Metamask Delegation Framework

Date	August 2024
Auditors	Rai Yang, François Legué

1 Executive Summary

This report presents the results of our engagement with **MetaMask** to review **Metamask Delegation Framework**.

The review was conducted over two weeks, from **August 19, 2024** to **August 30, 2024**, by **Rai Yang** and **François Legué**. A total of 20 person-days were spent.

The review focused on the updates implemented since last audit (commit hash `ee9f363e1128c7ef214f36e08dc0ce0b1069ef26`).

The main updates include:

- The refactor of the delegation manager to support delegation batching
- The ERC 7579 execution support.
- New caveat enforcers: `ArgsEqualityCheckEnforcer.sol`, `NativeBalanceGteEnforcer.sol`, `NativeTokenPaymentEnforcer.sol`, `NativeTokenTransferAmountEnforcer.sol`, `RedeemerEnforcer.sol`.

2 Scope

Our review focused on the commit hash b68ce99e9a545e3899e1c3634a837f3460c32370. The list of files in scope can be found in the Appendix.

2.1 Objectives

Together with the **MetaMask/delegation-framework** 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 known vulnerabilities particular to smart contract systems, as outlined in our Smart Contract Best Practices, and the Smart Contract Weakness Classification Registry.

3 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.

3.1 Open Delegations Combined With `NativeTokenPaymentEnforcer` Are Not Protected Against Front-Running Major ✓ Fixed

Resolution

Fixed in PR16.

`ArgsEqualityCheckEnforcer` enforcer terms contain the delegation hash and the redeemer. This prevents an external actor from front-running the payment when executing a delegation.

Description

The `NativeTokenPaymentEnforcer` is an enforcer that, combined with a delegation `D1`, ensures a `delegate` pays to redeem `D1`. The `delegate` that redeems the delegation `D1` must include in the arguments of this enforcer a delegation `D2` to the `NativeTokenPaymentEnforcer` itself. It will give the right to the enforcer to pull the amount specified in the first place in the terms of the `NativeTokenPaymentEnforcer` to exercise the delegation `D1`.

To prevent front-running and the use of the second delegation `D2` by a malicious actor who inspects the mempool, the usage of the `ArgsEqualityCheckEnforcer` enforcer is recommended with `terms` being the `delegationHash` of the first delegation `D1`. This will ensure that the delegation that is being paid for with the delegation `D2` is the delegation `D1`. While this works well when the delegation `D1` is restricted to a specific delegate, it is not sufficient in the case of open delegations.

The Delegator framework offers the possibility to create an open delegation. The purpose of an open delegation is that it can be redeemed by anyone.

The problem is that the computed `delegationHash`, is based on the fields of the delegation structure and the enforcers terms. However, in case of an open delegation, the `delegate` field will contain the same value (`ANY_DELEGATE`) regardless of who is currently redeeming the delegation.

The `DelegationManager` contract will accept the redeem from anyone when the delegation `delegate` field is `ANY_DELEGATE`.

src/DelegationManager.sol:L154

```
if (delegations_[0].delegate != msg.sender && delegations_[0].delegate != ANY_DELEGATE) {
```

In such a scenario, anyone using `D1` would share the same `delegationHash` in the arguments. Consequently, someone could watch the mempool and front-run a legitimate redeem by leveraging the delegation `D2`. The attacker would be able to execute his own action with the delegation `D1` but use the delegation `D2` (which was generated by someone else) to pay the right to exercise it.

Recommendation

Ensure that the consumer of the delegation `D2` (also known as immediate delegator) is the current redeemer of the delegation `D1`. This could be done using the `redeemer` parameter of the `afterHook` function of the `NativeTokenPaymentEnforcer` enforcer.

3.2 Front-Running in NativeTokenPaymentEnforcer Major ✓ Fixed

Resolution

Fixed in PR16.

`ArgsEqualityCheckEnforcer` is now enforced when using `NativeTokenPaymentEnforcer`. Its terms contain the delegation hash and the redeemer. This prevents an external actor from front-running the payment when executing a delegation.

Description

In `NativeTokenPaymentEnforcer`, the redeemer must include a payment delegation to pay the delegator to execute an execution. The payment delegation can be created by the redeemer or some other contract or account in behave of the redeemer(e.g. redeemer's smart account). However, this setup introduces a vulnerability where another redeemer can front-run the original redeemer by capturing and utilizing the same payment delegation specified in the `_args` parameter of the `afterHook` function in the `NativeTokenPaymentEnforcer`. In such cases, the payment would still be covered by the original redeemer or its root delegator.

To prevent front-running, it is recommended to use the `ArgsEqualityCheckEnforcer` to ensure that the redemption can only be executed by the original redeemer and the delegation is not open (`delegate != ANY_DELEGATE`), this is done by validating the delegation hash (`_delegationHash`), which is derived from the redeemed delegation. However the use of `ArgsEqualityCheckEnforcer` is currently optional within the payment delegation, not mandatory. Furthermore, even it's enforced, the vulnerability remains when the delegation is open (issue 3.1) . In this scenario, the `_delegationHash` would be identical for any redeemer, which fails to prevent front-running effectively.

Examples

src/enforcers/NativeTokenPaymentEnforcer.sol:L78-L102

```
Delegation[] memory delegations_ = abi.decode(_args, (Delegation[]));

// Assign the delegation hash as the args to the args equality enforcer.
for (uint256 x = 0; x < delegations_.length; ++x) {
    Delegation memory delegation_ = delegations_[x];
    for (uint256 i = 0; i < delegation_.caveats.length; ++i) {
        if (delegation_.caveats[i].enforcer == argsEqualityCheckEnforcer) {
            delegation_.caveats[i].args = abi.encodePacked(_delegationHash);
        }
    }
}

bytes[] memory permissionContexts_ = new bytes[](1);
permissionContexts_[0] = abi.encode(delegations_);

bytes[] memory executionCallDatas_ = new bytes[](1);
executionCallDatas_[0] = ExecutionLib.encodeSingle(recipient_, amount_, hex"");

ModeCode[] memory encodedModes_ = new ModeCode[](1);
encodedModes_[0] = ModeLib.encodeSimpleSingle();

uint256 balanceBefore_ = recipient_.balance;

// Attempt to redeem the delegation and make the payment
delegationManager.redeemDelegations(permissionContexts_, encodedModes_, executionCallDatas_);
```

Recommendation

In function `afterHook` add a require statement to check whether the immediate delegator (`delegation_[0].delegator`) of the payment delegation is equal to the redeemer (`_redeemer`).

3.3 Discrepancies Between the `ERC-7579` Draft Standard and Delegator's Implementation Minor

✓ Fixed

Resolution

Fixed in PR16.

The documentation now mentions what parts of the standard have been implemented and what aspects were excluded.

Description

The standard `ERC-7579` outlines the minimal required interfaces and behavior for modular smart accounts to ensure interoperability across implementations. However, It should be noted that this ERC is currently in draft status and may be prone to changes until it reaches the final version.

The current Delegator implementation contains discrepancies with the ERC-7579 draft. Here is a list of identified discrepancies:

- the account must implement the account config interface which is composed of the following functions:
  - `accountId()` : helps to identify the current account;
  - `supportsExecutionMode()` : helps to determine which execution mode is supported;
  - `supportsModule()` : helps to know which module is supported.
- the account must implement the module config interface which is composed of the following functions:
  - `installModule()` : installs a new module;
  - `uninstallModule()` : uninstalls a module;
  - `isModuleInstalled` : helps to know if a module is installed.
- Each component must implement the module interface which is composed of the following functions:
  - `onInstall()` : callback executed by the smart contract account on installation;
  - `onUninstall()` : callback executed by the smart contract account on uninstallation;
  - `isModuleType()` : helps to determine the type of the current module;
- The standard separate modules into the different types:
  - Validation (type: 1): The signature validation function implemented in `HybridDeleGator` and `MultiSigDelegator` must be in modules of Validation type;
  - Execution (type: 2): `DelegateManager` contract must be registered as a module of type Execution because it is the contract that can execute transactions on behalf of the smart account via a callback.
  - Fallback (type: 3);
  - Hooks (type: 4): Enforcers contracts must be registered as modules of type hooks.
- The hooks in `ERC-7579` standard are the equivalent of enforcers in the Delegator's codebase. To comply with the naming convention of the standard, the functions must be respectively changed from `beforeHook` and `afterHook` to `preCheck` and `postCheck`

Recommendation

Modify the current architecture and/or implement the missing features to comply with the ERC-7579 standard. However, as the standard is in draft status, it may change until its final version.

3.4 The `NativeTokenPaymentEnforcer` Enforcer May Change the Final State in Contradiction With Other Enforcers Minor ✓ Fixed

Resolution

Fixed in PR16.

The documentation now details how the `NativeTokenPaymentEnforcer` works and warns about its potential side effects.

Description

The `NativeTokenPaymentEnforcer` is an enforcer that, combined with a delegation, ensures a `delegate` pays to redeem it. This enforcer relies on a second delegation that gives the right (to the `NativeTokenPaymentEnforcer` enforcer) to pull the necessary amount to pay to exercise the first delegation. This is done through another call to the `redeemDelegation` with a specifically crafted `executionCallData` that will call the `recipient` sending the funds.

src/enforcers/NativeTokenPaymentEnforcer.sol:L94

```
executionCallDatas_[0] = ExecutionLib.encodeSingle(recipient_, amount_, hex"");
```

When this inner delegation is redeemed, a callback to the recipient is performed. During this callback, the recipient gains execution control flow and can perform arbitrary calls that can change the final state of the transaction.

It is to be kept in mind that this callback could contradict the restrictions put in place with the enforcers of the first delegation but after the executions of the `afterHook` of these ones.

Recommendation

A strong warning should be mentioned for the `NativeTokenPaymentEnforcer` enforcer that the payment gives potential code execution to the recipient and that it could completely change the final state of the transaction.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
src/DeleGatorCore.sol	`5aa88f81ecdf7584ca0934bc39809b2969fd3886`
src/DelegationManager.sol	`b8b1c17ff7d3c2e1e47fd7e461e778d0ad17cf09`
src/HybridDeleGator.sol	`8f12327f9c8e41c2f160459095affbd647ffff86`
src/MultiSigDeleGator.sol	`19b677d316285c78cefb13dd2cd08c3b708dcccd`
src/enforcers/AllowedCalldataEnforcer.sol	`2f72c6914706ea0fa846a0a664dfe2da7b29bcb6`
src/enforcers/AllowedMethodsEnforcer.sol	`cee4d9e021108918d6eb6ba436118f5d4e4f51ac`
src/enforcers/AllowedTargetsEnforcer.sol	`84c621ee778d0d66aa63952224f91a6a818367d2`
src/enforcers/ArgsEqualityCheckEnforcer.sol	`f53222364165890ac0d540ef9fcf058f02aee3f9`
src/enforcers/BlockNumberEnforcer.sol	`0bce813009249b92502840dbc4b93e21cb5fe988`
src/enforcers/CaveatEnforcer.sol	`49b200dbbaaa14b34d6b0adced6f7183e23e1ef0`
src/enforcers/DeployedEnforcer.sol	`d5c35230fe96ade72d0f7176c12759dc84f2edfa`
src/enforcers/ERC20BalanceGteEnforcer.sol	`581ef406e4dae732de3469b8b951599eac514578`
src/enforcers/ERC20TransferAmountEnforcer.sol	`31709de966f01b03eef00fcd23e013945a059ec3`
src/enforcers/IdEnforcer.sol	`8fc361f7dc8e60f4aeb94680e6baf297a062c78c`
src/enforcers/LimitedCallsEnforcer.sol	`157f32f206b5ef34abdb1ee8a54c15f3db8fefc9`
src/enforcers/NativeBalanceGteEnforcer.sol	`e64513c8e2f11e57bc77f503d2fbd3d7bbe95568`
src/enforcers/NativeTokenPaymentEnforcer.sol	`d1202ed7d87e70009edf82165d341e9ead629940`
src/enforcers/NativeTokenTransferAmountEnforcer.sol	`5e0fb9d00e71a3be917a09e60bf4c1cdd80a575b`
src/enforcers/NonceEnforcer.sol	`7deb2afd85df629cb02dcb7518d1eb05e61e080d`
src/enforcers/RedeemerEnforcer.sol	`6171a0434505cf50adb9912afd6a1503553b8c10`
src/enforcers/TimestampEnforcer.sol	`c4884e1416d318bb9f8a6f21385e6fb87b821d8e`
src/enforcers/ValueLteEnforcer.sol	`dcc1fa86b2d0d24cddee3b07daccf440f3e9512e`
src/interfaces/ICaveatEnforcer.sol	`6874e633d56fa166ec14eab87a656145c3c7dea4`
src/interfaces/IDeleGatorCore.sol	`e308d7809f972e58251a3a9d2fb6789df714ee32`
src/interfaces/IDelegationManager.sol	`8947ef1d8099a0ac5bd21793eb8bcf470ee06a14`
src/interfaces/IERC173.sol	`98840c9d3b5f5d090fdd45707a42bb99f206bfe8`
src/libraries/ERC1271Lib.sol	`227e93f2944c118ec6af8c720d388117b557a704`
src/libraries/EncoderLib.sol	`445bb6ed8377e48e13b9d96043785ee3665a542f`
src/libraries/P256FCLVerifierLib.sol	`53cef8bc609642ce6af3802112b88a6317537374`
src/libraries/P256VerifierLib.sol	`08a58e2610c01054a30d48ee7c4c49b73c3f8454`
src/libraries/WebAuthn.sol	`00c200ce9094677eda4342ccc4429b9d8143a5ab`
src/libraries/utils/Base64URL.sol	`16a170f5ade38bc457c265d8804c27cb2865215c`
src/utils/Constants.sol	`593a1a250e824b5b322f0c259bcf9ca3a013aaec`
src/utils/SimpleFactory.sol	`8bbb90b2b4c778a6d244b867e13bc970a89c9717`
src/utils/Types.sol	`c54fcebbbba459ae01dd87c1f9302bcede196988`

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.