Flare

Hybrid Band Reward & Distribution Update Smart Contract Audit

coinspect

Flare

Flare

Smart Contract Audit

V230220

Prepared for Flare • February 2023

1. Executive Summary

2. Assessment and Scope

Fixes review

3. Summary of Findings

4. Detailed Findings

FLR-32 Attackers can make executors waste money

FLR-33 Insecure random in finalizePriceEpoch random

FLR-34 Denial of service by preventing good random values

5. Disclaimer

1. Executive Summary

In January 2023, Flare engaged Coinspect to perform a source code review of changes to the **Flare** platform. These included modification to the reward band system for price providers, scaffolding for a new random generation system and optimization of the rewards distribution. The objective of the project was to evaluate the security of the smart contracts involved in these changes.

The following issues were identified during the initial assessment:

	High Risk	Medium Risk	Low Risk
Open	0	0	0
Fixed	0	0	2
Reported	0	0	3

Coinspect identified three low-risk issues. Although one of them (**FLR-32**) can involve monetary losses we decided to reduce its risk based on our analysis of the probabilities (Likelihood).

A brief summary of findings:

- Issue **FLR-32** describes how an attacker would put funds of an executor at risk (possible business impact but with very low likelihood).
- Issue **FLR-33** describes how one particular part of the system was not ready for new random generation.
- Issue **FLR-34** describes how smart contacts' availability could be impacted if the service does not give a good random number.

After the audit, Flare fixed both **FLR-33** and **FLR-34**. **FLR-32** was acknowledged but Flare stated it is the responsibility of the executors to avoid being impacted by it.

2. Assessment and Scope

The audit started on **January 23** and was conducted on the **flare_distribution_audit** branch of the git repository at https://gitlab.com/flarenetwork/flare-smart-contracts as of commit **`eb44f6f10da003cb11692cf0c0b953feedb51e10`** of **January 23**.

The audited files have the following sha256sum hash:

```
0a28649a98dd50c7eb9a347c6338a7c327883a544a4ddfdc713fb2f290e779b1  addressUpdater/implementation/AddressUpdater.sol
da08716ba1759e15755a890231989b7ed39b6de4eba51dba4e9b735c590b72d3  ftso/implementation/Ftso.sol
b7a9b1367ff17cc4bb42add954e084ff8f0615388a4dfcc6a1981a637a9eaa1d  ftso/implementation/FtsoManager.sol
5920d663ec76ee4486d88a8469698a10770e45b0f2b2de8ff306fd535f467122  ftso/lib/FtsoEpoch.sol
1c9f26687df89ce5e8126ad4163a7d0bf09cbef54f65bb0fb1a11ae38d2cd0d9  ftso/lib/FtsoManagement.sol
a44e7e7687de98e87355d51bccc09b813870ebe4fe1cfc9fde42be8152d09b71  ftso/lib/FtsoManagerSettings.sol
a89c462c2280ee1d6dac09e19ef52562f12d30f3771818da1ec5f0cd82ec9857  genesis/implementation/DistributionTreasury.sol
9b6f37d5709788c7f540deff9089b211085d266ea4bb88ad3bed4d7bc38c6656  genesis/implementation/InitialAirdrop.sol
cf373bd09d934acbf6ac6da723d684186b5368f459229c02855fecfd2d8f6440  tokenPools/implementation/DistributionToDelegators.sol
613d0a23161aa024617b18ecac1360554a0e9f7788106900db20537705b06a5c  utils/implementation/ValidatorRegistry.sol
```

Additionally the branch **699-update-voterwhitelister** was audited, containing the file with the following hash:

```
50d9ee16f8ef312ee5babb7ae6a3f43d0cef003296c0a125e1651bf3b980a8e3  VoterWhitelister.sol
```

For both branches, the focus of the audit was the differences with respect to the `master` branch of the repository.

The `flare_distribution_audit` branch introduces a few changes:

1. The addition of the Hybrid Band Reward: a new method for calculating the `FTSO` reward which attempts to discourage collusion.
2. Preparations to add a new *random generation scheme*, but said system is not yet in place.
3. Optimizations to the distribution to delegators: updating it to reflect the recent changes to the daemonize flow and the `ClaimSetupManager`.

The new hybrid band reward system adds a weight to those price providers that are close enough to the median value independently of the whole distribution of votes. This allows price submitters to have a guaranteed reward when voting near the expected value, without being affected by other price providers.

A new random scheme is toggled with a quality random flag. It was described by the team to be a commit and reveal scheme, where if any submitter fails to reveal, the good random flag will be turned off. This prevents submitters from commiting and not revealing values, a random manipulation strategy that could be performed with the current system.

On the other hand, the `699-update-voterwhitelister` branch adds logic to the `VoterWhitelister` contract, which now implements a temporary ban on voters. The governance has the possibility of disabling voting for specific addresses during a number of rewards specified on a case by case basis. This is performed by calling the `chillVoter` method.

When the `VoterWhitelister` is now updated, the new version of the contract will start on a `copyMode` state. This is a state that can be permanently disabled by the Immediate Governance and will prevent any voting during this state. The purpose is to copy the whitelisted price providers from the old `VoterWhitelister` contract. The `copyWhitelist` is restricted to the Immediate Governance. This restriction is probably unnecessary because the data can only be copied from the previous contract and not submitted by anyone.

Coinspect found no major issues with the changes presented. The code is well-tested and well documented, both with comments on the contracts and with external documentation which was provided by Flare.

Fixes review

On February 16th, 2023 a fix review was conducted on the the **flare_distribution_audit** branch of the git repository at https://gitlab.com/flarenetwork/flare-smart-contracts. The commits are included in the Merge Request 602.

The commits include fixes and a minor amount of changes. Coinspect found the fixes correctly mitigate bugs that were discussed with Flare and that the changes did not introduce any additional risk.

The commits reviewed were:

- `e07849d9120a6c48d3f15321715ad2029ff2093a`
- `fe8c5c38c7952cf53a57c331a74fdb992d514aba`
- `cc0130e9b23f5d8f2b3a44ff25f4ff92109095f4`
- `33744000b4a7e57cd668175ff1f5d2c53431e226`
- `31b45497b509fd083f3951a75f22293a53950e89`
- `85cd38a7d10dc8434644747b10691ffcbfb61446`
- `765aa940411a524d89fe5bfaf10c4f995a9213b7`
- `bb57dc77bfc4533787ef3dc1d3a32bf4a9620e3a`
- `6b463b7e65930c4aca3fd745df11fa136ce97d63`

The commit `e07849d9120a6c48d3f15321715ad2029ff2093a` moves bulk operations to the new contract registry contract.

Commit `fe8c5c38c7952cf53a57c331a74fdb992d514aba` adds new interfaces and adapts `AddressUpdater` so it conforms to `IIAddressUpdater`. On that same commit, a Flare contract registry was created which provides external view functions querying the addresses stored in `AddressUpdater`.

The commit `cc0130e9b23f5d8f2b3a44ff25f4ff92109095f4` adds an external view function to the `AddressUpdater` allowing to get an address by providing a name hash.

The commit `33744000b4a7e57cd668175ff1f5d2c53431e226` increases the number of epochs where an old reward manager could be used and restricts delegators from performing a claim multiple times.

On the commit `31b45497b509fd083f3951a75f22293a53950e89` the treasury was modified and now it only supports one distribution contract at a time. Two changes were made to `DistributionToDelegators`: it is possible to set an entitlement start time two weeks in the past to comply with publicly committed dates and the whole system is now stoppable in case of an emergency. Also, the older `Distribution` contract was removed and some constant values were modified to fit better with Flare's deployment and administration plans.

Commit `85cd38a7d10dc8434644747b10691ffcbfb61446` indirectly removes a retry logic that was active for updating the power blocks and rewards. Now, when `daemonize()` commits to some power blocks and distributable amounts, it will not ever change those values. This makes potential issues in the update of distributable amount more serious; as now if for some reason the treasury has no funds when calling `daemonize()` for the first time in a month, the rewards for that month will be zero **permanently**. This change also invalidates a bug where some users could get their rewards stuck when these values were updated by making that update impossible.

Commit `765aa940411a524d89fe5bfaf10c4f995a9213b7` modifies deployment scripts adding a redeployment mechanism. Some configurations were also changed, including the governance time lock which is now one hour.

The commit `bb57dc77bfc4533787ef3dc1d3a32bf4a9620e3a` fixes `FLR-34` by adding a timeout when waiting for good randoms on `FtsoManager.sol`.

Commit `6b463b7e65930c4aca3fd745df11fa136ce97d63` does the same for `DistributionForDelegators.sol` and makes `ownerNextClaimableMonth` increase monotonically, fixing an issue found by the Flare team.

3. Summary of Findings

Id	Title	Total Risk	Fixed
FLR-32	Attackers can make executors waste money	Low	!
FLR-33	Insecure random in finalizePriceEpoch random	Low	✔
FLR-34	Denial of service by preventing good random values	Low	✔

- ✔ The issue has been solved.
- ! The issue does not require immediate action but developers and users must exercise caution.
- ✘ The issue has not been addressed.

4. Detailed Findings

FLR-32	Attackers can make executors waste money	
Total Risk **Low**	Impact Medium	Location `contracts/tokenPools/implementation/DistributionToDelegators.sol`
Fixed !	Likelihood Low	

Description

Attackers can make an executor waste resources by setting an evil *_receiver*.

Executors can call the `claim` method to claim rewards for their clients. Importantly, the client is able to set any `_recipient` address via the `setAllowedClaimRecipients` method.

If an executor calls the `claim` method with a recipient address provided by an attacker and `_wrap` set to `false`; the contract will make a call to the `_recipient` address. This `_recipient` address can contain logic that makes the executor waste all their gas.

`DistributionToDelegators.sol`
```
        } else {
            (bool success, ) = _recipient.call{value: _rewardAmount}("");
            /* solhint-enable avoid-low-level-calls */
            require(success, ERR_CLAIM_FAILED);
        }
```

The problem is exacerbated if the executor contains off-chain logic that retries transactions that fail or if they set a high gas limit on their transactions. Both are common practices in automated systems that make transactions.

Recommendation

If executors are not supposed to call `_claim`, disallow them to do so.

In addition, it is encouraged that all calls to external contracts set a gas limit and use Nomad's Excessively Safe Call to prevent any kind of griefing.

Status

Flare has acknowledged the issue and stated that it's the responsibility of executors to avoid claiming for evil receivers.

FLR-33 Insecure random in finalizePriceEpoch random

Total Risk	Impact	Location
Low	Low	`contracts/ftso/implementation/FtsoManager.sol`
Fixed ✔	Likelihood Low	

Description

Random used for `finalizePriceEpoch` can still be manipulated.

The `finalizePriceEpoch` function in the `FTSOManager` uses the old random instead of the new `getCurrentRandomWithQuality` function.

The `getCurrentRandom` method may fail to provide a good random value and may also not be available in the new implementation.

Recommendation

Add support for the `getCurrentRandomWithQuality` `function.`

Status

Fixed at commit `eb44f6f10da003cb11692cf0c0b953feedb51e10` by following the recommendation.

FLR-34 Denial of service by preventing good random values

Total Risk	Impact	Location
Low	Low	`contracts/ftso/implementation/FtsoManager.sol`
Fixed ✓	Likelihood Low	

Description

The platform will become inoperational if the `getCurrentRandomWithQuality` method continually fails to return the `goodRandom` flag.

The `getCurrentRandomWithQuality` method provides a good random boolean flag. If this method returns false permanently, `_finalizeRewardEpoch` and `_updateVotePowerBlocksAndWeight` functions will stop working

This dismisses guarantees of well behaved providers receiving rewards.

Recommendation

Reward owners should be guaranteed to receive their rewards. Implement an escape-hatch system that allows owners to bypass the *good random* requirement if the system has not received a good random after a certain, long, amount of time.

Status

The issue has been fixed in commits `6b463b7e65930c4aca3fd745df11fa136ce97d63` and `bb57dc77bfc4533787ef3dc1d3a32bf4a9620e3a` by introducing a timeout for the wait of good random numbers.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.