FYEO

Secure Code Review of Solidity Contracts on the Flare Network

Flare Networks Ltd.

October 2022
Version 1.0

Presented by:
FYEO Inc.
PO Box 147044
Lakewood CO 80214
United States

Security Level
Public

Table of Contents

List of Figures

List of Tables

Executive Summary

Overview

Flare Networks Ltd. engaged FYEO Inc. to perform a Secure Code Review of Solidity Contracts on the Flare Network.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on September 05 - September 09, 2022, and focused on the following objectives:

- To provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the results of our tests.

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the FYEO Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

Key Findings

The following issues were identified during the testing period and should be prioritized for remediation to reduce the risk they pose:

- FYEO-FLARE-01 – Event emission is not consistent
- FYEO-FLARE-02 – Missing zero address check **(Partially Remediated)**
- FYEO-FLARE-03 – Unoptimized operation **(Remediated)**
- FYEO-FLARE-04 – Bool variable is explicitly compared to false
- FYEO-FLARE-05 – Prefer modifiers over repetitive checks
- FYEO-FLARE-06 – Public visibility is set for the function that is not called internally

Based on our review process, we conclude that the reviewed code implements the documented functionality to the extent of the reviewed code.

Scope and Rules of Engagement

The FYEO Review Team performed a Secure Code Review of Solidity Contracts on the Flare Network. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://gitlab.com/flarenetwork/flare-smart-contracts with the commit hash `7c2517f4e037e2cb62ab048b7bbc379f2cb78b36`.

The contract `GenericRewardManager.sol` was supplied with the commit hash `6dafd06160451d97bd3b9c30fa65364f8343a615` mid-review.

A re-review was performed on September 28, 2022, with the commit hash `694e199dff218f8df3d1be478367648f006a70c1`.

Files included in the code review

flare-smart-contracts/
└── contracts/
  ├── genesis/
  │ └── implementation/
  │ └── GovernanceSettings.sol
  ├── governance/
  │ └── implementation/
  │ └── GovernedBase.sol
  ├── tokenPools/
  │ └── implementation/
  │ ├── Escrow.sol
  │ └── GenericRewardManager.sol
  └── utils/
  └── implementation/
  └── ValidatorRegistry.sol

Table 1: Scope

Technical Analyses and Findings

During the Secure Code Review of Solidity Contracts on the Flare Network, we discovered:

- 3 findings with LOW severity rating.
- 3 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

