SMART CONTRACTS REVIEW

zokyo

April 22nd 2025 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING INERTIA FINANCE

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 0 Critical issues: 0 points deducted
- 2 High issues: 2 resolved = 0 points deducted
- 2 Medium issues: 2 resolved = 0 points deducted
- 4 Low issues: 3 resolved and 1 acknowledged = - 2 points deducted
- 0 Informational issues: 0 points deducted

Thus, 100 - 2 = 98

TECHNICAL SUMMARY

This document outlines the overall security of the Inertia Finance smart contract/s evaluated by the Zokyo Security team.

The scope of this audit was to analyze and document the Inertia Finance smart contract/s codebase for quality, security, and correctness.

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ton network's fast-paced and rapidly changing environment, we recommend that the Inertia Finance team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Inertia Finance repository:
Core Repo: https://github.com/inrt-fi/inertia-core
Move module repository : https://github.com/inrt-fi/move-modules
move-modules-main.zip - 79b66b519515e9a2155299b4f77d8e0d74535cad0945c748725adaa308af063b
nertia-core-main.zip - 72edb740771014bc4374bf8177489c6dcec197f3b6b837c81ff252f912c4d899

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- **Inertia Core:**
  - Go-lang WasmVM based on the MiniWast from Initia Labs
  - Approximately added over 100k lines of the Go code
  - Added various functionality
- **Move Module:**
  - 17 Move files
  - 3,537 lines of Move code
  - No test cases

- **The list of contracts, the MOVE part of the scope:**
  - asset_utils.move
  - chain.move
  - char_utils.move
  - common.move
  - cosmwasm_coin.move
  - decimals.move
  - farm.move
  - l2_denom_mapper.move
  - lst.move
  - numbers.move
  - point_farm.move
  - reward_manager.move
  - sinit.move
  - snapped_item.move
  - snapped_map.move
  - staking_utils.move
  - string_utils.move

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Inertia Finance smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Thorough manual review of the codebase line by line.

Executive Summary

The Inertia codebase exhibits a modular and layered architecture, with a clear separation between application logic and protocol-specific operations. The Move modules are well-structured, leveraging Initia's framework idioms effectively, and include helpful inline documentation, though a few areas could benefit from more extensive commenting on business logic and invariants. Overall, both parts demonstrate a mature and maintainable codebase, though aligning consistently with upstream Miniwasm improvements would further enhance long-term security and compatibility. The Inertia team has introduced new functionality to the code since the initial audit. Reviewing the latest commits would therefore require a full-scope audit from scratch.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Inertia Finance team and the Inertia Finance team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Significant Changes Detected in Wasm Contract (ics721_base.wasm)	High	Resolved
2	Repayments Allowed During Liquidation Despite Documentation Restriction	High	Resolved
3	Missing Error Check in IBC Hook May Cause Nil-Pointer Panic (DoS)	Medium	Resolved
4	Unbounded Reward Amounts in Distribution Schedule May Enable Misconfiguration or Abuse	Medium	Resolved
5	Removal of Mempool Service Registration May Reduce Observability	Low	Resolved
6	Auction and Mempool Swagger Specs Removed from Client Configuration	Low	Resolved
7	FetchGenesisCmd Removed from CLI Root Command	Low	Resolved
8	Refund Path in Auction May Revert Entire Bid on Failure	Low	Acknowledged

HIGH-1 | RESOLVED

Significant Changes Detected in Wasm Contract (ics721_base.wasm)

File: contrib/wasm/ics721_base.wasm

The .wasm contract changed without matching source code or reproducible build artifacts. Entry-point logic (execute, instantiate, query) was modified or restructured. Internal function definitions have shifted, suggesting behavioral changes. These undocumented changes may affect NFT minting/burning logic or IBC packet handling. This introduces the potential for unauthorized state changes, hidden logic, or weakened access control enforcement.

Recommendation:
Provide source code and the reproducible build pipeline for this .wasm contract. A separate audit should be conducted for the modified WASM code to ensure functional integrity and absence of backdoors.

HIGH-2 | RESOLVED

Repayments Allowed During Liquidation Despite Documentation Restriction

File: x/loan/keeper/msg_server_repay.go

According to the protocol's official architecture documentation, once liquidation is triggered, the user's loan account is to be locked, allowing only deposits during this period. However, in the current implementation, the MsgRepay handler does not check the IsLiquidating flag and allows repayments to proceed. This contradicts the documented behavior and introduces a logic inconsistency in the liquidation process.

Recommendation:
Either add a check in the MsgRepay handler to block repayments during liquidation, or—if repayment is an intended bailout feature—it must trigger auction cancellation, refund all active bids, and fully resolve the liquidation state across involved modules.

Missing Error Check in IBC Hook May Cause Nil-Pointer Panic (DoS)

File: app/ibc-hooks/receive.go @93

The check for err == nil was removed from the logic that unmarshals hookData. Without this check, if deserialization
fails (e.g., malformed IBC packet), hookData remains nil. Subsequent access to fields like hookData.Message will trigger a runtime nil-pointer panic, resulting in a chain halt or transaction rejection.

Example Impact:
An attacker or misconfigured counterparty could send a malformed IBC packet (invalid JSON, incorrect type). During packet handling, the hook attempts to access a field from the nil hookData. This causes a panic, crashing the chain halt or crash depending on panic handling logic.

Recommendation:

Reintroduce the err != nil check or add an explicit if hookData == nil guard. Add a unit test that simulates a malformed packet and ensures graceful failure. Consider wrapping hook handling in a recover() block if this logic is positioned in a chain-critical path.

Unbounded Reward Amounts in Distribution Schedule May Enable Misconfiguration or Abuse

File: point_farm.move
Function: register_distribution_schedule()

The function allows adding a new reward distribution schedule but does not enforce an upper bound on the amount field. This could result in unreasonably large or economically dangerous reward amounts being scheduled, either by mistake or through malicious governance. Since there is no check to ensure the scheduled rewards are backed by actual token reserves or capped against total supply, this may lead to future inconsistencies or economic failures in the staking program.

Recommendation:

Add validation logic to ensure amount is within reasonable limits. Optionally require the amount to be backed by actual treasury balance at schedule time. Consider adding a global cap or max per-schedule threshold

LOW-1 | RESOLVED

Removal of Mempool Service Registration May Reduce Observability

File: app/app.go

The removal of blockservice.RegisterMempoolService disables gRPC-based mempool transaction introspection. While this does not affect consensus or transaction validity, it may reduce observability for developers, relayers, or indexers. If this functionality was unused, its removal is benign. Otherwise, the change should be explicitly acknowledged and tested for downstream impact.

Recommendation:
Document the rationale for removal. Reintroduce the feature if observability or mempool simulation is required for tooling or operational purposes.

LOW-2 | RESOLVED

Auction and Mempool Swagger Specs Removed from Client Configuration

File: client/docs/config.json

The OpenAPI definitions for auction/v1 and mempool/v1 query endpoints were removed from the client documentation configuration. This affects auto-generated client documentation and reduces visibility into runtime query capabilities.
It may also cause issues with frontend tools that rely on Swagger-generated UIs or OpenAPI compliance.

Recommendation:
Confirm whether the auction and mempool query endpoints have been deprecated or relocated. If still active, reintroduce Swagger documentation or provide documentation through alternative means. If deprecated, communicate the removal clearly for integrators and frontend teams.

	NerdToken.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

LOW-3 | RESOLVED

FetchGenesisCmd Removed from CLI Root Command

File: cmd/minitiad/root.go

The CLI command FetchGenesisCmd was removed. This command typically supports automated retrieval of a network's genesis file. Without it, node operators must manually download genesis files, increasing the risk of misconfiguration and introducing operational friction. This may hinder automation or delay onboarding for devnets or testnets.

Recommendation:
If this feature was unused, document the rationale for removal. If operators previously relied on it, provide an alternative method for secure genesis file retrieval with checksum verification.

LOW-4 | ACKNOWLEDGED

Refund Path in Auction May Revert Entire Bid on Failure

File: x/liquidation/keeper/auction_liquidator.go

The ProcessLiquidation() function attempts to refund outbid participants using SendCoinsFromModuleToAccount. If any refund operation fails (e.g., due to invalid address or bank module rejection), the entire bid transaction is reverted. While this is acceptable in most cases, it creates fragility and may be abused if refund paths are deliberately broken.

Recommendation:
Wrap the refund loop with isolated error handling to prevent a single refund failure from reverting the entire bid process. Log refund errors for traceability, and add unit tests to simulate refund failure scenarios.

Client comment: we will leave this issue as "Acknowledged". I understand the intent of the review, but Refund is also an important part of the liquidation process. If any errors occur during this process, the Bid should not be accepted and such errors should be detected/ blocked in advance.

We are grateful for the opportunity to work with the Inertia Finance team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo Security recommends the Inertia Finance team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo