Metamask Delegation Framework Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

0kage

May 1, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

5 Audit Scope

The following contracts were included in the scope of the current audit:

Enforcers

- MultiTokenPeriodEnforcer.sol

Helpers

- DelegationMetaSwapAdapter.sol

Script

- DeployDelegationMetaSwapAdapter.s.sol

6 Executive Summary

Over the course of 5 days, the Cyfrin team conducted an audit on the Metamask Delegation Framework smart contracts provided by Metamask. In this period, a total of 2 issues were found.

Summary

Project Name	Metamask Delegation Framework
Repository	delegation-framework
Commit	0f8e128adebc...
Audit Timeline	April 14th - April 18th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	0
Informational	2
Gas Optimizations	0
Total Issues	2

Summary of Findings

[I-1] Missing zero address checks in DelegationMetaSwapAdapter	Resolved
[I-2] Ambiguous expiration timestamp validation in `DelegationMetaSwapAdapter`	Resolved

7 Findings

7.1 Informational

7.1.1 Missing zero address checks in DelegationMetaSwapAdapter

Description: Missing zero address checks in the `constructor` and `setSwapApiSigner` functions of `DelegationMetaSwapAdapter`.

```
  constructor(
        address _owner,
        address _swapApiSigner,
        IDelegationManager _delegationManager,
        IMetaSwap _metaSwap,
        address _argsEqualityCheckEnforcer
    )
        Ownable(_owner)
    {
        swapApiSigner = _swapApiSigner; //@audit missing address(0) check
        delegationManager = _delegationManager; //@audit missing address(0) check
        metaSwap = _metaSwap; //@audit missing address(0) check
        argsEqualityCheckEnforcer = _argsEqualityCheckEnforcer; //@audit missing address(0) check
        emit SwapApiSignerUpdated(_swapApiSigner);
        emit SetDelegationManager(_delegationManager);
        emit SetMetaSwap(_metaSwap);
        emit SetArgsEqualityCheckEnforcer(_argsEqualityCheckEnforcer);
    }
  function setSwapApiSigner(address _newSigner) external onlyOwner {
        swapApiSigner = _newSigner; //@audit missing address(0) check
        emit SwapApiSignerUpdated(_newSigner);
    }
```

Recommended Mitigation: Consider adding zero address checks.

Metamask: Resolved in commit 6912e73.

Cyfrin: Resolved.

7.1.2 Ambiguous expiration timestamp validation in `DelegationMetaSwapAdapter`

Description: In the DelegationMetaSwapAdapter.sol contract, the _validateSignature() method uses a "greater than" (>) comparison instead of a "greater than or equal to" (>=) comparison when validating signature expiration:

```
function _validateSignature(SignatureData memory _signatureData) private view {
    if (block.timestamp > _signatureData.expiration) revert SignatureExpired();

    bytes32 messageHash_ = keccak256(abi.encodePacked(_signatureData.apiData,
    ↪  _signatureData.expiration));
    bytes32 ethSignedMessageHash_ = MessageHashUtils.toEthSignedMessageHash(messageHash_);

    address recoveredSigner_ = ECDSA.recover(ethSignedMessageHash_, _signatureData.signature);
    if (recoveredSigner_ != swapApiSigner) revert InvalidApiSignature();
}
```

This implementation allows signatures to remain valid at the exact moment of their expiration timestamp, which creates ambiguity in the intended security model.

Impact: A signature marked as expired (with an expiration timestamp equal to the current block timestamp) is still considered valid, which may be counter-intuitive and could lead to confusion.

Recommended Mitigation: If the current behavior is intentional, consider renaming the `expiration` field to `validUpto`. Alternatively, to make it semantically clear with the term `expiration`, consider replacing > with >=.

Metamask: Resolved in commit 6912e73.

Cyfrin: Resolved.