Security Audit Report

Celestia: Q2 2023

Authors: Ivan Gavran, Andrija Mitrovic

Last revised 13 July, 2023

Table of Contents

Audit overview

The Project

In June 2023, Informal Systems has conducted a security audit for Celestia of the library rsmt2d.

The main focus of the audit was the correctness and functionality of the library and its integration with other repositories of Celestia (celastia-node and celestia-app).

The audited commit hash is 6515446b.

The audit took place from June 1, 2023 through June 29, 2023 by the following personnel:

- Ivan Gavran
- Andrija Mitrovic

Conclusions

In general, we found the codebase to be of very high quality: the code is well structured and easy to follow, and all functions contain good unit-tests. Repository does not have adequate specs, so those should be added. In the audit, we found 15 issues: most of them informational and one high severity, which was promptly addressed.

Further Increasing Confidence

For increasing confidence further, we would suggest enumerating all possible scenarios of the rsmt2d workflow and adding a test for each of them. Those tests go beyond unit tests as they need to include the interplay between different functions. (For instance, a scenario describing the situation in which an encoding error is detected in columns while rebuilding rows would have detected one of the problems found in this audit.)

Disclaimer

Blockchain technology and cryptographic assets in general and by definition present a high level of ongoing risk. Client is responsible for its own due diligence and continuing security in this regard.

Audit dashboard

Target Summary

- **Type**: Specification and Implementation
- **Platform**: Golang
- **Artifacts**:
    - rsmt2d

Engagement Summary

- **Dates**: 01.06.2023 to 29.06.2023
- **Method**: Manual code review, protocol analysis
- **Employees Engaged**: 2

Severity Summary

Finding Severity	**#**
Critical	0
High	1
Medium	0
Low	1
Informational	13
Total	**15**

System overview

Abstract

Rsmt2d is a library utilized for the erasure coding of data in Celestia. It serves as a specialized package offering functionalities to reorganize data into a square shape, expand it, and apply erasure coding to the original square data. Additionally, it incorporates the capacity to repair or restore incomplete data based on the underlying erasure coding technique. Furthermore, it provides proofs of tampered data, which play a vital role in ensuring data availability within the Celestia platform.

The Leopard codec serves as the underlying coding mechanism for Reed-Solomon erasure coding. Originally implemented in the C++ library, Leopard is now utilized by Rsmt2d through a Go port of the C++ library. For shards of 256 or less, the Leopard codec uses an 8-bit implementation, whereas shards exceeding 256 employ a 16-bit version of Leopard.

Data Structure

The Rsmt2d library is designed around a squared data structure for efficient data availability workflow. It uses two main structures: `dataSquare` and `extendedDataSquare`. The `dataSquare` structure serves as the foundation and provides the necessary functionality for storing data in a square format. On the other hand, the `extendedDataSquare` structure inherits from `dataSquare` and adds extra capabilities to provide necessary functionality for rsmt2d library.

Data Square (DS)

As a base data structure used for storing data, a data square is utilized. This data square is implemented by a struct `dataSquare` which is as follows:

```
type dataSquare struct {
    squareRow    [][][]byte // row-major
    squareCol    [][][]byte // col-major
    dataMutex    sync.Mutex
    width        uint
    chunkSize    uint
    rowRoots     [][]byte
    colRoots     [][]byte
    createTreeFn TreeConstructorFn
}
```

DS is a matrix where each cell is a byte array. It contains duplicated data arranged in both row-major and column-major order. This duplication allows for zero-allocation column slices to be provided. It also stores data about the width of the square, chunk size, roots by rows and columns and a delegate to tree constructor that is used for root calculation. All chunks/cells within the data square must be of the same size.

The provided functionality of the Data square is as follows:

- Creation of Data Square (`newDataSquare`): Rearranges the given data array into a square and creates a `dataSquare` object.
- Extending Data Square (`extendSquare`): Extends the original square horizontally and vertically by a predefined number of rows/columns and fills it with the same `fillerChunk` .
- Accessing Cells (`GetCell` , `SetCell` , `setCell`): Allows reading/writing data from/into cells.

- Accessing Row/Column Slices (`rowSlice` , `row` , `setRowSlice` , `colSlice, col, setColSlice`) : Enables reading/writing data from/into row/column slices.
- Accessing Roots (`resetRoots` , `computeRoots` , `getRowRoots` , `getRowRoot` , `getColRoots, getColRoot`): Provides functionality for resetting, computing, and reading row/column roots.

Extended Data Square (EDS)

The Extended data square is a data structure that inherits the DS (data square) data type and represents an extended portion of data. Its structure is as follows:

```
type ExtendedDataSquare struct {
    *dataSquare
    codec             Codec
    originalDataWidth uint
}
```

It includes a pointer to the DS struct and introduces additional attributes, `codec` and `originalDataWidth` . The `codec` attribute represents the codec used for encoding/decoding data (as mention, rsmt2d library uses the Leopard codec). The `originalDataWidth` field is used to store information about the width of the original data square.

The provided functionality of the Extended data square is as follows:

- Compute EDS (`ComputeExtendedDataSquare`): This function computes the extended data square for a given set of data chunks.
- Import EDS (`ImportExtendedDataSquare`): This function imports an extended data square that is represented as flattened chunks of data.
- Gets row/column (`Row` , `Col`): These functions return a copy of the internal slice, which represents a row or column slice of the extended data square.
- Gets row/column roots (`RowRoots` , `ColRoots`): These functions return the Merkle roots of all the rows or columns within the extended data square.

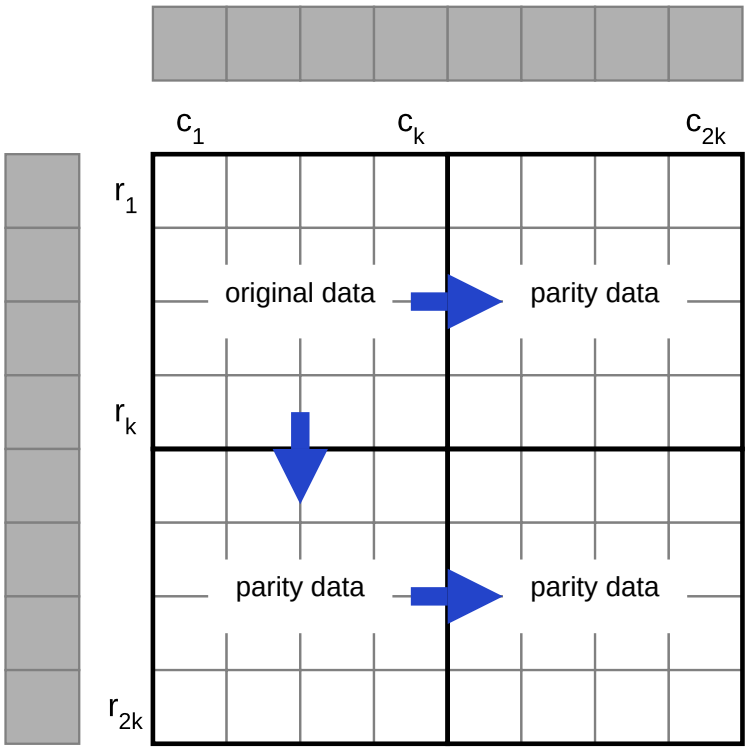
Extending data

Extending data is a part of the Reed Solomon encoding process. The original data square, named Q0, is extended to three more parity data squares: Q1, Q2, and Q3. These squares are filled with encoded original data. The functionality for this process is provided in the function called `ComputeExtendedDataSquare` .

The process of computing the extended data square with an array of data chunks is as follows (see figure for *Erasure extending of the original data square*):

1. Create a new data square (`ds`) from the provided array of data. This serves as the original data square.
2. Create an extended data square (`eds`) from the created `ds` in the previous step.
3. Perform erasure extending on the created `eds` :
    a. Extend the original data square to Q1, Q2, and Q3, filling them with empty data chunks.
    b. Fill Q1 and Q2 with erasure coded data. Q1 is the result of erasure coding Q0 row by row, and Q2 is the result of erasure coding Q0 column by column.
    c. Fill Q3 with erasure coded data. Q3 is the result of erasure coding Q2 row by row (it will be the same if Q3 is the result of erasure coding Q1 column by column).

Note that while the data is laid out in a two-dimensional square, the rows and columns are erasure coded using a standard one-dimensional encoding.



Erasure extending of original data square

Reconstruction of data

The reconstruction of the incomplete extended data square (EDS) is performed in the function Repair. This function continuously compares the repaired rows and columns against the expected Merkle roots. To ensure proper functioning, the missing shares must be set to `nil`.

The process of repairing is as follows:

1. A sanity check is performed (`prerepairSanityCheck`), which includes the following checks:
    a. Verify that the row/column roots from EDS are equal to the expected row/column roots.
    b. Check if the encoded original data from the row/column is equal to the extended erasure data from EDS.
2. Iterative solving of crossword (`solveCrossword`).

Solving the crossword involves looping through each row and column in an attempt to rebuild any incomplete rows or columns. This process is repeated until one of the following conditions is met:

1. The square is solved.
2. An error is returned from solving rows or columns.
3. No progress is made.

When solving a row/column, if it is completed, a check is performed against the expected Merkle roots (check for rows, check for columns). Additionally, a check for a completed orthogonal column/row is also performed. If the

calculated root for a newly completed row/column does not match the expected Merkle root, an `ErrByzantineData` error is returned.

This error is defined as follows:

```
type ErrByzantineData struct {
    Axis   Axis     // Axis of the data.
    Index  uint     // Row/Col index.
    Shares [][]byte // Pre-repaired shares. Missing shares are nil.
}
```

Identified Threats and Audit Plan

In this audit of the rsmt2d library, we were looking at the library from two angles:

1. logic within the library itself
2. usages of the library functions in the broader Celestia context (be it in `celestia-node` or `celestia-app`)

Threats

We identified the following threats to the chain stemming from the rsmt2d library:

1. Wrong usage of the encoding library, resulting in the extended data that does not have the desired recoverability properties.
2. Problems in the logic for recovering shares from pieces of data. If those problems exist, this would result in full nodes not being able to recover existing data, violating security assumptions of light-nodes.
3. Problems in sending and validating bad encoding proofs. If this is not done correctly, it could result in a) mistakenly blacklisting an honest peer, or b) accepting a fraudulent bad encoding proof.

Audit Plan

We set out to explore the following:

1. Inspection of how the data square is extended.
2. Inspection of updates of `squareRow` and `squareCol`, two independent variables that are referring to the same data. They have to be updated in sync.
3. A special attention is given to the function `solveCrossword`, inspecting termination and correctness of a rebuild process
4. Review of construction of `ErrByzantineData`, its transformation into a `BadEncodingProof`, and its validation in the `Validate` function.
5. Making sure that data is never changed without changing the corresponding roots.
6. Making sure that all assumptions are checked at all relevant places (e.g., chunk size, row/column indices within bounds, etc.)

Findings

Title	Type	Severity	Issue
ErrByzantineData.Shares are not filled	PROTOCOL	3 HIGH	https://github.com/celestiaorg/rsmt2d/issues/178
`OR` instead `AND` should be used in the test check	IMPLEMENTATION	1 LOW	https://github.com/celestiaorg/rsmt2d/issues/162
Use `ErrUnevenChunks` error	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/163
Unnecessary index calculation	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/164
Position and dimensions of row/column slice is not checked	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/169
Check chunk size	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/170
Code duplication	PRACTICE	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/171
Wrong comment for ErrByzabtineData	DOCUMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/173
Good coding practice	PRACTICE	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/177
Good naming practice	PRACTICE	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/179
Unused attribute in test	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/180

Title	Type	Severity	Issue
Comments about extending data are wrong	DOCUMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/192
Naming of test cases is confusing	DOCUMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/rsmt2d/issues/194
Unnecessary usage of errors.As instead errors.Is	IMPLEMENTATION	0 INFORMATIONAL	https://github.com/celestiaorg/celestia-node/issues/2391
Introduce a new variable for clarity	PRACTICE	0 INFORMATIONAL	https://github.com/celestiaorg/celestia-node/issues/2392

ErrByzantineData.Shares are not filled

Title	ErrByzantineData.Shares are not filled
Project	Celestia: Q2 2023
Type	PROTOCOL
Severity	3 HIGH
Impact	2 MEDIUM
Exploitability	3 HIGH
Issue	https://github.com/celestiaorg/rsmt2d/issues/178

Involved artifacts

- rsmt2d/extendeddatacrossword.go

Description

When `ErrByznatineData` is raised it is propagated through `solveCrossword`, `Repair`, `Reconstruct`, and `Retrieve`, where a new `ErrByzantine` is created. If the `ErrByzantineData.Shares` field is not filled, `GetProofsForShares` will return an empty sequence of `sharesWithProof`, and `ErrByzantine` is created with that empty sequence `sharesWithProof`. This is further propagated through GetEDS, SharesAvailable, sample; until a new `BadEncodingProof` is created, and which will end up propagated (the byzantine error is captured in SharesAvailable) to other nodes. Finally, when such a proof is received by other nodes, they call `Validate` on it. Since `p.Shares` is an empty array, an error will be raised upon checking the size of `p.Shares`, here.

`ErrByzantineData.Shares` are not set when verification of newly completed orthogonal vectors returns a `ErrByzantineData` error (first place and second place) that will lead to the previously mentioned flow. On the contrary, these shares are set if a verification of newly completed row or column returns `ErrByzantineData` error.

Problem Scenarios

The byzantine error for data will not be processed properly even if proven. Furthermore, honest peers can get blacklisted.

Recommendation

Fill in the shares with corresponding column or row data if the verification of that newly completed orthogonal column or row returns an `ErrByzantineData`.

Status

Resolved.

`OR` instead `AND` should be used in the test check

Title	`OR` instead `AND` should be used in the test check
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	1 LOW
Impact	2 MEDIUM
Exploitability	1 LOW
Issue	https://github.com/celestiaorg/rsmt2d/issues/162

Involved artifacts

- rsmt2d/datasquare_test.go

Description

The test `TestLazyRootGeneration` calculates root by root for each row and column and appends those to local arrays of roots. In the end, it compares these arrays with those that are calculated by `extendedDataSquare` function `computeRoots`. Test is supposed to fail if at least one of the root arrays is not equal to the expected value, but the check at the end of the test will lead to failure only if both of root arrays are not equal to the ones from EDS.

Problem Scenarios

Test will pass if one of the root arrays does not fit the criteria, but it should fail.

Recommendation

At the end of test if statement should check if any of the root arrays has changed and then return an error. Thus `or` instead of `and` should be used.

Status

Resolved.

Check chunk size

Title	Check chunk size
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	2 MEDIUM
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/170

Involved artifacts

- rsmt2d/datasquare.go

Description

When setting cell in SetCell and setCell a check for size of `newChunk` size should be done.

Problem Scenarios

This could lead to `datasquare` with chunks of different sizes.

Recommendation

Check if the `newCunk` size is equal to size of other chunks in data square and if not return `ErrUnevenChunks` (similar as in newDataSquare).

Status

Resolved.

Position and dimensions of row/column slice is not checked

Title	Position and dimensions of row/column slice is not checked
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	1 LOW
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/169

Involved artifacts

- rsmt2d/datasquare.go

Description

In function setRowSlice (setColSlice), which is used to set an arbitrary slice of row (column), no check if the `newRow` (`newCol`) can be inserted as a slice in the data square row (column) regarding their dimensions is done.

Problem Scenarios

Accessing array elements out of boundaries can lead to panic.

Recommendation

Check if `y+len(newRow)` is greater than `ds.with` (Check id `x+len(newCol)` is greater than `ds.with`).

Status

Resolved.

Good coding practice

Title	Good coding practice
Project	Celestia: Q2 2023
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/177

Involved artifacts

- rsmt2d/extendeddatacrossword.go

Description

Sanity checks within the `prerepairSanityCheck` function use four `if` statements but could use two. Because these checks are run in parallel using Go package `errgroup` the order of these does not matter. Thus grouping can be done that two `if` statements are used instead of four (I.e. Group two (first and second) statements under one `if rowIsComplete` and define `rowIsComplete` right before that if statement; do the same for `colIsComplete` if statements).

Problem Scenarios

Affects readability.

Recommendation

As in the description.

Status

Resolved.

Unnecessary index calculation

Title	Unnecessary index calculation
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/164

Involved artifacts

- rsmt2d/extendeddatasquare.go

Description

In the function `erasureExtendRow` when setting a row slice, the start index calculation `len(shares)-int(eds.originalDataWidth)` is unnecessary because `len(shares)` is the same as the `eds.originalDataWidth`.

Same problem exists in the erasureExtendCol function as well.

Problem Scenarios

This makes the code harder to read and understand and the calculation is unnecessary.

Recommendation

`shares` should be used instead `shares[len(shares)-int(eds.originalDataWidth):]`.

Status

Resolved.

Code duplication

Title	Code duplication
Project	Celestia: Q2 2023
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/171

Involved artifacts

- rsmt2d/extendeddatasquare.go
- rsmt2d/rsmt2d_test.go

Description

- Col function returns a column slice that is a copy of the internal column slice. Instead of doing the copying by itself, `Col` should use deepCopy function, the same as it is done in the function Row to avoid duplication of code.
- Flattened function that returns the concatenated rows of the data square, should be used in TestEdsRepairRoundtripSimple and TestEdsRepairTwice tests to avoid duplication of code.

Problem Scenarios

Duplication of code.

Recommendation

Noted in description.

Status

Resolved.

Wrong comment for ErrByzabtineData

Title	Wrong comment for ErrByzabtineData
Project	Celestia: Q2 2023
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/173

Involved artifacts

- rsmt2d/extendeddatacrossword.go

Description

Comment `ErrByzantineData is thrown when a repaired row or column does not match the expected row or column Merkle root.` is not complete. ErrByzantineData error is returned even if the parity data from a row or a column is not equal to the encoded original data.

Problem Scenarios

Misleading comment.

Recommendation

Update the comment with the case mentioned in description.

Status

Resolved.

Good naming practice

Title	Good naming practice
Project	Celestia: Q2 2023
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/179

Involved artifacts

- rsmt2d/extendeddatasquare.go

Description

Function deepCopy returns a copy of the extendedDataSquare. However, it is assigning the pointer to the deep-copied value to the object pointer `eds`. While this creates no issues because the original value `eds` was referring to remains unchanged, using the same name `eds` for the object pointer and the returned value is confusing.

Problem Scenarios

Makes the function unnecessarily confusing.

Recommendation

A different name for the return object should be used to avoid the confusion.

Status

Resolved.

Unused attribute in test

Title	Unused attribute in test
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/180

Involved artifacts

- rsmt2d/extendeddatacrossword_test.go

Description

There is an unused attribute axis in the test `TestCorruptedEdsReturnsErrByzantineData`.

There are two options:

- remove it
- used as an expected value in the test to check the result of the function Repair

Problem Scenarios

It affects the readability of the test, and there is additional unused data.

Recommendation

Noted in the description.

Status

Resolved.

Comments about extending data are wrong

Title	Comments about extending data are wrong
Project	Celestia: Q2 2023
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/192

Involved artifacts

- rsmt2d/extendeddatasquare.go

Description

The `erasureExtendSquare` function first does extending and filling with arrays of nulls as data chunks of the original data square, then does the filling in the extended parts with encoded original data. However even if the original square data is extended previously, the comments (first, second, third, fourth and fifth) are suggesting that extending is done after them where only filling in the extended shares with the encoded data.

Problem Scenarios

Comments can be misleading about the place of extending the data.

Recommendation

Use something like “filling in the data” instead “extending data within the mentioned comments.

Status

Resolved.

Naming of test cases is confusing

Title	Naming of test cases is confusing
Project	Celestia: Q2 2023
Type	DOCUMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/194

Involved artifacts

- rsmt2d/extendeddatacrossword_test.go

Description

First part of naming of test cases in `TestCorruptedEdsReturnsErrByzantineData` is confusing. There are two possibilities to understand does the naming refer to and both of them are not completely right:

1. This Bad Row or Column refers to the one Axis that should be returned as a part of the `byzData`. Trough debugging it has been confirmed that only for the third test case there is a overlapping between a name and the returned axis.
2. This Bad Row or Column refers to the corruption of data within a row or a column but this is not true for the fourth test case.

Problem Scenarios

Affects the understanding of the test.

Recommendation

Maybe just use the second part of naming (after “/“) which refers will the original or extended data be corrupted.

Status

Resolved.

Use `ErrUnevenChunks` error

Title	Use `ErrUnevenChunks` error
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/rsmt2d/issues/163

Involved artifacts

- rsmt2d/datasquare.go

Description

The `dataSquare` struct requires all chunks to be of the same size. If this is not the case, a predefined error named `ErrUnevenChunks` is returned. This is not done in the `newDataSquare` function (here), where a new error is created and returned.

Problem Scenarios

Affects readability.

Recommendation

Predefined error `ErrUnevenChunks` should be returned if chunk of different size is found.

Status

Resolved.

Unnecessary usage of errors.As instead errors.Is

Title	Unnecessary usage of errors.As instead errors.Is
Project	Celestia: Q2 2023
Type	IMPLEMENTATION
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/celestia-node/issues/2391

Involved artifacts

- celestia-node/share/availability/full/availability.go

Description

This issue is not a part of rsmt2d repository, but was found while inspecting the usage of rsmt2d in celestia-node.

Function `SharesAvailable` checks if an error is not `byzantine.ErrByzantine` here. For the check it uses `errors.As (..)` even if the `byzantineErr` is not used afterwards. Instead `errors.Is(..)` could be used because there is no need for a variable of type `byzantine.ErrByzantine`.

Problem Scenarios

Unnecessary casting.

Recommendation

In the description.

Status

Resolved.

Introduce a new variable for clarity

Title	Introduce a new variable for clarity
Project	Celestia: Q2 2023
Type	PRACTICE
Severity	0 INFORMATIONAL
Impact	0 NONE
Exploitability	0 NONE
Issue	https://github.com/celestiaorg/celestia-node/issues/2392

Involved artifacts

- celestia-node/share/eds/byzantine/bad_encoding.go

Description

This issue is not a part of rsmt2d repository, but was found while inspecting the usage of rsmt2d in celestia-node.

In the Validate function, `merkleRowRoots` and `merkleColRoots` variables are used. `len(merkleRowRoots)` is used to check the size of `p.Shares`. One minor improvement for clarity would be to factor out the `len(merkleRowRoots)` as a separate variable because there is no equivalence between `p.Shares` and `merkleRowRoots`.

Problem Scenarios

This can be misleading when understanding code.

Recommendation

In the description.

Status

Resolved.

Appendix: Vulnerability Classification

For classifying vulnerabilities identified in the findings of this report, we employ the simplified version of Common Vulnerability Scoring System (CVSS) v3.1, which is an industry standard vulnerability metric. For each identified vulnerability we assess the scores from the *Base Metric Group*, the Impact score, and the Exploitability score. The *Exploitability score* reflects the ease and technical means by which the vulnerability can be exploited. That is, it represents characteristics of the *thing that is vulnerable*, which we refer to formally as the *vulnerable component*. The *Impact score* reflects the direct consequence of a successful exploit, and represents the consequence to the *thing that suffers the impact*, which we refer to formally as the *impacted component*. In order to ease score understanding, we employ CVSS Qualitative Severity Rating Scale, and abstract numerical scores into the textual representation; we construct the final *Severity score* based on the combination of the Impact and Exploitability sub-scores.

As blockchains are a fast evolving field, we evaluate the scores not only for the present state of the system, but also for the state that deems achievable within 1 year of projected system evolution. E.g., if at present the system interacts with 1-2 other blockchains, but plans to expand interaction to 10-20 within the next year, we evaluate the impact, exploitability, and severity scores wrt. the latter state, in order to give the system designers better understanding of the vulnerabilities that need to be addressed in the near future.

Impact Score

The Impact score captures the effects of a successfully exploited vulnerability on the component that suffers the worst outcome that is most directly and predictably associated with the attack.

Impact Score	Examples
High	Halting of the chain; loss, locking, or unauthorized withdrawal of funds of many users; arbitrary transaction execution; forging of user messages / circumvention of authorization logic
Medium	Temporary denial of service / substantial unexpected delays in processing user requests (e.g. many hours/days); loss, locking, or unauthorized withdrawal of funds of a single user / few users; failures during transaction execution (e.g. out of gas errors); substantial increase in node computational requirements (e.g. 10x)
Low	Transient unexpected delays in processing user requests (e.g. minutes/a few hours); Medium increase in node computational requirements (e.g. 2x); any kind of problem that affects end users, but can be repaired by manual intervention (e.g. a special transaction)
None	Small increase in node computational requirements (e.g. 20%); code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation

Exploitability Score

The Exploitability score reflects the ease and technical means by which the vulnerability can be exploited; it represents the characteristics of the vulnerable component. In the below table we list, for each category, examples of actions by actors that are enough to trigger the exploit. In the examples below:

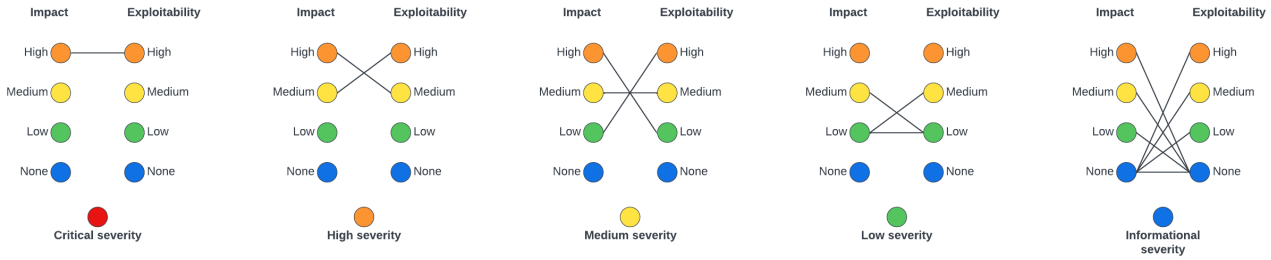
- *Actors* can be any entity that interacts with the system: other blockchains, system users, validators, relayers, but also uncontrollable phenomena (e.g. network delays or partitions).
- *Actions* can be

    - *legitimate*, e.g. submission of a transaction that follows protocol rules by a user; delegation/redelegation/bonding/unbonding; validator downtime; validator voting on a single, but alternative block; delays in relaying certain messages, or speeding up relaying other messages;
    - *illegitimate*, e.g. submission of a specially crafted transaction (not following the protocol, or e.g. with large/incorrect values); voting on two different alternative blocks; alteration of relayed messages.
- We employ also a *qualitative measure* representing the amount of certain class of power (e.g. possessed tokens, validator power, relayed messages): *small* for < 3%; *medium* for 3-10%; *large* for 10-33%, *all* for >33%. We further quantify this qualitative measure as relative to the largest of the system components. (e.g. when two blockchains are interacting, one with a large capitalization, and another with a small capitalization, we employ *small* wrt. the number of tokens held, if it is small wrt. the large blockchain, even if it is large wrt. the small blockchain)

Exploitability Score	Examples
High	illegitimate actions taken by a small group of actors; possibly coordinated with legitimate actions taken by a medium group of actors
Medium	illegitimate actions taken by a medium group of actors; possibly coordinated with legitimate actions taken by a large group of actors
Low	illegitimate actions taken by a large group of actors; possibly coordinated with legitimate actions taken by all actors
None	illegitimate actions taken in a coordinated fashion by all actors

Severity Score

The severity score combines the above two sub-scores into a single value, and roughly represents the probability of the system suffering a severe impact with time; thus it also represents the measure of the urgency or order in which vulnerabilities need to be addressed. We assess the severity according to the combination scheme represented graphically below.



As can be seen from the image above, only a combination of high impact with high exploitability results in a Critical severity score; such vulnerabilities need to be addressed ASAP. Accordingly, High severity score receive vulnerabilities with the combination of high impact and medium exploitability, or medium impact, but high exploitability.

Severity Score	Examples
Critical	Halting of chain via a submission of a specially crafted transaction

Severity Score	Examples
High	Permanent loss of user funds via a combination of submitting a specially crafted transaction with delaying of certain messages by a large portion of relayers
Medium	Substantial unexpected delays in processing user requests via a combination of delaying of certain messages by a large group of relayers with coordinated withdrawal of funds by a large group of users
Low	2x increase in node computational requirements via coordinated withdrawal of all user tokens
Informational	Code inefficiencies; bad code practices; lack/incompleteness of tests; lack/incompleteness of documentation; any exploit for which a coordinated illegitimate action of all actors is necessary