Three Sigma

Code Audit

Mini-ICO.cc

Mini-ICO Token Launcher

Disclaimer

Code Audit

Mini-ICO Token Launcher

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code's manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Mini-ICO Token Launcher

Table of Contents

Summary

Code Audit

Mini-ICO Token Launcher

Summary

Three Sigma audited Mini-ICO in a 4 person week engagement. The audit was conducted from 27/01/2025 to 07/02/2025.

Protocol Description

Mini-ICO is a high-energy, gamified token launch platform that turns fundraising into a dynamic and rewarding experience. It prioritizes early participation with a boost system, letting the fastest buyers access the best prices on a bonding curve. Beyond fueling community excitement, Mini-ICO also rewards fundraisers based on their achievements and progress, ensuring creators are incentivized to drive engagement and growth. With real-time leaderboards, referral rewards, and liquidity-backed momentum, it delivers a fair yet competitive launch experience that maximizes success for both participants and project creators.

Scope

Code Audit

Mini-ICO Token Launcher

Scope

Filepath	nSLOC
contracts/abstracts/ListingManager.sol	129
contracts/abstracts/UniSwapper.sol	104
contracts/CreatorLock.sol	50
contracts/FeeManager.sol	205
contracts/GovTreasury.sol	83
contracts/ICO.sol	8
contracts/Launchpad.sol	325
contracts/libraries/CurveLib.sol	28
contracts/MiniICO.sol	9
contracts/NoGasICOSwapper.sol	63
contracts/UniFeeHook.sol	143
contracts/UniPoolCreator.sol	96
contracts/Vote.sol	124
contracts/VoteManager.sol	172
contracts/Zapper.sol	144

Assumptions

OpenZeppelin and Uniswap v4 are considered secure.

Methodology

Code Audit

Mini-ICO Token Launcher

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Mini-ICO Token Launcher

Project Dashboard

Application Summary

Name	Mini-ICO
Commit	bd01c4d
Language	Solidity
Platform	EVM

Engagement Summary

Timeline	27/01/2025 to 07/02/2025
№ of Auditors	2
Review Time	4 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	1	0	1

None	6	3	3

Category Breakdown

Suggestion	6
Documentation	0
Bug	1
Optimization	0
Good Code Practices	0

Code Maturity Evaluation

Code Audit

Mini-ICO Token Launcher

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system's resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. No arithmetic errors were found.
Centralization	Satisfactory. The owner has control over a marginal functionality, but most of the design is decentralized.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Satisfactory. The contracts are not upgradeable.
Function Composition	Satisfactory. Functionality is well split into helper functions.
Front-Running	Satisfactory. No front-running issues were found.
Monitoring	Satisfactory. Events were correctly emitted.
Specification	Satisfactory. In-depth and well structured high-level specification as well as codebase documentation.
Testing and Verification	Satisfactory. Extensive test code coverage as well as usage of tools and different test methods.

Findings

Code Audit

Mini-ICO Token Launcher

Findings

3S-Mini-ICO-L01

`updateEpochConfig` when called with diff duration can cause bans to be lifted

Id	3S-Mini-ICO-L01
Classification	Low
Severity	Low
Likelihood	Low
Category	Bug
Status	Acknowledged with note “We’re aware of this issue, but we believe it’s more suitable to handle this logic off-chain. We will determine whether currentEpoch() should return a higher value when updateEpochConfig is called based on the actual situation.”

Description

The **`updateEpochConfig`** function is used to update the elements of **`epochConfig`**, **`duration`** included. If the **`duration`** gets changed, the **`currentEpoch()`** function might return a higher value than before.

A side effect of this situation is that banned tokens will get their **`fromEpoch`** and **`toEpoch`** values outdated, shortening the tokens' ban period possibly entirely.

```
function _currentEpoch() internal view returns (uint128) {
        EpochConfig memory config = epochConfig;
        if (config.startTime == 0) {
            revert EpochNotConfig();
        }
        return toUint128((block.timestamp - config.startTime) /
config.duration + 1);
```

```
    }
```

Recommendation

Add a storage list for banned tokens, so that when epoch duration is changed, the banned tokens' **`fromEpoch`** and **`toEpoch`** can be adjusted accordingly.

3S-Mini-ICO-N01

Missing Validation of msg.value in transferIn Function for ETH Transfers

Id	3S-Mini-ICO-N01
Classification	None
Category	Suggestion
Status	Addressed in #6195cf2.

Description

The **`transferIn`** function in the **`UniSwapper`** contract is responsible for transferring tokens from the sender to the contract. It checks if the **`token`** is not equal to the constant **`ETH_ADDRESS`** and, if so, uses the **`safeTransferFrom`** method to transfer the specified **`amount`** of tokens from the sender to the contract. However, the function currently lacks a validation check to ensure that when the **`token`** is **`ETH_ADDRESS`**, the **`amount`** parameter matches **`msg.value`**. While this omission does not pose any security threats, adding this check would enhance the contract's robustness.

Recommendation

Modify the **`transferIn`** function to include a validation check that ensures **`amount`** is equal to **`msg.value`** when the **`token`** is **`ETH_ADDRESS`**.

3S-Mini-ICO-N02

Claim Bribes Function Reverts on Individual Errors Impairing User Experience

Id	3S-Mini-ICO-N02
Classification	None
Category	Suggestion
Status	Acknowledged with note “In our design, we ensure data accuracy off-chain, so the transaction should either fully succeed or fully fail—there shouldn’t be cases of partial success.”

Description

The **`claimBribes`** function in the **`VoteManager`** contract allows users to claim bribes for specific tokens within specified epochs. It iterates through a list of **`epochIds`** and corresponding **`tokens`**, calculating and transferring claimable bribe amounts to the user. However, the function currently reverts the entire transaction if any of the following conditions are met: the bribe was already claimed for an epoch/token combination, there were no votes for a token in a given epoch, or no bribes are available for a user/token/epoch combination.

This behavior provides a poor user experience because when a revert occurs, the user receives no specific feedback about which (epoch, token) pair caused the failure, making it difficult to diagnose and resolve the issue.

Recommendation

Modify the **`claimBribes`** function to skip the current iteration of the inner loop and emit a specific event indicating the problematic (epoch, token) combination instead of reverting the entire transaction.

3S-Mini-ICO-N03

Incorrect Consecutive Win Count in _updateBannedToken Function

Id	3S-Mini-ICO-N03
Classification	None
Category	Suggestion
Status	Acknowledged with note “If we perform the check at the start of a new epoch, we'd have to traverse through consecWinEpochs + banEpochs, which could lead to inefficiency.”

Description

The **_updateBannedToken** function in the **VoteManager** contract is responsible for implementing the token ban feature, which aims to promote diversity in token votes. When a token wins for more than **epochConfig.consecWinEpochs** epochs in a row, it is banned for **epochConfig.banEpochs** epochs, preventing users from voting for it during the ban period. The issue lies in the **_updateBannedToken** function where the **consecutiveWinCount** counter is initialized to 1 instead of 0. This assumes that the current epoch is already finished, which is incorrect as the function is called during the epoch. As a result, the consecutive win count is miscalculated, potentially leading to premature or incorrect banning of tokens.

```
function _updateBannedToken(uint128 epochId, address previousWinner, address
winner) internal {
    // Clear the previous winner's ban
    tokenBanned[previousWinner] = Banned(0, 0);
    // Check consecutive wins for the new winner
    uint32 consecWinEpochs = epochConfig.consecWinEpochs;
    uint32 banEpochs = epochConfig.banEpochs;
    if (epochId >= consecWinEpochs) {
        uint256 consecutiveWinCount = 1;
        unchecked {
            for (uint128 i = epochId - 1; i > epochId - consecWinEpochs;
i--) {
                if (epochWinner[i] == winner) ++consecutiveWinCount;
                else break;
```

```
            }
        }
        if (consecutiveWinCount == consecWinEpochs) {
            tokenBanned[winner] = Banned(epochId + 1, epochId + banEpochs);
        }
    }
}
```

Recommendation

Initialize **`consecutiveWinCount`** to 0 in the **`_updateBannedToken`** function.

3S-Mini-ICO-N04

Missing Token Listing Validation in getPrice, estimateTokenForICO, and estimateICOForToken Functions

Id	3S-Mini-ICO-N04
Classification	None
Category	Suggestion
Status	Addressed in #6195cf2.

Description

The **`getPrice`** function calculates the price of a token based on its supply and reserve, **`estimateTokenForICO`** estimates the amount of tokens obtainable for a given ICO amount, and **`estimateICOForToken`** estimates the ICO amount required for a given token amount. These functions do not validate whether the provided **`token`** has been properly created via the **`create`** function, which can lead to unexpected behavior if called with an invalid token address.

Recommendation

Add a check at the beginning of the **`getPrice`**, **`estimateTokenForICO`**, and **`estimateICOForToken`** functions to ensure that **`tokenCreator[token]`** is not equal to **`address(0)`**. Revert with **`TokenNotCreated()`** if it is.

3S-Mini-ICO-N05

Incorrect Validation of newFee in setCreationFee Function

Id	3S-Mini-ICO-N05
Classification	None
Category	Suggestion
Status	Addressed in #6195cf2.

Description

The **`setCreationFee`** function in the **`Launchpad`** contract is designed to allow the contract owner to update the **`creationFee`** for token creation on the platform. The current validation logic checks if the **`newFee`** is greater than or equal to **`MINI_ICO_TOTAL_SUPPLY`**, which is not appropriate since the **`creationFee`** is denominated in **`$ICO`** tokens, not **`MINI_ICO`** tokens. This validation does not effectively ensure that the **`newFee`** is within a reasonable range for **`$ICO`** tokens, potentially allowing for an excessively high fee to be set, which could deter token creators from using the platform.

Recommendation

Modify the validation logic in the **`setCreationFee`** function to ensure that the **`newFee`** is bounded within a reasonable range for **`$ICO`** tokens. This can be achieved by setting a maximum allowable value for **`newFee`** that aligns with the intended economic model of the platform.

3S-Mini-ICO-N06

Unnecessary Upgradeability of MiniICO

Id	3S-Mini-ICO-N06
Classification	None
Category	Suggestion
Status	Acknowledged with note “The reason we designed MiniICO this way is to ensure that when a MiniICO contract is created, its source code is automatically verified on Etherscan and other blockchain explorers.”

Description

The **`MiniICO`** contract is designed to be an upgradeable contract, intended to be interacted with through a proxy. This design allows for future upgrades and modifications without altering the contract's address or state. However, within the **`Launchpad`** contract, the **`MiniICO`** is instantiated and used as a regular non-upgradeable contract. This approach negates the benefits of the upgradeable pattern, leading to potential confusion and misalignment with the intended design. The **`Launchpad`** contract's **`create`** function deploys **`MiniICO`** and directly interact with its implementation, bypassing the proxy mechanism, leading to reduced code clarity.

Recommendation

To align with the intended design and improve code clarity, refactor the **`MiniICO`** contract to be non-upgradeable, moving the **`initialize`** logic to the **`constructor`**.