Flood token-flow

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
MiloTruck, Lead Security Researcher

December 23, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Flood's mission is to solve these routing problems and distribute the solutions in a decentralized and verifiable way. Flood's Optimal Routing algorithm optimizes across all available sources of on-chain liquidity, and it is mathematically proven to deliver the best outcome every time.

From Dec 19th to Dec 21st the Cantina team conducted a review of flood-token-flow on commit hash 813a1da5. The team identified a total of **7** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 1
- Low Risk: 3
- Gas Optimizations: 0
- Informational: 3

3 Findings

3.1 Medium Risk

3.1.1 `moveOut()` can be exploited using reentrancy for tokens with callbacks

Severity: Medium Risk

Context: TokenFlow.sol#L54-L59

Description: `TokenFlow.moveOut()` calculates the amount of tokens transferred to the `payer` by taking the difference between the payer's balance before and after the transfer:

```
function moveOut(address token, uint128 amount) external requireScope(INTERNAL_SCOPE) {
    uint balanceBefore = token.balanceOf(payer);
    token.safeTransferFrom(msg.sender, payer, amount);
    uint received = token.balanceOf(payer) - balanceBefore;
    TransientNetflows.add(token, int256(uint256(received)));
}
```

However, while this pattern works for fee-on-transfer tokens, it can be exploited through reentrancy if `token.transferFrom()` gives external callbacks to other addresses. For example, ERC-777 gives an external callback to the token sender and receiver. If `token` is an ERC-777 token, `moveOut()` can be exploited as such:

- Attacker calls `moveOut()` with `amount = 100`:
  - `balanceBefore = 0`.
  - `transferFrom()` grants a callback to the attacker (ie. the token sender), which he uses to call `moveOut()` with `amount = 200`:
    * `balanceBefore = 0`.
    * 200 tokens are transferred to `payer`.
    * `received = 200 - 0 = 200`.
    * `netflow = 200`.
  - 100 tokens are transferred to `payer`.
  - `received = 300 - 0 = 300`.
  - `netflow = 300 + 200 = 500`.

In the example above, even though the attacker only transferred 300 tokens, a positive netflow of 500 was added. An attacker could abuse this to transfer less tokens to the `payer` than specified in the constraints.

Recommendation: Include some form of reentrancy protection for `moveIn()` and `moveOut()`, such as a `nonReentrant` modifier. Alternatively, document that token with external callbacks are not supported, including ERC-777 tokens.

Flood: Fixed in commit e434c000 and a64b2a9e.

Cantina Managed: Verified, the recommendation has been implemented.

3.2 Low Risk

3.2.1 Redundant transient storage clearing before revert

Severity: Low Risk

Context: TokenFlow.sol#L38

Description: The `clearTransientState()` call before reverting in the `main` function is redundant, as a revert discards all the state changes made during the call, including transient storage modifications:

```
function main(Constraint[] calldata constraints, IFlowScope internalScope, bytes calldata data) external
↪  requireScope(EXTERNAL_SCOPE) {
    initTransientState(constraints, msg.sender);

    (bool ok, bytes memory err) = address(internalScope).call(abi.encodeCall(IFlowScope.enter,
↪  (SELECTOR_EXTENSION, constraints, msg.sender, data)));

    if (!ok) {
        clearTransientState(); // <-------------------
        // bubble up the error
        assembly {
            revert(add(err, 0x20), mload(err))
        }
    }

    bool netflowsPositive = TransientNetflows.arePositive();
    clearTransientState();

    if (!netflowsPositive) {
        revert BadNetflows();
    }
}
```

The only reason to actively clear transient storage is to enable future operations within the same transaction. Therefore, clearing transient storage immediately before reverting is unnecessary, as the revert itself halts the execution and guarantees that no state changes persist.

Recommendation: Remove the `clearTransientState()` call before reverting:

```
  if (!ok) {
-     clearTransientState();
      // bubble up the error
      assembly {
          revert(add(err, 0x20), mload(err))
      }
  }

  bool netflowsPositive = TransientNetflows.arePositive();
- clearTransientState();
-
  if (!netflowsPositive) {
      revert BadNetflows();
  }
+
+ clearTransientState();
```

Flood: Fixed in commit e434c000.

Cantina Managed: Verified, the recommendation was implemented.

3.2.2 MoveIn and moveOut function can be reentered

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `moveIn` and `moveOut` functions can be called by any address during the internal execution of the `main` function, rather than being restricted solely to the `internalScope` contract. In a scenario where an external call executed by the `internalScope` passes the control temporary to a malicious address or interacts with tokens that have on-transfer hooks, an attacker can exploit these callbacks to re-enter the contract and invoke `moveIn` or `moveOut` directly.

This could be abused in multiple ways:

1. To steal any possible net profit that the caller has just accrued since the final verification only checks that netflows remain nonnegative and consequently the attacker can reset a profitable token's net-flow to zero siphoning off all the user gains. (Requires an extra approval by the payer).
2. To reset all allowances given beforehand by the payer to 0 by doing a `moveOut` followed by a `moveIn` of the same amount.

Recommendation: Consider ensuring that only the trusted `internalScope` address can call the `moveIn` and `moveOut` functions.

Flood: Acknowledged.

Cantina Managed: Acknowledged.

3.2.3 Unsafe cast from `uint256` to `int256` could overflow

Severity: Low Risk

Context: TokenFlow.sol#L57-L58

Description: In `moveOut()`, `received` is cast from `uint256` to `int256` directly:

```
uint received = token.balanceOf(payer) - balanceBefore;
TransientNetflows.add(token, int256(uint256(received)));
```

However, this is unsafe - while unlikely, if `received` is more than `int256.max`, the cast to `int256` would overflow. This would result in an incorrect netflow being added. Additionally, the cast to `uint256` is redundant as `received` is already a `uint256`.

Recommendation: Consider checking that `received <= type(int256).max` to ensure the cast does not overflow:

```
- TransientNetflows.add(token, int256(uint256(received)));
+ if (received > type(int256).max) revert Overflow();
+ TransientNetflows.add(token, int256(received));
```

Flood: Fixed in commit e434c000.

Cantina Managed: Verified, the cast is now performed with Solady's `SafeCast`.

3.3 Informational

3.3.1 Unbounded gas consumption by internalScope contract

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `internalScope` contract passed as a user controlled parameter during the `main` call. This `internalScope` contract is supposed to perform multiple complex operations which in some cases might give the control of the execution temporary to external addresses (for example during the transfer of an ERC777 token). In this situation, those external addresses would be able to perform unintended calls that would be paid with the user gas (as long as the transaction gas limit is not reached).

Recommendation: Users should carefully set conservative gas limits when initiating transactions that involve potentially untrusted `internalScope` contracts. By doing so, they can limit the impact of abusive or wasteful operations and ensure that no excessive gas is consumed during external calls.

Flood: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Fee on transfer tokens are treated different in the `moveIn` and `moveOut` functions

Severity: Informational

Context: TokenFlow.sol#L65

Description: The current implementation of the `moveOut` function computes the actual received amount by measuring the `payer`'s token balance before and after the transfer, accommodating fee on transfer tokens accurately. In contrast, `moveIn` assumes the entire requested amount is transferred without verifying how much was actually received by the `to` address.

Recommendation: Consider updating the `moveIn` function to use the same approach as `moveOut` by measuring the `to`'s balance before and after the transfer.

Flood: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 Minor improvements to code/comments

Severity: Informational

Context: TokenFlow.sol#L12-L13, TokenFlow.sol#L35-L43, TransientNetflows.sol#L6

Description/Recommendation:

1. TransientNetflows.sol#L6 -- Typo: `An helper` should be `A helper`.
2. TokenFlow.sol#L12-L13 -- Visibility is missing for both constants, consider declaring them private:

```
- uint constant EXTERNAL_SCOPE = 0;
- uint constant INTERNAL_SCOPE = 1;
+ uint256 constant private EXTERNAL_SCOPE = 0;
+ uint256 constant private INTERNAL_SCOPE = 1;
```

3. TokenFlow.sol#L35-L43 -- `main()` performs a low-level call to `internalScope` when calling `enter()`. However, since `abi.encodeCall()` is used and the logic in `main()` reverts on failure, a high-level call can be used instead:

```
- (bool ok, bytes memory err) = address(internalScope).call(abi.encodeCall(IFlowScope.enter,
↪  (SELECTOR_EXTENSION, constraints, msg.sender, data)));
-
- if (!ok) {
-     clearTransientState();
-     // bubble up the error
-     assembly {
-         revert(add(err, 0x20), mload(err))
-     }
- }
+ IFlowScope(internalScope).enter(SELECTOR_EXTENSION, constraints, msg.sender, data);
```

The only difference is when using a low-level call, `main()` does not revert `internalScope` is an address without code. However, this is not desired behavior.

Flood: All issues have been fixed in commit e434c000.

Cantina Managed: Verified, all recommendations have been implemented.