

StakingP2
Off-chain services
Security Review



Staking P2 Off-chain Services

Source Code Review

Version: v240220

Prepared for: Flare

November 2023

Off-chain Services Review

Executive Summary

Summary of Findings

Solved issues & recommendations

Assessment and Scope

Integration With the Mirroring Protocol

Architecture Overview

Trust Assumptions

External Calls Handling

Interactions With External Services

Changes introduced by new commit

Fix Review

Detailed Findings




Disclaimer

File hashes

Executive Summary

In September 2023, Flare engaged Coinspect to review the off-chain **Mirroring Services** implementing the Flare Staking Phase 2 Mirroring Protocol. The objective of the project was to evaluate the security of the scripts responsible for voting and mirroring staking positions. These operate with other peripheral services such as Indexers, resulting in a system in charge of linking the state of the P-Chain with the C-Chain's.

The following issues were identified during the initial assessment:

 Solved	 Caution Advised	 Resolution Pending
High 1	High 0	High 0
Medium 2	Medium 0	Medium 0
Low 1	Low 0	Low 0
No Risk 3	No Risk 0	No Risk 0
Total 7	Total 0	Total 0

PSOS-001, a high-risk issue, highlights how the current project's environment setup is prone to leak user's credentials. The first medium-risk issue, PSOS-002, depicts how adversaries can force a mirroring service to stop working by mirroring the stake themselves. Additionally, PSOS-003 outlines a lack of user segregation in the database. Lastly, the remaining low-risk refers to the fact that reset epochs are not properly

handled and will never be voted again by the service, thereby requiring manual voting and root calculation, reflected in PSOS-004.

It should be noted that the Flare team addressed PSOS-002 in the most recent commit, prior to Coinspect reporting the issue.

Summary of Findings

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
PSOS-001	Insecure handling of voters private keys	High
PSOS-002	Adversary can force mirroring services to stop working	Medium
PSOS-003	Lack of database privilege segregation	Medium
PSOS-004	The voting service will never include transactions from reset epochs	Low
PSOS-005	Script will attempt to mirror epochs twice	None
PSOS-006	Key contextual time-based variables could be constant	None
PSOS-007	Service API disclosing internal error information	None

Assessment and Scope

The source code review of the Flare Staking Phase 2 Mirroring Services project started on September 11th, 2023, and was conducted on the `staking-clients` branch of the git repository located at <https://github.com/flare-foundation/flare-p-chain-indexer> as of commit `ee0632512bd70d12e2b2738df9321a23e8b5e394`.

Overall, the code was easy to read and understand. However, Coinspect suggests including more documentation and specifications regarding its use cases considering the strong trust assumptions of the system. Coinspect also recommends adding more integration testing cases to include adversarial scenarios or situations where a condition is changed directly on the C-Chain (smart contracts). For example, tests considering root resets, threshold changes, etc.

The mirroring services are comprised by several jobs, each one in charge of a specific key task required by the **Mirroring Protocol**. Particularly, the services implemented cover the following tasks: **Uptime Tracking and Voting**, **Root Voting**, **Stake Mirroring**, and **Transaction Indexing**.

Integration With the Mirroring Protocol

The reviewed off-chain services work as a nexus between the transactions made on the P-Chain and the corresponding smart contracts on the C-Chain. As a result, users are able to stake or delegate into a P-Chain node, and receive rewards on the C-Chain. This is achieved by the **Transaction Mirroring Protocol** comprised by actions made off-chain and on-chain. Trusted entities called `voters` receive a set of indexed transactions for each epoch, calculate their merkle root and vote for that root on the C-Chain's smart contract. This process is responsibility of the `Root Voting` service.

When the voting threshold is reached, a root is committed and anyone can mirror a proved transaction into that chain, hence, a `Mirroring` service handling those transactions is also included into the off-chain suite. In addition, voters are in charge of tracking the node uptime by emitting an event on the same C-Chain contract, carried out by the `Uptime Voting` script. Those actions require the collection of each transaction and node status from the P-Chain, which is performed by the `Uptime` and `Indexer` services.

Architecture Overview

The architecture is based on goroutines that run each job concurrently, relying on an external transaction database that is populated by the indexing service. It is flexible, allowing users to choose between which jobs they will run. For instance, a non-voter user is able to run the indexer and the mirroring service, calling the smart contract subsidizing the mirroring of stakes. Likewise, voters will also run the voting services providing merkle roots to the voting contract and submit validators uptime records. Coinspect identified that the current project's architecture requires users to store their credentials on the same configuration file along with other parameters, and most importantly does not protect that file against accidental commits or pushes to the version control system, PSOS-001.

Each service relies on the database's state which is populated by the indexer service. Moreover, because all other services work in an epoch-based time frame, a critical aspect is updating internal indexes (used in for loops) and database registries properly. In the event of having a delay or disarrangement between the current block, epoch or internal indexes, each service would likely stop working as intended. For example, as each service loops over each epoch in a sequential way, if the Governance resets a root, voters will require to manually calculate and vote for the new root of that epoch, PSOS-004. Also, Coinspect identified that the database can be altered from different sources, such as API calls. In the event of the API key compromise, adversaries would be able to alter information in this database, and therefore affect voting results.

Trust Assumptions

Regarding voters and their incentives to act in an honest manner, the system assumes that they are all trusted entities that will not become rogue or collude. In fact, this assumption combined with the insecure private key handling could lead to a denial of service if a supply chain attack is used to compromise the nodes and/or several credentials are leaked. For instance, with a voting threshold of 6, and 5 keys compromised the system would become non operational.

Censorship and Collusion

The incentive program for voters to act honestly is unclear. In other words, there are no punishments or incentives that prevent voting collusion or censorship against a specific set of transactions or accounts. In short, in the event of having compromised keys or voters turning adversarial, the system does not implement preventive actions,

protecting the mirroring protocol with corrective countermeasures such as merkle root resets and revoke stakings made directly on the C-Chain contracts relying on the Protocol's Governance.

External Calls Handling

Interactions with foreign data sources such as contracts and nodes are made through two different clients: Avalanche Indexer Client and Avalanche RPC Client. They both require an API Key in order to establish connection, facing the same issue as the private keys mentioned on the [Architecture Overview](#) section in the present [Assessment](#). Calls made to contracts on the C-Chain have no specific gas estimation rules, relying on the estimation of the node.

Native Balance Checks

Currently, there are no checks to ensure that a user running the off-chain services has enough native balance to cover the gas fees. Running out of native tokens will trigger a revert on those jobs performing calls to the contracts, until the account is funded. Coinspect suggests adding monitoring services into Prometheus that track each user's balance, which trigger an alert when the account's balance falls below a threshold. Additionally, a balance check could be added when each service is started.

Smart Contract reverted transactions handling

In terms on how the services handle revert conditions, Coinspect identified that this field could be improved. Specifically, several critical updates such as next epoch indexes of the mirroring services are made only if the call to the contract on the C-Chain was successful. In other words, a revert on the smart contract triggers an early return on every single service which should consider when updating internal indexes, for example.

Regarding the voting service, this condition is handled by the call made to the `shouldVote` function, ensuring to submit a vote only for the current epoch if the root has not been committed or the voter has not voted. However, Coinspect identified that reversals are not properly handled in the mirroring service, where adversaries can front-run stake mirroring transactions to stop each service potentially leaving all mirroring services non-operational, PSOS-002.

Interactions With External Services

During this engagement, Coinspect did not review how the validator uptime data is consumed once the respective event (`PChainStakeMirrorValidatorUptimeVoteSubmitted`) is emitted. As the design of the uptime voting system found in the smart contracts is lightweight, it only emits the event provided by its caller. Services consuming its data should handle adversarial scenarios such as double voting, voting for fake `nodeIds`, among others.

A similar situation relates to nodes providing P-chain and uptime data. The current engagement has not addressed potential issues from nodes set up incorrectly or with weak security protections.

Changes introduced by new commit

On September 20th, 2023, a new commit (c38fa4913cfd529a764ced791e111e22f5217c7f, tag audit-09-20) was provided by Flare's team. These code modifications introduce several changes and bug fixes which were reviewed during the last two days of the engagement.

- Fixed PS0S-001: Private keys are now retrieved from an external file.
- Added new queries to the database allowing to fetch a list of stakers available at a specific time.
- Added epoch cronjob support.
- Split jobs into two files, `main` and `stub`. The `main` job file performs core actions whereas `stubs` provide each job with peripheral actions (such as utility functions). Several functions were moved from the `utils` file to each `stub`.
- Fixed PS0S-002 (independently discovered by the Flare team): Expected reverts when mirroring transactions are now handled.
- Added two new API endpoints allowing querying mirroring information.
- Added new functions to the staking endpoint.
- Added several unit tests to each service. Coinspect suggests more scenarios are tested, for instance, evaluate how the scripts behave if a condition or parameter is modified directly on the C-Chain contracts.

Fix Review


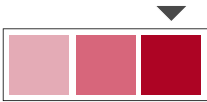
On October 2nd, 2023 Flare provided an updated repository commit with fixes for all the issues reported. Additionally, they made fixes and improvements to the source code.

- **Improvement:** Added the configuration option `delete_old_uptimes_epoch_threshold` to remove uptime info for epochs already voted for (commit `c9b1b3cc89cd4fa16a9600626579940c042a1742`). This feature calls on each loop a cleanup function that deletes from the database registries of old uptime epochs to prevent it from growing too large. Only data older than 5 epochs can be deleted.
- **Fix:** Voting and mirroring is only performed for input with index 0. Previously, voting and mirroring was done for all input addresses with the same weight (total staked amount) which could potentially lead to rewarding a staker multiple times (commit `9c23c230a8dec0b505b1c41539b2dc1899989792`).
- **Improvement:** `gasLimit` was estimated automatically and it was tight (by library we use). Since the gas used depends on the order of execution after estimation if two or more votes were submitted at about the same time, then and they happen to use the same estimate, the one that puts in the block later fails due to all gas used (note the limit is tight). Now there is option to set `gasLimit` through env variable (commit `a8a4e2c98b3a5c8277a74bcb7a4bedfd1364a873`).

Detailed Findings

PSOS-001

Insecure handling of voters private keys

Status Solved	Risk High
	
Resolution Fixed	Impact High Likelihood High
Location <code>config.toml</code>	

Description

The current project's structure does not fully protect voter's and operator's private keys or endpoints, increasing the likelihood of disclosing sensitive data as the default configuration file is already pushed into the main branch.

Each voter has two different ways, as per the project's documentation, to create configuration files:

```
Config file can be specified using the command line parameter `--config`, e.g., `./services --config config.local.toml`. The default config file name is `config.toml`.
```

A sample `config` file includes not only sensitive information (such as the voter's private key) but also the parameters used to customize how the script works (for example, refresh timeouts, chain ids, start indexes, etc.). This structure is error prone as users are forced to mix on the same file configuration parameters required to run the script along with sensitive credentials.

Additionally, voters cloning this repository could accidentally push configuration files with their private key because the default config file is named after `config.toml` (when there is no `config.LOCAL.toml`). The current exceptions of `.gitignore` don't protect users against this kind of mistakes, potentially exposing their private key when pushing.

It is worth mentioning that adding a `*.toml` exception won't protect voters as `config.toml` was previously committed and pushed into the main branch, commit hash `a71d0d3c6c068a817db229e389ab77166c9c139e` on February 16th, 2023.

Recommendation

Don't use the same file to store sensitive data and configuration parameters. Additionally, improve the private key and endpoint (API key) handling to reduce the risk of expos.

Status

Fixed in commit `7a18b8be487663ab0c70e632497f22cced24ef60`.

Private keys are now retrieved from an external file.

However, Coinspect observed that:

- Minimum file system permissions are not enforced for the file containing the private key. It is strongly suggested to add checks in the service that prevent reading private keys from files that have overly permissive permissions.
- RPC API keys could also be exposed in the configuration file, which is risky as detailed in `PS0S-003`.
- Private key management is still a relevant threat as mentioned in the assessment. Continuous improvements can be made to support safer key storage devices and key rotation policies.

PSOS-002

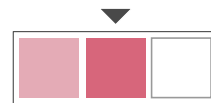
Adversary can force mirroring services to stop working

Status
Solved



Resolution
Fixed

Risk
Medium



Impact
Low
Likelihood
High

Location

`indexer/cronjob/mirror.go`

Description

An adversary can indefinitely halt off-chain clients from mirroring transactions, potentially impacting the calculation of rewards.

When a client attempts to mirror a stake that has already been mirrored, the transaction will be reverted. This will trigger an error in `mirror.go:mirrorTx()`, abruptly terminating the mirroring script. As a consequence, the `NextDBIndex` epoch does not get updated.

Subsequently, when the next cycle begins, the mirroring script will attempt to mirror that same previously reverted stake, as `NextDBIndex` remained unchanged. This causes the script to be trapped in a perpetual error loop.

To illustrate, here's a brief breakdown of the logic that leads to the mirroring scripts getting stuck indefinitely:

```
for epoch := epochRange.start; epoch <= epochRange.end; epoch++ {
    // Skip updating if indexer is behind
    if c.indexerBehind(&idxState, epoch) {
        logger.Debug("indexer is behind, skipping mirror")
        return nil
    }

    logger.Debug("mirroring epoch %d", epoch)
    if err := c.mirrorEpoch(epoch); err != nil {
        //This error is thrown when the mirroring transaction
        reverts,
        //preventing the script from finishing iterating the
        for loop

        return err
    }
}

logger.Debug("successfully mirrored epochs %d-%d", epochRange.start,
epochRange.end)

//NextDBIndex will not be updated due to early finish
//Therefore, in the next iteration, the script will attempt
//to mirror the same stake that caused the revert

if err := c.updateJobState(epochRange.end); err != nil {
    return err
}
```

Note that:

- When working in parallel with multiple mirroring clients, this issue hinders all but one from effectively mirroring. An adversary can exploit this weakness to force the remaining operative client into an error loop.
- Any third-party with a valid `_merkleProof` and `_stakeData` can still manually mirror the transaction on demand.

Proof of Concept

The following scenario uses two different mirroring services and prints the EVM revert that is triggered when the second service tries to mirror. Then, it is shown how this service does not update its index trying to mirror the transactions on the same epoch were the revert was initially triggered.

To run this test, add the following scripts into `indexer/cronjob/voting_test.go`.

Run:


```
go test ./indexer/cronjob/ -run
"TestCoinspect_VotingWithTwoMirroringServices" -v -count=1
```

Output

```
=== RUN    TestCoinspect_VotingWithTwoMirroringServices
=== RUN    TestCoinspect_VotingWithTwoMirroringServices/Run_indexer_1
=== RUN    TestCoinspect_VotingWithTwoMirroringServices/Run_indexer_2
=== RUN
TestCoinspect_VotingWithTwoMirroringServices/Run_voting_clients_1_and_2
=== RUN
TestCoinspect_VotingWithTwoMirroringServices/Reverts_if_two_mirroring_j
obs_mirror_the_same_Tx

Waiting 2 secs...

Runnning mCronjob1
mirroring epoch: 0
mirroring epoch: 1
mirroring epoch: 2

Waiting 2 secs...

Runnning mCronjob2
mirroring epoch: 0
Failed on Epoch: 0
mirroringContract.MirrorStake: Error: VM Exception while processing
transaction: reverted with reason string 'transaction already mirrored'

Waiting 2 secs...

Runnning mCronjob1
<nil>

Runnning mCronjob2
mirroring epoch: 0
Failed on Epoch: 0
mirroringContract.MirrorStake: Error: VM Exception while processing
transaction: reverted with reason string 'transaction already mirrored'
--- PASS: TestCoinspect_VotingWithTwoMirroringServices (10.95s)
    --- PASS:
TestCoinspect_VotingWithTwoMirroringServices/Run_indexer_1 (0.10s)
    --- PASS:
TestCoinspect_VotingWithTwoMirroringServices/Run_indexer_2 (0.12s)
    --- PASS:
TestCoinspect_VotingWithTwoMirroringServices/Run_voting_clients_1_and_2
(0.75s)
    --- PASS:
TestCoinspect_VotingWithTwoMirroringServices/Reverts_if_two_mirroring_j
obs_mirror_the_same_Tx (8.46s)
PASS
ok      flare-indexer/indexer/cronjob    11.259s
```

Script

```
func TestCoinspect_VotingWithTwoMirroringServices(t *testing.T) {
    now := time.Unix(1675349340, 0) // 2023-02-02 14:49:00 UTC
    vCronjob1, vCronjob2, mCronjob1, mCronjob2, indexer1, indexer2,
    err := createTestVotingClientsTwoMirrorings(now)
    require.NoError(t, err)

    t.Run("Run indexer 1", func(t *testing.T) {
        err := indexer1.IndexBatch()
        require.NoError(t, err)
    })
    t.Run("Run indexer 2", func(t *testing.T) {
        err := indexer2.IndexBatch()
        require.NoError(t, err)
    })

    t.Run("Run voting clients 1 and 2", func(t *testing.T) {
        vCronjob1.time.SetNow(now)
        vCronjob2.time.SetNow(now)
        for i := 0; i < 10; i++ {
            err := vCronjob1.Call()
            require.NoError(t, err)
            err = vCronjob2.Call()
            require.NoError(t, err)
            vCronjob1.time.AdvanceNow(30 * time.Second)
            vCronjob2.time.AdvanceNow(30 * time.Second)
        }
    })

    t.Run("Reverts if two mirroring jobs mirror the same Tx", func(t
    *testing.T) {
        mCronjob1.time.SetNow(now)
        mCronjob1.time.AdvanceNow(10 * 30 * time.Second)

        mCronjob2.time.SetNow(now)
        mCronjob2.time.AdvanceNow(10 * 30 * time.Second)

        fmt.Println("\nWaiting 2 secs...")
        time.Sleep(2 * time.Second)
        fmt.Println("\nRunning mCronjob1")
        err11 := mCronjob1.Call()
        require.NoError(t, err11)

        fmt.Println("\nWaiting 2 secs...")
        time.Sleep(2 * time.Second)
        fmt.Println("\nRunning mCronjob2")
        err2 := mCronjob2.Call()
        fmt.Println(err2)

        fmt.Println("\nWaiting 2 secs...")
        time.Sleep(2 * time.Second)

        fmt.Println("\nRunning mCronjob1")
        err12 := mCronjob1.Call()
        fmt.Println(err12)
        require.NoError(t, err12)

        time.Sleep(2 * time.Second)
    })
}
```

```

        fmt.Println("\nRunning mCronjob2")
        err22 := mCronjob2.Call()
        fmt.Println(err22)
    })
}

```

Where createTestVotingClientsTwoMirrorings() is:

```

func createTestVotingClientsTwoMirrorings(epochStart time.Time)
(*votingCronjob, *votingCronjob, *mirrorCronJob, *mirrorCronJob,
*shared.ChainIndexerBase, *shared.ChainIndexerBase, error) {
    ctx1, err :=
context.BuildTestContext(votingCronjobTestConfig(epochStart,
"flare_indexer_indexer", privateKey1))
    if err != nil {
        return nil, nil, nil, nil, nil, nil, err
    }
    cronjob1, err := NewVotingCronjob(ctx1)
    if err != nil {
        return nil, nil, nil, nil, nil, nil, err
    }
    ctx2, err :=
context.BuildTestContext(votingCronjobTestConfig(epochStart,
"flare_indexer_indexer_2", privateKey2))
    if err != nil {
        return nil, nil, nil, nil, nil, nil, err
    }
    cronjob2, err := NewVotingCronjob(ctx2)
    if err != nil {
        return nil, nil, nil, nil, nil, nil, err
    }
}

mirror1, err := NewMirrorCronjob(ctx1)
if err != nil {
    return nil, nil, nil, nil, nil, nil, err
}

mirror2, err := NewMirrorCronjob(ctx2)
if err != nil {
    return nil, nil, nil, nil, nil, nil, err
}

indexer1 := &shared.ChainIndexerBase{
    StateName:    pchain.StateName,
    IndexerName: "P-chain Blocks Test",
    Client:       testClient,
    DB:           ctx1.DB(),
    Config:       ctx1.Config().PChainIndexer,
    BatchIndexer: pchain.NewPChainBatchIndexer(
        ctx1, testClient, testRPCClient,
        pchain.NewPChainDataTransformer(transformPChainTx),
    ),
}
indexer2 := &shared.ChainIndexerBase{
    StateName:    pchain.StateName,

```

```

IndexerName: "P-chain Blocks Test",
Client:      testClient,
DB:         ctx2.DB(),
Config:     ctx2.Config().PChainIndexer,
BatchIndexer: pchain.NewPChainBatchIndexer(
                ctx1, testClient, testRPCClient,

pchain.NewPChainDataTransformer(transformPChainTx),
    ),
    }
    return cronjob1, cronjob2, mirror1, mirror2, indexer1,
indexer2, nil
}

```

Recommendation

Skip mirroring transactions that were already mirrored. Improve the testing suite to account for similar scenarios.

Status

Fixed in commit [86d532eda23569b5a6fcd384bc8ae4fbd000f3e8](#).

The following checks were added to prevent bubbling expected errors when mirroring stakes:

```

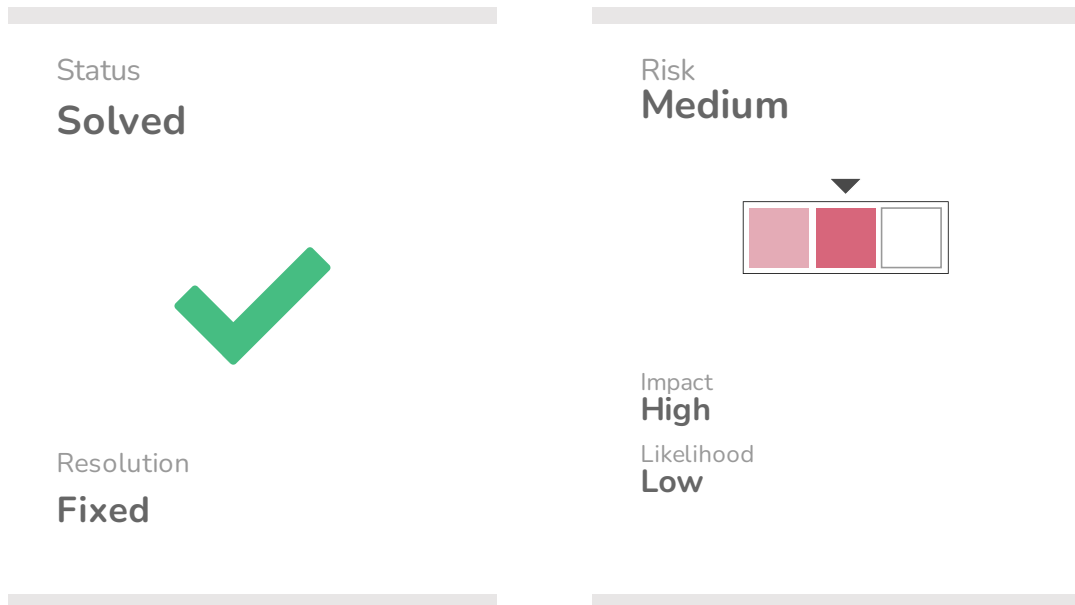
    if strings.Contains(err.Error(), "transaction already
mirrored") {
        logger.Debug("tx %s already mirrored", *in.tx.TxID)
        return nil
    }

    if strings.Contains(err.Error(), "staking already ended") {
*in.tx.TxID)
        logger.Debug("staking already ended for tx %s",
        return nil
    }
}

```

PSOS-003

Lack of database privilege segregation



Description

The service API and indexer currently use the same database without any privilege restrictions. This means that if the API is compromised, an attacker could alter the indexed information.

Such alterations could influence a voter's choice on which P-chain transactions to mirror and also impact validator uptime information.

Recommendation

Implement a read-only database user for the service API.

Status


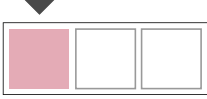
Fixed on commit dc637d5b6b5b9fbfeb023642ac6287e09a9e30e1.

A note recommending database segregation was added to the project's readme:

```
**Note:** We recommend that the user accessing the database is not the same as for the indexer. The user for the services should only have read permissions enabled!
```

PSOS-004

The voting service will never include transactions from reset epochs

Status Solved	Risk Low
	
Resolution Fixed	Impact Medium
	Likelihood Low
Location <code>indexer/cronjob/voting.go</code>	

Description

P-Chain transactions from epochs that the governance reset on the smart contracts will never be proved by the voting service. Fixing this situation will require manual root calculation and voting, potentially leading to omission.

The voting service retrieves the latest epoch where the root is zero, meaning that a commitment is still required. Then, according to the transactions of the database it calculates the root and proceeds to vote for that root on the Multisig Voting contract. However, if the governance resets a root for a past epoch, the voting service will never re-calculate and vote for that root as epochs are increased sequentially:

```

    // Last epoch that was submitted to the contract
    nextEpochToSubmit := utils.Max(state.NextDBIndex,
c.epochs.first)
    lastEpochToSubmit := c.epochs.getEpochIndex(now) - 1
    for e := int64(nextEpochToSubmit); e <= lastEpochToSubmit; e++
{

```

In other words, those transactions on the P-Chain that were not mirrored before the root was reset will require manual actions by all voters (calculating and voting for that new root) so they have a valid proof.

It is worth noting this problem is also present in the mirroring service. Transactions of past epochs that had their root reset and then re-committed will never be mirrored by the service. However, they can be mirrored by the interested party.

Proof of Concept

The following proof of concept requires some configurations on the flare-smart-contracts test/staking2/StakeE2ETest.ts file. The account with the privateKey1 of the Go tests was assigned as the Governance when deploying the contracts on that script:

```

const MOCK_GOVERNANCE = web3.eth.accounts.privateKeyToAccount(
  "0xd49743deccbccc5dc7baa8e69e5be03298da8688a15dd202e20f15d5e0e9a9fb"
);

```

This precondition allows testing onlyGovernance calls directly from the Go testing suite.

Run this test with:

```

go test ./indexer/cronjob/ -run "TestCoinspect_VotingResetingRoot" -v -
count=1

```

Output

```

=== RUN    TestCoinspect_VotingResetingRoot
=== RUN    TestCoinspect_VotingResetingRoot/Run_indexer_1
=== RUN    TestCoinspect_VotingResetingRoot/Run_indexer_2
=== RUN    TestCoinspect_VotingResetingRoot/Run_voting_clients_1_and_2
=== RUN    TestCoinspect_VotingResetingRoot/Check_merkle_roots
Root at epoch 0: [220 121 237 247 149 216 125 17 22 224 50 223 102 125
26 58 96 246 154 40 10 66 178 32 117 1 44 38 66 181 124 6]

```



```

Root at epoch 1: [41 13 236 217 84 139 98 168 214 3 69 169 136 56 111
200 75 166 188 149 72 64 8 246 54 47 147 22 14 243 229 99]
=== RUN    TestCoinspect_VotingResetingRoot/Reset_root_of_Epoch_0
Waiting 2 seconds...
Root at epoch 0: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
Root at epoch 1: [41 13 236 217 84 139 98 168 214 3 69 169 136 56 111
200 75 166 188 149 72 64 8 246 54 47 147 22 14 243 229 99]
=== RUN
TestCoinspect_VotingResetingRoot/Run_voting_clients_1_and_2,_again
Waiting 2 seconds...
Root at epoch 0 After New Voting: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
--- PASS: TestCoinspect_VotingResetingRoot (5.74s)
    --- PASS: TestCoinspect_VotingResetingRoot/Run_indexer_1 (0.08s)
    --- PASS: TestCoinspect_VotingResetingRoot/Run_indexer_2 (0.08s)
    --- PASS:
TestCoinspect_VotingResetingRoot/Run_voting_clients_1_and_2 (0.64s)
    --- PASS: TestCoinspect_VotingResetingRoot/Check_merkle_roots
(0.01s)
    --- PASS: TestCoinspect_VotingResetingRoot/Reset_root_of_Epoch_0
(2.09s)
    --- PASS:
TestCoinspect_VotingResetingRoot/Run_voting_clients_1_and_2,_again
(2.09s)
PASS
ok      flare-indexer/indexer/cronjob    6.072s

```

Test

```

func TestCoinspect_VotingResetingRoot(t *testing.T) {
    now := time.Unix(1675349340, 0) // 2023-02-02 14:49:00 UTC
    vCronjob1, vCronjob2, _, indexer1, indexer2, err :=
createTestVotingClients(now)
    require.NoError(t, err)

    txOpts, err := TransactOptsFromPrivateKey(privateKey1, 31337)
    require.NoError(t, err)

    t.Run("Run indexer 1", func(t *testing.T) {
        err := indexer1.IndexBatch()
        require.NoError(t, err)
    })
    t.Run("Run indexer 2", func(t *testing.T) {
        err := indexer2.IndexBatch()
        require.NoError(t, err)
    })

    t.Run("Run voting clients 1 and 2", func(t *testing.T) {
        vCronjob1.time.SetNow(now)
        vCronjob2.time.SetNow(now)
        for i := 0; i < 10; i++ {
            err := vCronjob1.Call()
            require.NoError(t, err)
            err = vCronjob2.Call()
            require.NoError(t, err)
        }
    })
}

```

```

        vCronjob1.time.AdvanceNow(30 * time.Second)
        vCronjob2.time.AdvanceNow(30 * time.Second)
    }
})
t.Run("Check merkle roots", func(t *testing.T) {
    root, err :=
getMerkleRootFromContract(vCronjob1.votingContract, 0)
    require.NoError(t, err)
    fmt.Println("Root at epoch 0:", root)

    root_1, err := getMerkleRootFromContract(vCronjob1.votingContract, 1)
    require.NoError(t, err)
    fmt.Println("Root at epoch 1:", root_1)
})

t.Run("Reset root of Epoch 0", func(t *testing.T) {
    vCronjob1.votingContract.ResetVoting(txOpts,
big.NewInt(0))

    fmt.Println("Waiting 2 seconds...")
    time.Sleep(2 * time.Second)

    root, err := getMerkleRootFromContract(vCronjob1.votingContract, 0)
    require.NoError(t, err)
    fmt.Println("Root at epoch 0:", root)
    root_1, err :=
getMerkleRootFromContract(vCronjob1.votingContract, 1)
    require.NoError(t, err)
    fmt.Println("Root at epoch 1:", root_1)
})

t.Run("Run voting clients 1 and 2, again", func(t *testing.T) {
    vCronjob1.time.SetNow(now)
    vCronjob2.time.SetNow(now)
    for i := 0; i < 10; i++ {
        err := vCronjob1.Call()
        require.NoError(t, err)
        err = vCronjob2.Call()
        require.NoError(t, err)
        vCronjob1.time.AdvanceNow(10 * time.Second)
        vCronjob2.time.AdvanceNow(10 * time.Second)
    }

    fmt.Println("Waiting 2 seconds...")
    time.Sleep(2 * time.Second)

    root, err := getMerkleRootFromContract(vCronjob1.votingContract, 0)
    require.NoError(t, err)
    fmt.Println("Root at epoch 0 After New Voting:", root)
})
}

```

Recommendation

Handle potential merkle root resets in the voting script. This might require tweaking the voting smart contract to report the epochs ids where the voting was reset.



Status

Fixed on commit `e4811f18f130c5199b865d3bb99a461e5ddf993f`.

Bot operators now can start running the script from a particular epoch by using the command line options `-reset-voting` and `-reset-mirroring`.

PSOS-005

Script will attempt to mirror epochs twice

Status Solved	Risk None
	
Resolution Fixed	Impact Recommendation
	Likelihood -

Location

`indexer/cronjob/mirror.go`

Description

The `epochRange.end` of a range will be processed twice. In the event of processing a range of length 1, it could imply sending a duplicate mirror transaction and thereby a waste of gas.

The for loop below processes every epoch in the range, including `epochRange.end`. Then, it updates `jobState.NextDBIndex` in the `updateJobState` function with the `epochRange.end` value.

```
for epoch := epochRange.start; epoch <= epochRange.end; epoch++ {
    ...
}
...
if err := c.updateJobState(epochRange.end); err != nil {
```

```
    return err  
}
```

In the next cycle, it sets the start epoch value (`epochRange.start`) to be equal to `jobState.NextDBIndex`. This is, the value of the `epochRange.end` last processed.

```
func (c *mirrorCronJob) getStartEpoch() (int64, error) {  
    jobState, err := database.FetchState(c.db, mirrorStateName)  
    if err != nil {  
        return 0, err  
    }  
  
    return int64(utils.Max(jobState.NextDBIndex, c.epochs.first)), nil  
}
```

Recommendation

The `updateJobState` function should set `jobState.NextDBIndex` as `epoch+1`. Otherwise, do not include `epochRange.end` in the for loop by removing the `=` from the `<=` operator.

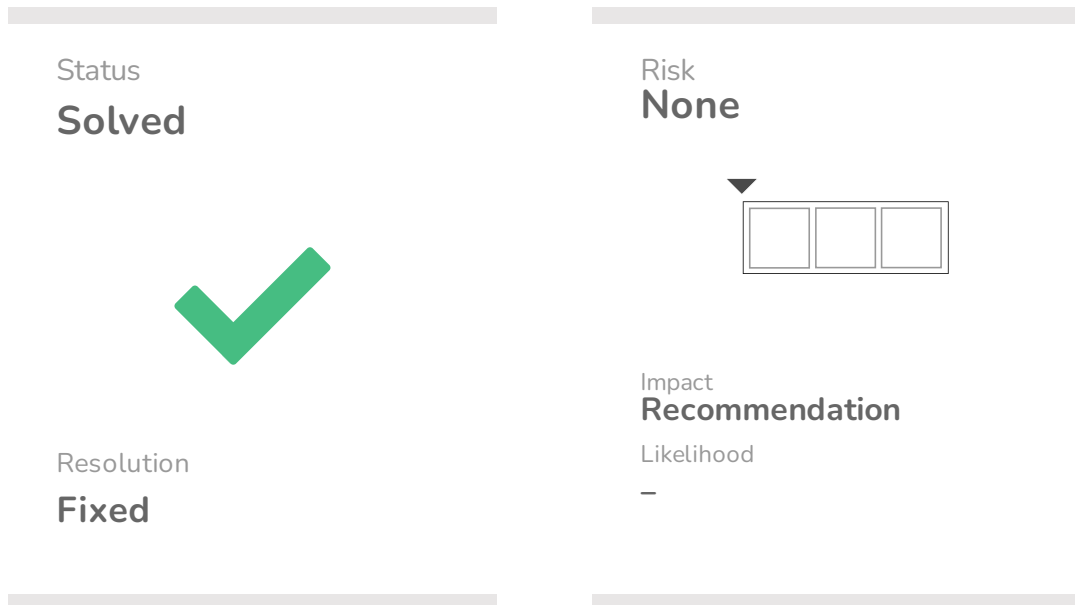
Status

Fixed on commit [9e3655b5fa50cb81280dcc53f514c733dd2c7c6f](#).

The `updateJobState` function now sets `jobState.NextDBIndex` as `epoch+1`.

PSOS-006

Key contextual time-based variables could be constant



Description

A mismatch between the actual contextual variables (start timestamp and epoch duration) and the configured by users will disarrange the epoch index calculation disrupting voting and mirroring.

Currently, users need to setup on the configuration file the first epoch start timestamp and the epoch duration. However, those variables are immutable and known from before, according to the PChainStakeMirrorMultiSigVoting contract:

```
// immutable settings
uint256 internal immutable firstEpochStartTs; // start timestamp
of the first epoch instance
uint256 internal immutable epochDurationSeconds; // duration of an
epoch instance
```

```
type EpochConfig struct {
    Period time.Duration `toml:"period" envconfig:"EPOCH_PERIOD" `
    Start  utils.Timestamp  `toml:"start" envconfig:"EPOCH_TIME" `
    First  uint64             `toml:"first" envconfig:"EPOCH_FIRST" `
}
```

This structure is error prone and the services might behave unexpectedly in the event of using different values than the ones submitted to the smart contracts.

Recommendation

Use constant values instead of configurable ones. Alternatively, make the contract variables public and cache their return values when starting the off-chain services.

Status

Fixed on commit [f935d10cb712ebb800e9b8ee2142a1e2a4345e73](#).

Contextual values are now retrieved from the deployed contracts.

PSOS-007

Service API disclosing internal error information

Status
Solved



Resolution
Fixed

Risk
None



Impact
Recommendation

Likelihood

–

Location

`services/utills/router.go`

Description

The service API discloses internal errors in the HTTP responses. Exposing such internal details aids adversaries in gathering information about the server, database, or software. This can subsequently be exploited to target and compromise the infrastructure.

For instance, when sending a POST request to `/exports/transactions` with the following request body:

```
{
  "address": "string",
  "limit": 0,
  "nodeId": "string",
  "offset": 0,
```



```
"time": "2023-09-21T17:50:00.785Z"  
}
```

The server returns internal information about the database structure:

```
Error 3065 (HY000): Expression #1 of ORDER BY clause is not in SELECT  
list, references column 'flare_indexer_services.p_chain_txes.id' which  
is not in SELECT list; this is incompatible with DISTINCT
```

Recommendation

Return generic error messages instead. If desired, log error these internal error messages.

Status

Fixed on [a37bf39e2c6d047806438c043b4a5b4c5b8709ba](#).

Logs now return a generic error message.

Disclaimer

The information presented in this document is provided “as is” and without warranty. Security Audits are a “point in time” analysis, and as such, it's possible that something in scope may have changed since the tasks reflected in this report were executed. This report shouldn't be considered a perfect representation of the risks threatening the analyzed systems and/or applications in scope.

File hashes

File hashes of the commit `ee0632512bd70d12e2b2738df9321a23e8b5e394`, of September 11th, 2023.

```
375c3926b7ad6023b8dd1578272aef846cbcf00e1dd1f47033b941fb018fae4a ./database/methods.go
9b3ed5a86f896c3cf041b3f708fc43849a5148e0c0bbce55f0d4f4e511897ca6 ./database/pchain_queries.go
bf6a7e0d6d0cfd258f5e3beaf4640293df1ece19e1e135be05337166ad95c7e ./database/types.go
f53e5e47b484dacc3b871c540a4d6b355b8558a17987b325ed2acbaee9e56129 ./database/queries.go
6ae416e0e136b7fd307df12dae3b4888b0e0ab75b51fe55ff68df03d39a2ae0c ./database/entity_utils.go
563b5e02ac69f438048b37cd10caec75510e8c1a3475a479362d618c5c6bc992 ./database/pchain_entities.go
a5500b068d54b1a93d475d030dc36661f9d81b46d43e9c479296ba0569ae0c35 ./database/creators.go
72cce777dc9a5d8052c7d944794181f1045261d0f9f478fc4058ff789c25788d ./database/utills.go
876def176dae9f05e10ea5a8ed4c272610f2c6db8c8e24e7760b8a161653a865 ./database/cronjob_entities.go
dada26cde92c0a6d5602c36ac09376edba17962adf043f4f054563f4f8380512 ./database/testing.go
f8f26b72e85663ad4be568608a9e2bf526274e047abebcfffca19c6827c54d04 ./database/entities.go
bd97335d117cc2beb1c941901b4c16ac230b81144f978aa7f808db882d75b551 ./logger/logger.go
14ec7310214a5301e11a5ca802444000a9c37fdd07e67de8a2d0377f3eb429 ./logger/colors.go
f1c6492c62ad8cbf1856240a3109b0952cb4320d1da566319275adda2882902f ./config/config.go
086eb26af541dc54c2f3aa14dc01c374369df492542fe1e72b405d04a45a878b ./config/callback.go
6302a2fc084f1ce3fed796efb8f7cb4133145f35789bab369a374b675e689482 ./utils/encoding.go
807ac0abd82bc379b3ff69f2f5f247a1be4ccd353b71f463f3ea5166e56b4a55 ./utils/cache_test.go
b454a74beb7b5e6e8da10415a2d9f3a0ca8b8579a3c611653ecdaf4ce7b087ca ./utils/address.go
c4ed82ac652c1a8b79f2e015ab58cc8b275ccb72b527f0fa6659aee74b0e2e0 ./utils/time.go
17ad9f76d72211fe22bba335619092c49ecc1b6cc984f32c7990bffd0a25a006 ./utils/contracts/voting/autogen.go
cd6c63c61f6e77da1f05eecb6e8cee0caeb8ea4403dbb1968c7015aaa466cbbf ./utils/contracts/voting/voting.go
b474696022be4eb2e17018de0aa2e65603d5180328d090cfeec562bf45dc60b
./utils/contracts/mirroring/autogen.go
e41fbc618f8f260ee0213871ae6fe0d1fd0c754d19976d13fea40ab41b4c7398
./utils/contracts/mirroring/mirroring.go
ee3ac541007429737631c0fa9219c03f1f725949163560993e6d32cdf5dbcdcc ./utils/cache.go
1519d2ec178ecb29d6f6daa39da04bd35453d27729d8ed0808c9d28bc24b394a ./utils/merkle/merkle.go
6ea34b8f7aa64fa043c93dd5572df50e4066e03072e7c0d78e493e81bb34fcd4 ./utils/merkle/merkle_test.go
b175e51239ebd04b494ad6cc0dd3b342c98bc2e6a7f73af57097775b93d31e37
./utils/chain/indexer_client_test.go
72b91b0f56cd4f8e796eb0b61ab1f58b171804f214561631c518dad71b07de3 ./utils/chain/indexer_client.go
e52abaf1facdbf40bd6b9a93f298d64ebde65228a72f9c7dac144d41a562ecbc ./utils/chain/client.go
5973141db5314b3ef2190f6f50badd8020a791859a62fa1b9497e0901522141 ./utils/chain/p_chain_rpc_client.go
956415f58c14219f09a5283dd6a6f5291a7dce1d6fc1e09c9cf81be276d70dff
./utils/chain/p_chain_rpc_client_test.go
bc185a0231eea05846d111dd3242e74905f0b65833864a7e88a8619d2d710ff3 ./utils/chain/uptime_client.go
94a22a25ae5875ff7772d5b1cbbbc507c95e8e8e76aca0bb7838ebc2ba56022fb ./utils/chain/uptime_client_test.go
8bed42e03d7fd8202ba83f4e29d0c066414bcc01111b68cd31deea4468edb72 ./utils/chain/testing.go
d6d7c84ee5ce4d3f98c46ccba5cdc08766a97c83882a145a7e598a324557b584 ./utils/url.go
62bfc0b2de4771984c7295ba7ad0d5f74ee97608b3c9017f676f9e458a8221aa ./utils/toml.go
98e287b61a63836dd79db06e7ca055e0d744001ef102b26c174757ecdb03a0e1 ./utils/math.go
827d97a9780269460088e0f85541db22bbdf1b09a266f79dfed2dcf50c3ec916 ./utils/errors.go
294dd66551b7b0a81fc4fd9822271f75e23c5f24099f0d33139a6dee736d9ea8 ./utils/structures.go
0fefff438b4a2bc69703b48b9d695791eab08d7406c4405c6f3ed3f2bb7b64073 ./indexer/pchain/indexer_test.go
df0f42c56848a9545efce3969bfc45bcb82e28a7687197683267f02b8acd037 ./indexer/pchain/batch_indexer.go
f478823bfc2af47f06599b39003d6b035ab75d8ff7880d6625a9a2874eb9a73a ./indexer/pchain/entity_creator.go
ef5cb73bb33324f1c5f247a7604c6f31dc2a83e17403d866e98f66802908735 ./indexer/pchain/migrations.go
14e4d401b1660dd6d47d224197ff990d2c52a46f5d304e0ddf045b8a2a9ad56e ./indexer/pchain/in_updater.go
8b1a4ca5d238bcf878465ea5fb724a754321f35722236d471d1fcb71579927b4 ./indexer/pchain/indexer.go
8801701b4c9e9f746fa288b06e4815916ecde11bc24d7d3cd59efc5d5578699e ./indexer/pchain/utills.go
9cde08026adc0ff0dbcb7618d10acadc37e5fba5ac6f76f36f99866a2549e6b8 ./indexer/pchain/main_test.go
2ca728fc82e9829979d2600197f4bf9973a47d89b56a6efe59bcb4b056483c53 ./indexer/migrations/container.go
9ff32583b70ba35a045a607677fc0b4da268519b3cd07a8e9ea18ae566cf8cdc ./indexer/context/context.go
```

```
014b96b073fe35d99c673dd8d9a7cd68f097204099eb38e3a00a0c42af2826f6
e81268e5e6f2120373047b8c7bfe08f1be67e4c46f5d80720e6d9e3f5e2b31ca
1dab200cfecdb728b8e78aa4c8d991c61f4bc7ec15d11274034a57fe61184516
6660cb5b40f6d5caa4fbfa6d70c6934ec33b56d74b54476da76423d01188e946
6023c662a56a9f659bab53b02450f613582b03b9c4483c38b4b28a272ab1af9a
6566fa4ca799a51ad70ee5944d7c6617f780f86114f244f7abbedf722a5c97c9
749230f955d75e75c3f8940a7e7f36b8fab5a7bcb9bbac3da42ef735143f7574
03804504cd4119415caa459e2081115ad25f603859fbbba5a50bab9612b75eba3
913e50a79e6297a0b9fe8ba5811050d22e031b232e14a2fd1607210ba27b0bbe
6ed4fde06d204227e4689ce4d6dbc1b250648409987c9c77b268ccd00fa4d5e9
ebf112693eebbcbea43cb89e4e5fe2762ce14030dc0ceddf7ec7167440bf0e79
3768301b949bbeb944818d8ba0bcb8360e249cf91905b27eb4e12171a15f8912
bafb6dd54bf84844eb7e667560b0a55f600a8e04b18d3ae2581eccad031ccfce
c91fa8eb9c44753acae45fbb02203dfb246969d6c4a86142762abdc581b7b527
d8f743b36b9712dbf5138a19e0a5fae3a0c3e25ab769889b0b3e86b4cfe8e938
./indexer/cronjob/uptime_voting_test.go
42ea489b4248d0e0cb973df8763659da28dde5852929bd5f05995288bef6aef9
a47145207e1d589c9a177a19b4ce53c55d47a23b1ef12b5bc727e48b033250d8
af0001daecca65b2973ea3bfa3bed4a2f9d6de1f5c6fff0ed13b52ef9104bfa04
8eaa49996b48f226ef3a37524ad1389a00da0a06148e04c103db4dd6c06d606e
106567e604abb67607f7e05d0a24ac2481ef55432218cd93029dcf772572c24c
fe0de45a263d08352a42fb986956c0db25746437b607b0b5b7c416ba2804acd74
d5b055aa8cecc51c87acd5cd7d892b3f0023600b3fdb5a3765982f85d806ea1d
ba0efae2c0ca55dd794113f8f134ef24c75a34c954acd1b7fb0d6596d212e30e
632a3d9f0a6c17d0a0bd82100076e3bd970e528aafc2fb3337a5fc1cb28cc6ef
8233cc62d0d4782710ffe1ef2eb15a8ca911c036068e5a5a828e10b37fc47e87
df96990da592f11d24c191364bc75f2ae8d38e4c9f92fdbc68a6bcd04544e00e
6ebbb8792ffe8061a9aced38feec0e613b52b8c96123a208959606e5471d560c
ee20cae9ad5d385cb0623ec3f5acd7b2a65b75223c83d7465f3bc92ca70b3005
0fd449aef232097a67e27606d1d781ee9c6b7d180cd4d28546d179983823734e
f8508962c659930780eced801b50cc5af7686ab8d465331e3b996a3323e6d247
4c1f2895e46803a7af18040862355d2cab140a8a5913366ec7532b4f34306b2b
303dc6dc816fc713822ff4c17c88fd6f0bfa103c4eeb38597a35a7e007b8fa9b
6e0c1a13c6c51b72fb26d66551e8ed4c34c0f6432395d7ace3bd8ccf19d04980
3b007dfde6aabc2a0a21f5261be1c00203533189aa90fe0a7c7c5f15e0aff7b2
d43b9bae78f88fe14ccd88df1004c8bb68bf4095bfd1c2c7eb52f4b974da9bab
6acacd74704de91ddf89adbe4d16773e7651c7aac58c75b99ab61a12ab443656
31132d4ecf58a332079c52e6d7e0f1b764457746e32df82bc210036edd0d1aae
3d0691a28399fbc7edd126788a77a9b8a50b2867750952883d99482688ecb7e5
1d091cde41022f072beffb7577387508680063fe20193e427f53a22774b31c04
efbafb47668da5cc51a67690e03d8c2fa1be76994bde3d8c8e3406bb50009f8c
./indexer/context/testing.go
./indexer/config/config.go
./indexer/runner/runner.go
./indexer/shared/types.go
./indexer/shared/indexer_metrics.go
./indexer/shared/inout_indexer.go
./indexer/shared/base_tx.go
./indexer/shared/in_updater.go
./indexer/shared/indexer.go
./indexer/cronjob/mirror.go
./indexer/cronjob/uptime_voting.go
./indexer/cronjob/voting.go
./indexer/cronjob/migrations.go
./indexer/cronjob/utills.go
./indexer/cronjob/uptime_test.go
./indexer/cronjob/uptime.go
./indexer/cronjob/voting_test.go
./indexer/cronjob/cronjob.go
./indexer/cronjob/main_test.go
./indexer/main/indexer.go
./services/context/context.go
./services/context/testing.go
./services/config/config.go
./services/utills/encoding.go
./services/utills/services.go
./services/utills/validate.go
./services/utills/testing.go
./services/utills/router.go
./services/api/pchain.go
./services/api/attestation.go
./services/api/shared.go
./services/main/services.go
./services/routes/query.go
./services/routes/query_test.go
./services/routes/staking.go
./services/routes/types.go
./services/routes/transactions.go
./services/routes/transfer.go
./services/routes/main_test.go
```

File hashes of the commit c38fa4913cfd529a764ced791e111e22f5217c7f, of September 20th, 2023.

```
375c3926b7ad6023b8dd1578272aef846cbcf00e1dd1f47033b941fb018fae4a
58bab7915fd8bd0af8398aa7df4531f39c322f87052401013e8e730148978267
bf6a7e0dd6d0cfdb258f5e3beaf4640293df1ece19e1e135be05337166ad95c7e
f53e5e47b484dacc3b871c540a4d6b355b8558a17987b325ed2acbaee9e56129
6ae416e0e136b7fd307df12dae3b4888b0e0ab75b51fe55ff68df03d39a2ae0c
563b5e02ac69f438048b37cd10caec75510e8c1a3475a479362d618c5c6bc992
a5500b068d54b1a93d475d030dc36661f9d81b46d43e9c479296ba0569ae0c35
72cce777dc9a5d8052c7d944794181f1045261d0f9f478fc4058ff789c25788d
876def176dae9f05e10ea5a8ed4c272610f2c6db8c8e24e7760b8a161653a865
dada26cde92c0a6d5602c36ac09376edba17962adf043f4f054563f4f8380512
f8f26b72e85663ad4be568608a9e2bf526274e047abebcfffca19c6827c54d04
bd97335d117cc2beb1c941901b4c16ac230b81144f978aa7f808db882d75b551
14ec7310214a5301e11a5ca802444000a9c37fddf07e67de8a2d0377f3eb429
25a7478144737775a5bcbfb88aafe58d665c724788bf1969d5824f7606f8adf4
086eb26af541dc54c2f3aa14dc01c374369df492542fe1e72b405d04a45a878b
6302a2f0c884f1ce3fed796efb8f7cb4133145f35789bab369a374b675e689482
807ac0abd82bc379b3ff69f2f5f247a1be4ccd353b71f463f3ea5166e56b4a55
c4ed82ac652c1a8b79f2e015ab58cc8b275ccbd72b527f0fa6659aee74b0e2e0
17ad9f76d72211fe22bba335619092c49ecc1b6cc984f32c7990bffd0a25a006
cd6c63c61f6e77da1f05eeceb6e8cee0caeb8ea4403dbb1968c7015aaa466cbbf
c78f5d7dda7a53ffed8f6bdc116545b502a99e8347ccf9fc24ce52d4b2564141
./database/methods.go
./database/pchain_queries.go
./database/types.go
./database/queries.go
./database/entity_utills.go
./database/pchain_entities.go
./database/creators.go
./database/utills.go
./database/cronjob_entities.go
./database/testing.go
./database/entities.go
./logger/logger.go
./logger/colors.go
./config/config.go
./config/callback.go
./utills/encoding.go
./utills/cache_test.go
./utills/time.go
./utills/contracts/voting/autogen.go
./utills/contracts/voting/voting.go
```

