SECURITY AUDIT REPORT

Fun Token

DATE
25 Sep 2025

PREPARED BY

0xTeam.
WEB3 AUDITS

info@0xTeam.Space
0xTeamSpace
0xTeamSpace

Contents

0.0	Revision History & Version Control	3
1.0	Disclaimer	4
2.0	Executive Summary	5
3.0	Checked Vulnerabilities	7
4.0	Techniques , Methods & Tools Used	8
5.0	Technical Analysis	9
6.0	Auditing Approach and Methodologies Applied	28
7.0	Limitations on Disclosure and Use of this Report	29

Revision History & Version Control

Version	Date	Author’s	Description
1.0	23 Sep 2025	Gurkirat, Manoj & Aditya	Initial Audit Report
2.0	24 Sep 2025	Gurkirat, Manoj & Aditya	Re-Audit Report
3.0	25 Sep 2025	Gurkirat, Manoj & Aditya	Final Audit Report

0xTeam conducted a comprehensive Security Audit on the Fun Token to ensure the overall code quality, security, and correctness. The review focused on ensuring that the code functions as intended, identifying potential vulnerabilities, and safeguarding the integrity of Fun Token’s operations against possible attacks.

Report Structure

The report is divided into two primary sections:

1. **Executive Summary** : Provides a high-level overview of the audit findings.
2. **Technical Analysis** : Offers a detailed examination of the Smart contracts code.

Note :
The analysis is static and manual, exclusively focused on the smart contract code. The information provided in this report should be used to assess the security, quality, and expected behavior of the code.

1.0 Disclaimer

This is a summary of our audit findings based on our analysis, following industry best practices as of the date of this report.However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. The audit focuses on Smart contracts coding practices and any issues found in the code, as detailed in this report. For a complete understanding of our analysis, you should read the full report. We have made every effort to conduct a thorough analysis, but it's important to note that you should not rely solely on this report and cannot make claims against us based on its contents. We strongly advise you to perform your own independent checks before making any decisions. Please read the disclaimer below for more information.

DISCLAIMER: By reading this report, you agree to the terms outlined in this disclaimer. If you do not agree, please stop reading immediately and delete any copies you have. This report is for informational purposes only and does not constitute investment advice. You should not rely on the report or its content, and 0xTeam and its affiliates (including all associated companies, employees, and representatives) are not responsible for any reliance on this report.The report is provided "as is" without any guarantees. 0xTeam excludes all warranties, conditions, or terms, including those implied by law, regarding quality, fitness for a purpose, and use of reasonable care. Except where prohibited by law, 0xTeam is not liable for any type of loss or damage, including direct, indirect, special, or consequential damages, arising from the use or inability to use this report. The findings are solely based on the Smart contracts code provided to us.

2.0 Executive Summary

2.1 Overview

0xTeam has meticulously audited the Fun Token Smart contracts project. The primary objective of this audit was to assess the security, functionality, and reliability of the Fun Token's before their deployment on the blockchain. The audit focused on identifying potential vulnerabilities, evaluating the contract's adherence to best practices, and providing recommendations to mitigate any identified risks. The comprehensive analysis conducted during this period ensures that the Fun Token is robust and secure, offering a reliable environment for its users.

2.2 Scope

The scope of this audit involved a thorough analysis of the Fun Token Smart contracts, focusing on evaluating its quality, rigorously assessing its security, and carefully verifying the correctness of the code to ensure it functions as intended without any vulnerabilities.

Files in Examination:

Language	Solidity
In-Scope	• contracts\FUNGiveway.sol • contracts\GeomeanOracle.sol • contracts\USDTtoFUNOracle.sol
Github	https://github.com/FunTokenHubs/5m-staking-contracts
Fixed Commit Hash	ee37470a587b3124f4b8bee70dbac33874ab3d09

OUT-OF-SCOPE: External Smart contracts code, other imported code.

2.3 Audit Summary

Name	Verified	Audited	Vulnerabilities
Fun Token	Yes	Yes	Refer Section 5.0

2.4 Vulnerability Summary

High	Medium	Low	Informational
2	1	9	3

High Medium Low Informational

2.5 Recommendation Summary

Issues \ Severity	🔴 High	🟡 Medium	🟢 Low	🔵 Informational	Total (Σ)
Open					
Resolved	2	1	7	1	**11**
Acknowledged			1	2	**3**
Partially Resolved			1		**1**
Total (Σ)	2	1	9	3	**15**

- **Open**: Unresolved security vulnerabilities requiring resolution.
- **Resolved**: Previously identified vulnerabilities that have been fixed.
- **Acknowledged**: Identified vulnerabilities noted but not yet resolved.
- **Partially Resolved**: Risks mitigated but not fully resolved.

2.6 Summary of Findings

ID	Title	Severity	Fixed
H-01	Incorrect Deadline Initialization Contract Uses Expired Timestamp (Jan 1, 2025 Instead of Dec 31, 2025)	High	✔
H-02	Lack of Minimum Bound on maxInterestDuration Enables Yield Manipulation	High	✔
M-01	state.cardinality not updated correctly in _updatePool, causing observe/_floorObservation to miss valid history	Medium	✔
L-01	Unused State Variables: maxInterestDuration, contractDeployTime, totalWeight	Low	✔
L-02	Misleading Revert Message in lockTokens function	Low	✔
L-03	Misleading Revert Message in withdrawWithInterest function	Low	✔
L-04	Misleading Revert Message in _beforeInitialize() function	Low	✔
L-05	Redundant code in withdrawWithInterest()	Low	✔
L-06	Use Named Return Variable to Save Gas	Low	📝
L-07	Redundant Checks Between manuallyTriggerMilestone and _triggerPriceMilestone	Low	🟡
L-08	Redundant Getter Functions for Public State Variables	Low	✔
L-09	State variables that could be declared immutable	Low	✔
I-01	nonReentrant Modifier is Not the First Modifier	Info	✔
I-02	Multiple Roles Granted to Same Admin in Constructor	Info	📝
I-03	Redundant Check in updateMaxInterestDuration	Info	📝

✔ - Fixed 🟡 - Partially Fixed ✘ - Not Fixed 📝 - Acknowledged

3.0 Checked Vulnerabilities

We examined Smart contracts for widely recognized and specific vulnerabilities. Below are some of the common vulnerabilities considered.

Category	Check Items
Source Code Review	➔ Reentrancy Vulnerabilities ➔ Ownership Control ➔ Time-Based Dependencies ➔ Gas Usage in Loops ➔ Transaction Sequence Dependencies ➔ Style Guide Compliance ➔ EIP Standard Compliance ➔ External Call Verification ➔ Mathematical Checks ➔ Type Safety ➔ Visibility Settings ➔ Deployment Accuracy ➔ Repository Consistency
Functional Testing	➔ Business Logic Validation ➔ Feature Verification ➔ Access Control and Authorization ➔ Escrow Security ➔ Token Supply Management ➔ Asset Protection ➔ User Balance Integrity ➔ Data Reliability ➔ Emergency Shutdown Mechanism

4.0 Techniques , Methods & Tools Used

The following techniques, methods, and tools were used to review all the smart contracts

- **Structural Analysis:**
  This involves examining the overall design and architecture of the smart contract. We ensure that the contract is logically organised, scalable, and follows industry best practices. This step is crucial for identifying potential structural issues that could lead to vulnerabilities or maintenance challenges in the future.

- **Static Analysis:**
  Static analysis is conducted using automated tools to scan the contract's codebase for common vulnerabilities and security risks without executing the code. This process helps identify issues such as reentrancy, arithmetic errors, and potential denial-of-service (DOS) vulnerabilities early on, allowing for quick remediation.

- **Code Review / Manual Analysis:**
  A manual, in-depth review of the smart contract's code is performed to verify the logic and ensure it matches the intended functionality as described in the project's documentation. During this phase, we also confirm the findings from the static analysis and check for any additional issues that may not have been detected by automated tools.

- **Dynamic Analysis:**
  Dynamic analysis involves executing the smart contract in various controlled environments to observe its behaviour under different conditions. This step includes running comprehensive test cases, performing unit tests, and monitoring gas consumption to ensure the contract operates efficiently and securely in real-world scenarios.

- **Tools and Platforms Used for Audit:**
  Utilising tools such as Remix , Slither, Aderyn, Solhint for static analysis, and platforms like Hardhat and Foundry for dynamic testing and simulation.

Note: The following values for "Severity" mean:

- High: Direct and severe impact on the funds or the main functionality of the protocol.
- Medium: Indirect impact on the funds or the protocol's functionality.
- Low: Minimal impact on the funds or the protocol's main functionality.
- Informational: Suggestions related to good coding practices and gas efficiency.

5.0 Technical Analysis

High

[H-01] Incorrect Deadline Initialization Contract Uses Expired Timestamp (Jan 1, 2025 Instead of Dec 31, 2025)

Severity
High

Location	Functions
Contracts\FUNGiveaway.sol	→ `_initializePriceMilestones`

Issue Description
The contract is designed to operate with a program deadline of December 31, 2025, as referenced throughout the code and documentation. However, the hardcoded deadline timestamp (1735689600) corresponds to January 1, 2025, 00:00:00 UTC.

This means the system immediately considers the deadline as passed upon deployment, breaking core functionality.

Impact / Proof of Concept

- All functions that depend on **priceMilestones[i].deadline** (e.g., lockTokens, withdrawPrincipal, withdrawWithInterest) will behave incorrectly.
- This essentially **invalidates the program logic and lifecycle**, making the contract unusable in practice.

```
function _initializePriceMilestones() private {
    uint256 deadline = 1735689600; // 31 DEC 2025 00:00:00 UTC
```

Recommendation
Correct the timestamp to match the intended deadline of **Dec 31, 2025, 23:59:59 UTC → 1767225599.**

Status - Recommendation accepted and fixed

[H-02] Lack of Minimum Bound on maxInterestDuration Enables Yield Manipulation

Severity
High

Location	Functions
Contracts\FUNGiveaway.sol	➔ updateMaxInterestDuration

Issue Description

The original issue was addressed by moving the if check before the interest calculation.
However, the new implementation introduces a different problem, the **admin now has unrestricted control over the interest duration** via the updateMaxInterestDuration(uint256 _newMaxDuration) function.

If the deadline has already passed, the admin (or a compromised admin wallet) could call this function and set maxInterestDuration to a very low value (even below the intended deadline or even 1). This reduces the effective interest calculation in withdrawWithInterest() in below line

```
uint256 interest = (locks[i].amount * ANNUAL_INTEREST_RATE *
lockDuration) /(100 * SECONDS_PER_YEAR);
```

Impact / Proof of Concept
This could cause losses for users who locked their tokens under the assumption of guaranteed returns.

This introduces both financial risk for users and centralization risk, as the interest calculation becomes entirely dependent on the admin's actions.

```
function updateMaxInterestDuration(uint256 _newMaxDuration) external
onlyRole(DEFAULT_ADMIN_ROLE) {
       if (_newMaxDuration == 0) revert InvalidAmount();

       maxInterestDuration = _newMaxDuration;

```

```
        emit MaxInterestDurationUpdated(_newMaxDuration);
    }
```

1. Users may receive significantly less interest than expected.
2. Admin or compromised admin wallet can manipulate interest accrual arbitrarily.
3. Breaks the assumption of minimum guaranteed yield after tokens are locked.

Recommendation

Introduce a safeguard by enforcing a minimum duration in updateMaxInterestDuration function.

For example, revert if _newMaxDuration is below a predefined threshold (such as the protocol's intended minimum deadline). This ensures that users always receive at least the minimum interest they were promised, even if the admin wallet gets compromised.

```
function updateMaxInterestDuration(uint256 _newMaxDuration) external
onlyRole(DEFAULT_ADMIN_ROLE) {
        // Enforce a minimum duration to protect users
        if (_newMaxDuration < 1767225599) revert
DurationLessThanDeadline();

        maxInterestDuration = _newMaxDuration;
        emit MaxInterestDurationUpdated(_newMaxDuration);
    }
```

Status - Recommendation accepted and fixed

Medium

[M-01] state.cardinality not updated correctly in _updatePool, causing observe/_floorObservation to miss valid history

Severity
MEDIUM

Location	Functions/Variables
Contracts\GeomeanOracle.sol	→ _updatePool / _floorObservation

Issue Description

The _updatePool function defers updating state.cardinality unless the updated index wraps around to 0

```
if (state.cardinalityNext > state.cardinality && indexUpdated ==
0) {
        state.cardinality = state.cardinalityNext;
    }
```

_updatePool computes capacity = state.cardinalityNext and computes indexUpdated = (state.index + 1) % capacity. It then writes to observations[poolId][indexUpdated] and sets state.index = indexUpdated.

Hence state.cardinality is left unchanged until indexUpdated == 0. Thus **state.index may become ≥ state.cardinality**, and observe() / _floorObservation() uses card = state.cardinality but would only be able to use the latest observation newest = observations[poolId][state.index]

Eg:-

Start state:

- state.cardinality = 1
- state.cardinalityNext = 1

- state.index = 0
- observations[poolID][0].blockTimestamp = 100 (initial observation)

User calls increaseCardinalityNext(4):

- state.cardinality = 1
- state.cardinalityNext = 4

Now a swap occurs at time 200 which triggers _updatePool:

1. The function sets capacity = state.cardinalityNext = 4.
2. It computes indexUpdated = (state.index + 1) % capacity = (0 + 1) % 4 = 1.
3. Writes observations[1].timestamp = 200 (initialized = true)
4. Sets state.index = 1
5. state.cardinality remains 1 (because indexUpdated != 0) so state.cardinality is not updated.
6. Now call observe is called which internally calls _floorObservation(targetTime = 150):
    a. card = state.cardinality = 1
    b. newest = observations[state.index] = observations[1] (timestamp 200)
    c. targetTime < newest.timestamp → for loop to card = 1 steps:
    d. Check observations[1]: timestamp 200 > 150 → not good
    e. Loop ends (only 1 step allowed) → revert **InsufficientObservationHistory()**
7. But observations[0] (t = 100) exists and would satisfy the query

Impact / Proof of Concept

- A caller trying to compute a TWAP in the time window that should be available will get **InsufficientObservationHistory()** reverts even though the data physically exists.

Recommendation

```
if (state.cardinalityNext > state.cardinality) {
    state.cardinality = state.cardinalityNext;
}
```

Status - Recommendation accepted and fixed

Low

[L-01] Unused State Variables: maxInterestDuration, contractDeployTime, totalWeight

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	➔ `uint256 public maxInterestDuration;, function updateMaxInterestDuration, contractDeployTime, totalWeight`

Issue Description
The state variables maxInterestDuration, contractDeployTime, and totalWeight are declared in the contract but are never actually used in any computation or logic. Additionally, the function updateMaxInterestDuration is defined but not utilized anywhere. Unused state variables and functions increase contract size unnecessarily which leads to higher deployment gas costs.

Impact / Proof of Concept
Although these unused variables do not affect the current functionality or security of the contract, they contribute to higher gas costs during deployment. Removing these variables and unused functions will optimize the contract and reduce deployment costs.

Recommendation
- Remove the unused state variables: maxInterestDuration, contractDeployTime, totalWeight.
- Remove the unused function updateMaxInterestDuration.

Status - Recommendation accepted and fixed

[L-02] Misleading Revert Message in lockTokens function

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	→ `lockTokens`

Issue Description
In the lockTokens function, the contract reverts with DeadlineNotReached if the first milestone's deadline has already passed:

```
if (block.timestamp >= priceMilestones[0].deadline) revert
DeadlineNotReached();
```

This is misleading, because the user is actually too late to lock tokens, not that the deadline hasn't been reached yet.

Impact / Proof of Concept
It doesn't have a direct impact on the protocol.

Recommendation
- Update the revert message to accurately reflect the condition,

```
error DeadlineExceeded();

if (block.timestamp >= priceMilestones[0].deadline) revert
DeadlineExceeded();
```

Status - Recommendation accepted and fixed

[L-03] Misleading Revert Message in withdrawWithInterest function

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	➔ `withdrawWithInterest`

Issue Description
In the withdrawWithInterest() function, the following code is used:

```
if (distributionActive) {
    revert DistributionNotActive();
}
```

DistributionActive being true indicates that milestone distributions are active, but the error message DistributionNotActive is misleading.

Impact / Proof of Concept
It doesn't have a direct impact on the protocol.

Recommendation
- Update the revert message to accurately reflect the condition,

```
error DistributionActive();

if (!distributionActive) {
    revert DistributionActive();
}
```

Status - Recommendation accepted and fixed

[L-04] Misleading Revert Message in _beforeInitialize() function

Severity
LOW

Location	Functions/Variables
Contracts\GeomeanOracle.sol	➔ `_beforeInitialize()`

Issue Description
In the _beforeInitialize() function, the following code is used:

```
function _beforeInitialize(address, PoolKey calldata key, uint160)
internal view override returns (bytes4) {
        if (key.fee != 0 || key.tickSpacing !=
TickMath.MAX_TICK_SPACING) {
            revert OnlyOneOraclePoolAllowed();
        }
        return BaseHook.beforeInitialize.selector;
 }
```

OnlyOneOraclePoolAllowed() is a confusing name for a check that simply requires fee == 0 && tickSpacing == TickMath.MAX_TICK_SPACING. Rename to InvalidOraclePoolConfig() or similar.

Impact / Proof of Concept
- It doesn't have a direct impact on the protocol.

Recommendation
- Update the revert message to accurately reflect the condition,

```
error InvalidOraclePoolConfig();
function _beforeInitialize(address, PoolKey calldata key, uint160)
internal view override returns (bytes4) {
        if (key.fee != 0 || key.tickSpacing !=
TickMath.MAX_TICK_SPACING) {
            revert InvalidOraclePoolConfig();
        }
        return BaseHook.beforeInitialize.selector;}
```

Status - Recommendation accepted and fixed

[L-05] Redundant code in withdrawWithInterest()

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	→ withdrawWithInterest

Issue Description
In the withdrawWithInterest() function, the following code appears inside the for loop:

```
if (lockDuration > maxInterestDuration) {
    lockDuration = maxInterestDuration;
}
```

- This variable (lockDuration) is updated after the interest calculation, which means the cap on lockDurationhas no effect on the computed interest.
- maxInterestDuration is also therefore redundant in the current implementation.

Impact / Proof of Concept
- No functional or security impact, the interest is still calculated using the original lockDuration.

- However, the presence of unused logic costs gas.

Recommendation
- Remove the "if" block entirely, cause there is no use of it in the computation.

```
function withdrawWithInterest() external whenNotPaused nonReentrant {
        if (hasWithdrawn[msg.sender]) revert AlreadyWithdrawn();
        if (block.timestamp < priceMilestones[0].deadline) revert
DeadlineNotReached();

        if (distributionActive) {
            revert DistributionNotActive(); // Reuse error: milestones
active, can't withdraw with interest
```

```
        }

        uint256 totalAmount = 0;
        uint256 totalInterest = 0;

        UserLock[] storage locks = userLocks[msg.sender];

        for (uint256 i = 0; i < locks.length; i++) {
            if (!locks[i].withdrawn) {
                uint256 lockDuration = block.timestamp - locks[i].lockTime;
                uint256 interest = (locks[i].amount * ANNUAL_INTEREST_RATE *
lockDuration) /
                                (100 * SECONDS_PER_YEAR);

                totalAmount =  totalAmount + locks[i].amount;
                totalInterest = totalInterest + interest;
                locks[i].withdrawn = true;
            }
        }

        hasWithdrawn[msg.sender] = true;

        if (totalAmount != 0) {
            totalLockedTokens = totalLockedTokens  - totalAmount;
            uint256 totalPayout = totalAmount + totalInterest;

            // Transfer principal + interest from treasury
            funToken.safeTransferFrom(treasuryWallet, msg.sender,
totalPayout);

            emit TokensWithdrawn(msg.sender, totalPayout, totalInterest);
        }
    }
```

Status - Recommendation accepted and fixed

[L-06] Use Named Return Variable for to Save Gas

Severity
Low

Location	Functions/Variables
Contracts\FunGiveaway.sol	→ `All functions with unnamed return variable`

Issue Description

Several functions in the contract use explicit return statements with locally defined variables rather than leveraging Solidity's named return variables. Using named return variables can reduce gas usage by avoiding extra stack operations and eliminating the explicit return statement.

`Example:`
In getUnclaimedRewards, the function defines a local variable and explicitly returns it:

```
uint256 totalRewards = 0;
// ... computation ...
return totalRewards;
```

Impact / Proof of Concept:

- No functional impact, the function works correctly.
- Gas savings are minimal, but cumulative savings can matter in large-scale usage.

Recommendation:

- Refactor functions to use named return variables instead of explicit return statements where applicable.
- Convert to a **named return variable** pattern like below:

```
function getUnclaimedRewards(address _user) external view returns
(uint256 totalRewards) {
    UserLock[] storage locks = userLocks[_user];
    if (locks.length == 0) return 0;

    for (uint256 i = 0; i < currentPriceLevel; i++) {
        if (priceMilestones[i].triggered && !hasClaimed[_user][i]) {
            uint256 userEligibleWeight = 0;
            for (uint256 j = 0; j < locks.length; j++) {
                if ((locks[j].eligibleMilestones & (1 << i)) != 0) {
                    userEligibleWeight = userEligibleWeight +
locks[j].weight;
                }
            }
            if (userEligibleWeight != 0 && milestoneEligibleWeight[i]
!= 0) {
                totalRewards = totalRewards + (userEligibleWeight *
priceMilestones[i].rewardAmount) / milestoneEligibleWeight[i];
            }
        }
    }
}
```

Status - Recommendation accepted but Acknowledged

[L-07] Redundant Checks Between manuallyTriggerMilestone and _triggerPriceMilestone

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	→ `manuallyTriggerMilestone` → `_triggerPriceMilestone`

Issue Description
Both manuallyTriggerMilestone and _triggerPriceMilestone functions perform the same validations:

1. manuallyTriggerMilestone already ensures that _level < priceMilestones.length.
2. _triggerPriceMilestone again checks for _level >= priceMilestones.length and priceMilestones[_level].triggered.

Impact / Proof of Concept
- No functional impact; the contract works correctly.
- Gas inefficiency unnecessary logic in _triggerPriceMilestone.

Recommendation
- Consolidate validation to only one place: Remove the checks in _triggerPriceMilestone.

```
function _triggerPriceMilestone(uint256 _level) private {

        priceMilestones[_level].triggered = true;
        priceMilestones[_level].triggeredAt = block.timestamp;
       // Activate distribution for this level and all previous levels
        distributionActive = true;
        currentPriceLevel = _level + 1;
        emit PriceMilestoneTriggered(_level,
priceMilestones[_level].priceThreshold, block.timestamp); }
```

Status - Recommendation accepted but Partially Fixed

[L-08] Redundant Getter Functions for Public State Variables

Severity
LOW

Location	Functions/Variables
Contracts\FunGiveaway.sol	➔ `getter functions`

Issue Description
The contract defines multiple getter functions for state variables that are already declared as public, which automatically generates a getter in Solidity. Making the following functions redundant:

- `getMilestoneEligibleWeight(uint256 _milestoneIndex)`
- `getContractBalance()`
- `getCurrentPrice()`
- `getUserLock(address _user, uint256 _index)`

Impact / Proof of Concept
- No functional impact; the contract works correctly.
- Removing redundant functions can reduce deployment gas, because every function increases the contract bytecode size.

Recommendation
- Remove the above redundant functions and protocol could use the automatically generated getters for public state variables.

Status - Recommendation accepted and fixed

[L-9] State variables that could be declared immutable

Severity
LOW

Location	Functions/Variables
Contracts\GeomeanOracle.sol	➔ `_beforeInitialize()`

Issue Description
contractDeployTime and _decimals are only assigned in the constructor. Therefore it can be made immutable as immutable values are cheaper to read. State variables that are not updated following deployment should be declared immutable to save gas.

Impact / Proof of Concept
While this issue doesn't directly impact the functionality of the contract, the contract can benefit from the use of constant and immutable keywords for variables that do not change after deployment. This could save gas costs by storing the variables directly in the bytecode.

Recommendation
- Add the immutable attribute to state variables that never change or are set only in the constructor.

```
uint256 public immutable contractDeployTime;
uint8 private immutable  _decimals;
```

Status - Recommendation accepted and fixed

Info

[I-01] nonReentrant Modifier is Not the First Modifier

Severity
INFO

Location	Functions/Variables
Contracts\FUNGiveaway.sol	➔ Line 137: lockTokens ➔ Line 182: withdrawPrincipal ➔ Line 233: claimRewards ➔ Line 279: withdrawWithInterest

Issue Description
The functions use multiple modifiers, but the nonReentrant modifier is not placed first. Best practices recommend placing nonReentrant as the first modifier to ensure it executes before other modifiers.

Impact / Proof of Concept
This issue does not directly impact the protocol's functionality. However, following this defensive coding practice reduces the attack surface if future modifiers are updated to include external calls.

Recommendation

- Reorder modifiers so that nonReentrant is always placed first.

Eg.,

```
function lockTokens(uint256 _amount) external nonReentrant
whenNotPaused {
    ...
}
```

Status - Recommendation accepted and fixed

[I-02] Multiple Roles Granted to Same Admin in Constructor

Severity
INFO

Location	Functions
Contracts\FUNGiveaway.sol	→ `constructor`

Issue Description
Protocol grants the same _admin address for all three roles: DEFAULT_ADMIN_ROLE, OPERATOR_ROLE, and PRICE_UPDATER_ROLE. While this is functionally valid, it may be unnecessary if the protocol intends for role separation.

Impact
Currently, this setup does not break functionality, but it reduces clarity of the role-based access control design.

Recommendation
- If separation is intended, assign roles to distinct addresses.
- If separation is not intended, protocol could consolidate roles for clarity.

Status - Recommendation accepted but Acknowledged

[I-03] Redundant Check in updateMaxInterestDuration

Severity
INFO

Location	Functions
Contracts\FUNGiveaway.sol	➔ updateMaxInterestDuration

Issue Description
The current implementation contains a redundant check:

if (_newMaxDuration == 0) revert InvalidAmount();
This condition is unnecessary because _newMaxDuration == 0 will always be caught by the subsequent check:

if (_newMaxDuration < 1767225599) revert DurationLessThanDeadline();
Since 0 < 1767225599 evaluates to true, the InvalidAmount check will never be reached.

Impact
- No functional issues introduced.
- Slightly increases contract bytecode size and gas costs for deployment.

Recommendation
Remove the redundant zero check to streamline the code:

```
function updateMaxInterestDuration(uint256 _newMaxDuration) external
onlyRole(DEFAULT_ADMIN_ROLE) {
    if (_newMaxDuration < 1767225599) revert
DurationLessThanDeadline();
    maxInterestDuration = _newMaxDuration;
    emit MaxInterestDurationUpdated(_newMaxDuration);
}
```

Status - Recommendation accepted but Acknowledged

6.0 Auditing Approach and Methodologies Applied

The Solidity smart contract was audited using a comprehensive approach to ensure the highest level of security and reliability. Careful attention was given to the following key areas to ensure the overall quality of the code:

- **Code quality and structure**: We conducted a detailed review of the codebase to identify any potential issues related to code structure, readability, and maintainability. This included analysing the overall architecture of the Solidity smart contract and reviewing the code to ensure it follows best practices and coding standards.
- **Security vulnerabilities**: Our team used manual techniques to identify any potential security vulnerabilities that could be exploited by attackers. This involved a thorough analysis of the code to identify any potential weaknesses, such as buffer overflows, injection vulnerabilities, signatures, and deprecated functions.
- **Documentation and comments**: Our team reviewed the code documentation and comments to ensure they accurately describe the code's intended behaviour and logic. This helps developers to better understand the codebase and make modifications without introducing new issues.
- **Compliance with best practices**: We checked that the code follows best practices and coding standards that are recommended by the Solidity community and industry experts. This ensures that the Solidity smart contract is secure, reliable, and efficient.

Our audit team followed OWASP and Ethereum (Solidity) community security guidelines for this audit. As a result, we were able to identify potential issues and provide recommendations to improve the smart contract's security and performance.

Throughout the audit of the smart contracts, our team placed great emphasis on ensuring the overall quality of the code and the use of industry best practices. We meticulously reviewed the codebase to ensure that it was thoroughly documented and that all comments and logic aligned with the intended behaviour. Our approach to the audit was comprehensive, methodical, and aimed at ensuring that the smart contract was secure, reliable, and optimised for performance.

6.1 Code Review / Manual Analysis

Our team conducted a manual analysis of the Solidity smart contracts to identify new vulnerabilities or to verify vulnerabilities found during static and manual analysis. We carefully analysed every line of code and made sure that all instructions provided during the onboarding phase were followed. Through our manual analysis, we were able to identify potential vulnerabilities that may have been missed by automated tools and ensure that the smart contract was secure and reliable.

6.2 Tools Used for Audit

In the course of our audit, we leveraged a suite of tools to bolster the security and performance of our program. While our team drew on their expertise and industry best practices, we also integrated various tools into our development environment. Noteworthy among them are Remix,Slither, Aderyn, Solhint for Static Analysis and Hardhat & Foundry for Dynamic Analysis. This holistic approach ensures a thorough analysis, uncovering potential issues that automated tools alone might overlook. 0xTeam takes pride in utilising these tools, which significantly contribute to the quality, security, and maintainability of our codebase

7.0 Limitations on Disclosure and Use of this Report

This report contains information concerning potential details of the Fun Token Project and methods for exploiting them. 0xTeam recommends that special precautions be taken to protect the confidentiality of both this document and the information contained herein. Security Assessment is an uncertain process, based on past experiences, currently available information, and known threats. All information security systems, which by their nature are dependent on human beings, are vulnerable to some degree. Therefore, while 0xTeam considers the major security vulnerabilities of the analysed systems to have been identified, there can be no assurance that any exercise of this nature will identify all possible vulnerabilities or propose exhaustive and operationally viable recommendations to mitigate those exposures. In addition, the analysis set forth herein is based on the technologies and known threats as of the date of this report. As technologies and risks change over time, the vulnerabilities associated with the operation of the Fun Token Smart contracts Code Base described in this report, as well as the actions necessary to reduce the exposure to such vulnerabilities, will also change. 0xTeam makes no undertaking to supplement or update this report based on changed circumstances or facts of which 0xTeam becomes aware after the date hereof, absent a specific written agreement to perform the supplemental or updated analysis. This report may recommend that 0xTeam use certain software or hardware products manufactured or maintained by other vendors. 0xTeam bases these recommendations upon its prior experience with the capabilities of those products. Nonetheless, 0xTeam does not and cannot warrant that a particular product will work as advertised by the vendor, nor that it will operate in the manner intended. This report was prepared by 0xTeam for the exclusive benefit of Fun Token and is proprietary information. The Non-Disclosure Agreement (NDA) in effect between 0xTeam and Fun Token governs the disclosure of this report to all other parties including product vendors and suppliers.