce1a6736e1eac5465f2a6`
packages/permissions-kernel-snap/src/rpc/rpcMethod.ts	`83699d8982c18bd66c0f3e3a0153bb973cf6994e`
packages/permissions-kernel-snap/src/utils/error.ts	`87671ab801b32df74000cd4f7a505c5328c9940f`
packages/permissions-kernel-snap/src/utils/index.ts	`8d5ec6b8918c0c3532e51237a89a54dbd12228d0`
packages/permissions-kernel-snap/src/utils/validate.ts	`d51fa025396afc4352db584b22621cf273b17371`
packages/shared/jest.config.js	`2c7fbd0b105504e3de2afda34842356d68b04c48`
packages/shared/src/constants/index.ts	`4c674df45ba5501d0de3e4d7a7d3f9caf5c0aaa7`
packages/shared/src/types/7715-permissions-request.ts	`c2273df9528a3b839eeccce3a0797ccf61411c07`
packages/shared/src/types/7715-permissions-response.ts	`d507e914fa723dc9dd1dd421f52bf052e3e5e0cc`
packages/shared/src/types/7715-permissions-types.ts	`20e1d31454ee988e25805e085484c5d5600e3f59`
packages/shared/src/types/common.ts	`b9bef1f6d457606dd2646d99294429c7040342b7`
packages/shared/src/types/index.ts	`ea43b5ad0e4ef08bd813fd47ac55539ad8248d30`
packages/shared/src/types/snap-permission-registry.ts	`c1040dca93ae4bed9013d3400c4bef1391c93b43`
packages/shared/src/types/snap.ts	`53458a1caced5d72c1b4fd480dc5ed72238f763c`
packages/shared/src/utils/common.ts	`f8d42802679d48307ac9b0529adf6767edea37b0`
packages/shared/src/utils/error.ts	`6eee9d3e7d525e570c07a7de8eaa6bd3463ffdb5`
packages/shared/src/utils/index.ts	`b99980208c35ab16669225346a38bf7b539253a7`
packages/shared/src/utils/logger.ts	`bce3117d83080b7e727232ee195febde9cb0119b`

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.