coinspect

You build, we defend.

flare

Source Code Audit

FDC v2

Nov 2024

FDCv2

Source Code Audit

Version: v250113 Prepared for: Flare November 2024

Security Assessment

1. Executive Summary

In **November 2024**, **Flare** engaged Coinspect to perform a Source Code Audit of the second version of the Flare Data Connector codebase. Coinspect was specifically tasked with identifying security risks in updates to the previously reviewed repositories, as well as in the newly added verifier and indexer-related repositories.

✓ Solved	⚠ Caution Advised	✗ Resolution Pending
High 0	High 0	High 0
Medium 2	Medium 0	Medium 0
Low 5	Low 1	Low 0
No Risk 9	No Risk 0	No Risk 0
Total **16**	Total **1**	Total **0**

Coinspect identified three medium-risk issues, including fee inefficiencies caused by the use of a fixed transaction fee, the use of a default valid and compliant payment reference string for invalid references, and discrepancies in source address roots or UTXO coinbase transactions between the indexer and the MCC library.

Additionally, the assessment highlighted seven low-risk problems related to:

- The indexer API returning wrong or invalid information,
- The possibility of voter transactions being rejected due to potentially insufficient fees,
- The storage of plaintext credentials in the indexer framework,

- An insecure default block confirmation value on the indexer framework,
- The possibility of sanctioned addresses causing a Denial-of-Service on given UTXO transactions within block ranges,
- The indexer database returning inconsistent responses for different underlying chain information,
- The possibility of causing a Denial-of-Service to the Flare System Client by the reward distribution data source.

Lastly, it is worth highlighting the number of informational issues included in the report.

Overall, accurately assessing the risk of each issue was inherently complex, as the majority of the reviewed repositories serve or are utilized by the core logic application layer, which falls outside the scope of this review. Consequently, the risk level depends on how their information is implemented and utilized within the broader system.

2. Summary of Findings

This section provides a concise overview of all the findings in the report grouped by remediation status and sorted by estimated total risk.

2.2 Finding where caution is advised

Issues with risk in this list have been addressed to some extent but not fully mitigated. Any future changes to the codebase should be carefully evaluated to avoid exacerbating these issues or increasing their probability.

Findings with a risk of None pose no threat, but document an implicit assumption which must be taken into account. Once acknowledged, these are considered solved.

Id	Title	Risk
FDC-019	Inconsistent indexer database State responses	Low

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
FDC-012	Attacker can use null-payment reference as a valid payment reference on UTXO chains	Medium
FDC-018	Mismatching source address root for UTXO coinbase transactions	Medium
FDC-007	XRP indexer provides wrong response	Low

FDC-008	Voters using Type-2 transactions risk having their transaction rejected	Low
FDC-010	Indexer framework stores database credentials as plaintext	Low
FDC-011	Indexer framework has an insecure default confirmation value	Low
FDC-020	ReadAll execution exposes Flare System Client to Denial-of-Service	Low
FDC-006	Voters using Type 2 transactions waste funds by having a set priority fee	None
FDC-009	Slice data can be lost as return variable is not replaced	None
FDC-013	XRP verifier and indexer disagree on which transactions are native	None
FDC-014	Misleading log message	None
FDC-015	Sanctioned address can cause a Denial-of-Service on ANYONECANPAY UTXO transactions	None
FDC-016	Anyone can prevent an input from being associated to their UTXO wallet address	None
FDC-017	UTXO and XRP source addresses are inconsistent	None
FDC-021	Fetching signing policies from previous relay contract versions	None
FDC-022	Inconsistent payment reference processing	None

3. Scope

The scope was defined to include the following repositories at their specified commits:

- https://gitlab.com/flarenetwork/fdc/evm-verifier, commit **9c648a5e445b6b360dbb9c9fa9d93cd90f85792f**
- https://gitlab.com/flarenetwork/libs/go-flare-common, commit **2bb70a08182cb2fa032ee138843c0e6856f45d9d**
- https://gitlab.com/flarenetwork/fdc/fdc-client, commit **1bf98b05d3307dd1535ea2ca59988f11ea621fdb**
- https://gitlab.com/flarenetwork/flare-system-client, commit **6caa7b61f4794588de5d1564fb19fa50f61f691d**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-framework, commit **706f9e280fc6067bb90d8e92043973de2cf13499**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-api, commit **4a4a863bc5021b64d7fdba597f567d8a2cd5c5ff**
- https://gitlab.com/flarenetwork/fdc/verifier-utxo-indexer, commit **7bbe6621ccd7d2c2d20b82937d5dd0138bb82e4b**
- https://gitlab.com/flarenetwork/fdc/verifier-xrp-indexer, commit **664c80164be1fecca22e95deae2818b6a3247b0a**

On November 21, Flare provided commits that updated the code of the projects to support a new field for transactions called `SourceAddressRoot`. The new commits were only analyzed with this functionality in mind and by diffing them with the previously provided commits. The updated commits are:

- https://gitlab.com/flarenetwork/fdc/evm-verifier, no update
- https://gitlab.com/flarenetwork/libs/go-flare-common, no update
- https://gitlab.com/flarenetwork/fdc/fdc-client, commit **94399c4315132824298c7db901196b888cfdb890**
- https://gitlab.com/flarenetwork/flare-system-client, commit **390e4e632ce96b2a641708d626201e69625146a2**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-framework, commit **3009437a61fa2457f4d5d5b2b79971c796c38efe**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-api, commit **daa1937df6bab7d911b7239fa3c75d6aa8ae2bbf**
- https://gitlab.com/flarenetwork/fdc/verifier-utxo-indexer, commit **f17e70312b94380b426fb162f9b013fbe51427b8**
- https://gitlab.com/flarenetwork/fdc/verifier-xrp-indexer, commit **c09a0fd8ee5f270b9604a8de0ab26d245c93fc88**
- https://gitlab.com/flarenetwork/fdc/mcc, diff between commits **b312535c73707ffdeebf47f5ba9b90435a73e85f** and **b1c75793adb928bd6c3bd50261c7bf5c884c3838**

Coinspect identified the `py_flare_common` Python library, maintained by the Flare team, as a key component for computing the Merkle tree and the Source Address Root. However, the team was unable to investigate the behavior of the `MerkleTree` function when provided with an empty array of addresses due to the lack of access to the library.

3.1 Fixes review

on December 18, Flare contacted Coinspect to perform a fix review. The fix review was carried in the following commits:

- https://gitlab.com/flarenetwork/fdc/evm-verifier, commit **c105809f0b444a977be1e01d9a0a735a1fd995ab**
- https://gitlab.com/flarenetwork/libs/go-flare-common, commit **aeaae3b73cabd56024df220f36523ddca2e0d4ba**
- https://gitlab.com/flarenetwork/fdc/fdc-client, commit **78b38a479e27a2c12015f6029e145d992672f03c**
- https://gitlab.com/flarenetwork/flare-system-client, commit **b7deb363d38f36f478489953d1e552b10037b0ce**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-framework, commit **c7bd102cb88a984ca2adda96544acccd27bd2cb6**
- https://gitlab.com/flarenetwork/fdc/verifier-indexer-api, commit **29e4c0a419d291249e767640324cd1f3becafddc**
- https://gitlab.com/flarenetwork/fdc/verifier-utxo-indexer, commit **7faaf78f97f8f48ba02d536b199adebe2826bed0**

The fix review only dealt with changes related to the issues detailed in this report.

4. Assessment

The Flare Data Connector protocol (FDC for short) is a system of interconnected systems which aim to provide an oracle service for the Flare chain. The system as a whole consists of both on-chain and off-chain programs which work together to identify valid voters for the oracle data, gather their signatures and provide third-parties with a way to verify arbitrary data on-chain.

This particular review had a primary and secondary targets:

1. **Main target**: Indexing API and attestation types implementations
2. **Secondary target**: Updates to the FDC client and the Flare System Client (FSC for short)

The threat model for the indexers, the attestation types and the FDC/FSC clients are quite different.

On the indexers side, the biggest threat identified is an attacker that can generate a transaction in one of the supported chains that is *not* parsed correctly or unable to be parsed. For example, an adversarial agent in the FAsset system could try to move collateral without it being detected by the indexer (see MCC-1 for an specific instance of this threat). An attacker can also try to generate transactions that cause the system to halt to cause a denial of service (see ATC-30).

For attestation types implementations, the concerns are similar: the implementations must match the specification exactly, and it should not be possible for an adversary to create transactions that for all intent and purposes *should* signal a certain event, but the system does not identify them as such (for an example, see ATC-09). Attestations also need to prevent attackers from leveraging them against users due to having wrong assumptions (as in ATC-22).

The FDC and FSC are systems that work in tandem. While the FDC prepares the correct (from the voters' point of view) merkle root, the FSC is responsible for querying it and posting the signatures on the Flare blockchain when the threshold is reached. Note that a critical assumption is that the FDC and FSC trust each other and the FDC trusts the verifier servers that report data about the attestations.

For these systems, the threat model is mainly concerned with a denial of service attack. Other threats, such as signature verification, while critical, are not the main responsibility of the off-chain components. These checks can be found on the Relay.sol contract, which was out of scope for this review.

Reviewers identified as the most important changes to the FDC/FSC repositories the following additions:

- Support for new `Type 2` transactions
- Changes in the `FSC`'s `finalizer` package, which now uses an additional channel the `messageChannel`, to which the voter's own votes are sent when they `submitSignatures` on-chain.

It is worth mentioning that this project's scope included reviewing the Verifier Indexer Framework, a blockchain-agnostic, generic framework written in Go for creating indexers. However, it is important to note that the UTXO indexer reviewed does not implement this framework and is instead written in Python. The lack of a unified ecosystem not only exposes the project to higher maintenance costs but also requires additional resources to manage compatibility issues, code consistency, and integration efforts across different platforms.

4.1 Security assumptions

1. The JSON-RPC contacted by the indexer is trusted by the operators
2. The verifiers used by the FDC voters are trusted by the voters
3. More than half the voters are honest and live

5. Detailed Findings

FDC-006

Voters using Type 2 transactions waste funds by having a set priority fee

Status

Solved

✓

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

–

Location

```
flare-system-client/utils/chain/tx_utils.go
```

Description

Voters taking advantage of the Type-2 transactions will see their funds wasted due to the system using a set max priority fee, either hardcoded on the code or via the configuration.

To understand the issue, consider the following snippet in the `SendRawType2Tx` method:

```
        tipCap := new(big.Int)
        if gasConfig.MaxPriorityFeePerGas != nil &&
gasConfig.MaxPriorityFeePerGas.Cmp(big.NewInt(0)) == 1 {
                tipCap.Set(gasConfig.MaxPriorityFeePerGas)
        } else {
                tipCap.Set(DefaultTipCap)
        }

gasFeeCap = gasFeeCap.Add(gasFeeCap, tipCap)

// ... redacted for brevity...

txData := types.DynamicFeeTx{
                ChainID:   chainID,
                Nonce:     nonce,
                GasTipCap: tipCap,
                GasFeeCap: gasFeeCap,
                Gas:       gasLimit,
                To:        &toAddress,
                Value:     value,
                Data:      data,
        }
```

The `tipCap` is a value obtained either directly from the configuration or a default (set at 20 GWei). In any case, the result is the same: whenever the priority fee for inclusion is lower than the set priority fee, the voter will be wasting funds, as the difference will be burned.

Note that the current calculation goes against Avalanche's recommendations on gas-fee calculations, which states that the transaction's priority fee should be calculated using the `eth_maxPriorityFeePerGas` endpoint.

Recommendation

Change the configuration parameters so that users can set a bound to `MaxPriorityFeePerGas` instead of using that value directly.

The value should be gotten from the node's `eth_maxPriorityFeePerGas` endpoint and increased by a small multiplier, as it is likely the protocol wants to prioritize inclusion.

Status

Acknowledged. Flare has stated that `submit1`, `submit2` and `submitSignatures` transactions are refunded via a consensus-level mechanism. In light of this, the issue's severity has also been lowered from `medium` to `informational`.

Flare stated that other minor concerns reflected in this issue such as inclusion speed and gas pricing are not a priority right now and will be revisited later.

FDC-007

XRP indexer provides wrong response

Status

Solved

Resolution

Fixed

Risk

Low

Impact

Low

Likelihood

Low

Location

`verifier-indexer-api/src/entity/xrp-entity-definitions.ts:92`

Description

The `XRP` entity definition is currently returning XRP transactions from the database with an incorrect `chainType`, causing the `transactionsWithinBlockRange` API function to return inaccurate data.

As shown below, the `chainType` property is set to `DOGE` instead of `XRP`:

```
id: 0,
chainType: ChainType.DOGE, // TODO: add a chainType variable
transactionId: this.hash,
```

Furthermore, Coinspect identified several `TODO` comments scattered throughout the file and the repository, suggesting that the `verifier-indexer-api` might still be in the development phase.

Recommendation

Fix the `chainType` value returned. Review pending `TODO` comments.

Status

Fixed by commit `29e4c0a419d291249e767640324cd1f3becafddc`. The `chainType` is now correctly returned as `XRP`.

FDC-008

Voters using Type-2 transactions risk having their transaction rejected

Status

Solved

Resolution

Fixed

Risk

Low

Impact

Low

Likelihood

Low

Location

```
flare-system-client/utils/chain/tx_utils.go
```

Description

Voters that set their `BaseFeePerGasCap` are at risk of having their transaction rejected because the result from `eth_baseFee` might not be enough to cover the transaction.

The base fee is a dynamic value that cannot be predicted accurately by users. All users that set the `baseFee` are at risk of having transactions rejected when the base fee spikes.

