Security Assessment

Charli3.io

Nov 13th, 2021

Table of Contents

Summary

This report has been prepared for Charli3.io to discover issues and vulnerabilities in the source code of the Charli3.io project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from minor to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices.

Overview

Project Summary

Project Name	Charli3.io
Platform	Cardano
Language	Plutus
Codebase	
Commit	https://github.com/Charli3-Official/charli3-oracle-prototype/commit/f4d4cf13e11de506297986dbbc63a60d6e7d3dc8 https://github.com/Charli3-Official/charli3-oracle-prototype/commit/ea5d23c750172b57021f1393f2a47c9d5c8c9f11 https://github.com/Charli3-Official/charli3-oracle-prototype/commit/acfd9a0e56ee30ff248323befe46fdd60a7bba27

Audit Summary

Delivery Date	Nov 13, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	0	0	0	0	0	0
Medium	0	0	0	0	0	0
Minor	5	0	0	1	0	4
Informational	10	0	0	2	0	8
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
ACO	Oracle/AggregateConditions.hs	9db7482198ab84cf443b317e47a60c6c88e84b3911b4cd35fb55f3fbccd8eb55
COC	Oracle/Consensus.hs	9057c60117ce7b21a18a5da123d970f8f5acfdddd90d7cce419bb8b95a679a43
DFO	Oracle/DataFeed.hs	5e86002762f5a11b3b52dd8b47f8dedc130085c4b275140726b2dba7841b2e56
OCO	Oracle/OnChain.hs	72d4454a7ae1583c15211e6bb65e1958c5deb3591a338b0b981e3c29a1b26364
TOC	Oracle/Tokens.hs	9f3f79f2ec421eafb8580fd68646d7752b754cb853cae8da0e177a263ddb0f96
TOK	Oracle/Types.hs	09a84a70b83fee653515cfc0d6a2525f72e54d07049a2387b2eee087ebc3b134
VOC	Oracle/Validator.hs	12c053d2e2f2ff9d3ff54f7d19c0e72d160d555b317b72946314e83bf338bd62
OCP	Oracle.hs	7d765e6ebb4817dfab2a7034a4faa28a49033883ec470c3da162857ac0650e37
SCC	ScriptContext.hs	7b9296489983686ef4aec7f625865d8ec2795afc3de4c2b76687e8a42dec7ed2
ACC	Oracle/AggregateConditions.hs	a69e1e1f347a6fcc1dc2e225fab8f6fdd41f9b324201bc1716af07b7230395fe
COK	Oracle/Consensus.hs	269db865d36ca8e98bcf8e16e4c9b559c7e2653fc4187eebed271e4bb2e37604
DFC	Oracle/DataFeed.hs	5e86002762f5a11b3b52dd8b47f8dedc130085c4b275140726b2dba7841b2e56
OCC	Oracle/OnChain.hs	b34753833aa3564e8551f0a367738db3c40fd412ebe6684395dcc3fde10a6948
TOP	Oracle/Tokens.hs	2a4edc35a9c90d413a3e69a4e3a41a2be1b46e9b31e8dae2d790f8a6846282cf
TCK	Oracle/Types.hs	b5f832f3e1dea336247ae06af19e184fcd2d775469ef4d81e20adbd4c1640569
VOK	Oracle/Validator.hs	5324fad8c6db39e821f19d6ab00e8c5ba085e14a4e7c1479dd29f1956c6a6e96
ORA	Oracle.hs	7d765e6ebb4817dfab2a7034a4faa28a49033883ec470c3da162857ac0650e37
SCK	ScriptContext.hs	7b9296489983686ef4aec7f625865d8ec2795afc3de4c2b76687e8a42dec7ed2

File Dependency diagram :

Oracles

A blockchain oracle is a third-party service providing outside information to the blockchain, in order to make it usable in smart contracts. Smart contracts alone can not access outside data, but trusting a single centralized data provider would nullify the advantage of blockchain; this is known as the *Oracle problem*.

One of the proposed solutions is decentralized oracles like `Charli3`: the data is obtained from a group of trusted entities called `Node Operators` who fetch the data from a reliable source and submit it to the oracle. The oracle contracts implement a mechanism of consensus and rewards to try to ensure that the data is reliable even if some of the node operators fail or misbehave.

High-level state transitions of the Charli3 protocol

A Charli3 oracle provides a feed of time-stamped numeric data (typically asset prices). The data is provided by a fixed set of node operators, whose inputs are aggregated into a single value through a particular update logic. In order to support this logic, the oracle allows data providers to perform certain operations. Here we give a high-level view of how the oracle functions on-chain as a state transition system. It is a fairly conventional distributed oracle, and the main novelty is instead of how this state is realized using the Cardano eUTxO model, which we will describe later.

The state of the Oracle can be considered as two parts: the fixed setup, and the dynamically changing state. The setup has the public key of the oracle creator, a list of (the public keys of) node operators, and the values of the configurable parameters (e.g. percentage of nodes needed for consensus, see below). The dynamic state is the price and time of the last update for the oracle itself, the price and time of the last update for each of the nodes, and the amount of funds (`C3` tokens) currently credited to the oracle and to each node.

Each contract endpoint provided by `Charli3` can be considered as an action that transitions from one state to another. They are:

- **Add funds to the Oracle Contract:**
  These funds are used for payments to the node operators. The only part of the state that this action change is the amount of funds credited to the oracle.

- **Update the data provided by a node:**
  At any time, the operator of a node can submit a new value and update the time stamp (signing the transaction with their key). Such a data update action changes the data stored in the node and the time stamp. Every other element of the state representation remains unchanged.

- **Use the data:**
  Anyone can use the current oracle (value, update time) pair as an input to other transactions. This is how the data is provided to other smart contracts. This action doesn't change the state.

- **Collect fees:**
  At any time, a node operator can transfer the `C3` tokens credited to their node to an address of their choice. This action will reduce the amount of `C3` tokens credited to the relevant node by the value specified. It leaves all the other state components unaltered.

- **Update the oracle settings**
  At any time, the oracle owner can add or delete node operators from the oracle, and toggle whether the oracle is marked as enabled or not. Deleting a node operator should also send them their current `C3` token balance.

- **Compute the aggregate data:**
  This is the most important action in the state transition system. Computing the aggregate data involves reading the most recent data feed from the selected nodes, checking that the computed aggregate value using this data satisfies certain constraints, and then accordingly updating the oracle feed and its timestamp. At the same time, it pays the data providers their fees as `C3` tokens. Hence, the aggregation step changes the oracle feed, the time stamp associated with it, the amount of funds credited to the oracle, and the amount of funds credited to the participating data providers.

- **Update data provided by the node operators and compute the aggregate data in a single step:**
  This is the case when an aggregation request and a data feed update request are simultaneously submitted to the oracle. In this situation, the oracle allows the update for the relevant data feed to take place but excludes the data provider's value from the aggregation step. The state change involved in this case is the combination of the state change for a data feed update and that of an aggregation step.

The most interesting transition is the aggregation step, which computes the new oracle value. It uses parameters (`osUpdatedNodes`, `osUpdatedNodeTime`, `osAggregateTime`, `osAggregateChange`) and applies the following checks:

- `checkAggregatorPermission`/`checkAggregationEnabled`:
  the transaction is signed by a node operator or oracle owner, and the oracle is marked as enabled.
- `checkNodeUpdatesCondition`:
  a minimum percentage osUpdatedNodes of nodes have a fresh data value (i.e. one which was updated after the previous oracle value was computed and updated recently). It then computes the new aggregated value as the median of the fresh values.

- `checkAggregationUpdateTime`: Either a certain minimum time has passed since the previous oracle value, or the newly computed aggregated value is different from the previous by more than a certain percentage.

If all conditions pass the Oracle value is updated. Finally, the contract computes the difference between each node's reported value and the aggregated value. If the difference is too large, that node is considered "out of consensus". Otherwise, the node is considered "in consensus" and is paid a fee for participating in this round of the oracle.

Analysis: Comparisons

The logic of requiring a certain fraction of the node operators to provide values and then taking the median is quite standard among existing distributed oracles. In the Ethereum world, for example, *Chainlink* (FluxAggregator.sol), *Compound*, *AmpleForth*, and other systems use this rule.

The system of detecting outliers and withholding payment to those node operators is intended to incentivize high-quality data.
We do not do any detailed economic analysis of how well this will work.

Analysis: Attacker Model

When analyzing the correctness of the oracle we must consider two classes of attacks, on data availability and data correctness. Both these properties can only partially be guaranteed by the on-chain logic, so they also require some assumptions about the trustworthiness of the participants.

For data correctness, it is inherently impossible to ensure that oracle data providers submit correct data using code on the chain itself. Instead, the design assumes that someone will review the data afterward and that node operators have incentives to act honestly. To this end, the oracle owner selects the node operators, and each node update is signed by the operator. The contract code can therefore assume an *honest majority* security model, where it should return correct values as long as most node operators provide good data.
Specifically, because of the median rule, we know that as long as more than 50% of the nodes that participated in the aggregation are honest, the final selected value is within the range of values that the honest operators reported.
Each aggregation is allowed to proceed if a fraction `osUpdatedNodes` of the nodes provide data, so it is safe as long as the fraction of honest nodes is (strictly) greater than (1 - `osUpdateNodes`/2).
For example, if there are 4 operators of the oracle, and `osUpdateNodes` is set to 74% so it can make progress if one of them fails, then the oracle can tolerate 1 malicious node operator submitting bad data because there will still be 2 honest ones in every aggregation.

We review the code to ensure that the median calculation is done correctly and to make sure that node operators have no way to influence the aggregation result except through the value they submit.

For data availability, we assume that a majority of the node operators keep submitting data because it is in their interest to earn fees. The oracle owner should estimate how many nodes may fail and set the threshold `osUpdatedNodes` low enough that the oracle can still make progress with that many nodes missing.

We review the code to make sure that the protocol can make progress even if third parties or a minority of node operators try to interfere.
This is assured because:

1. there are sufficient access checks in the validators that only node operators can affect their own data feed UTxOs and
2. the aggregate transitions can happen as long as a majority of the data feeds are up to date.

We note that the design of the `Charli3` protocol, and in particular the fact that it is based on the Cardano eUTxO model, helps make the data availability argument easier and more trustworthy because in `Charli3` the operations on the data feeds are completely independent of each other. Ethereum-based oracles need code to either *push* data into a central contract or to have the contract *pull* from sources, and this interaction can be a source of programming errors (e.g. possible reentrancy when calling a data source to pull data from it, ampleforth-core/v1.0.0/Trail-Of-Bits-Audit.pdf)

The Oracle Contract as a Constraints Emitting Machine

The main novelty of the `Charli3` contract is that it is implemented on top of the Cardano eUTxO model. This means that the contract state must be represented using UTxO datums, and the state transition logic is implemented by transaction validators rather than imperative code. These validators are the focus of our code review.

The contract supports usage by three different kinds of users (and a corresponding UTxO type):

- The **data providers** or the **Node Operators** use the Node State UTxO
- The **data consumers** or the **End users** use the Oracle Feed UTxO
- The **oracle owner** (could be the set of whitelisted node operators) use the Aggregate State UTxO.

Each of these carries a datum (`NodeDatum`, `OracleFeedDatum`, `AggDatum`) which together keep the entire contract state. In order to distinguish the different kinds of UTxOs, it used a scheme based on non-fungible tokens. There are three minted NFTs (`nodeNFT`, `oracleNFT`, and `aggStateNFT`) which are used to mark the

types of the UTxOs. In other words, the value of each `Charli3` UTxO will contain exactly one of those NFTs, in addition to the `C3` tokens used for payments.

Each of the high-level state transitions is then triggered by a transaction submitted by the off-chain code, and checked by an on-chain validator. The above figure summarizes the possible transactions and the associated checks.

Each validator needs to carry out the checks associated with the high-level logic, and also enough checks that the representation invariant is preserved.

In the Cardano model, validation logic is divided into several scripts associated with each type of node datum. In our case there are 12 such functions:

Analysis: Invariants and simulation relation

The oracle is modeled using the data type `Oracle` which is a record type defined as follows :

```
data Oracle = Oracle { oracleCreator :: PubKeyHash
                     , oracleNFT     :: AssetClass
                     , aggStateNFT   :: AssetClass
                     }
```

All the other oracle states should be reachable from the oracle script address and this information. In particular, the `OracleFeedDatum` has a list of all the nodes `osNodeList`.

Specifically we can define a ***representation invariant***:

- There is exactly one UTxO with address `(oracleAddress oracle)` and NFT `(oracleNFT oracle)` and its datum is `(OracleFeedDatum)`
- There is exactly one UTxO with address `(oracleAddress oracle)` and NFT `(aggStateNFT oracle)` and it's datum is `(AggDatum agstate)`
- For each item in `osNodeList` there is exactly one UTxO `(oracleAddress oracle)` and the NFT from `osNodeList`, and it's datum should be a `NodeDatum`.

The correctness property we want to prove about the validators is that they correctly implement the business logic of the contract. To make this precise, we can state it mathematically in terms of the high-level transition system: we want to say that `mkOracleValidator` returning true implies a successful transition can take place.
In more mathematical terms, we define a function from the set of Oracle Redeemers to transitions (say F), and review the following property:

Claim: For a given Oracle o, Asset Class a, Oracle Datum d, Oracle Redeemer r, Script Context s and a Contract State c (represented by the pair (datum, value)), if we have :

mkOracleValidator a o d r s = true, then there exists a contract state c' such that
c $\xrightarrow{F(r)}$ c'.

(the $F(r)$ in the bracket represents the transition corresponding to the redeemer, the correspondence is straightforward), and the state stored in the datum of the transaction outputs is c' (this information can be obtained from the script context s, and some helper functions). Furthermore, the outputs corresponding to the state c' satisfy the relevant constraints from the transition $F(r)$.

In this work, we merely check that this holds true by manual code review, but such a theorem would also be suitable for stating and proving with formal verification.

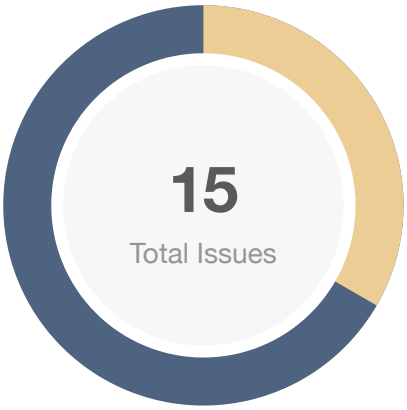
Analysis: Throughput and Denial-of-service considerations

In the extended UTxO model (eUTxO) each transaction consumes a particular UTxO, which always raises the question of whether there will be *contention* for the UTxOs. In the case of `Charli3`, this can be considered separately for the different types of transactions. Node updates are only done by the node operator and can be done independently from each other, so they are completely scalable.
Node operators and the oracle operators will race each other to submit aggregation transactions, but this is benign because any successful transaction will allow the protocol to make progress.

However, there is potential for *contention* for the oracle output token. As described above, an aggregation action creates a single token, and a client of the oracle needs to consume it (producing an identical new token). Any further clients will then need to wait and observe what the new token is before they can make new transactions, so the oracle output can only be used once per block.
Whether this is an issue depends on the client: many existing example Cardano contracts are built around a state machine that uses a single token for the state, and such a contract can in any case only make a single transaction per block, so the use of `Charli3` would not be a limiting factor. Cardano users are investigating different strategies to create *low-contention* contracts, and (according to the whitepaper) future versions of `Charli3` will accommodate such contracts by producing multiple output UTxOs.

As noted in the `Charli3` whitepaper, the current version of the `Charli3` contracts could also be vulnerable to *denial-of-service* attacks where an attacker submits many transactions competing for the oracle UTxO to prevent legitimate oracle users from making progress.
There are plans to address this in future versions of Charli3, but in this version clients of the oracle must be aware of this possibility and check that any potential data availability issues would be tolerable.

Findings



Critical	0 (0.00%)
Major	0 (0.00%)
Medium	0 (0.00%)
Minor	5 (33.33%)
Informational	10 (66.67%)
Discussion	0 (0.00%)

ID	Title	Category	Severity	Status
ACO-01	Typos	Coding Style	Informational	Resolved
COC-01	Unoptimized Function	Coding Style, Gas Optimization	Informational	Resolved
OCC-01	Redundant Statements	Volatile Code, Gas Optimization	Informational	Resolved
OCC-02	missing check in `unchangedSettings`	Inconsistency, Logical Issue	Minor	Resolved
OCC-03	No Check That `addNodes` Do Not Delete Nodes And Vice/Versa	Logical Issue	Minor	Resolved
OCC-04	No Check For Payment To Node Operator	Logical Issue	Minor	Resolved
OCO-01	Typos	Coding Style	Informational	Resolved
OCO-02	Incorrect Error Message	Logical Issue	Minor	Resolved
OCO-03	Unclear Comment	Coding Style	Informational	Resolved
OCO-04	No validation of the update time	Logical Issue	Minor	Acknowledged
TOC-01	Similar Functions Names	Coding Style	Informational	Resolved
TOC-02	Third Party Dependencies	Control Flow	Informational	Acknowledged
TOK-01	Error In Comments	Language Specific	Informational	Acknowledged

ID	Title	Category	Severity	Status
TOK-02	Function Not Recommended For Production	Volatile Code	Informational	Resolved
TOK-03	Field Not Used	Data Flow	Informational	Resolved

ACO-01 | Typos

Category	Severity	Location	Status
Coding Style	Informational	projects/charli3io/Oracle/AggregateConditions.hs (20fe893): 140, 232, 59	Resolved

Description

Trace error typos:

`src/Oracle/AggregateConditions.hs`:

```
140         "New aggregation feed didn't *changed* "
```

Should be changed to something like:

```
140         "New aggregation feed didn't *change* "
```

`src/Oracle/AggregateConditions.hs`:

```
232         , "- Oracle feed *don't changed* enough"
```

Should be changed to something like:

```
232         , "- Oracle feed *didn't change* enough"
```

`src/Oracle/OnChain.hs`:

```
70          traceFalse "Invalid datum for the provided *reedemer.*"
```

Should be changed to something like:

```
70          traceFalse "Invalid datum for the provided *redeemer.*"
```

Comment typos:

`src/Oracle/AggregateConditions.hs`

```
59          -- | Checks if the last update of a data feed *succeded* after the last
```

Should be changed to something like:

```
59          -- | Checks if the last update of a data feed *succeeded* after the last
```

`src/Oracle/OnChain.hs`

```
73          - The datum *doesn't changed*.
```

Should be changed to something like:

```
73          - The datum *doesn't change*.
```

Recommendation

We recommend correcting typos in the contract.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27.

COC-01 | Unoptimized Function

Category	Severity	Location	Status
Coding Style, Gas Optimization	Informational	projects/charli3io/Oracle/Consensus.hs (20fe893): 62	Resolved

Description

In the file `Consensus.hs`, the function `mad` will recompute many time the median of a list which never change.

Recommendation

We advise rewriting the function to avoid unnecessary recomputation, for example:

```
mad :: [Integer] -> Integer
mad xs = median (map f xs)
  where
    md = median xs

    f :: Integer -> Integer
    f a = abs md a
```

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27, the new implementation only computes the median once.

OCC-01 | Redundant Statements

Category	Severity	Location	Status
Volatile Code, Gas Optimization	Informational	projects/charli3io/Oracle/OnChain.hs (fb1df58): 322~324	Resolved

Description

Commit: ea5d23c750172b57021f1393f2a47c9d5c8c9f11

In the file `OnChain.hs`, on line 322, the function `mkDelNodesAggStateValidator` makes several checks including :

```
322 unchangedSettings     inAggState outAggState     &&
323 burningNodeNFTs       inAggState outAggState ctx &&
324 unchangedSettings     inAggState outAggState
```

The two `unchangedSettings` checks are identical.

Recommendation

We advise removing one of the `unchangedSettings` checks

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27

OCC-02 | missing check in `unchangedSettings`

Category	Severity	Location	Status
Inconsistency, Logical Issue	Minor	projects/charli3io/Oracle/OnChain.hs (fb1df58): 990~1001	Resolved

Description

Commit: ea5d23c750172b57021f1393f2a47c9d5c8c9f11

In the file `OnChain.hs` a new function is defined : `unchangedSettings`.
The comment describing it says that it checks that all the components of the settings of an oracle didn't change, except for the node whitelist.

However, the function doesn't perform any check on the new parameters, `osMadMultiplier` and `osDivergence`, added to the `OracleSettings` type in the file `Types.hs`.

In particular, this means that an Oracle operator can use an "add nodes" transaction to change these settings after the oracle has been deployed, which is not documented and possibly unintended.

Recommendation

We advise checking if the comment or the logic of the function needs to be changed, and otherwise documenting the current behavior.

In case rewriting the function is necessary, we advise checking all the functions using the type `OracleSettings` to be sure that no other omission happened related to the added field. For example, the function `checkOracleSettings` in the file `AggregateConditions.hs` hasn't changed and doesn't perform any check on the new parameters (but that function is only used in off-chain code).

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27. Also, the code has been refactored to keep the unchanged settings in a `OracleSettingsPayload` subrecord, which is better programming practice.

OCC-03 | No Check That `addNodes` Do Not Delete Nodes And Vice/Versa

Category	Severity	Location	Status
Logical Issue	Minor	projects/charli3io/Oracle/OnChain.hs (fb1df58): 284	Resolved

Description

Commit: ea5d23c750172b57021f1393f2a47c9d5c8c9f11

The intention is that the `AddNodes` transactions should only add nodes to the list, but currently, there is no check that it did not delete any. The checks in this function (`mintedNewNodeNFTs` and `checkNewNodeUTxOs`) are written in terms of `getNewNodes`, which will ignore if any nodes were deleted. This may mean that a malformed addNodes transaction could delete nodes without deleting the corresponding node NFTs, breaking an intended invariant of the contract.

Similarly, the `DelNodes` transaction does not seem to check that no nodes were added.

Recommendation

We advise adding some checks to ensure the safety of these functions.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27. There is a new a new validation function `correctNodeListUpdate`, which is used in `mkAddNodesAggStateValidator` and `mkDelNodesAggStateValidator`.

OCC-04 | No Check For Payment To Node Operator

Category	Severity	Location	Status
Logical Issue	Minor	projects/charli3io/Oracle/OnChain.hs (fb1df58): 344	Resolved

Description

Commit: ea5d23c750172b57021f1393f2a47c9d5c8c9f11

In the `OffChain.hs` fils, the intention is that whenever the node operator deletes a node they will also pay the C3 tokens,

```
        Constraints.mustPayToPubKey (nodeStatePKH nutxoState)
            (getValueOf c3Asset (nodeTxOutTx ^. ciTxOutValue))
```

However, there is no check in the validator for this. A malicious oracle owner could cheat the node operator out of their payment.

Recommendation

Add a check using e.g. `valuePaidTo` that the C3 value of the deleted node was sent.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27. The check is done using the helper function `payedTo`.

OCO-01 | Typos

Category	Severity	Location	Status
Coding Style	Informational	projects/charli3io/Oracle/OnChain.hs (20fe893): 70, 73	Resolved

Description

Trace error typos:

`src/Oracle/AggregateConditions.hs`:

```
140         "New aggregation feed didn't *changed* "
```

Should be changed to something like:

```
140         "New aggregation feed didn't *change* "
```

`src/Oracle/AggregateConditions.hs`:

```
232         , "- Oracle feed *don't changed* enough"
```

Should be changed to something like:

```
232         , "- Oracle feed *didn't change* enough"
```

`src/Oracle/OnChain.hs`:

```
70          traceFalse "Invalid datum for the provided *reedemer.*"
```

Should be changed to something like:

```
70          traceFalse "Invalid datum for the provided *redeemer.*"
```

Comment typos:

`src/Oracle/AggregateConditions.hs`

```
59          -- | Checks if the last update of a data feed *succeded* after the last
```

Should be changed to something like:

```
59          -- | Checks if the last update of a data feed *succeeded* after the last
```

`src/Oracle/OnChain.hs`

```
73          - The datum *doesn't changed*.
```

Should be changed to something like:

```
73          - The datum *doesn't change*.
```

Recommendation

We recommend correcting typos in the contract.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27.

OCO-02 | Incorrect Error Message

Category	Severity	Location	Status
Logical Issue	Minor	projects/charli3io/Oracle/OnChain.hs (20fe893): 384	Resolved

Description

In the `OnChain.hs` file, the error message in the function `increasedC3Value` is not a complete description of the check:

```
384  traceIfFalse "increasedC3Value: outVal lower than inVal" $
```

the error can be caused by either a lower outVal than inVal, *or* failing to preserve the NFT token, so in the latter case the error message could be misleading.

Recommendation

We advise rewriting the error message.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27, there are now separate error messages for the two cases.

OCO-03 | Unclear Comment

Category	Severity	Location	Status
Coding Style	Informational	projects/charli3io/Oracle/OnChain.hs (20fe893): 103~104	Resolved

Description

In the file `OnChain.hs`, the comments of the function `mkAddFundsValidator` say that :

`The entire input value is equal to the entire output value plus some funds (possibly any currency).`

However, the extra funds can only be in the `c3` token (c.f. function `increasedC3Value`), not any currency.

Recommendation

We advise rewriting this comment to be more clear about what the function checks.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27.

OCO-04 | No validation of the update time

Category	Severity	Location	Status
Logical Issue	Minor	projects/charli3io/Oracle/OnChain.hs (20fe893): 135	Acknowledged

Description

When you do an Aggregate action, the validator checks that a sufficient percentage of nodes have an update time (`dfLastUpdate`) in the relevant time window, where it get the current time from `txInfoValidRange` (in `mkAggregateAggStateValidator -> checkNodeUTxOs -> checkAmountOfNodes -> checkFeedLastUpdate`). However, when you update a node there is no check that the value of `dfLastUpdate` actually is at the current time. So for example, a node operator could specify some time next week, and then a week from now that value could get used in an aggregation.

This should not lead to any security violations: it is still in the interest of the node operator to provide timely updates according to the intended protocol, rather than guessing a value far into the future which probably will be incorrect and excluded from the consensus. And any value provided can be overwritten by another update, so it does not provide any kind of communication channel beyond what could already be done off-chain. However, it is counter-intuitive that the timestamp is validated in some contract transactions but not others, and for defense-in-depth, it would be preferable to make sure that it is always valid.

Recommendation

Add a validation check that `txvalid` range of the update transaction is reasonably short (comparable to the update time window of the oracle) and agrees with the provided `dfLastUpdate`.

Alleviation

`[Charli3]`: We thought of some preliminary solutions but none of them were feasible: any check we want to add requires that in the single node-update transaction we should read the oracle settings adding congestion issues (not possible to have two different nodes updating values concurrently). This will instead be addressed by monitoring node updates externally and taking this kind of misbehavior into account for overall node reputation, thus filtering out misbehaving nodes.

TOC-01 | Similar Functions Names

Category	Severity	Location	Status
Coding Style	Informational	projects/charli3io/Oracle/Tokens.hs (20fe893): 71, 79	Resolved

Description

In `Token.hs` the functions `forgeNFT` and `forgeNFTs` are very similar. This could cause confusion or misuses.

Recommendation

We advise renaming `forgeNFT` since it is not called by any other function. This will avoid confusion.

Alleviation

`[CertiK]`: Resolved in commit : 954192a57595979ba75e01ed19bdc7de41d2394f

TOC-02 | Third Party Dependencies

Category	Severity	Location	Status
Control Flow	Informational	projects/charli3io/Oracle/Tokens.hs (20fe893): 23, 75, 83	Acknowledged

Description

The contract relies on the `Plutus.Contracts.Currency` to ensure the logic's consistency with NFTs.

Recommendation

We understand that logic of `Charli3` protocol requires the use of external modules. We encourage the team to constantly monitor the statuses of those 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`Charli3`: We will continually monitoring the libraries we use for changes and updates, to ensure no unvetted changes are allowed to slip in.

TOK-01 | Error In Comments

Category	Severity	Location	Status
Language Specific	Informational	projects/charli3io/Oracle/Types.hs (20fe893): 97~108	Acknowledged

Description

In the function `mkOracleSettings` each comment detailing the parameter of the function is shifted by one line.

Recommendation

We advise correcting this by rewriting the function as follows :

```
95  mkOracleSettings
96      :: [PubKeyHash]
97      -> Integer
98      -- ^ The percentage of nodes needed for aggregation (0-100)
99      -> Integer
100     -- ^ The max time since last node update for aggregation (in milliseconds)
101     -> Integer
102     -- ^ The min time since last aggregation for calculating a new one
103     --   (in milliseconds)
104     -> Integer
105     -- ^ The percentage of change between last aggregated value and the new one
106     --   (0-100)
107     -> NodeFee
108     -- ^ The amount of c3 to pay for aggregation to each node.
109     -> OracleSettings
110 mkOracleSettings ns pun tun tuagg pagg feePrice =
111     OracleSettings
112     { osNodeList        = map (`mkNodeInfo` Nothing) ns
113     , osUpdatedNodes    = pun
114     , osUpdatedNodeTime = tun
115     , osAggregateTime   = tuagg
116     , osAggregateChange = pagg
117     , osNodeFeePrice    = feePrice
118     }
```

TOK-02 | Function Not Recommended For Production

Category	Severity	Location	Status
Volatile Code	Informational	projects/charli3io/Oracle/Types.hs (20fe893): 293~300	Resolved

Description

These linked functions all used `unstableMakeIsData` which is not recommended for production usage.

Recommendation

We advise using `makeIsDataIndexed` for production usage to ensure that the output is compatible if the Plutus library is updated in the future.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27, which now uses `makeIsDataIndexed`.

TOK-03 | Field Not Used

Category	Severity	Location	Status
Data Flow	Informational	projects/charli3io/Oracle/Types.hs (20fe893): 268	Resolved

Description

The `srOldNodes` field is currently always set to the empty list. In the function `oracleState` in `OffChain.hs` there is a helper function called `splitNodes`, which name suggests that it should split the set of nodes into old and new ones, but in fact, it only filters for new nodes.

Recommendation

We advise, either update the logic to also compute the list of old nodes or delete the field.

Alleviation

`[CertiK]`: Resolved in commit : acfd9a0e56ee30ff248323befe46fdd60a7bba27. The field is now deleted.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a struct assignment operation affecting an in-memory struct rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.