

F Y E O

Secure Code Review of Golang Validator on the Flare Network

Flare Networks Ltd.

February 2023

Version 1.0

Presented by:

FYEO Inc.

PO Box 147044

Lakewood CO 80214

United States

Security Level

Public

TABLE OF CONTENTS

- Executive Summary2
- Overview2
- Key Findings2
- Scope and Rules of Engagement2
- Technical Analyses and Findings4
- Findings5
- Technical Analysis5
- Technical Findings6
- General Observations6
- Call is not replaced to DaemonCall in GetAttestation function8
- Our Process9
- Methodology9
- Kickoff9
- Ramp-up9
- Review9
- Code Safety10
- Technical Specification Matching10
- Reporting10
- Verify11
- Additional Note11
- The Classification of vulnerabilities11

LIST OF FIGURES

- Figure 1: Findings by Severity4
- Figure 2: Methodology Flow9

LIST OF TABLES

- Table 1: Scope3
- Table 2: Findings Overview5

EXECUTIVE SUMMARY

OVERVIEW

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

The assessment was conducted remotely by the FYEO Security Team. Testing took place on January 30 - February 03, 2023, 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 should be prioritized for remediation to reduce the risk they pose:

- FYEO-FL-01 – Call is not replaced to DaemonCall in GetAttestation function

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 Golang Validator 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 public repository at <https://github.com/flare-foundation/go-flare> with the commit hash 9351b98bee5af7481996cd6b6bbf6f3004b87a27.

A re-review was conducted using the commit hash 033b1f5ba188d5712ca4185a82cbbbc06f9d6a97

Files included in the code review

```
go-flare/  
└─ coreth/
```

Files included in the code review

```
├── core/
│   ├── governance_settings.go
│   ├── state_transition.go
│   └── state_connector.go
└── avalanche/
    ├── utils/
    │   └── constants/
    │       └── validator_config.go
    ├── hashing/
    │   └── hashing_test.go
    ├── version/
    │   └── constants.go
    └── vms/
        ├── platformvm/
        │   └── txs/
        │       └── executor/
        │           ├── proposal_tx_executor.go
        │           └── reward_validator_test.go
```

Table 1: Scope

TECHNICAL ANALYSES AND FINDINGS

During the Secure Code Review of Golang Validator on the Flare Network, we discovered:

- 1 finding 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-FL-01	Informational	Call is not replaced to DaemonCall in GetAttestation function

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.

CALL IS NOT REPLACED TO DAEMONCALL IN GETATTESTATION FUNCTION

Finding ID: FYEO-FL-01

Severity: **Informational**

Status: **Remediated**

Description

In State Connector and Governance Setting EVM calls, `st.evm.Call` is replaced by `st.evm.DaemonCall`. These two calls are equal, with the exception that `DaemonCall` does not permit the `value` field to be passed to the call. However, there is still an EVM call in the `GetAttestation` function that has not changed.

Proof of Issue

File name: state_connector.go

Line number: 178-181

```
func (st *StateTransition) GetAttestation(attestor common.Address,
instructions []byte) (string, error) {
    merkleRootHash, _, err := st.evm.Call(vm.AccountRef(attestor), st.to(),
instructions, params.TxGas, big.NewInt(0))
    return hex.EncodeToString(merkleRootHash), err
}
```

Severity and Impact Summary

Inconsistent code.

Recommendation

We recommend changing `Call` to `DaemonCal` for consistency.

Flare Response

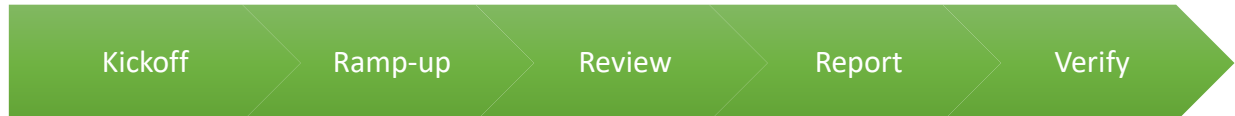
The team will leave it as-is for now and then leave a note for changing it in a future release since we've already prepared the release and tested it on Coston2.

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