Flare

Security Review

FTSO Scaling

coinspect

flare

FTSO Scaling

Source Code Security Review

Version: v240515 Prepared for: Flare May 2024

Source Code Security Review

1. Executive Summary

In **January 2024**, **Flare** engaged Coinspect to perform a source code review of its **FTSO Scaling** system. The objective of the engagement was to evaluate the security of the application, which is in charge of coordinating price feeds obtained by the operators of the system with the Flare Systems Client and providing proofs to end-users.

	✓ Solved	⚠ Caution Advised	✗ Resolution Pending
High	2	0	0
Medium	1	0	0
Low	1	0	0
No Risk	1	0	0
Total	**5**	**0**	**0**

`FTSOS-001`, `FTSOS-002` and `FTSOS-003` all show how the lack of authentication in the server leads to reveal data being public and also liable to be replaced by attackers. `FTSOS-004` shows that it is possible -- although unlikely -- to have the same commit hash for different rounds.

2. Summary of Findings

2.3 Solved issues & recommendations

These issues have been fully fixed or represent recommendations that could improve the long-term security posture of the project.

Id	Title	Risk
FTSOS-001	Attacker can access reveal data	High
FTSOS-002	Attacker can force the reveal of wrong data due to random data in reveal	High
FTSOS-003	Attacker can force revealing of wrong data by following reference price provider	Medium
FTSOS-004	Possible to repeat hashes for different rounds	Low
FTSOS-005	Unused utilities	None

3. Scope

The scope was set to be the repository at ftso-scaling on commit `783a7be11d6fdf7610dc2991471129d5b7f08ea0`.

The review included all the components of the project, except those related to rewards. As such,the following files and folders have been excluded from this review:

```
apps/ftso-reward-calculator/*
libs/ftso-core/src/rewards-calculation/*
libs/ftso-core/src/data-calculation-interfaces.ts
libs/ftso-core/src/utls/RewardClaim.ts
libs/ftso-core/src/utils/PartialRewardOffer.ts
libs/ftso-core/src/events/InflationRewardsOffered.ts
```

Methods dealing with rewards such as `getRewardOffers` in `libs/ftso-core/src/IndexerClient.ts` are also excluded.

It is worth noting that the **FTSO Scaling** project is one part of a more complex system. The correctness of the prices posted by voters require several other components to work timely and correctly, including data feeds that need to be connected to the **FTSO Scaling** project and are out of scope of the audit and the responsibility of voters to implement. Coinspect did review an example implementation to be provided as reference, but this reference is not intended to be put in production.

4 Assessment

The **FTSO Scaling** program is composed of one Next.js server which is the main piece of software in charge of communication with the price providers and with the `top-level-client` application. The server fetches data from the price providers on request of the `top-level-client` and it is also in charge of assembling the merkle root of these price feeds to be posted on the blockchain by the `top-level-client`.

It is worth noting that the program does not handle private keys: all the signing is done in the `top-level-client`'s scope.

There are libraries that the server uses for operations: `fsp-utils` deal with encoding and decoding of data while `fsp-core` is a bigger library that holds core logic for the application, including the ordering of the price feeds and calculation of the random value.

Reviewers considered that the project was exposed to the Internet, as some endpoints are clearly tagged as `external`. What is more, users *need* to contact a provider to get the proof of their claims to be posted on chain.

4.1 Security assumptions

Some security assumptions were made when reviewing the project:

1. The `top-level-client` fetches data in a timely manner.
2. The price feeds set by the voter are accurate
3. The provider trusts the price feed software and the `top-level-client` software

4.2 Testing

Coinspect found that the project was well tested, with a 65% coverage by line. 86 tests pass in the project, while 2 integration tests are currently failing due to wrong arguments being passed to functions:

```
  1) ftso-data-provider.service (test/apps/integration/ftso-data-
provider.service.test.ts)
       should return correct reveal data:
     TypeError: lru_cache_1.LRUCache is not a constructor
      at new FtsoDataProviderService (apps/ftso-data-provider/src/ftso-
data-provider.service.ts:2:4558)
      at Context.<anonymous> (test/apps/integration/ftso-data-
provider.service.test.ts:85:21)

2) ftso-data-provider.service (test/apps/integration/ftso-data-
provider.service.test.ts)
       should compute results - multiple voters, same price:
     TypeError: lru_cache_1.LRUCache is not a constructor
      at new FtsoDataProviderService (apps/ftso-data-provider/src/ftso-
data-provider.service.ts:2:4558)
```

4.3 Additional Fixes

In April 8, 2024 Coinspect reviewed additional changes and features as of commit `3de282a31c78dc85972bb004412dfa8dd375df0e`. The scope did not change: the example applications and the rewards components were not considered for this commit.

Flare indicated that the most relevant changes made from the original commit were:

- Malformed submit transactions are now skipped instead of failing.
- Improved edge case handling for the benching window.
- Removed a failure on `TIMEOUT` when retrieving data from the indexer.

In addition to the changes outlined by Flare, an analysis of the differences between the new commit and the originally reviewed version revealed numerous other modifications to the codebase. Due to the extensive nature of these changes and their integration with existing code, conducting an isolated analysis of each change is impractical. Because of this, Coinspect focused the review on the changes specifically outlined by Flare.

5. Detailed Findings

FTSOS-001

Attacker can access reveal data

Status

Solved

✓

Resolution

Fixed

Risk

High

🟥

Impact

High

Likelihood

High

Location

```
apps/ftso-data-provider/src/ftso-data-provider.controller.ts
```

Description

An attacker can query the `submit2/:votingRoundId/:submitAddress` endpoint to get the reveal data associated with the commit of `votingRoundId`.

This is because the `submit2` endpoint does not check the timing of the request and does not enforce that it is during the reveal epoch of the sub-protocol. Instead, the reveal data is simply accumulated in a map when generated:

```
  async getCommitData(
    votingRoundId: number,
    submissionAddress: string
  ): Promise<IPayloadMessage<ICommitData> | undefined> {
    ...
    const hash = CommitData.hashForCommit(submissionAddress,
revealData.random, revealData.encodedValues);
    const commitData: ICommitData = {
      commitHash: hash,
    };
    this.votingRoundToRevealData.set(votingRoundId, revealData);
    ...
  }
```

And then retrieved and returned when the `submit2` endpoint is called:

```
  async getRevealData(votingRoundId: number):
Promise<IPayloadMessage<IRevealData> | undefined> {
    this.logger.log(`Getting reveal for voting round
${votingRoundId}`);

const revealData = this.votingRoundToRevealData.get(votingRoundId);
    if (revealData === undefined) {
      // we do not have reveal data. Either we committed and restarted
the client, hence lost the reveal data irreversibly
      // or we did not commit at all.
      this.logger.error(`No reveal data found for epoch
${votingRoundId}`);
      return undefined;
    }

const msg: IPayloadMessage<IRevealData> = {
      protocolId: FTSO2_PROTOCOL_ID,
      votingRoundId: votingRoundId,
      payload: revealData,
    };
    return msg;
  }
```

Recommendation

Implement authentication so only the `top-level-client` can call the endpoints that it should use.

Consider checking the timing of the request to only reveal data that can be revealed if the endpoints needs to be public.

Status

Fixed on commit `d6b70937d03a632f72899472b17d66d444fdb041`.

An API Key is now required to authenticate calls to the methods exposed by the `FtsoDataProviderController`.

FTSOS-002

Attacker can force the reveal of wrong data due to random data in reveal

Status

Solved

✓

Resolution

Fixed

Risk

High

Impact

High

Likelihood

High

Location

```
apps/ftso-data-provider/src/ftso-data-provider.controller.ts
```

Description

An attacker can manipulate a victim to reveal data that does not match their initial commit due to the randomness of reveal data, which changes with each request, and the way this data is stored.

When the `top-level-client` requests data for commitment, it calls the `submit1/:votingRoundId/:submitAddress` endpoint. This data is then posted on-chain. The FTSO Data Provider stores the actual data in the `votingRoundToRevealData` map, mapped from `votingRoundId` to `revealData`.

The problem arises because `revealData` includes a random component: `revealData.random`, which changes on different calls to the endpoint.

This allows an attacker to wait until the `top-level-client` has requested a commit from the victim and then call `submit1/:votingRoundId/:submitAddress` with the same voting round ID.

Since the `votingRoundId` remains the same, the map will overwrite the previous `revealData` with the new `revealData.random`.

When the `top-level-client` later requests the reveal data, this newly overwritten data is returned. This reveal data will not match the one originally committed by the victim, exposing them to penalties and potentially causing loss of rewards.

Recommendation

Implement authentication for methods that should only be accessed by the `top-level-client`.

Additionally, consider revising the storage mechanism for `revealData` to prevent overwriting. One approach is to assign a unique random ID to each piece of reveal data, shared only with the requester, who can then retrieve the reveal data using this ID.

Status

Fixed on commit `d6b70937d03a632f72899472b17d66d444fdb041`.

The issue is no longer exploitable by an attacker due to authentication being implemented at the controller's level.

FTSOS-003

Attacker can force revealing of wrong data by following reference price provider

Status
Solved

✓

Resolution
Fixed

Risk
Medium

Impact
High

Likelihood
Medium

Location

`apps/ftso-data-provider/src/ftso-data-provider.controller.ts`

Description

An attacker can force a victim to reveal data that does not match their commit because the example provider implementation always returns the most up-to-date prices instead of considering the `roundId`.

This issue is very similar to FTSOS-002. The difference is that instead of relying on the `random` data contained in the reveal, the attacker relies on the fact that the reference implementation for a provider is unsafe by default.

When the `ftso-data-provider` queries the `example-provider` for its price feed, it will send an HTTP request to the endpoint `preparePriceFeeds/:votingRoundId`. Unfortunately, by default, this endpoint completely ignores the `votingRoundId` parameter.

```
  @Post("preparePriceFeeds/:votingRoundId")
  async getPriceFeeds(
    @Param("votingRoundId", ParseIntPipe) votingRoundId: number,
    @Body() body: PriceFeedsRequest
  ): Promise<PriceFeedsResponse> {
    const prices = await
this.priceProviderService.getPrices(body.feeds);
    return {
      votingRoundId,
      feedPriceData: prices,
    };
  }
```

This means that no matter the implementation of `priceProviderService`, it will not have information on which `votingRoundId` to return prices for. Therefore, it will almost certainly return the most up to date information. Which will cause a mismatch between commit and reveal, as described in FTSOS-002.

The likelihood of this issue is lower because there is a chance that the provider implements a correct version of the example.

Recommendation

See the recommendation for `FTSOS-002`.

Status

Fixed on commit `d6b70937d03a632f72899472b17d66d444fdb041`.

The issue is no longer exploitable by an attacker due to authentication being implemented at the controller's level.

FTSOS-004

Possible to repeat hashes for different rounds

Status
Solved

✓

Resolution
Fixed

Risk
Low

🟥☐☐

Impact
Low

Likelihood
Low

Location

```
client/finalizer/finalizer_client.go
```

Description

The hash for a `commit` can be repeated in different voting rounds because the hash does not include the voting round or any other kind of timestamps.

This means that a misbehaving voter that does not participate correctly in the protocol and chooses a fixed random parameter to post would get the same hash for the same prices each time. As prices are not entirely random nor volatile, the chance of prices repeating somewhat often is non-negligible.

The issue is in the `hashForCommit` method:

```
  export function hashForCommit(voter: Address, random: string, prices:
string) {
    const types = ["address", "uint256", "bytes"];
    const values = [voter.toLowerCase(), random, prices];
    const encoded = encodeParameters(types, values);
    return soliditySha3(encoded)!;
  }
```

Note as well that his goes against the specification of the FTSO protocol, which states that `i`, the round ID, is included in the commit hash.

```
Then the commit hash sha3(random_number, i,  address, data) is
calculated from the random number, round id i,
data provider's address and the feed value vector.
```

Recommendation

Include the round ID in the commit hash.

Status

Fixed on commit `96b969348154bd205d6e1a029af14601367b41e2`.

The round ID was included in the commit hash.

FTSOS-005

Unused utilities

Status
Solved

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
–

Location

```
fsp-utils/src
```

Description

There are unused utilities for signing messages within the `fsp-utils/src` folder, like the `signMessageHash` function.

This functions are not only unused, but strictly unsupported by the platform.

Recommendation

Move the functions used in tests to a corresponding folder.

Status

Acknowledged.

The Flare Team stated that some functions are used in the rewarding logic calculation.

6. Disclaimer

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