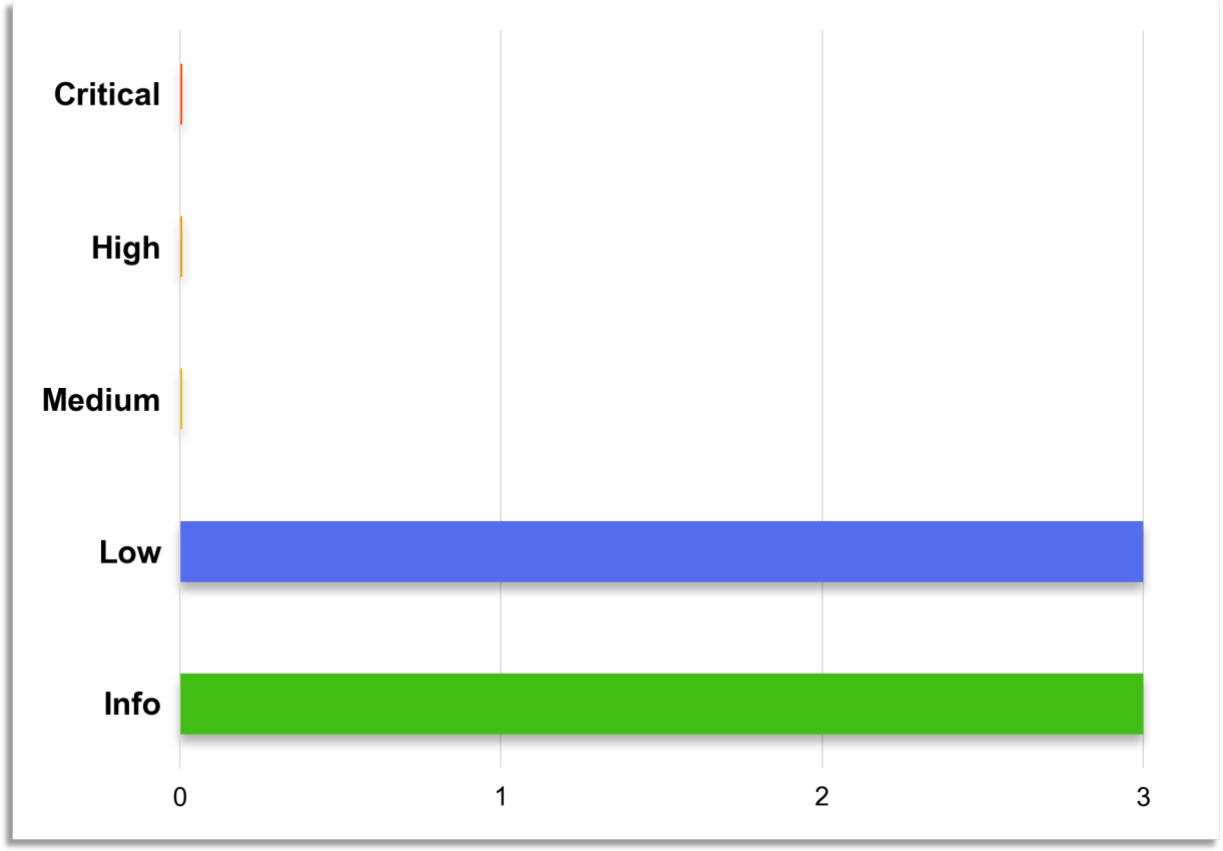


Figure 1: Findings by Severity

FINDINGS

The *Findings* section provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the findings.

Finding #	Severity	Description
FYEO-FLARE-01	**Low**	Event emission is not consistent
FYEO-FLARE-02	**Low**	Missing zero address check
FYEO-FLARE-03	**Low**	Unoptimized operation
FYEO-FLARE-04	**Informational**	Bool variable is explicitly compared to false
FYEO-FLARE-05	**Informational**	Prefer modifiers over repetitive checks
FYEO-FLARE-06	**Informational**	Public visibility is set for the function that is not called internally

Table 2: Findings Overview

TECHNICAL ANALYSIS

The source code has been manually validated to the extent that the state of the repository allowed. The validation includes confirming that the code correctly implements the intended functionality.

TECHNICAL FINDINGS

GENERAL OBSERVATIONS

The scope of this audit included `Escrow.sol`, `ValidatorRegistry.sol`, and `RewardManager.sol` as well as their dependency contracts.

`Escrow.sol` is used to lock a user's Wei for a period of time with the ability to claim 2.37% of the locked amount every 30 days. The contract is planned to be used by the Flare team with the primary purpose of balancing staking capability with other users.

`ValidatorRegistry.sol` provides addresses of registered validators with the ability to add new or remove existing validators.

`RewardManager.sol` is used to distribute Wei according to inflation.

Contracts allow issuing the ability to withdraw for executors, and this code is properly secured.

During the assessment, only a few low-severity issues were found regarding optimization, additional checks, and code consistency. No security issues were found.

Following re-review, the FYEO team has found that the optimization recommendation was headed, and some zero-address checks were implemented.

EVENT EMISSION IS NOT CONSISTENT

Finding ID: FYEO-FLARE-01
Severity: **Low**
Status: **Open**

Description

The event is emitted only when `claimStartTs` is set for the first time in the constructor, but not when this value is changed in the setter.

Proof of Issue

File name: contracts/tokenPools/implementation/Escrow.sol

Line number: 210

```
    function setClaimingStartTs(uint256 _claimStartTs) public onlyGovernance {
        require(claimStartTs > block.timestamp, "Already started");
        require(_claimStartTs >= block.timestamp && _claimStartTs <=
latestClaimStartTs,
            "Wrong start timestamp");
        claimStartTs = _claimStartTs;
    }
```

Line number: 77

```
    constructor(
        address _governance,
        address _addressUpdater,
        uint256 _latestClaimStartTs
    )
        Governed(_governance)
        AddressUpdatable(_addressUpdater)
    {
        require(_latestClaimStartTs >= block.timestamp, "In the past");
        latestClaimStartTs = _latestClaimStartTs;
        claimStartTs = _latestClaimStartTs;
        emit ClaimStart(_latestClaimStartTs);
    }
```

Severity and Impact Summary

Event emission logic is not consistent and may cause errors for off-chain handlers.

Recommendation

It is recommended to emit `ClaimStart` event in `setClaimingStartTs` function.

MISSING ZERO ADDRESS CHECK

Finding ID: FYEO-FLARE-02
Severity: **Low**
Status: **Partially Remediated**

Description

A zero address Validation is a sanity check.

Proof of Issue

File name: contracts/governance/implementation/GovernedBase.sol

Line number: 111

```
initialGovernance = _initialGovernance;
```

File name: contracts/genesis/implementation/GovernanceSettings.sol

Line number: 64

```
governanceAddress = _governanceAddress;
```

File name: contracts/tokenPools/implementation/GenericRewardManager.sol

Line number: 242

```
newRewardManager = _newRewardManager;
```

Line number: 248

```
rewardDistributor = _rewardDistributor;
```

Line number: 391

```
(bool success, ) = _recipient.call{value: _rewardAmount}("");
```

Line number: 406

```
wNat.depositTo{value: _rewardAmount}(_recipient);
```

Severity and Impact Summary

Zero addresses may lead to errors or blocking functionality and may also cause a loss of funds.

Recommendation

It is recommended to add a zero-address check to all mentioned cases.

Remediation

Following the audit, zero address checks were implemented for `_newRewardManager` and `_recipient`.

The Flare team has indicated that the initialize function in `GovernedBase.sol` is called only in two contracts which were out of the scope of this review:

- by the constructor in `Governed.sol` where a check for the zero-address is performed
- and by the `initialiseFixedAddress()` function in `GovernedAtGenesis.sol` where direct calls to initialize are forbidden by overload

Since no contract inherits `GovernedBase.sol` directly, there is no danger of calling a zero-address

UNOPTIMIZED OPERATION

Finding ID: FYEO-FLARE-03
Severity: **Low**
Status: **Remediated**

Description

The contract changes the storage variable inside the loop

Proof of Issue

File name: contracts/tokenPools/implementation/GenericRewardManager.sol

Line number: 228

```
        for(uint256 i = 0; i < len; i++) {
            beneficiaryRewardAmount[_addresses[i]] += _rewardAmounts[i];
            totalAwardedWei += _rewardAmounts[i];
        }
```

Severity and Impact Summary

Operations with storage are among the most expensive in Solidity. Modifying storage in a loop increases the cost of transactions significantly.

Recommendation

It is recommended to move the storage variable outside of the loop:

```
        uint256 totalAmount = 0;
        for(uint256 i = 0; i < len; i++) {
            beneficiaryRewardAmount[_addresses[i]] += _rewardAmounts[i];
            totalAmount += _rewardAmounts[i];
        }
        totalAwardedWei += totalAmount;
```

BOOL VARIABLE IS EXPLICITLY COMPARED TO FALSE

Finding ID: FYEO-FLARE-04
Severity: **Informational**
Status: **Open**

Description

Bool variable doesn't need to be compared with 'true' or 'false'.

Proof of Issue

File name: contracts/governance/implementation/GovernedBase.sol

Line number:

```
require(initialised == false, "initialised != false");
```

Severity and Impact Summary

Clean code recommendation, no impact on security.

Recommendation

It is recommended to remove explicit comparison:

```
require(!initialised, "initialised != false");
```

PREFER MODIFIERS OVER REPETITIVE CHECKS

Finding ID: FYEO-FLARE-05
Severity: **Informational**
Status: **Open**

Description

The check implemented in the modifier is duplicated in the code.

Proof of Issue

File name: contracts/governance/implementation/GovernedBase.sol

Line number: 97

```
    function switchToProductionMode() external {
        _checkOnlyGovernance();
        require(!productionMode, "already in production mode");
        initialGovernance = address(0);
        productionMode = true;
        emit GovernedProductionModeEntered(address(governanceSettings));
    }
```

Line number: 50

```
    modifier onlyImmediateGovernance () {
        _checkOnlyGovernance();
        _;
    }
```

Severity and Impact Summary

Clean code recommendation, no impact on security.

Recommendation

It is recommended to apply `onlyImmediateGovernance` modifier to `switchToProductionMode` function.

PUBLIC VISIBILITY IS SET FOR THE FUNCTION THAT IS NOT CALLED INTERNALLY

Finding ID: FYEO-FLARE-06
Severity: **Informational**
Status: **Open**

Description

Public visibility is used for functions that should be accessible from other contracts, via transactions, and from the current contract. Functions that are not meant to be called internally should have external visibility.

Proof of Issue

File name: contracts/tokenPools/implementation/Escrow.sol

Line number: 206

```
function setClaimingStartTs(uint256 _claimStartTs) public
```

Severity and Impact Summary

External functions may be more GAS efficient, especially with functions that receive a lot of data in parameters.

Recommendation

It is recommended to set external visibility for functions that are not called internally.

References

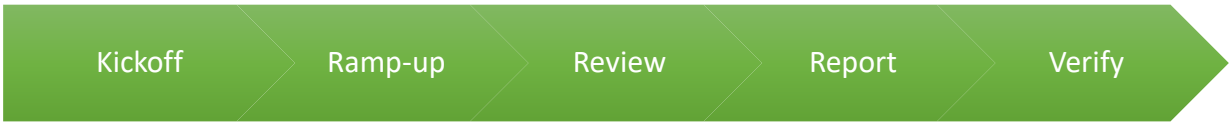
- https://docs.soliditylang.org/en/v0.7.4/contracts.html?highlight=external#visibility-and-getters

Our Process

Methodology

FYEO Inc. uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

Figure 2: Methodology Flow



Kickoff

The project is kicked off as the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

FYEO Inc. delivers a draft report that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project. We may conclude that the overall risk is low but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We report security issues identified, as well as informational findings for improvement, categorized by the following labels:

- Critical
- High
- Medium

- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

VERIFY

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes within a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

ADDITIONAL NOTE

It is important to note that, although we did our best in our analysis, no code audit or assessment is a guarantee of the absence of flaws. Our effort was constrained by resource and time limits along with the scope of the agreement.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination.

THE CLASSIFICATION OF VULNERABILITIES

Security vulnerabilities and areas for improvement are weighted into one of several categories using, but is not limited to, the criteria listed below:

Critical – vulnerability will lead to a loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probability of exploit is high

High - vulnerability has potential to lead to a loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that cannot be found in the code
- All mismatches from the stated and actual functionality

- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Txn signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - vulnerability hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-reviewed crypto functions
- Program crashes, leaves core dumps or writes sensitive data to log files

Low – vulnerability has a security impact but does not directly affect the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations