FYEO

Secure Code Review of the Flare Network's Validator Codebase

Flare Networks Ltd.

September 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 the Flare Network.

The assessment was conducted remotely by the FYEO Security Team. Testing took place on July 25 - August 11, 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 were prioritized for remediation to reduce the risk they pose:

- FYEO-FLARE-01 – Anyone can call add_validator on permitted NodeIDs
- FYEO-FLARE-02 – Deferring unsafe method "Close" on type "*os.File"
- FYEO-FLARE-03 – Missing checks for lengths of "latestConfigName" and "peneltimateConfigName"
- FYEO-FLARE-04 – Nodes can have different permitted validator sets
- FYEO-FLARE-05 – State Connector code is not complete

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 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://github.com/sprwn/go-flare with the commit hash 5e45574e39303cb9137b3a21f776004b295bbdf3.

The scope of the review included all changes made to the repository since commit hash 0ed919efba40d5d7cb3e63a3f330dfcf4a1bf4c7 and up to commit hash 5e45574e39303cb9137b3a21f776004b295bbdf3.

A re-review was performed on September 7, 2022, with the commit hash b596931f85367ca5db272e962b2fded14a9733f1

TECHNICAL ANALYSES AND FINDINGS

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

- 1 finding with HIGH severity rating.
- 3 findings with LOW severity rating.
- 1 finding with INFORMATIONAL severity rating.

The following chart displays the findings by severity.