Recommendation

Allow the user to set independent values for `maxPriorityFeePerGas` and `maxFeePerGas`. Users should not set the `BaseFeePerGasCap`, as the base fee is a protocol-dictated value that users have, in general, no way of predicting.

Because the blockchain only charges the minimum between `(maxFeePerGas, baseFee + maxPriorityFeePerGas)` users should set the absolute maximum they are willing to pay in `maxFeePerGas` configuration, while setting a smaller priority fee in `maxPriorityFeePerGas` which should cover the priority fee for normal scenarios.

Status

Fixed. The usage of the `BaseFeePerGasCap` parameter is discouraged in the FSC README file. This parameter is required for testing purposes mainly.

FDC-009

Slice data can be lost as return variable is not replaced

Status

Solved

Resolution

Fixed

Risk

None

Impact

Recommendation

Likelihood

–

Location

`fdc/fdc-client/client/attestation/verification.go`

Description

The `slices.Replace` method is used in the `Response::addRound` method, but the result is discarded:

```
_ = slices.Replace(r, roundIDStartByte, roundIDEndByte, roundIDSlot...)
```

While in this particular scenario the variable being replaced is inconsequential, because the length of the underlying array is not modified, calling `slices.Replace` and discarding the result is not recommended: depending on the length of the arguments the slice might now represent an outdated view of the underlying array.

Recommendation

Assign to `r` the result of the call to `Replace`.

Status

Fixed by commit `78b38a479e27a2c12015f6029e145d992672f03c`. `r` is now assigned.

FDC-010

Indexer framework stores database credentials as plaintext

Status

Solved

✓

Resolution

Fixed

Risk

Low

Impact

Low

Likelihood

Low

Location

`fdc/verifier-indexer-framework/pkg/config/config.go`

Description

The indexer framework stores the `postgresql` databases credentials in a `toml` file. This prevents operators from using secret management solutions offered by cloud services, which provide features such as key rotation and monitoring.

Recommendation

Allow operators to store the credentials in environment variables.

Status

Fixed in commit **c7bd102cb88a984ca2adda96544acccd27bd2cb6**. Database credentials can now be specified via environment variables.

FDC-011

Indexer framework has an insecure default confirmation value

Status

Solved

✓

Resolution

Fixed

Risk

Low

▼
🟥 □ □

Impact

Low

Likelihood

Low

Location

```
fdc/verifier-indexer-framework/pkg/config/config.go
```

Description

The indexer framework uses a default `indexer` configured with only one confirmation. Most blockchains systems do not have single-block confirmation, making this default overly optimistic and making operators that forget the configuration parameter ingest non-confirmed blocks.

Recommendation

Make the number of confirmation blockchain dependent. The indexer framework should know which blockchain is the underlying one, and provide an appropriate number of confirmations.

Note that the `confirmation` parameter is consensus critical, as all voters need to reach the same conclusion. This means voters should not normally need to change the value.

Status

Fixed in commit **c7bd102cb88a984ca2adda96544acccd27bd2cb6**. The default value for `Confirmations` has been removed from the indexer framework and now varies based on the specific implementation for each different blockchain.

FDC-012

Attacker can use null-payment reference as a valid payment reference on UTXO chains

Status

Solved

✓

Resolution

Fixed

Risk

Medium

Impact

High

Likelihood

Low

Location

```
flarenetwork/fdc/verifier-utxo-
indexer/utxo_indexer/models/transaction.py:79
mcc/src/base-objects/transactions/UtxoTransaction.ts:48
mcc/src/base-objects/transactions/XrpTransaction.ts:47
```

Description

Implementations using standard payment references can check for a Payment with a `000...000` reference, which is actually the default reference for non-existent UTXO payment reference. The zero payment reference complies with the Flare standard payment reference.

This is enabled by the UTXO indexer, which returns such payment reference for transactions do not include one:

```
def _extract_payment_reference(response: TransactionResponse):
    ...
    if len(std_references) == 1:
        return std_references[0]
    return ZERO_REFERENCE
```

There are two scenarios where this would be exploitable: an honest developer that reads the specification of the `Payment` attestation and wants to check for a standard payment reference. Assume they are following the official verification workflow example, which states:

```
The user first needs to call the requestServiceUsage(...) function. The
contract then stores a record that a specific msg.sender has requested
to use the service, and issues a personal 32-byte payment reference for
the request. It returns a request for payment containing:
```

The developer issues a personal 32-byte payment reference to the user, and decides that payment references can be auto-incremental and zero-indexed. All users that register can now make a `Payment` *without* a payment reference and have it recognized as a valid attestation.

The other scenario involves a scammer trying to exploit users. They *on purpose* develop a smart contract that leverages this issue. For example, it could be a betting contract that bets on the non-existence of a transaction with a zero payment reference. Users check that the transaction does not exist and bet on the contract. At that point, the scammer shows a transaction that has no payment reference is actually attested for.

Note that the XRP indexer does not have this problem, as it correctly specifies an invalid payment reference as `""`.

```
func paymentReference(tx XRPTransaction) string {
        if len(tx.Memos) == 1 {
                if memo, ok := tx.Memos[0]["Memo"]; ok {
                        if memoData, ok := memo["MemoData"]; ok {
                                if len(memoData) == 64 {
                                        return memoData
                                }
                        }
                }
        }

        return ""
}
```

Also note that the ReferencedPaymentNonexistence attestation is not affected, as it checks for the zero-reference in `generic-chain-verifications.ts`. All other attestation types will have the wrong payment reference.

```
  if (
unPrefix0x(request.requestBody.standardPaymentReference).toLowerCase()
===
    unPrefix0x(ZERO_BYTES_32).toLowerCase()
  ) {
    return { status:
VerificationStatus.ZERO_PAYMENT_REFERENCE_UNSUPPORTED };
  }
```

Following the same inconsistency, the MCC library also generates a valid payment reference (`ZERO_BYTES_32`) for payments with invalid or missing payment references. The following code snippets demonstrate the `stdPaymentReference` function.

The first snippet applies to UTXO-based transactions:

```
public get stdPaymentReference(): string {
    let paymentReference = this.reference.length === 1 ?
prefix0x(this.reference[0]) : "";
    if (!isValidBytes32Hex(paymentReference)) {
        paymentReference = ZERO_BYTES_32;
    }
    return paymentReference;
}
```

The next example shows the implementation for XRP transactions:

```
public get stdPaymentReference(): string {
    const paymentReference = this.reference.length === 1 ?
prefix0x(this.reference[0]) : "";
    if (isValidBytes32Hex(paymentReference)) {
        return paymentReference;
    } else {
        const alternative = bytesAsHexToString(paymentReference);
        if (isValidBytes32Hex(alternative)) {
            return alternative;
        }
        return ZERO_BYTES_32;
    }
}
```

In both implementations, the default value for an invalid or missing payment reference is the string `ZERO_BYTES_32` (`000...000`).

Recommendation

The null value for UTXO payment references should not fit the standard payment reference format. Align the null payment reference for invalid or non-existent payment references across the platform.

Add integration tests to validate and compare the expected outputs between MCC and the UTXO indexers, ensuring uniform behavior.

Status

Fixed in commit **5bd32b1abc37cf86ace99cbe03e0258b70f32ec3** from the Developer Hub Documentation. The documentation provides a warning about the default value assigned to non-standard references.

FDC-013

XRP verifier and indexer disagree on which transactions are native

Status

Solved

✓

Resolution

Deferred

Risk

None

Impact

Recommendation

Likelihood

–

Location

`verifier-xrp-indexer/internal/xrp/xrp.go``

Description

The XRP verifier and its indexer disagree about the definition of a `native payment`.

The indexer requires the transaction to be of type `Payment`:

```go
func isNativePayment(tx XRPTransaction) bool {
        if tx.TransactionType == "Payment" {
                var amountStr string
                err := json.Unmarshal(tx.Amount, &amountStr)
                if err == nil {
                        _, err = strconv.Atoi(amountStr)
                        if err == nil {
                                return true
                        }
                }
                var amountStruct XRPAmount
                err = json.Unmarshal(tx.Amount, &amountStruct)
                if err == nil && amountStruct.Currency == XRPCurrency {
                        return true
                }
        }

        return false
}
```

While the `verifier` uses the MCC's library different definition, which only requires the transaction to have been in XRP native's token.

```
public get isNativePayment(): boolean {
        return this.currencyName === XRP_NATIVE_TOKEN_NAME;
}
```

The indexer's `isNativePayment` data is then not used in the rest of the program; nevertheless it is stored in the database.

Recommendation

Make sure the database data matches what is used in the program.

Status

Deferred. The Flare team stated that the native definition is no longer needed and will be removed.

FDC-014

Misleading log message

Status
Solved

Resolution
Fixed

Risk
None

Impact
Recommendation

Likelihood
-

Location

`flare-common/pkg/policy/storage.go:50`

Description

The `Add` function in the `storage` file produces an inaccurate error message, which complicates troubleshooting by failing to accurately reflect the actual issue.

In the code snippet below, the error message suggests that the current signing policy has a larger start voting round ID than the previous one. However, the condition actually checks if the current policy's start voting round ID is earlier than the previous policy's ID.

```
// should be sorted by voting round ID, should not happen
if sp.StartVotingRoundID < s.spList[len(s.spList)-1].StartVotingRoundID
{
        return fmt.Errorf("signing policy for reward epoch ID %d has
larger start voting round ID than previous policy",
                    sp.RewardEpochID)
}
```

Recommendation

Update the error message to accurately reflect the logic of the `if` condition.

