Inferium

SMART CONTRACTS REVIEW

zokyo

February 12nd 2025 | v. 1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING INFERIUM

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: 0 points

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

Thus, 100 - 2 = 98

TECHNICAL SUMMARY

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

The scope of this audit was to analyze and document the Inferium 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 Ethereum network's fast-paced and rapidly changing environment, we recommend that the Inferium 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 Inferium repository:
Repo: https://github.com/inferiumai/inferium_contracts/commit/35d918e013a4ceab11c71491e48fb8b261826b35

Last commit - d5e0a077919654fcdfd93f823d8f570e0017e3c8

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

- ./contracts/token/InferiumToken.sol
- ./contracts/token/MintableBaseToken.sol
- ./contracts/token/BaseToken.sol
- ./IMintable.sol

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 Inferium 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 provided Solidity smart contracts define a hierarchical token system with additional features for minting, pausing, blacklisting, and secure token withdrawals. The `BaseToken` contract extends OpenZeppelin's `ERC20Burnable`, `Pausable`, and `Ownable` modules, allowing the owner to pause/unpause transfers and withdraw tokens. `MintableBaseToken` builds on `BaseToken` by introducing minting capabilities restricted to designated minters and enforcing a supply cap through `ERC20Capped`. It also incorporates a blacklist mechanism to restrict transfers for flagged accounts. `InferiumToken` is a specific implementation of `MintableBaseToken` with a fixed maximum supply of 250 million tokens. The contracts leverage OpenZeppelin's `SafeERC20` for secure token transfers and include event emissions for tracking administrative actions

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 Inferium team and the Inferium 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	Centralization Risk	Low	Acknowledged
2	Lack of Two-Step Ownership Transfer	Low	Resolved
3	Floating Pragma	Informational	Resolved

LOW-1 | ACKNOWLEDGED

Centralization Risk

The smart contracts grant significant control to the contract owners through several functions. This centralization poses a substantial risk, as it places considerable trust and control in a single entity. If the owner's private key is compromised, it could lead to catastrophic disruptions or malicious misuse of the contract.

Recommendation:
Use a multi-signature wallet for executing Owner functions. This requires multiple authorized signatures to approve critical actions, reducing the risk of a single point of failure.

Client's comment: We can set the owner to be gnosis later.

LOW-2 | RESOLVED

Lack of Two-Step Ownership Transfer

The BaseToken contract does not implement a two-step process for transferring ownership. In its current state, ownership can be transferred in a single step, which can be risky as it could lead to accidental or malicious transfers of ownership without proper verification.

Recommendation:
Implement a two-step process for ownership transfer where the new owner must explicitly accept the ownership. It is advisable to use OpenZeppelin's Ownable2Step.

Floating Pragma

The smart contract uses a floating pragma version `(^0.8.28)`. Contracts should be deployed using the same compiler version and settings as were used during development and testing. Locking the pragma version helps ensure that contracts are not inadvertently deployed with a different compiler version.

Recommendation:
Consider locking the pragma version to a specific, tested version to ensure consistent compilation and behavior of the smart contract.

	./contracts/token/InferiumToken.sol ./contracts/token/MintableBaseToken.sol ./contracts/token/BaseToken.sol ./IMintable.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

We are grateful for the opportunity to work with the Inferium 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 Inferium team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo