Security Audit Scope Document

Project Overview

This document outlines the security audit scope for the MetaMask Snap-7715-Permissions project, which implements ERC-7715 permission management through two complementary snaps. The system enables fine-grained permission management for blockchain interactions through a kernel-provider architecture.

We have proposed a number of changes to the ERC-7715 specification, in this pull request. We are adopting these changes preemptively.

Snaps Under Audit

1. Permissions Kernel Snap (`@metamask/permissions-kernel-snap`)

Purpose: Exposes a developer facing API (to be proxied via Wallet API in MetaMask) that allows a caller to request an EIP-7715 permission from the user. Manages a permissions offer registry that contains all permission types the system is capable of granting. Acts as a gatekeeper that forwards valid requests to permission provider snaps and rejects invalid requests, and requests for unsupported permission types. Presently this snap supports only the Gator Permission Snap, but in the future it will support multiple permission providers.

Core Functionality:

- Maintains ephemeral permissions offer registry
- Validates incoming permission requests against registry
- Forwards supported requests to permission provider snaps

2. Gator Permissions Snap (`@metamask/gator-permissions-snap`)

Purpose: Leverages EIP-7702 Delegator smart accounts to fulfil EIP-7715 permission requests, that are received from the Permission Kernel Snap. Presents confirmation dialogs for users to review, attenuate, and grant permissions that may be used to execute specific actions on chain.

Core Functionality:

- Processes permission grant requests with user confirmation
- Provides interactive UI for permission review and attenuation
- Manages permission lifecycle and delegation creation
- Surfaces token metadata and pricing information
- Integrates with profile sync for permission persistence

@metamask/delegation-core is included as a dependency of Gator Permissions Snap and provides basic encoding and hashing functionality for delegations and caveats.

Architecture

The system follows a kernel-provider architecture where the Permissions Kernel Snap acts as a central coordinator and the Gator Permissions Snap serves as a permission provider. For detailed architectural information, see the Gator Permissions Snap architecture doc.

The permission is granted from an EOA in the user's Wallet. The delegation is signed via `eth_signTypedData_v4` RPC call.

By delegating all high-risk operations to the wallet, we prevent the snap from executing sensitive operations, maintaining a strong security boundary between the snap and the user's assets.

Presently the `eth_signTypedData_v4` RPC method in MetaMask disallows signing delegations, in order to ensure that users are not signing authority that they do not understand. We will shortly undertake work to clearly represent permissions being signed within MetaMask, so that this method can be allowed in cases where the permission being signed is able to be identified and decoded by MetaMask's confirmation system. This means that MetaMask will reject any signature requests proposed by Gator Permissions Snaps, although the rest of the system works as expected.

Areas of Specific Security Concern

1. Permission Validation and Filtering

- **Risk**: Confirmation bypass or unauthorized permission granting
- **Components**:
  - `packages/permissions-kernel-snap/src/registryManager.ts` - Registry validation
  - `packages/permissions-kernel-snap/src/utils/validate.ts` - Input validation
  - `packages/gator-permissions-snap/src/core/permissionRequestLifecycleOrchestrator.ts`
- **Concerns**: A requested permission may not be granted to the caller without the user first being presented with the details of the permission, and pressing "Grant". This triggers a Sign Typed Data confirmation screen where the delegation is signed.

2. Permission Persistence

- **Risk**: Successfully granting permission without successful persistence
- **Components**:
  - `packages/gator-permissions-snap/src/profileSync/` - Profile synchronization
- **Concerns**: It is critical that any granted permission is stored in profile sync, before being returned to the caller, as this makes the permission available for revocation. If storage of the permission fails, the granting of the permission should also fail, and the signature and delegation should never be made available to the caller.

3. User Input and Confirmation Flow

- **Risk**: UI manipulation, or misrepresentation
- **Components**:
  - `packages/gator-permissions-snap/src/core/confirmation.tsx`
  - `packages/gator-permissions-snap/src/userEventDispatcher.ts`
  - `packages/gator-permissions-snap/src/ui/components/`
- **Concerns**: Misrepresenting the permission being granted via UI redressing, or otherwise making the details of the permission that the user is granting unclear.

4. External API Integration

- **Risk**: API manipulation, data injection
- **Components**:
  - `packages/gator-permissions-snap/src/clients/` - External API clients
  - `packages/gator-permissions-snap/src/services/` - Token and price services

- **Concerns**: External API responses could be manipulated to misrepresent the details of the permission being granted, such as displaying incorrect token prices, balances, or metadata that could mislead users during the permission granting process.

5. Permission Context and Metadata Handling

- **Risk**: Context manipulation, metadata injection
- **Components**:
  - `packages/gator-permissions-snap/src/permissions/` - Permission-specific implementations
  - Permission context builders and validators
- **Concerns**: Manipulation of the context and metadata, or interruption of the rendering loop may result in a permission being granted that is not the same as the permission that the user is seeing.

Future Changes and Integration Plans

Integration into MetaMask

Both Permissions Kernel, and Gator Permissions snaps will be pre-installed into MetaMask extension (and in the future MetaMask mobile). A number of specific changes are planned for MetaMask extension in order to support the permissions system.

1. **RPC Ingress via Wallet API** presently the Permissions Kernel Snap must be invoked directly via the `wallet_invokeSnap` RPC. The Wallet API will implement the RPC methods described in ERC-7715 and forward those methods to the Permissions Kernel snap.
2. **Sign Permission Confirmation** when the snap calls `eth_signTypedData_v4` with a delegation representing a valid, supported ERC-7715 permission, the request will be decoded to present the permission details to the user. If the delegation is unable to be decoded, the request will be rejected. Metadata will be passed with the delegation, including the justification and origin of the request. This functionality will only be available to the Gator Permissions Snap snap, delegation signing requests made by other parties will be rejected.
3. **EIP-7702 Upgrade Account** presently a user is able to manually delegate their EOA to the MetaMask Delegator Stateless 7702 account, via the MetaMask UI, or by accepting a proposed EIP-5792 request that requires the smart contract functionality. The Gator Permissions Provider snap will utilize a new RPC method to request the user upgrade the account before granting the permission.

These changes are described in the Readable permissions requirements ADR.

Planned Enhancements

These planned enhancements are considered in the mid to long term, and are documented here for context only.

1. **Additional Permission Types**: Additional permission types beyond current token-based permissions.
2. **Multi-Provider Support**: Expansion beyond single gator-permissions-snap to multiple permission providers is considered for the very long term. This includes expansion of the permission type registry to support multiple permission providers.

Additional Documentation

- ERC-7715: Grant Permissions from Wallets
- Update ERC-7715: Simplify specification to ease wallet implementation

- Architecture Documentation - Detailed system architecture
- Permission Persistence - Permission storage mechanisms
- Ephemeral Registry - Registry implementation details