Status

Fixed by commit `aeaae3b73cabd56024df220f36523ddca2e0d4ba`.

FDC-015

Sanctioned address can cause a Denial-of-Service on ANYONECANPAY UTXO transactions

Status
Solved

✓

Resolution
Acknowledged

Risk
None

Impact
Recommendation

Likelihood
-

Location

`verifier-utxo-indexer/utxo_indexer/models/transaction.py:63`

Description

UTXO-based systems enable the creation of `ANYONECANPAY` transactions, allowing any participant to add arbitrary inputs. However, if a malicious actor with a sanctioned address includes a single input, it could result in the rejection of the entire transaction.

As demonstrated in the snippet below, the UTXO indexer retrieves the address of each input to construct the Merkle tree:

```python
for input in inputs:
    if input.script_key_address != "":
        addresses.append(input.script_key_address)
    else:
        addresses.append(None)
tree = merkle_tree_from_address_strings(addresses)
```

Since `ANYONECANPAY` transactions are unlikely to be utilized during the minting process, the overall risk of this issue is low.

Recommendation

Clearly document the limitations regarding the use of `ANYONECANPAY` in transactions.

Status

Acknowledged. The Flare team has acknowledged this possibility, emphasizing that FDC is not concerned with how a transaction is created or signed. Instead, FDC focuses solely on proving the results of signed transactions. This issue primarily concerns protocols that utilize FDC.

FDC-016

Anyone can prevent an input from being associated to their UTXO wallet address

Status

Solved

✓

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

-

Location

```
verifier-utxo-indexer/utxo_indexer/models/transaction.py:63
```

Description

Anyone spending from non-standard scripts can use the funds without having their address included in the Merkle tree.

Consider the snippet below: if the `script_key_address` in the input is empty, which is the value when the input spends from a non-standard script, the address is simply interpreted as `None`.

```
for input in inputs:
    if input.script_key_address != "":
        addresses.append(input.script_key_address)
    else:
        addresses.append(None)
tree = merkle_tree_from_address_strings(addresses)
```

This allows anyone to avoid being indexed by simply managing their UTXO coins with non-standard scripts so that an address is not readily available.

The severity of this issue depends on the implementation responsible for deciding whether a given Merkle tree is accepted or not.

Recommendation

Consider preventing the indexing of inputs from non-standard Bitcoin scripts, and document this limitation

Status

Acknowledged. This is the intended behavior of the FDC and consumers of its data should be aware of it.

FDC-017

UTXO and XRP source addresses are inconsistent

Status

Solved

Resolution

Acknowledged

Risk

None

Impact

Recommendation

Likelihood

-

Description

The implementation of the `SourceAddresses` root is inconsistent for XRP and UTXO chains.

For XRP, an address is part of the merkle tree only if they end with less funds than they had before the transaction took place:

```
diff := new(big.Int).Sub(finalVal, previousVal)
if diff.Cmp(big.NewInt(0)) < 0 {
        hashedAddress :=
crypto.Keccak256Hash(crypto.Keccak256Hash([]byte(modifiedNode.FinalFiel
ds.Account)).Bytes())
        sourceAddresses = append(sourceAddresses, hashedAddress)
}
```

For UTXO chains, there is no such check.

As with `FDC-016`, the impact of this issue will depend on how this information is consumed by other layers in the system.

Recommendation

Unify the logic to handle `SourceAddresses`. If impossible due to the differences in the underlying chains, document the behavior.

Status

Acknowledged. The Flare team stated that Payment XRP transactions would only involve at most two addresses (the payer and the receiver).

FDC-018

Mismatching source address root for UTXO coinbase transactions

Status

Solved

Resolution

Fixed

Risk

Medium

Impact

High

Likelihood

Medium

Location

`mcc/src/base-objects/transactions/UtxoTransaction.ts:60`

Description

The Merkle tree root for a UTXO coinbase transaction differs between the computation performed by the MCC library and the UTXO indexer. The severity of this discrepancy depends on how the source address root is consumed upstream, but it could potentially lead to a Denial-of-Service for users interacting with agents that enforce the handshake.

In the MCC library, the `merkleTreeFromAddressStrings` function generates the Merkle tree by processing the `sourceAddresses`. Here's the relevant snippet:

```
public get sourceAddressesRoot(): string {
    return merkleTreeFromAddressStrings(this.sourceAddresses).root ||
ZERO_BYTES_32;
}
```

From the sourceAddresses function, we observe that it returns undefined in the case of a coinbase transaction:

```
public get sourceAddresses(): (string | undefined)[] {
    if (isCoinbase(this.data)) {
        // Coinbase transactions mint coins
        return [undefined];
    } else if (hasPrevouts(this.data)) {
        return this.data.vin.map((vin) => {
            return vin.prevout.scriptPubKey.address; // are we sure
that every prevout has an address
        });
        // This indicates faulty assumptions about the transaction data
    } else throw MccError(`transaction ${this.txid} that does not have
prevout and is not coinbase`);
}
```

In merkleTreeFromAddressStrings, an undefined address is replaced with ZERO_BYTES_32, and the Merkle tree uses this value as a leaf:

```
export function merkleTreeFromAddressStrings(addresses: (string |
undefined)[]): MerkleTree {
    const hashedAddresses = [];
    for (const address of addresses) {
        if (address === undefined) {
            hashedAddresses.push(ZERO_BYTES_32);
        } else {

hashedAddresses.push(singleHash(singleHash(decodeAsciiString(address)))
);
        }
    }
    return new MerkleTree(hashedAddresses);
}
```

In contrast, the UTXO indexer directly assigns the source address root of a coinbase transaction as ZERO_BYTES, bypassing Merkle tree computation:

```
def update_source_addresses_root_cb(self, inputs:
List["TransactionInputCoinbase"]):
    self.source_addresses_root = ZERO_SOURCE_ADDRESS_ROOT
```

This is, the UTXO indexer does not perform MerkleTree([ZERO_BYTES]), unlike the MCC library.

Recommendation

Align the source address root computation for UTXO coinbase transactions across the system to ensure consistency.

Add integration tests to validate and compare the expected outputs between MCC and the UTXO indexers, ensuring uniform behavior.

Status

Fixed by commit `7faaf78f97f8f48ba02d536b199adebe2826bed0`. The UTXO indexer behavior now matches the one observed in the MCC.

FDC-019

Inconsistent indexer database State responses

Status

Caution Advised

Resolution

Deferred

Risk

Low

Impact

Low

Likelihood

Medium

Location

```
verifier-indexer-api/src/services/indexer-services/utxo-
indexer.service.ts:86
```

Description

The `getStateSetting` function for UTXO currently returns a hardcoded, fixed timestamp of `-1` for block information, whereas the same function for XRP indexed data provides the actual block timestamp. This inconsistency can lead to issues for clients relying on valid timestamps—like those provided for XRP—potentially resulting in incorrect decisions or actions.

```
bottom_indexed_block: {
    height: resPrune.latest_indexed_tail_height,
    timestamp: -1, // FUTURE FEAT: (Luka) add to db
    last_updated: resPrune.timestamp,
},
top_indexed_block: {
    height: resTop.latest_indexed_height,
    timestamp: -1, // FUTURE FEAT: (Luka) add to db
    last_updated: resTop.timestamp,
},
chain_tip_block: {
    height: resTop.latest_tip_height,
    timestamp: -1, // FUTURE FEAT: (Luka) add to db
    last_updated: resTop.timestamp,
},
```

Recommendation

Return the correct UTXO block timestamp.

Status

Deferred. This fix will be included in the next release, that is expected to add more information to the indexer state.

FDC-020

ReadAll execution exposes Flare System Client to Denial-of-Service

Status

Solved

✓

Resolution

Fixed

Risk

Low

▼
🟥 □ □

Impact

Medium

Likelihood

Low

Location

```
flare-system-client/client/epoch/rewards_utils.go:123
```

Description

Using `ReadAll` in Go can lead to memory exhaustion or denial-of-service vulnerabilities when excessively large inputs, as it loads the entire input into memory.

The code snippet below illustrates a `GET` request to fetch the `reward-distribution-data.json` file. According to the README, such files are uploaded to a GitHub repository. If an attacker gains access to this repository, they could potentially crash all Flare System Clients consuming this data.

```
url := fmt.Sprintf("%s/%d/reward-distribution-data.json",
config.UrlPrefix, epochId)

logger.Info("Fetching reward data at: %s", url)
```

```
result := <-shared.ExecuteWithRetryChan(func() ([]byte, error) {
        resp, err := http.Get(url)
        if err != nil {
                return nil, err
        }
        defer resp.Body.Close()

        if resp.StatusCode == http.StatusNotFound {
                return nil, nil // 404 is expected if data is not yet
published, don't retry
        }
        if resp.StatusCode != http.StatusOK {
                return nil, errors.Errorf("unexpected status code: %s",
resp.Status)
        }

        bytes, err := io.ReadAll(resp.Body)
```

This issue also highlights a broader security risk. If all Flare System Clients rely on the same repository for reward distribution data, compromising that repository could disrupt the functionality of all dependent clients.

Recommendation

Implement size restrictions to prevent excessive memory consumption when processing input.

Ensure clients can access a secondary, trusted repository if the primary repository becomes unavailable or compromised.

Status

Fixed by commit `b7deb363d38f36f478489953d1e552b10037b0ce`. The code now uses a limit reader.

FDC-021

Fetching signing policies from previous relay contract versions

Status

Solved

Resolution

Fixed

Risk

None

Impact

Recommendation

Likelihood

-

Location

`flare-system-client/client/finalizer/relay_client.go:96`

Description

A temporary modification was implemented in the testnet environment to enable the FSC relay client to read events from both the old and new versions of the relay contract. This behavior differs from the mainnet environment. As indicated in the code comments, this change was intended to be removed prior to this review.

```go
// TEMP CHANGE for upgrading Relay contract, should be removed after 17
Oct 2024

// If using new Songbird Relay, query the old one as well.
// Note: this won't have any effect on other networks as we currently
have unique Relay addresses for each network.
if r.address ==
common.HexToAddress("0x67a916E175a2aF01369294739AA60dDdE1Fad189") {
          logsOld, err :=
db.FetchLogsByAddressAndTopic0(common.HexToAddress("0xbA35e39D01A3f5710
d1e43FC61dbb738B68641c4"), r.topic0SPI, from, to)
        if err != nil {
                return nil, err
        }
        allLogs = append(allLogs, logsOld...)
}
// If using new Coston Relay, query the old one as well.
if r.address ==
common.HexToAddress("0x92a6E1127262106611e1e129BB64B6D8654273F7") {
        logsOld, err :=
db.FetchLogsByAddressAndTopic0(common.HexToAddress("0xA300E71257547e645
CD7241987D3B75f2012E0E3"), r.topic0SPI, from, to)
        if err != nil {
                return nil, err
        }
        allLogs = append(allLogs, logsOld...)
}
// END TEMP CHANGE
```

Recommendation

Consider removing the temporary change.

Status

Fixed in commit ***b7deb363d38f36f478489953d1e552b10037b0ce**. The code pointed out by this issue was removed.

FDC-022

Inconsistent payment reference processing

Status

Solved

✓

Resolution

Fixed

Risk

None

▼

□□□

Impact

Recommendation

Likelihood

–

Location

`verifier-xrp-indexer/internal/xrp/xrp.go:237`

Description

As illustrated in the snippet below, the XRP indexer converts the extracted payment reference to a lowercase hexadecimal string, whereas the UTXO indexer does not enforce this behavior. A third-party client consuming these strings in hex format could be impacted by a potential case mismatch.

Note that this conversion does not impact the payment reference's functionality when converted back to bytes, as the case of the hex string is irrelevant.

```
func paymentReference(tx XRPTransaction) string {
        if len(tx.Memos) == 1 {
                if memo, ok := tx.Memos[0]["Memo"]; ok {
                        if memoData, ok := memo["MemoData"]; ok {
                                if len(memoData) == 64 {
                                        return
strings.ToLower(memoData)
```

```
                                }
                        }
                }
        }

        return ""
}
```

However, it was observed that the Bitcoin indexer does not enforce lowercase formatting for payment references. While most RPC nodes encode these references in lowercase hexadecimal, some may encode them in uppercase, which does not affect the byte values but introduces inconsistency in representation.

It is important to note that this issue is informational, as the Standard Payment Reference documentation defines the payment reference as a 32-byte string, making the case of the hex string irrelevant for functionality.

Recommendation

The UTXO indexer should enforce storing payment references in lowercase hexadecimal.

Status

Fixed in commit **7c76d788f20d7371a632730fd359030c42640a1c**. Flare updated the UTXO indexer to use the lowercase format for payment references.

5. Disclaimer

The contents of this report are provided "as is" without warranty of any kind. Coinspect is not responsible for any consequences of using the information contained herein.

This report represents a point-in-time and time-boxed evaluation conducted within a specific timeframe and scope agreed upon with the client. The assessment's findings and recommendations are based on the information, source code, and systems access provided by the client during the review period.

The assessment's findings should not be considered an exhaustive list of all potential security issues. This report does not cover out-of-scope components that may interact with the analyzed system, nor does it assess the operational security of the organization that developed and deployed the system.

This report does not imply ongoing security monitoring or guaranteeing the current security status of the assessed system. Due to the dynamic nature of information security threats, new vulnerabilities may emerge after the assessment period.

This report should not be considered an endorsement or disapproval of any project or team. It does not provide investment advice and should not be used to make investment decisions.