OpenZeppelin | security

# Reserve ThrottleWallet Audit

September 22, 2023

# Table of Contents

# Summary

| | |
|---|---|
| Type | DeFi |
| Timeline | From 2023-09-03<br>To 2023-09-04 |
| Languages | Solidity |

| | |
|---|---|
| Total Issues | 23 (7 resolved, 3 partially resolved) |
| Critical Severity Issues | 0 (0 resolved) |
| High Severity Issues | 0 (0 resolved) |
| Medium Severity Issues | 4 (0 resolved) |
| Low Severity Issues | 7 (3 resolved, 2 partially resolved) |
| Notes & Additional Information | 12 (4 resolved, 1 partially resolved) |

# Scope

We audited Reserve Protocol's `throttle-wallet` repository at the e9a56a0 commit.

- `ThrottleWallet.sol`

# System Overview

Reserve Protocol is a protocol that allows for the permissionless creation of 1:1 asset-backed stablecoins. Such stablecoins are called RTokens, and are backed by a collection of assets chosen by the creator of the RToken. To protect RToken holders in the event of collateral token default, Reserve Protocol utilizes the `RSR` token. RSR tokens can be staked in the protocol to provide over-collateralization to a selection of RTokens. In return for the risk, stakers receive a portion of the RToken fees.

Over the course of this audit, we audited a vesting contract, which is used for the rate-limited distribution of RSR tokens, subject to a timelock.

# Trust Assumptions and Privileged Roles

It is assumed that the admin address is controlled by a good faith actor. This party can modify the beneficiary of the vesting contract, cancel pending withdrawal requests, and renounce their role.

# Medium Severity

## M-01 Admin Address Is Not Modifiable

The admin, a key operational role, lacks the ability to update its address. It is crucial to facilitate address changes due to potential security threats or operational adjustments, especially considering that operational requirements may last years due to throttling.

Reasons to change the admin address include:

- Suspected address compromise.
- Admin's desire to upgrade wallet security (e.g., switching to a hardware wallet, a multi-sig, or an account abstraction wallet). As security practices evolve and if locked funds appreciate in value, the initial admin wallet may no longer be adequately secure.
- Dissatisfaction with the initial multi-sig wallet's functionality.

Consider implementing OpenZeppelin's `Ownable2Step` for more secure role management.

***Update:*** *Acknowledged, not resolved. Reserve Protocol intends the admin to be a contract with its own ownership management functionality. The Reserve Protocol team stated:*

> *Admin is not intended to be modifiable. It is intended to be a high M in a an M-of-N multisig, where all the private keys are extremely cold.*

## M-02 User Address Becomes Unmodifiable if Admin Is Renounced

Given the throttling and variations in amounts and user activity, the release process could span over several years. If adminship is renounced during this period, updating the user's wallet becomes impossible.

Over such an extended timeframe, wallet updates may be necessary due to:

- Users' desire to enhance wallet security or usability, possibly by transitioning to a hardware wallet, modifying their multi-sig setup, or upgrading to an account abstraction wallet.

- A wallet being potentially compromised, but with funds not fully withdrawn from the contract due to timelocks and throttling.

Consider transferring the admin role to the `user` rather than setting it to `address(0)` when renouncing adminship.

***Update:*** *Acknowledged, not resolved. Reserve Protocol intends the user to be a contract with its own ownership management functionality. The Reserve Protocol team stated:*

> *The user does not need to be modifiable once the admin is renounced. The intention is for the user to eventually be set to some immutable smart contract that handles long-term distribution from the `ThrottleWallet`. Once this user contract is set, the Admin should be renounced to solidify the immutability.*

## M-03 Cancelled Withdrawals Do Not Release Consumed Throttle Capacity

Throttling capacity is consumed when initiating a withdrawal request. However, on cancellation, only the `totalPending` balance is freed. The throttling capacity, however, remains consumed. Despite this design being documented, it can unjustly restrict users from subsequent valid and legitimate withdrawal requests.

For instance, if a user's wallet is compromised, and the pending withdrawal must be cancelled, it is likely a new wallet address would be assigned by the admin. In such scenarios, not releasing the consumed throttling capacity seems counterintuitive and unfairly prevents the user from accessing their funds. Similarly, if a user accidentally specifies a wrong target address and the request needs to be cancelled, the remaining throttling limit unduly impedes subsequent withdrawals.

Consider adding the cancelled `amount` back to the `lastRemainingLimit`, thereby partially restoring the throttle capacity. Moreover, consider allowing users to cancel their own requests, minimizing throttling capacity loss from delay due to communication and action required from the admin.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Cancelled withdrawals should only happen in the event that the user key is compromised and a bad withdrawal is initiatied. The additional complexity of restoring the throttle capacity is deemed unnecessary.*

## M-04 Withdrawal Requests Uncancellable When Admin Is Renounced or Lost

The ability to cancel a withdrawal request lies solely with the admin. Yet, the contract provides a method to renounce adminship. Moreover, there is a risk that an admin might lose access to their wallet or choose not to collaborate with a user.

In such scenarios, if there is a need to cancel an initiated request, it becomes impossible. Crucially, since these requests reserve a "pending" balance, they must be cancelled to release the funds for withdrawal, otherwise these funds are permanently lost.

Several scenarios can warrant the need to cancel a request:

- Specifying an address that, although once controlled by the user, gets compromised between the initiation and the unlock time.
- Incorrectly designating an address not under the user's control due to oversight.
- Using a zero address or the RSR token contract's address, which would lead to transfer failures.

Consider enabling users to cancel their own withdrawal requests. Additionally, consider transferring the admin role to the user's address when adminship is renounced.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *The intention is for the user to eventually be set to some immutable smart contract that handles long term distribution from the* `ThrottleWallet`*. Once this user contract is set, the admin should be renounced to solidify the immutability. At this point, no withdrawals should be cancellable.*

# Low Severity

## L-01 Missing Docstrings

Throughout the codebase there are several parts that do not have docstrings. For instance:

- Line 176 in `ThrottleWallet.sol`
- Line 175 in `ThrottleWallet.sol`
- Line 31 in `ThrottleWallet.sol`
- Line 33 in `ThrottleWallet.sol`

- Line 32 in `ThrottleWallet.sol`
- Line 30 in `ThrottleWallet.sol`
- Line 51 in `ThrottleWallet.sol`
- Line 55 in `ThrottleWallet.sol`
- Line 41 in `ThrottleWallet.sol`
- Line 40 in `ThrottleWallet.sol`
- Line 52 in `ThrottleWallet.sol`
- Line 50 in `ThrottleWallet.sol`
- Line 48 in `ThrottleWallet.sol`
- Line 42 in `ThrottleWallet.sol`
- Line 54 in `ThrottleWallet.sol`

Consider thoroughly documenting all functions (and their parameters) that are part of any contract's public API. Functions implementing sensitive functionality, even if not public, should be clearly documented as well. When writing docstrings, consider following the Ethereum Natural Specification Format (NatSpec).

***Update:*** *Resolved in pull request #9 at commit 3019f5c.*

## L-02 Uncovered `require` Statements in Tests

Several `require` statements lack test coverage: 1, 2, 3, 4, 5, 6.

Consider enhancing test coverage to ensure full branch coverage, particularly input validation checks.

***Update:*** *Resolved in pull request #9.*

## L-03 Hardcoded Token Address Limits Testing

The hardcoded address prevents proper testing using a live testnet.

Consider using an immutable variable instead of a constant, and setting the value in the constructor.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *The `ThrottleWallet` is currently only intended to be used for RSR. This modification can be made at a later date if the contract needs to be used for any other tokens. Testing can be performed on a fork of mainnet.*

## L-04 Griefing Due to Lack of Access Control for Withdrawal Completion

There is a risk that if a withdrawal request's cancellation is delayed past the `unlockTime`, any external party can invoke `completeWithdrawal`, griefing the user, or causing the funds to be transferred to a compromised address.

Consider limiting access to the `completeWithdrawal` method to only the user.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Withdrawals should only be canceled in the event that a user key is compromised and a malicious withdrawal is initiatied. 4 weeks is more than enough time to notice a malicious withdrawal, and it will be promptly canceled well before the delay is expired.*

## L-05 `availableToWithdraw` Method Does Not Reflect the True Withdrawable Amount

The `availableToWithdraw()` method only computes the maximum based on time elapsed, neglecting other prerequisites outlined in `initiateWithdrawal`. Particularly, the contract must hold enough balance to satisfy both the withdrawal `amount` and the sum recorded in `totalPending`. Given its name, this method could misguide and result in errors in integrations and user experiences. Additionally, users of the contract are required to inspect its implementation to calculate the correct amount available to withdraw.

Consider including the balance checks within the view to return the correct amounts. Additionally, consider providing the logic implemented currently in this method under a different name, for example, `availableThrottleLimit`.

***Update:*** *Partially resolved in pull request #9 at commit 5b36323. The full balance of the contract is checked, but pending amounts are included as available to withdraw.*

## L-06 Risk of Trapped ERC-20 or ETH

As the contract will be supplied with RSR tokens by directly sending RSR tokens to it, there is a higher-than-usual risk of mistakenly sending other ERC-20 tokens or ETH. This can occur especially if multiple contracts are in use by different parties.

Consider implementing a function, accessible by the admin, that facilitates the retrieval of inadvertently deposited non-RSR ERC-20 tokens or ETH.

***Update:*** *Resolved in pull request #9 at commit 3aff08a. Notably, the added method uses* `.transfer` *to transfer ETH, which is no longer the recommended way to send ETH. Additionally, the method lacks access control which may enable social (e.g., phishing) attack vectors on Reserve Protocol users if used to legitimize malicious token flows between protocol-controlled contracts.*

## L-07 Missing Input Validation

There are two places in the code that would benefit from additional input validation:

- The beneficiary of the wallet can initiate a withdrawal of tokens to an arbitrary address. In order to prevent accidental initiation of a withdrawal to the zero address, consider checking for the `target` input parameter to be a non-zero address.
- The admin can change the `user` storage variable to an arbitrary new address, with the exception of the old value of `user`. In order to prevent `user` from accidentally being set to the zero address, consider checking for the `_newUser` input parameter to be a non-zero address.

***Update:*** *Partially resolved in pull request #9 at commit 5f10bf4. A check was added for* `initiateWithdrawal`*, but not for setting the user. The Reserve Protocol team stated:*

> *No need to disallow the setting of user to* `0x0`*. The admin will validate that the user is set correctly before they renounce their role.*

# Notes & Additional Information

## N-01 Excessive Gas Usage From Redundant Storage Reads

In `completeWithdrawal`, multiple storage reads of `WithdrawalRequest` values increase the gas costs.

Consider loading the `WithdrawalRequest` struct into a memory variable initially, then updating only the request's `status` in storage.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes. Safety / low-complexity / minimal changes are prioritized over small gas improvements.*

## N-02 Overly Complex and Inefficient Storage and Nonce Mechanism

The contract employs a mapping paired with a nonce for tracking withdrawals. Given the sequential nature of requests and the increment-by-one pattern of the nonce, a storage array seems more appropriate.

Shifting to a storage array offers advantages:

- Eliminates the need to maintain the `nonce` storage.
- Ensures only initiated requests are accessed, preventing out-of-bounds requests.
- Potentially improves gas efficiency, especially after the anticipated Verkle Trees hard fork, due to using sequential storage.

Consider simplifying the code by discarding the `nonce` and utilizing a storage array for requests.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes. Safety / low-complexity / minimal changes are prioritized over small gas improvements.*

## N-03 Excessive Gas Consumption in `initiateWithdrawal` Due to Inefficient Storage Usage

The `initiateWithdrawal` method updates four storage variables: `lastWithdrawalAt`, `lastRemainingLimit`, `totalPending`, and `nextNonce`. Presently, each variable occupies an individual storage slot, leading to high gas costs.

To optimize gas consumption:

- Pack the variables into fewer storage slots by adjusting variable sizes.
- Group state variable declarations (note that `nonce` is declared separately).

Specifically, for scenarios where users might make over 65535 withdrawals:

- Use `uint128` for amounts and `uint32` for time and nonce. This approach takes up 2 slots.

Alternatively, if anticipated future withdrawals for any user are safely below 65535, opt for a single-slot strategy:

- Use `uint32` for `lastWithdrawalAt`.
- Use `uint16` for nonce, assuming the impossibility of users surpassing 65535 withdrawals on L1.
- Use `uint104` for both `lastRemainingLimit` and `totalPending`, taking advantage of RSR's fixed supply being below this size's maximum value.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes. Safety / low-complexity / minimal changes are prioritized over small gas improvements.*

## N-04 Inefficient Storage Layout of `WithdrawalRequest` Increases Gas Costs

`WithdrawalRequest` utilizes a storage layout that is not optimized for gas efficiency. By adjusting the size of `unlockTime` to `uint32`, the layout can be condensed to consume 2 slots rather than 3.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes. Safety / low-complexity / minimal changes are prioritized over small gas improvements.*

## N-05 Unnecessary Use of `SafeERC20` Library Increases Gas Costs

Given that the contract is designed to work with a specific token that already reverts on failed transfers, the inclusion of the SafeERC20 library is redundant. This unnecessary use adds to both the deployment and usage gas costs.

Consider removing the `SafeERC20` library to streamline the contract and reduce associated gas costs.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes. Safety / low-complexity / minimal changes are prioritized over small gas improvements.*

## N-06 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

The file `ThrottleWallet.sol` has the `solidity ^0.8.19` floating pragma directive.

Consider using a fixed pragma version, such as `solidity 0.8.19`.

***Update:*** *Resolved in pull request #9 at commit 9b3136f.*

## N-07 Constants Not Using `UPPER_CASE` Format

Throughout the codebase, there are constants not using `UPPER_CASE` format. For instance:

- The `throttledToken` constant declared on line 39 in `ThrottleWallet.sol`
- The `throttlePeriod` constant declared on line 40 in `ThrottleWallet.sol`
- The `amountPerPeriod` constant declared on line 41 in `ThrottleWallet.sol`
- The `timelockDuration` constant declared on line 42 in `ThrottleWallet.sol`

According to the Solidity Style Guide, constants should be named with all capital letters with underscores separating words. For better readability, consider following this convention.

***Update:*** *Acknowledged, not resolved. The Reserve Protocol team stated:*

> *Not enough of a concern to make the changes.*

## N-08 Missing Events in Constructor Impedes Off-Chain State Reconstruction

To accurately reconstruct the contract's state from events off-chain, it is important to emit events that record all state changes. Consider emitting both `UserChanged` and `AdminChanged` events after the initial assignment in the constructor.

***Update:*** *Resolved in pull request #9 at commit 3d153e4.*

## N-09 Redundant Return Value in `initiateWithdrawal` Method

The `initiateWithdrawal` method returns a nonce, yet its intended use is unclear since this value is not described to be used on-chain and is not currently used in tests.

Consider omitting this return value unless a specific on-chain use case is identified.

***Update:*** *Acknowledged, not resolved. Reserve Protocol team stated:*

> *Once the user role is updated to be a smart contract with its own distribution logic, this return value will be useful.*

## N-10 Inaccurate `revert` Message

The error message "must be less than max" does not accurately reflect the implemented comparison logic. Consider updating the message to "must be less than or equal to max".

***Update:*** *Resolved in pull request #9 at commit b0c8fba.*

## N-11 Variable Naming Improvements for Clarity

Consider updating these names for improved clarity:

1. `user` to `beneficiary`, for specificity
2. `pendingWithdrawals` to `withdrawalRequests`, as not all requests are pending
3. `lastWithdrawalAt` to `lastWithdrawalInitiatedAt`, since the stored time is for initiation, not withdrawal
4. `throttledToken` to `TOKEN`, since there are no other tokens to distinguish from

5. `lastRemainingLimit` to `previousRemainingThrottleLimit` to reduce ambiguity

***Update:*** *Partially resolved in pull request #9 at commit 5d3f9f6. Reserve Protocol team stated:*

> *Only felt the need to update one name (`pendingWithdrawals` to `withdrawalRequests`).*

## N-12 Inappropriate `require` Check

The `completeWithdrawal` method allows a user to complete a previously initiated withdrawal. It checks whether `withdrawal.amount` is different from zero. However, since there should be no scenario where this value is zero, this check has an invariant character, for which a `require` statement should not be used.

Consider omitting the `require` check, or using an `assert` statement instead.

***Update:*** *Resolved in pull request #9 at commit 58b098f.*

# Conclusion

During our engagement, we found four medium-severity issues, as well as several low-severity issues and notes. We found the Reserve Protocol team to be responsive to our questions and open to re-factoring parts of the codebase to address the identified issues.