./utils/contracts/addresses/autogen.go
4169db264b0c08e2629b3e5caa2bf7a252b6eb9f9be27fbfd587a067427e42eb9
./utils/contracts/addresses/binder.go
b474696022be4eb2e17018de0aa2e65603d5180328d090cfeec562bf45dc60b
./utils/contracts/mirroring/autogen.go
e41fbc618f8f260ee0213871ae6fe0d1fd0c754d19976d13fea40ab41b4c7398
./utils/contracts/mirroring/mirroring.go
ee3ac541007429737631c0fa9219c03f1f725949163560993e6d32cdf5dbcdcc
1519d2ec178ecb29d6f6daa39da04bd35453d27729d8ed0808c9d28bc24b394a
6ea34b8f7aa64fa043c93dd5572df50e4066e03072e7c0d78e493e81bb34fcd4
2133684912c3e554344b7694f76686ab2f32d05cf71a1f87537bc594af4fe763
./utils/chain/indexer_client_test.go
8dabe0db49cc194b61817d1e22c8b624305b3d31d60dc28a6f22e6efdb76d9f
72b91b0f56cd4f8e796eb0b61ab1f58b171804f214561631c518dad71b07de3
e52abaf1facdbf40bd6b9a93f298d64ebde65228a72f9c7dac144d41a562ecbc
5973141db5314b3ef2190f6f50badd8020a791859a62fa1b9497e0901522141
956415f58c14219f09a5283dd6a6f5291a7dce1d6fc1e09c9cf81be276d70dff
./utils/chain/p_chain_rpc_client_test.go
bc185a0231eea05846d111dd3242e74905f0b65833864a7e88a8619d2d710ff3
1c12d9c8c0a438dce8a52f2007e7e4fa99693ef8d893174a8617bc2793ddd8e
8bed42e03d7fd8202ba83f4e2e9d0c066414bcc01111b68cd31deea4468edb72
2957afdea1ecedb3a9228b39d91ee25211e66109e042d982eb1ee9e118404f08
c601f256c0cb20bd7eddf263ca2d513af492f3fddd5a3329c3a5666efef3809e
d6d7c84ee5ce4d3f98c46ccba5c5dc08766a97c83882a145a7e598a324557b584
62bfc0b2de4771984c7295ba7ad05f74ee97608b3c9017f676f9e458a8221aa
98e287b61a63836dd79db06e7ca055e0d744001ef102b26c174757ecdb03a0e1
827d97a9780269460088e0f85541db22bbdf1b09a266f79dfed2dcf50c3ec916
294dd66551b7b0a81fc4fd9822271f75e23c5f24099f0d33139a6dee736d9ea8
05287575644134c4cc503c0f5c767f66d3095a849e73dff89852439f1addcb2d
df0f42c56848a9545efce3969bfc45bcbb82e28a7687197683267f02b8acd037
f478823bfc2af47f06599b39003d6b035ab75d8ff7880d6625a9a2874eb9a73a
4f5cb73bb333324f1c5f247a7604c6f31dc2a83e17403d866e98f66802908735
14e4d401b1660dd6d47d224197ff990d2c52a46f5d304e0ddf045b8a2a9ad56e
8b1a4ca5d238bcf878465ea5fb724a754321f35722236d471d1fcb71579927b4
8801701b4c9e9f746fa288b06e4815916ecde11bc24d7d3cd59efc5d5578699e
f67d41b2f28914ae2802b9b74200240ee714bca5d4d85b939be9dccc3a84dad80
2ca728fc82e9829979d2600197f4bf9973a47d89b56a6efe59bcb4b056483c53
9ff32583b70ba35a045a607677fc0b4da268519b3dc07a8e9ea18ae566cf8cdc
014b96b073fe35d99c673dd8d9a7cd68f097204099eb38e3a0a0c42af2826f6
d4cd97b71e23efedba10a6111608c04a8edc0a87a49661b8ba21acd5b8e1aa84
1dab200cfeadb728b8e78aa4c8d991c61f4bc7ec15d11274034a57fe61184516
6660cb5b40f6d5caa4fbfa6d70c6934ec33b56d74b54476da76423d01188e946
6023c662a56a9f659bab53b02450f613582b03b9cd483c38b4b28a272ab1af9a
6566fa4ca799a51ad70ee5944d7c6617f780f86114f244f7abbedf722a5c97c9
d207f31f9b320450427696c33453020d7749e6b527fffd74c682652b4461aed1f
03804504cd4119415caa459e2081115ad25f603859fbba5a50bab9612b75eba3
913e50a79e6297a0b9fe8ba5811050d22e031b232e14a2fd1607210ba27b0bbe
85e7432f2c1123ab3fd0be10e1379987f472ab1f582c787bf4f22f86dd94a434
5fa49f501112952babd06045aec7a4eb8705a897b810d445dd8782025ffac8f4
6a168b2e02994b1d0a1f15265d78d6c259cc30533e5e061eba42fef56b514119
./indexer/cronjob/voting_integration_test.go
567da2348064e2caabde0b170dfb7b8121a6a462e4217cc69a3349032a7c0a98
dac2892645b9bc5b230334717e00bdcf4686948f1dab7155caa4b56bf79b5395
baf6bd54bf84844eb7e667560b0a55f600a8e04b18d3ae2581eccad031ccfce
8efc41678532ad7d8bb2da4eb0118b03ad6c792c3d2f8483cfd3696bb10e98e1
7495f3028a309743286730f4c5ffc1e257e67119f3eb2cfff87359a6f2dc5543
e37596f9e7cea07dc21fa8da1d3874bc83119d5ffcc735120782783def20f3b2
./indexer/cronjob/uptime_voting_test.go
1ebe02289f3b4cc07ad367659100ab2e34b86ece9b33913223c57c64e6a65d3
dd5225148effdf57745cccd310b661bb17305926e02c5da42f028cf6c1b491f0
a47145207e1d589c9a177a19b4ce53c55d47a23b1ef12b5bc727e48b033250d8
8366d6c1ef091150ea027da681e172c57d94716ff3f14d108a85584236352bf0
4bb6c72dee6a1e0eaab241a9d6a31bfc2151008bccfcc6ac2e7e596ccefc8a1b
0d1222445fade1aaa536097b3360aee7fed43465f53eb98a6af721f622f96c72
fe0de45a263d08352a42fb986956cdb25746437b607b0b5b7c416ba2804acd74
d5b055aa8cec51c87acd5cd7d892b3f0023600b3fdb5a3765982f85d806ea1d
ba0efae2c0ca55dd794113f8f134ef24c75a34c954acd1b7fb0d6596d212e30e
45cd6ce6b49e56d9abe37426270d2aa61e994a3f15e3929bfa8c5fcf06af604d
daecef5f08fdfac028a8e990bcd24e6271d62e83aa428652a5bb7710610ec206

```
df96990da592f11d24c191364bc75f2ae8d38e4c9f92fdb68a6bcd04544e00e ./services/utils/services.go
6ebbb8792ffe8061a9aced38feec0e613b52b8c96123a208959606e5471d560c ./services/utils/validate.go
ee20cae9ad5d385cb0623ec3f5acd7b2a65b75223c83d7465f3bc92ca70b3005 ./services/utils/testing.go
22813667ea76d2a8b2dac3d1bf0c8c65ce30c7180e4f7d8bb2f262d8b72aea13 ./services/utils/router.go
f8508962c659930780eced801b50cc5af7686ab8d465331e3b996a3323e6d247 ./services/api/pchain.go
4c1f2895e46803a7af18040862355d2cab140a8a5913366ec7532b4f34306b2b ./services/api/attestation.go
303dc6dc816fc713822ff4c17c88fd6f0bfa103c4eeb38597a35a7e007b8fa9b ./services/api/shared.go
ead2c1d2cecb0acbb01d939f372b5aeba30500bb657056054fb4674a07e483e6 ./services/main/services.go
3b007dfde6aabc2a0a21f5261be1c00203533189aa90fe0a7c7c5f15e0aff7b2 ./services/routes/query.go
4d05d06931ca7c145898a6d9e1052f133faeb6feb0ca5c8395ab5c7e1fde8a88 ./services/routes/query_test.go
6ff8ef94ac1e8c959f2f183922381a8a7348383d97d319e09c530dc39d7aa8e4 ./services/routes/staking.go
31132d4ecf58a332079c52e6d7e0f1b764457746e32df82bc210036edd0d1aae ./services/routes/types.go
5790e734cdf1f6a8bcaede7fb93b082f151dfdddc9740bd584ec7a9bbcbab2ab ./services/routes/mirroring.go
3d0691a28399fbc7edd126788a77a9b8a50b2867750952883d99482688ecb7e5 ./services/routes/transactions.go
bc77c0080b451bb826f4b4aab2d24f4ef6f1d732822d37bd785906c3337b26fb ./services/routes/transfer.go
d5f92ebca70efe65baf0861c6f3751d889e5208d6644379d4170785dea5a8783 ./services/routes/mirroring_test.go
aca99b7167b536c182413a7e36a6498e25f0d8c69c1267b0d422d1ed7f298c77 ./services/routes/main_test.go
```