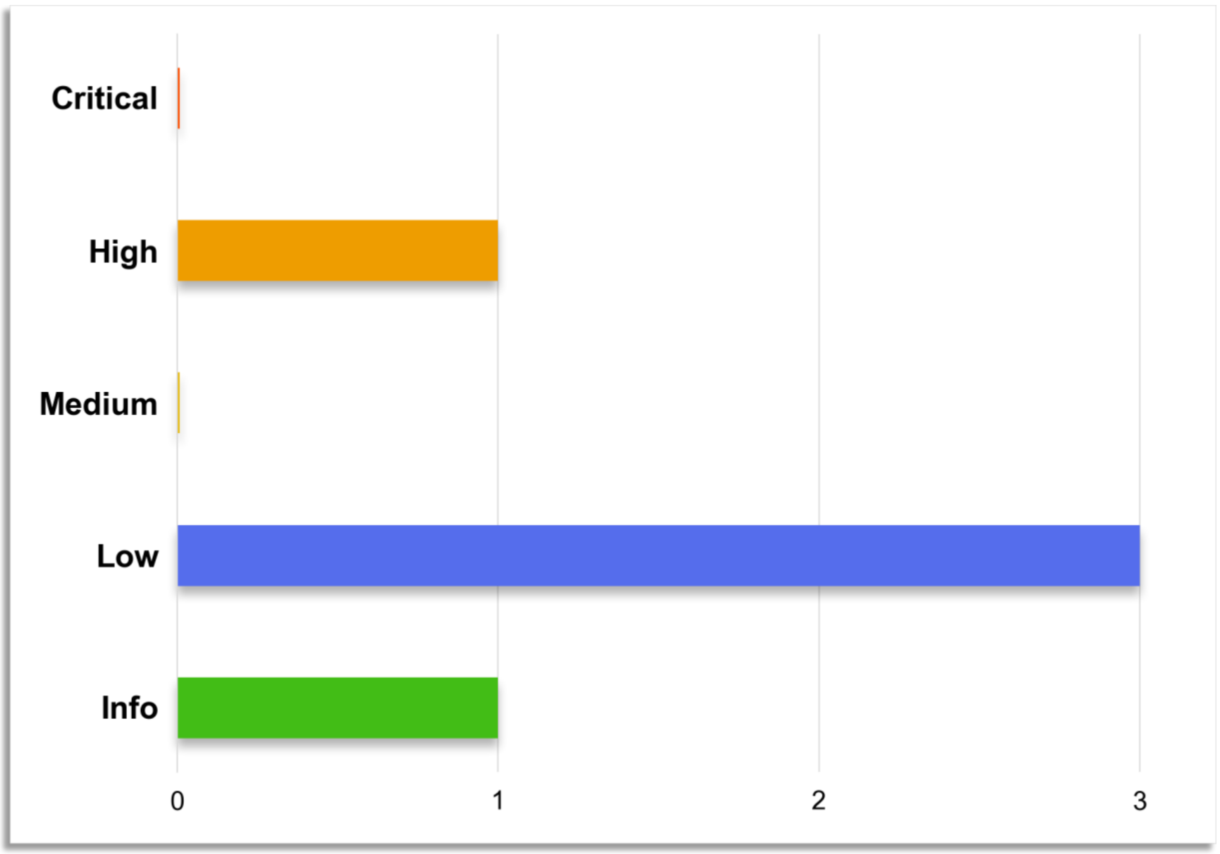


Figure 1: Findings by Severity

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-FLARE-01	**High**	Anyone can call add_validator on permitted NodeIDs
FYEO-FLARE-02	**Low**	Deferring unsafe method "Close" on type "*os.File"
FYEO-FLARE-03	**Low**	Missing checks for lengths of "latestConfigName" and "peneltimateConfigName"
FYEO-FLARE-04	**Low**	Nodes can have different permitted validator sets
FYEO-FLARE-05	**Informational**	State Connector code is not complete

Table 1: 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.

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

Technical Findings

General Observations

This audit is focused on the validator changeover mechanism, along with several other details in the Golang codebase of the Flare Network: https://flare.xyz.

The core premise of the validator changeover mechanism on the Flare Network is to reuse the traditional staking mechanism of Avalanche with an additional restriction on the transaction that permits new stakers to join the network. Overall, the code was well written and the Flare development team was very communicative, quickly providing responses to the auditing team.

ANYONE CAN CALL ADD_VALIDATOR ON PERMITTED NODEIDS

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

Description

The execute function in `add_validator_tx.go` will pull a list of permitted validators from local cache and then use the `(networkID, nodeID, weight, duration)` to search for a match in the cache. When a new validator config is deployed to the network, an attacker can initiate a new staking transaction that set valid `nodeID`s as active on the network. These `nodeID`s would be valid and can't be updated. The actual owners may not yet have spun up nodes on the network, which if set as new staked validators could cause a reduction in the percentage of live nodes on the network at large.

Proof of Issue

File name: add_validator_tx.go
Line number: 357-362

```
cacheSearchHash := hashing.ToValidatorConfigHash(
        networkIDStr,
        tx.Validator.NodeID.String(),
        strconv.FormatUint(tx.Validator.Wght, 10),
        strconv.FormatFloat(duration.Seconds(), 'f', 0, 64),
)
```

File name: hashing.go
Line number: 106-113

```
func ToValidatorConfigHash(networkID string, nodeID string, weight string,
duration string) string {
    salt := "flare" + networkID + "-"
    nodeIDHash := sha256.Sum256([]byte(salt + nodeID))
    nodeWeightHash := sha256.Sum256([]byte(salt + weight))
    nodeDurationHash := sha256.Sum256([]byte(salt + duration))
    validatorConfigHash := sha256.Sum256(append(append(nodeIDHash[:],
nodeWeightHash[:]...)[:], nodeDurationHash[:]...))
    return hex.EncodeToString(validatorConfigHash[:])
}
```

Link: https://docs.avax.network/apis/avalanchego/apis/p-chain#platformaddvalidator

```
platform.addValidator(
    {
        nodeID: string,
        startTime: int,
        endTime: int,
        stakeAmount: int,
        rewardAddress: string,
        delegationFeeRate: float,
        from: []string, // optional
        changeAddr: string, // optional
        username: string,
        password: string
    }
) ->
{
    txID: string,
    changeAddr: string
}
```

Severity and Impact Summary

This could cause a reduction in the percentage of live nodes on the network at large.

Recommendation

We recommend adding a check that the public key used for signing the staking transactions is included in the cache hashes, so that only the `nodeID` owners are ever permitted to choose the point in time in which their validators become active on the network.

DEFERRING UNSAFE METHOD "CLOSE" ON TYPE "*OS.FILE"

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

Description

`f.Close()` could return error which is not checked when calling `defer`.

Proof of Issue

File name: add_validator_tx.go
Line number: 202-206

```
if err != nil {
   f.Close()
   return time.Time{}, []string{}, fmt.Errorf("failed to open validator
config: %w", err)
}
defer f.Close()
```

Severity and Impact Summary

This could lead to a crash if `Close()` returns an error.

Recommendation

We recommend using this code snippet instead

```
defer func() {
    if err := f.Close(); err != nil {
         logger.Printf("Error closing file: %s\n", err)
    }
}()
```

References

- https://www.joeshaw.org/dont-defer-close-on-writable-files/
- https://golang.org/pkg/os/#File.Close

Missing checks for lengths of "latestConfigName" and "peneltimateConfigName"

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

Description

The length of `latestConfigName` or `penultimateConfigName` could be less than 4. This will lead to underflow and panic.

File name: add_validator_tx.go
Line number: 172-179

```
if latestConfigName[len(latestConfigName)-4:] != ".csv" ||
penultimateConfigName[len(penultimateConfigName)-4:] != ".csv" {
   return time.Time{}, []string{}, fmt.Errorf("invalid file extension for
validator config")
}
latestConfigExpiresAtInt, err :=
strconv.Atoi(latestConfigName[:len(latestConfigName)-4])
if err != nil {
   return time.Time{}, []string{}, fmt.Errorf("failed to decode
latestConfigExpiresAtInt: %w", err)
}
penultimateConfigExpiresAtInt, err :=
strconv.Atoi(penultimateConfigName[:len(penultimateConfigName)-4])
```

Severity and Impact Summary

This could lead to a panic if length of `latestConfigName` or `penultimateConfigName` is less than 4

Recommendation

We recommend adding a check if `len(latestConfigName)>4` and `len(penultimateConfigName)>4`.

NODES CAN HAVE DIFFERENT PERMITTED VALIDATOR SETS

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

Description

The core premise of the validator changeover mechanism on Flare is that the system reuses the traditional staking mechanism of Avalanche, except that it adds an additional restriction on the transaction that permits new stakers to join the network.

The `add_validator_tx` functionality reads a permitted validator set from a csv file and loads to the cache, then checks that a new staker belongs to the updated validator cache. All validators will produce the same result (reaching consensus) only if they read from the same csv. At present, there is no way to guarantee that they are reading the same file (same validator set). An attacker just needs to compromise that file instead of the private key. If enough validators are attacked, then this can bring down consensus.

Severity and Impact Summary

This can affect consensus safety.

Recommendation

We recommend committing the hash of the validator set (or the set itself if not too big) to the chain first, so that all validators would first agree on the same validator set, then there should be an additional check in `add_validator` that compares the committed hash with the hash of the local file.

Another recommendation is to drive the permitted validators from participants in Flare's time series oracle (FTSO): https://flaremetrics.io which is aligned with Flare's future plan for the validator changeover mechanism.

Flare Response

Our plan is to issue a hard fork each month that adds a new permitted validator config which is transparently derived from the recent FTSO performance such that anyone can verify the configs off-chain - we think this creates the most stable way to operate the system, and it enables off-chain elements in the validator config derivation such as collusion detection in the FTSO.

In the near term we'll also be including FTSO performance information from our canary network Songbird into the validator config list for Flare mainnet - this is also why we opted for this method so that we can include this data from a different chain

State Connector code is not complete

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

Description

The implementation of the Flare state connector is not complete. At present, if default and local attestors reach different majority decisions or local attestors don’t reach majority, it follows the default attestors’ decision.

Proof of Issue

File name: state_connector.go
Line number: 224-232

```
if len(localAttestors) > 0 {
   localAttestationVotes :=
CountAttestations(st.GetAttestations(localAttestors, instructions))
   if finalityReached && defaultAttestationVotes.majorityDecision !=
localAttestationVotes.majorityDecision {
      // Different actions are available to take here depending on the node
operator's preference.
      // 1) Create a backup of the DB
      // 2) Push an alert to the node operator
      // 3) Fork the node now from the default path -> return err
   }
}
```

Severity and Impact Summary

Not completed code.

Recommendation

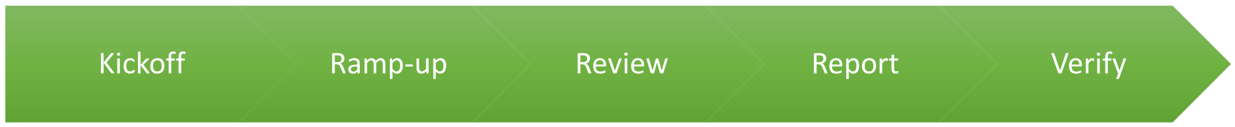
We recommend completing the implementation as described in https://docs.flare.network/tech/state-connector/

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