Audit Report

Rusk Node Library

v1.0

September 20, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by Dusk to perform a security audit of the consensus implementation of Rusk.

The objectives of the audit are as follows:

1. Determine the correct functioning of the system, in accordance with the system specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/dusk-network/rusk
Commit	`6a2b0478dbd3d7dfa131461ab5d24a862ffa15ea`
Scope	`node/` `node-data/`
Fixes verified at commit	`253ff91ce8becc1384bccd6393a5f717f5e67f0a` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Rusk is the official Dusk protocol node client and smart contract platform. It is written in Rust and adheres to the Dusk protocol specifications.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of documentation	Medium	-
Test coverage	Low	The node library only contains a few inline tests

Summary of Findings

No	Description	Severity	Status
1	Faulty header validation in the attestation	Critical	Resolved
2	Faults are not written into a block	Critical	Resolved
3	`MAX_PENDING_SENDERS` flag gets ignored	Major	Resolved
4	Transaction filters are unimplemented	Major	Resolved
5	Hardcoded genesis date can lead to a stuck network	Major	Resolved
6	Mempool can grow arbitrarily large	Major	Resolved
7	Incorrect faults processing during the first epoch	Major	Resolved
8	No upper limit for the number of nullifiers	Major	Acknowledged
9	`GetInv` ignores max entry limit	Minor	Resolved
10	Inefficient attestation cache cleanup	Minor	Acknowledged
11	Potential panic during logging	Minor	Resolved
12	Excessive warnings logging due to an underflow	Minor	Resolved
13	Cloning large data structure	Minor	Acknowledged
14	Deleting from immutable view of the database	Minor	Resolved
15	Wrong comments	Informational	Resolved
16	`ConsensusHeader` included in every message	Informational	Acknowledged
17	Missing validation and unenforced invariants	Informational	Resolved
18	Outstanding `TODO` and `FIXME` comments	Informational	Acknowledged
19	Unused code	Informational	Resolved
20	Code duplicates	Informational	Resolved
21	Use of magic numbers decreases maintainability	Informational	Acknowledged

Detailed Findings

1. Faulty header validation in the attestation

Severity: Critical

In `node/src/chain/header_validation.rs:409`, the `verify_block_att` identifies all committee members as voters by merging them without validating that they voted.

This might lead to faulty validation of the block header, i.e. in labeling a non-valid candidate header as valid, which is considered a critical error.

Recommendation

We recommend that only voting members are considered when verifying the block header.

Status: Resolved

2. Faults are not written into a block

Severity: Critical

In `consensus/src/proposal/block_generator.rs:140`, the vector `faults` is saved into the new block.

However, the vector is empty because it is only initialized on line `114` with expression `Vec::<Fault>::new()`. The header of this block doesn't contain any information about the faults as well, because the merkle root is derived from the empty vector.

Recommendation

We recommend initializing the fields `faults` and `faultroot` field of the block and its header with actual information about faults.

Status: Resolved

3. `MAX_PENDING_SENDERS` flag gets ignored

Severity: Major

In `node/src/network.rs:43-51` the `reroute` function ignores if there are too many messages to be rerouted. If the `MAX_PENDING_SENDERS` flag is exceeded, the `pending_senders` counter is reset to zero and a warning message is logged. However no

other actions such as limiting the messages are performed and logic for intervention for operators is provided. This could lead to an excessive rerouting of messages and potentially an overload of the operating or other nodes.

Recommendation

We recommend implementing a logic that prevents excessive rerouting of messages.

Status: Resolved

4. Transaction filters are unimplemented

Severity: Major

In `node/src/mempool.rs:51-59` the `TxFilter` is unimplemented.

As a consequence is it not checked if a transaction is a duplicate, nullified, or exists in the mempool. While this can be a critical issue in production, unimplemented features are reported as major.

Recommendation

We recommend implementing the unfinished feature and resolving all `TODO`s.

Status: Resolved

5. Hardcoded genesis date can lead to a stuck network

Severity: Major

At node startup, as defined in `node/src/chain.rs:276`, the genesis block is generated if the current chain tip is not yet initialized in local storage. However, the genesis date is hardcoded to March 25, 2024 as can be seen in `node/src/chain/genesis.rs:14`. Not only a network based on this particular commit is difficult to be launched, but also this approach in general poses challenges if the network launch fails and needs rescheduling. Genesis date modification requires source code modification, followed by recompilation and distribution of the binary to the node environment. Providing the genesis date via a genesis config could streamline this process.

More importantly, the genesis date is not validated against the local clock. The timestamp for the first candidate block, built atop the genesis block, is determined by `consensus/src/proposal/block_generator.rs:122`. This timestamp must fall within a specific time range, relative to clocks of other provisioners in the network. Delay in block production is a slashable event, so a block timestamp must not be too far in the past. Delayed blocks are rejected in `node/src/chain/header_validation.rs:105`. This

means that the next block after the genesis cannot be built without violating the protocol rules.

At the same time, setting the timestamp of the genesis block too far into the future is also problematic, as it prevents the block's acceptance by other nodes, see `node/src/chain/header_validation.rs:115`.

Recommendation

We recommend enabling users to provide the genesis date as well as validating it against the local clock, aborting the launch if this validation fails. Provisioner instructions should highlight the need to synchronize node clocks via NTP.

Status: Resolved

6. Mempool can grow arbitrarily large

Severity: Major

In `node/src/mempool.rs`, the function `accept_tx` does not perform any checks on the number of transactions that are already in the mempool. An attacker could abuse this by spamming a lot of low-value transactions and overwhelm nodes with these transactions, which can ultimately lead to a Denial of Service for particular nodes or the whole network.

Recommendation

We recommend introducing an upper limit on the number of transactions in the mempool and developing eviction rules that prevent cheap attacks. Ethereum clients like `geth` can be an inspiration for this implementation, as they contain such a limit and corresponding eviction rules.

Status: Resolved

7. Incorrect faults processing during the first epoch

Severity: Major

In `node/src/chain/header_validation.rs:299-330`, function `verify_faults` is defined. The function iterates through all input faults, validates them and checks for double faults. However, arithmetic underflows involving unsigned integers disrupt the correct execution of this function.

First, in `node/src/ledger/faults.rs:146`, the `validate` function ensures a fault hasn't expired by subtracting the constant `EPOCH` from its `current_height`. If only a few rounds have elapsed since genesis, this subtraction will result in a number almost as large as `u64::MAX`, causing the fault to be mistakenly marked as expired.

Secondly, even with an accurate expiration check, a similar underflow occurs in `node/src/chain/header_validation.rs:320` during the calculation of the `start_height` parameter, which determines which faults are evaluated to detect double faults. A large number near `u64::MAX` would cause all faults to be retrieved, starting from the chain's tip, leading to any fault during the first epoch being misclassified as a double fault.

Recommendation

We recommend using checked arithmetic and enabling overflow checks to prevent functioning in the presence of arithmetic issues in production.

Status: Resolved

8. No upper limit for the number of nullifiers

Severity: Major

The function `accept_tx` in `node/src/mempool.rs` iterates over all the nullifiers that a user provided and performs a database lookup for each ID. Because there is no upper limit for the number of nullifiers, an attacker can abuse this feature to perform DoS attacks on a node: If they send a lot of transactions that each contain a large number of nullifiers, a node may be overloaded and not able to process any other instructions.

Recommendation

We recommend introducing an upper limit on the number of nullifiers in a transaction.

Status: Acknowledged

Team response: This limit is already enforced by the Phoenix transaction proof. In other words, a valid transaction cannot have more than 4 nullifiers.

9. `GetInv` ignores max entry limit

Severity: Minor

The function `handle_inv` in `node/src/databroker.rs` has an argument `m` with the type `node_data::message::payload::Inv`. This struct has a field `max_entries` to limit the number of entries returned. The function `handle_get_resource` reads this field and limits the response by the value (if it is greater than zero). However, `handle_inv` ignores the field completely and does not limit the returned entries in any way.

Recommendation

We recommend reading the field and limiting the number of entries returned, i.e. adapting the logic from `handle_get_resource`.

Status: Resolved

10. Inefficient attestation cache cleanup

Severity: Minor

The functions `attach_att_if_needed` and `on_idle` in `node/src/chain/fsm.rs` call `retain` on the `HashMap` `self.attestations_cache` to clean it up. According to the Rust documentation, "this operation takes O(capacity) time instead of O(len) because it internally visits empty buckets too."

Therefore, the cleanup can be highly inefficient. Even if the hash map only contains a few entries, every bucket will be visited whenever the function is called.

Recommendation

We recommend optimizing the cleanup operation (and potentially using other data structures) to make sure that the node is not overloaded when cleaning up the cache.

Status: Acknowledged

Team response: The team agrees with the issue and the recommended solution, which is planned for version 1.2.0 of the protocol (Issue #2155).

11. Potential panic during logging

Severity: Minor

Within `node-data/src/ledger.rs:35-46`, the `to_str` function is defined. It takes a byte array as input, converts it into a hexadecimal text string, and formats it for readability.

However, at line `42`, the hexadecimal string undergoes a split operation using the `split_at` function, with a constant parameter `OFFSET` set to `16`. This operation could cause a panic if the string is shorter than `OFFSET`.

Although no specific issue has been identified, this function is used extensively throughout the codebase.

Recommendation

We recommend validating the size of the hexadecimal string before formatting it. If it is short enough, it can be returned as-is.

Status: Resolved

12. Excessive warnings logging due to an underflow

Severity: Minor

Within `node/src/network.rs:43-68`, the reroute function implements forwarding messages to the upper layer. An atomic counter `self.pending_senders` is used to monitor the current number of messages in transit. Should incrementing this counter surpass the `MAX_PENDING_SENDERS` threshold, a warning is logged, and the counter is reset to `0`, see lines `49-50`.

Then a future is spawned which sends the messages. Upon completion, it decrements the counter on line `64`. However, after hitting the `MAX_PENDING_SENDERS` limit the counter is `0`, so decrementing it leads to an underflow and very large number `u64::MAX`.

If another simultaneous future ends before the next time the `reroute` is called, the counter is decreased further to `u64::MAX - 1`. With a steady influx of messages, the counter remains near `u64::MAX`, causing the system to interpret the threshold as hit and reset the counter to zero repeatedly. The pattern repeats until exactly 1 future completes between successive `reroute` calls.

This behavior leads to the logging of a warning with every message dispatch. Under heavy traffic, this frequent logging could significantly hinder node performance.

Recommendation

We recommend introducing a separate counter or timestamp to rate limit warnings, lowering the severity level of the log message and enabling overflow checks.

Status: Resolved

13. Cloning large data structure

Severity: Minor

In `consensus/src/user/provisioners.rs:91`, the mapping `self.current` is cloned. This mapping contains public keys of all current provisioners with their stakes. Cloning of huge data structures can affect the node performance.

Recommendation

We recommend utilizing `std::mem::replace` for efficient move of the value without cloning.

Status: Acknowledged

Team response: The team agrees with the issue and the recommended solution, which is planned for version 1.2.0 of the protocol (Issue #2151).

14. Deleting from immutable view of the database

Severity: Minor

Within `node/src/mempool.rs:126-155`, the `accept_tx` function is implemented to accept new transactions into the node's mempool. On line `140`, the variable view is introduced, representing a read-only view of the database. Starting with line `155`, the code iterates through all transactions in the database identified by the nullifiers of the transaction being accepted. If a transaction's gas price is lower than that of the accepted transaction, it is removed from the view on line `158`.

However, this operation has no effect because the view is read-only.

Recommendation

We recommend using the `update` function to obtain a mutable view of the database.

Status: Resolved

15. Wrong comments

Severity: Informational

The comment above the function `load_provisioners_keys` in `node-data/src/bls.rs` states that the `RUSK_WALLET_PWD` variable is read to unlock the wallet files. However, the function reads the environment variable `RUSK_CONSENSUS_KEYS_PASS`.

Another instance of misleading comments can be observed in the `node-data/src/bls.rs` file. Functions `load_keys` and `read_from_file` are documented with `/// Panics on any error` comments despite that the functions already implement proper error propagation using `Result`.

Recommendation

We recommend correcting the comments.

Status: Resolved

16. `ConsensusHeader` included in every message

Severity: Informational

The struct `Message` that is defined in `node-data/src/message.rs` has a field `header` with type `ConsensusHeader`. This header is only required for consensus messages, but it is included in every message.

Recommendation

We recommend refactoring the `Message` struct such that no unnecessary fields are sent and the network usage is reduced.

