Smart Contract Audit Report

1. Overview

- **Project Name:** MAIGA
- **Contracts Audited:**
  - File: INCENTIVES.SOL
- **Code base:** Provided by file

The contract is upgradeable.

Note: The deployer can replace the old contract with a new one with new features. Be aware of this because the owner can add new features that may have a negative impact on your investments. Our team did not audit other contracts associated with the project. We recommend investors do their own research before investing.

2. Executive Summary

This audit aims to assess the security of the smart contracts for the ICO protocol. We identified issues:

- High Risk
- Medium Risk
- Low Risk
- Informational / Gas Optimization Suggestions

All high and medium severity issues have been addressed / partially addressed / not addressed by the development team at the time of publishing this report.

3. Methodology

Our audit involved the following steps:

1. **Manual Code Review** – Analyzed logic for correctness, authorization, error handling, etc.
2. **Automated Testing** – Used static analysis tools such as Slither, MythX, or Hardhat plugins.
3. **Gas Usage Review** – Identified areas for optimization.
4. **Dependency Analysis** – Assessed third-party libraries (e.g., OpenZeppelin).

4. Risk Classification

Severity	Description
High	Critical vulnerabilities that can lead to loss.
Medium	Significant issues but not critical.
Low	Minor issues, often best practices.
Informational	No direct risk but useful suggestions.

5. Inheritance Graph

N/A

6. Ownership Privileges

INCENTIVES.SOL

- The owner can withdraw oMaiga tokens from the contract.
- The owner can update the authorizedSigner address.

7. Findings

7.1 High Severity

7.2 Medium Severity

[M-01] Inconsistency in tier progression logic.

- **Description:** The claim() function only advances to the next tier when the current tier is completely exhausted (when tokensClaimed >= tokenAllocation), while getCurrentClaimableAmount() unconditionally increments the tier index (tempTierIndex++) after every loop iteration regardless of whether the tier is full. This causes the view function to calculate token amounts using incorrect tier pricing, leading to discrepancies between what users see as claimable amounts and what they actually receive when claiming. The bug manifests when processing contributions that span multiple tiers or when tiers have remaining capacity, resulting in frontend displaying wrong information and potential user confusion about expected token allocations.

- **Mitigation:** Modify the getCurrentClaimableAmount() function to use the same tier progression logic as the claim() function by replacing the unconditional tempTierIndex++ increment with a conditional check: only increment tempTierIndex when the current tier becomes fully allocated (if (currentTierInfo.tokensClaimed + tokensForCurrentBnbAmount >= currentTierInfo.tokenAllocation) { tempTierIndex++; }). This ensures both functions maintain identical tier progression behaviour and return consistent results for the same input parameters.

- File: INCENTIVES.SOL

- Line: 70-148

- **Status:** Open

[M-02] The owner can drain the token.

- **Description:** The withdrawTokens() function reduces totalTokenSupply after transferring tokens to the owner, but doesn't validate that the remaining supply stays above totalTokensClaimed. This creates a scenario where if the owner withdraws too many tokens, totalTokensClaimed becomes greater than or equal to the reduced totalTokenSupply, causing the safety check require(totalTokensClaimed < totalTokenSupply, "Sale completed") in the claim() function to permanently fail. Once this condition is triggered, all future claim attempts will revert with "Sale completed" even though the contract may still hold sufficient tokens for legitimate claims. This represents both an accidental risk (owner miscalculating withdrawal amounts) and a potential centralization attack vector where the owner could maliciously lock user funds.

- **Mitigation:** Add a safety check in the withdrawTokens() function to ensure totalTokenSupply - amount >= totalTokensClaimed before allowing any withdrawal. This

validation should occur before the token transfer to prevent the owner from withdrawing more tokens than the excess amount (tokens not yet allocated to users), ensuring that all legitimate claims can still be processed after any withdrawal operation. Additionally, consider implementing a time-lock or multi-signature requirement for withdrawal operations to prevent accidental or malicious disabling of core contract functionality

- File: INCENTIVES.SOL
- Line: 194-200
- **Status:** Open

[M-03] Unlimited ICO Pause Duration

- **Description:** The Owner can pause the ICO indefinitely without time restrictions. This centralized control could be abused to manipulate market timing or prevent user participation, effectively locking the ICO state and blocking purchases for an unlimited duration. Implement maximum pause duration and allow forced resume after timeout.
- **Mitigation:** Add a pause cooldown period between consecutive pauses.
- File: ICO.sol
- Line: 96-99, 101-105
- **Status:** Open

[M-04] Missing Parameter Thresholds and Validation.

- **Description:** Critical contract parameters lack validation checks and thresholds. The owner can set prices to zero or extremely high values, modify caps without maintaining soft cap < hard cap relationship, and set illogical timeline sequences. This could break core contract functionality or enable manipulation. For example, setting presale price higher than public price, or setting hard cap below soft cap would create invalid sale conditions and potentially lock user funds.
- **Mitigation:** Implement strict bounds checking and logical validation for all parameter modifications. Add minimum/maximum thresholds for prices, enforce cap relationships, and validate timeline sequences. Use constants for boundaries and require proper validation in all setter functions.
- File: ICO.sol
- Line: 264-268, 284-291
- **Status:** Open

7.3 Low Severity

[L-01] Missing events arithmetic.

- **Description:** It is recommended to emit all the critical parameter changes.
- File: INCENTIVES.SOL
- Line: L194-200, L202-204
- **Status:** Open

[L-02] Missing zero or dead address check.

- **Description:** It is recommended to check that the address cannot be set to zero or dead.
- File: INCENTIVES.SOL
- Line: L202-204
- **Status:** Open

[L-03] Missing 'isContract' check.

- **Description:** The contract lacks a validation check to ensure that specific parameters are contract addresses. Without this check, there is a risk that non-contract addresses (such as externally owned accounts, or EOAs) could be mistakenly set for parameters intended to reference other contracts. This could lead to failures in executing critical interactions within the contract, as EOAs do not support contract-specific functions.
- **Mitigation:** Implement a validation check to ensure that parameters designated as contract addresses are verified as such. This can be done using Solidity's Address library function isContract, which checks if an address has associated contract code.
- File: INCENTIVES.SOL
- Line: 42-68
- **Status:** Open

7.4 Informational/Optimization Severity

[I-01] Floating pragma solidity version.

- **Description:** Adding the constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.
- File: INCENTIVES.SOL

- Line: L2
- **Status:** Open

8. Conclusion

The smart contracts are generally well-written and follow modern best practices. After addressing the findings noted, the contracts should be considered safe for deployment, assuming no changes are made without further auditing.

9. Disclaimer

This audit report is not the final deployed version of the contract. So, any changes made after the audit will be considered out of the scope of the audit, and we will not be responsible for any changes related to that.