FYEO

Secure Code Review of Solidity Smart Contracts on the Flare Network

Flare Networks Ltd.

December 2022
Version 1.0

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

Security Level
Public

Table of Contents

Executive Summary 2
- Overview 2
- Key Findings 2
- Scope and Rules of Engagement 2

Technical Analyses and Findings 4
- Findings 5
- Technical Analysis 5

Technical Findings 6
- General Observations 6
- Annotation: return values marked as parameters 7
- Array filled twice with the same values 8
- Misspelling 9
- Public visibility is set for the function that is not called internally 10
- Unused constants 11
- Update function description 12
- Use enum value 13

Our Process 14
- Methodology 14
  - Kickoff 14
  - Ramp-up 14
  - Review 14
  - Code Safety 15
  - Technical Specification Matching 15
  - Reporting 15
  - Verify 16
- Additional Note 16
- The Classification of vulnerabilities 16

List of Figures

Figure 1: Findings by Severity 4

Figure 2: Methodology Flow 14

List of Tables

Table 1: Scope 3

Table 2: Findings Overview 5

EXECUTIVE SUMMARY

OVERVIEW

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

The assessment was conducted remotely by the FYEO Security Team. Testing took place on December 16 - December 23, 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. These were remediated during the audit process:

- FYEO-FLARE-01 – Annotation: return values marked as parameters
- FYEO-FLARE-02 – Array filled twice with the same values
- FYEO-FLARE-03 – Misspelling
- FYEO-FLARE-04 – Public visibility is set for the function that is not called internally
- FYEO-FLARE-05 – Unused constants
- FYEO-FLARE-06 – Update function description
- FYEO-FLARE-07 – Use enum value

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

SCOPE AND RULES OF ENGAGEMENT

The FYEO Review Team performed a Secure Code Review of Solidity Smart 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/btblock-cybersec/multichain/flare/flare-smart-contracts** with the commit hash 177d841d8137f702f59b5690e4452e9e835ab02d.

Files included in the code review

```
flare-smart-contracts/
└── contracts/
    ├── assetRegistry/
    │   └── implementation/
    │       ├── FlareAssetRegistry.sol
    │       └── WNatRegistryProvider.sol
    ├── ftso/
    │   └── implementation/
    │       ├── FtsoManager.sol
    │       └── Ftso.sol
    ├── personalDelegation/
    │   └── implementation/
    │       ├── DelegationAccountClonable.sol
    │       └── DelegationAccountManager.sol
    ├── tokenPools/
    │   └── implementation/
    │       └── FtsoRewardManager.sol
    ├── utils/
    │     └── implementation/
    │       ├── FtsoRegistry.sol
    │       ├── FtsoRegistryProxy.sol
    │       └── ProxyGoverned.sol
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

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

- 7 findings with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

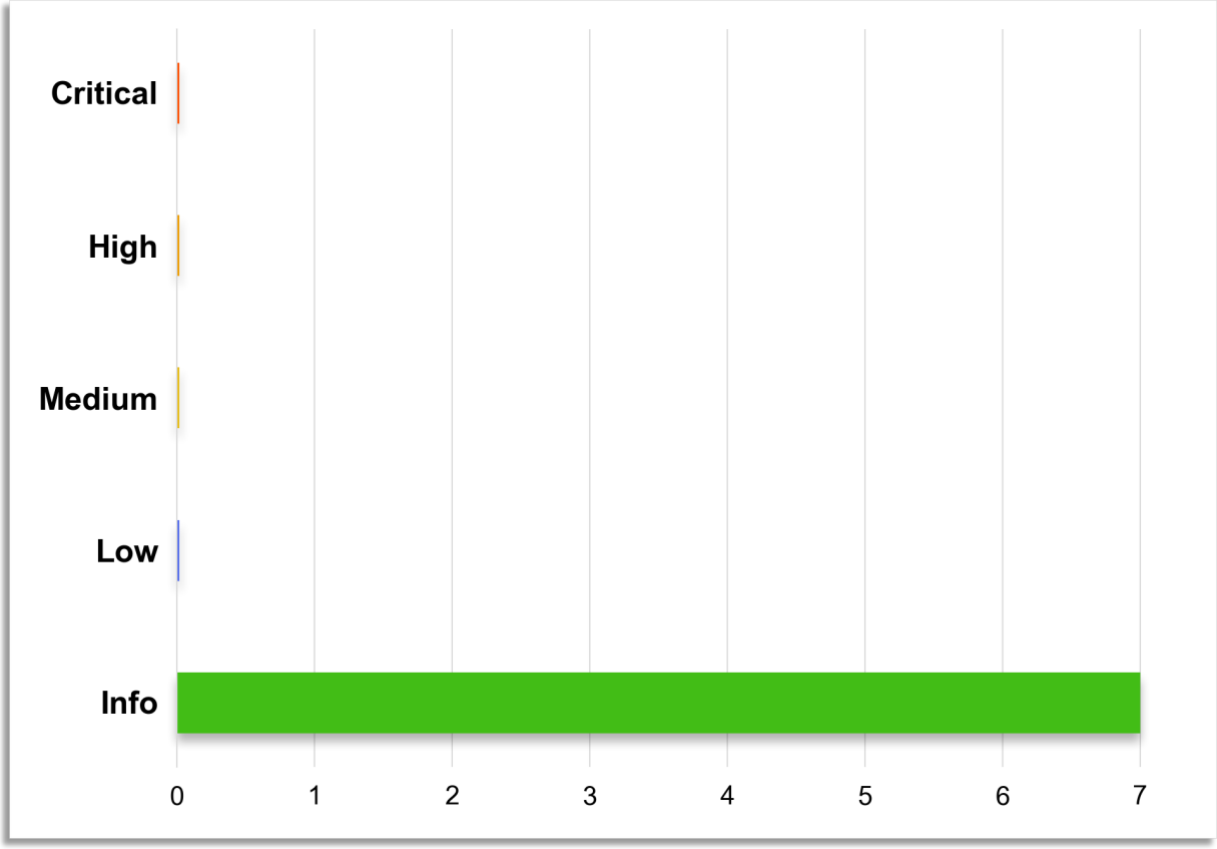


Figure 1: Findings by Severity

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	**Informational**	Annotation: return values marked as parameters
FYEO-FLARE-02	**Informational**	Array filled twice with the same values
FYEO-FLARE-03	**Informational**	Misspelling
FYEO-FLARE-04	**Informational**	Public visibility is set for the function that is not called internally
FYEO-FLARE-05	**Informational**	Unused constants
FYEO-FLARE-06	**Informational**	Update function description
FYEO-FLARE-07	**Informational**	Use enum value

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 contracts audited, provide a clever solution to reward distributions, thoroughly aggregating rewards by providers and epochs. The level of security of the reward distribution also depends on the usage of the contracts, i.e. the integrity and validity of inputs passed.

We have tried to identify the assumptions made, presented in the findings below. After discussing some of these findings with the team, it is clear that correct usage of the contracts is also guaranteed. An example of this also includes the usage of unchecked arithmetic, which is safe in its usage.

We would also like to note that lower-level code such as `createClone` was not thoroughly investigated.

Finally, the team chose to use its own pattern when it comes to updating contracts, using `AddressUpdatable`. It is not clear why this approach was chosen over simple `setters` as:

- Contracts' `_updateContractAdresses` do not perform any checks on the updated addresses
- Address can only be updated in groups (visible clearly in `DelegationAccountManager.sol` where adding a rewards manager requires the updating of all addresses). This approach seems to allow room for error.

Nevertheless, the structures and methods used throughout the project are well thought out and secure.

In regard to communication, the development team was very helpful in answering all our queries in a prompt manner.

ANNOTATION: RETURN VALUES MARKED AS PARAMETERS

Finding ID: FYEO-FLARE-01
Severity: **Informational**
Status: **Remediated**

Description

Some functions use incorrect annotation for return values

Proof of Issue

File name: contracts/assetRegistry/implementation/FlareAssetRegistry.sol

Line number: 210

```
    /**
     * @notice Returns a generic asset attribute value.
     * @param _token The token's address
     * @param _nameHash attributes name's hash
     * @param _defined true if the attribute is defined for this token
     * @param _value attribute value, may have to be cast into some other type
     */
    function getAttribute(address _token, bytes32 _nameHash)
        public view override
        returns (bool _defined, bytes32 _value)
```

File name: contracts/assetRegistry/implementation/WNatRegistryProvider.sol

Line number: 45

```
    /**
     * @notice Returns a generic asset attribute value.
     * @param _token The token's address
     * @param _nameHash attributes name's hash
     * @param _defined true if the attribute is defined for this token
     * @param _value attribute value, may have to be cast into some other type
     */
    function getAttribute(address _token, bytes32 _nameHash)
        external view override
        returns (bool _defined, bytes32 _value)
```

Severity and Impact Summary

The outdated description may cause the generation of invalid documentation

Recommendation

It is recommended to use `@return` instead of `@param` to describe return values.

ARRAY FILLED TWICE WITH THE SAME VALUES

Finding ID: FYEO-FLARE-02
Severity: **Informational**
Status: **Remediated**

Description

The same elements of `_ftsos` array are readded.

Proof of Issue

File name: contracts/utils/implementation/FtsoRegistry.sol

Line number: 232

```
(_supportedIndices, _ftsos) = _getSupportedIndicesAndFtsos();
```

Line number: 237

```
_ftsos[len] = ftsoHistory[_supportedIndices[len]][0];
```

Severity and Impact Summary

Meaningless operation increases GAS costs.

Recommendation

It is recommended to remove the assignment to `_ftsos` in the loop

MISSPELLING

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

Description

There are spelling errors in the code and commentaries.

Proof of Issue

```
cummulative - cumulative
attibute - attribute
undelagated - undelegated
immediatelly - immediately
setings - settings
beneficary - beneficiary
othervise - otherwise
occures - occurs
recipent - recipient
```

Severity and Impact Summary

There is no impact on security

Recommendation

It is recommended to fix misspelling

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

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

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/ftso/implementation/Ftso.sol

Line number: 698

```
function getEpochId(uint256 _timestamp) public view override returns (uint256)
```

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

UNUSED CONSTANTS

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

Description

Some of the constants are not used in the contract

Proof of Issue

File name: contracts/ftso/implementation/FtsoManager.sol

Line number: 51

```
    string internal constant ERR_GOV_PARAMS_NOT_INIT_FOR_FTSOS = "Gov. params
not initialized";
```

Line number: 54

```
    string internal constant ERR_ALREADY_ADDED = "Already added";
```

Severity and Impact Summary

There is no impact on security

Recommendation

It is recommended to delete unused constants

UPDATE FUNCTION DESCRIPTION

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

Description

After adding `_explicitDelegation` parameter, the description of `_claimReward` is outdated

Proof of Issue

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

Line number: 948

```
    /**
     * @notice Claims `_rewardAmounts` for `_dataProviders`.
     * @dev Internal function that takes care of reward bookkeeping
     * @param _recipient            address representing the recipient of the
reward
     * @param _rewardEpoch          reward epoch number
     * @param _rewardState          object holding reward state
     * @return Returns the total reward amount.
     */
    function _claimReward(
        address _rewardOwner,
        address _recipient,
        uint256 _rewardEpoch,
        RewardState memory _rewardState,
        bool _explicitDelegation
    )
```

Severity and Impact Summary

The outdated description may cause the generation of invalid documentation

Recommendation

It is recommended to add the description for `_explicitDelegation` parameter.

USE ENUM VALUE

Finding ID: FYEO-FLARE-07
Severity: **Informational**
Status: **Remediated**

Description

There is a comparison to a raw number, even so, an enum structure is exist

Proof of Issue

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

Line number: 922

```
require(wNat.delegationModeOf(_rewardOwner) == 2, "explicit delegation only");
// DelegationMode.AMOUNT
```

Severity and Impact Summary

In the case of enum modifications, the code is prone to errors.

Recommendation

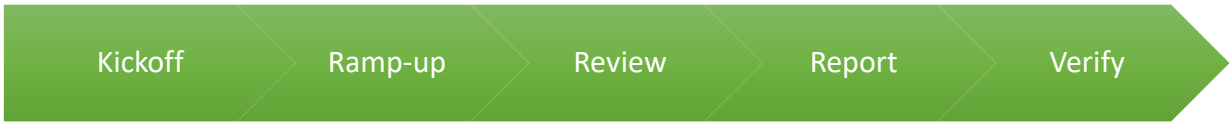
It is recommended to use the enum value

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