Status: Acknowledged

Team response: The `ConsensusHeader` field is only serialized for Consensus messages, so it does not affect other messages at a protocol level.

Nevertheless, it is planned to be removed for version 1.2.0 (Issue #1754)

17. Missing validation and unenforced invariants

Severity: Informational

1. In `node/src/network/frame.rs:25`, field checksum is declared. It is initialized on line `35`, but it is never utilized to verify a header consistency.
2. In `node-data/src/ledger/attestation.rs:35`, the structure `StepVotes` is considered to be empty if any of `bitset` or `aggregate_signature` is zero. However, if only one of these fields is zero then it must be treated as an inconsistent structure.
3. In `node/src/databroker.rs:303`, a child block was added to the inventory. This is part of the traversal from one block specified by its hash through all of its descendants. To ensure consistency of the database, the value of `prev_block_hash` of each child pair should be checked against the hash of its parent.
4. Within `consensus/src/user/provisioners.rs:255-263`, edge case of a single active provisioner is handled. Set `members` should be checked to be non-empty in the end. It can be useful to double-check that `exclusion` set is equal to `eligibles` set in this edge case.
5. In `node-data/src/ledger.rs:38`, the function `to_str` returns an error because the length of the hexadecimal string is not even. However, the function `hex::encode` called on line `36` always returns even strings.

Recommendation

We recommend adding validation checks and utilizing `debug_assert!` for invariants checks.

Status: Resolved

18. Outstanding `TODO` and `FIXME` comments

Severity: Informational

The codebase contains multiple `TODO`, `FIXME` and `FIX_ME` comments, totaling to 24 instances. This approach to documenting issues and outstanding work has several disadvantages:

- `FIXME` and `TODO` comments become stale more easily comparing to proper tasks tracked in an issue tracker, as a consequence such comments usually do not progress towards resolution
- Such comments highlight the weakest spots in the codebase, this can be exploited by a malicious party since the code is publicly available
- Information in comments of any kind must be maintained, otherwise it becomes misleading for the future developers or reviewers

Recommendation

We recommend moving the comments into an issue tracking system and resolving them.

Status: Acknowledged

Team response: As suggested, all items are planned to be removed from the code and converted into issues before version 1.2.0 (Issue #2148)

19. Unused code

Severity: Informational

The codebase contains several declarations that are not utilized:

- Function `send_and_wait` is declared in `node/src/lib.rs:69-75` is declared and defined within `node/src/network.rs:327-361`, but it is never called from anywhere.
- Function `verify_block_header` declares parameter `disable_winning_att_check` in `consensus/src/operations.rs:88`, however its only usage in `consensus/src/validation/step.rs:84` sets the parameter to true.

Unused definitions reduce maintainability and readability of the codebase.

Recommendation

We recommend removing unused functions and parameters.

Status: Resolved

20. Code duplicates

Severity: Informational

File consensus/src/commons.rs defines the function get_current_timestamp that creates a timestamp of the current moment in time, converts it into Unix time format and unwraps any potential errors.

The very similar code snippets occurs in 4 other places:

- `node/src/network.rs:250-253`
- `node/src/chain/header_validation.rs:110-113`
- `consensus/src/proposal/step.rs:114-117`
- `node-data/src/message.rs:943-946`

Code duplicates reduce maintainability and readability of the codebase.

Recommendation

We recommend replacing code duplicates with calls to the function.

Status: Resolved

21. Use of magic numbers decreases maintainability

Severity: Informational

Throughout the codebase, hard-coded number literals without context or a description are used. Using such “magic numbers” goes against best practices as they reduce code readability and maintenance as developers are unable to easily understand their use and may make inconsistent changes across the codebase.

Instances of magic numbers are listed below:

- `node/src/lib.rs:230,237` with an inconsistency as the comment says: "Result code 1 means abort all services.", but a `2` causes `abort_all()`.
- `node/src/databroker/conf.rs:24,25`
- `node/benches/accept.rs:114,120,123,126`

Recommendation

We recommend defining magic numbers as constants with descriptive variable names and comments, where necessary.

Status: Acknowledged

Team response: A thorough review of magic numbers in the code is planned for version 1.2.0 of the protocol (Issue #2126)