Audited by

CredShields

Smart Contract Audit

August 21, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of BitSafe SRL between July 23rd, 2025, and July 30th, 2025. A retest was performed on August 21st, 2025.

Author
Shashank (Co-founder, CredShields) shashank@CredShields.com

Reviewers
Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor), Yash Shah (Auditor), Prasad Kuri (Auditor)

Prepared for
BitSafe SRL

Table of Contents

1. Executive Summary --------------------

BitSafe SRL engaged CredShields to perform a smart contract audit from July 23rd, 2025, to July 30th, 2025. During this timeframe, 29 vulnerabilities were identified. **A retest was performed on August 21st, 2025, and all the bugs have been addressed.**

During the audit, 18 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "BitSafe SRL" and should be prioritized for remediation.

The table below shows the in-scope assets and a breakdown of findings by severity per asset. Section 2.3 contains more information on how severity is calculated.

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
FUN Token Giveaway Contracts	11	7	4	5	0	2	**29**
	11	**7**	**4**	**5**	**0**	**2**	**29**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the FUN Token Giveaway Contract's scope during the testing window while abiding by the policies set forth by BitSafe SRL's team.

State of Security

To maintain a robust security posture, it is essential to continuously review and improve upon current security processes. Utilizing CredShields' continuous audit feature allows both BitSafe SRL's internal security and development teams to not only identify specific vulnerabilities but also gain a deeper understanding of the current security threat landscape.

To ensure that vulnerabilities are not introduced when new features are added or code is refactored, we recommend conducting regular security assessments. Additionally, by analyzing the root cause of resolved vulnerabilities, the internal teams at BitSafe SRL can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, BitSafe SRL can future-proof its security posture and protect its assets.

2. The Methodology --------------------

BitSafe SRL engaged CredShields to perform the FUN Token Giveaway Contract audit. The following sections cover how the engagement was put together and executed.

2.1 Preparation Phase

The CredShields team meticulously reviewed all provided documents and comments in the smart contract code to gain a thorough understanding of the contract's features and functionalities. They meticulously examined all functions and created a mind map to systematically identify potential security vulnerabilities, prioritizing those that were more critical and business-sensitive for the refactored code. To confirm their findings, the team deployed a self-hosted version of the smart contract and performed verifications and validations during the audit phase.

A testing window from July 23rd, 2025, to July 30th, 2025, was agreed upon during the preparation phase.

2.1.1 Scope

During the preparation phase, the following scope for the engagement was agreed upon:

IN SCOPE ASSETS
https://github.com/FunTokenHubs/5m-staking-contracts/tree/766a67344487baafdc384352f6e72f1eb5992ca3

2.1.2 Documentation

Documentation was not required as the code was self-sufficient for understanding the project.

2.1.3 Audit Goals

CredShields employs a combination of in-house tools and thorough manual review processes to deliver comprehensive smart contract security audits. The majority of the audit involves manual inspection of the contract's source code, guided by OWASP's Smart Contract Security Weakness Enumeration (SCWE) framework and an extended, self-developed checklist built from industry best practices. The team focuses on deeply understanding the contract's core logic, designing targeted test cases, and assessing business logic for potential vulnerabilities across OWASP's identified weakness classes.

CredShields aligns its auditing methodology with the OWASP Smart Contract Security projects, including the Smart Contract Security Verification Standard (SCSVS), the Smart Contract Weakness Enumeration (SCWE), and the Smart Contract Secure Testing Guide (SCSTG). These frameworks, actively contributed to and co-developed by the CredShields team, aim to bring consistency, clarity, and depth to smart contract security assessments. By adhering to these OWASP standards, we ensure that each audit is performed against a transparent, community-driven, and technically robust baseline. This approach enables us to deliver structured, high-quality audits that address both common and complex smart contract vulnerabilities systematically.

2.2 Retesting Phase

BitSafe SRL is actively partnering with CredShields to validate the remediations implemented towards the discovered vulnerabilities.

2.3 Vulnerability classification and severity

CredShields follows OWASP's Risk Rating Methodology to determine the risk associated with discovered vulnerabilities. This approach considers two factors - Likelihood and Impact - which are evaluated with three possible values - **Low**, **Medium**, and **High**, based on factors such as Threat agents, Vulnerability factors, and Technical and Business Impacts. The overall severity of the risk is calculated by combining the likelihood and impact estimates.

Overall Risk Severity				
Impact	HIGH	● Medium	● High	● Critical
	MEDIUM	● Low	● Medium	● High
	LOW	● None	● Low	● Medium
		LOW	MEDIUM	HIGH
Likelihood				

Overall, the categories can be defined as described below -

1. Informational

We prioritize technical excellence and pay attention to detail in our coding practices. Our guidelines, standards, and best practices help ensure software stability and reliability. Informational vulnerabilities are opportunities for improvement and do not pose a direct risk to the contract. Code maintainers should use their own judgment on whether to address them.

2. Low

Low-risk vulnerabilities are those that either have a small impact or can't be exploited repeatedly or those the client considers insignificant based on their specific business circumstances.

3. Medium

Medium-severity vulnerabilities are those caused by weak or flawed logic in the code and can lead to exfiltration or modification of private user information. These vulnerabilities can harm the client's reputation under certain conditions and should be fixed within a specified timeframe.

4. **High**

   High-severity vulnerabilities pose a significant risk to the Smart Contract and the organization. They can result in the loss of funds for some users, may or may not require specific conditions, and are more complex to exploit. These vulnerabilities can harm the client's reputation and should be fixed immediately.

5. **Critical**

   Critical issues are directly exploitable bugs or security vulnerabilities that do not require specific conditions. They often result in the loss of funds and Ether from Smart Contracts or users and put sensitive user information at risk of compromise or modification. The client's reputation and financial stability will be severely impacted if these issues are not addressed immediately.

6. **Gas**

   To address the risk and volatility of smart contracts and the use of gas as a method of payment, CredShields has introduced a "Gas" severity category. This category deals with optimizing code and refactoring to conserve gas.

2.4 CredShields staff

The following individual at CredShields managed this engagement and produced this report:

- Shashank, Co-founder CredShields shashank@CredShields.com

Please feel free to contact this individual with any questions or concerns you have about the engagement or this document.

3. Findings Summary -------------------

This chapter contains the results of the security assessment. Findings are sorted by their severity and grouped by asset and OWASP SCWE classification. Each asset section includes a summary highlighting the key risks and observations. The table in the executive summary presents the total number of identified security vulnerabilities per asset, categorized by risk severity based on the OWASP Smart Contract Security Weakness Enumeration framework.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 29 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SCWE | Vulnerability Type
Incorrect price calculation for Identical Oracle Cumulative Values	Critical	Business Logic Issue (SCWE-001)
Incorrect Observation Indexing in _updatePool() Causes Oracle Data Overwrite	Critical	Business Logic Issue (SCWE-001)
Oracle Users Can Receive Inflated Token Amounts Due to Incorrect Decimal Adjustment	Critical	Business Logic Issue (SCWE-001)
Oracle Users Can Receive Zero USDT Values Due to Incorrect Decimal Division	Critical	Business Logic Issue (SCWE-001)
Improper observation indexing can corrupt time-series data	Critical	Business Logic Issue (SCWE-001)
Misconfigured Milestone Triggering Logic Can Delay or Deny Legitimate Reward Distribution to Participants	Critical	Business Logic Issue (SCWE-001)

Oracle Observation Contains Hardcoded Zero Values Leading to Invalid TWAP	Critical	Business Logic Issue (SCWE-001)
Misconfigured Oracle Hook Prevents Observation Recording	Critical	Dead Code (SCWE-062)
Users Can Lose Access To Locked Funds If Price Threshold Is Reached	Critical	Business Logic Issue (SCWE-001)
Malicious Claimer can Claim Both Rewards and Interest After Milestone Trigger	Critical	Business Logic Issue (SCWE-001)
User Can Repeatedly Claim Rewards From Already-Triggered Milestones	Critical	Business Logic Issue (SCWE-001)
Malicious Actor Can Reduce Observation Buffer Size	High	Access Control
Invalid External Call to Missing observe() Function in geomeanOracle	High	Broken Functionality
Unauthorized Pool Manipulation via Missing Access Control in setPool()	High	Access Control
Missing permission declaration can revert PoolManager execution for integrated hook	High	Business Logic Issue (SCWE-001)
Invalid Oracle Implementation Can Permanently Disable Price Checks	High	Business Logic Issue (SCWE-001)
Stale Price Oracle Due to Improper Observation Selection in observe()	High	Business Logic Issue (SCWE-001)
Hook misconfiguration can prevent lifecycle execution in Uniswap V4 pools	High	Business Logic Issue (SCWE-001)
Improper initialization can break oracle functionality for pool participants	Medium	Business Logic Issue (SCWE-001)
Insecure Manual Milestone Triggering Allows Skipping of Reward Levels	Medium	Business Logic Issue (SCWE-001)
Delayed Oracle Update Can Prevent Milestone Triggering for Eligible Users	Medium	Business Logic Issue (SCWE-001)
Missing cardinalityNext Tracking in Observation Recording	Medium	Incorrect Accounting

getContractBalance() Always Returns Zero Misleading Users about locked token state	Low	Business Logic Issue (SCWE-001)
Admin Inability to Adjust Deadline	Low	Hardcoded Constants (SCWE-008)
Missing zero address validations	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Floating and Outdated Pragma	Low	Floating Pragma (SCWE-060)
Liquidity Providers Can Compromise Oracle Accuracy Through Partial Range Positions	Low	Missing Input Validation (SC04-Lack Of Input Validation)
Cheaper conditional operators	Gas	Gas Optimization
Gas Optimization for State Variables	Gas	Gas Optimization

Table: Findings in Smart Contracts

4. Remediation Status ------------------

BitSafe SRL is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on August 21st, 2025, and all the issues have been addressed.**

Also, the table shows the remediation status of each finding.

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Incorrect price calculation for Identical Oracle Cumulative Values	Critical	Fixed [August 21, 2025]
Incorrect Observation Indexing in _updatePool() Causes Oracle Data Overwrite	Critical	Fixed [August 21, 2025]
Oracle Users Can Receive Inflated Token Amounts Due to Incorrect Decimal Adjustment	Critical	Fixed [August 21, 2025]
Oracle Users Can Receive Zero USDT Values Due to Incorrect Decimal Division	Critical	Fixed [August 21, 2025]
Improper observation indexing can corrupt time-series data	Critical	Fixed [August 21, 2025]
Misconfigured Milestone Triggering Logic Can Delay or Deny Legitimate Reward Distribution to Participants	Critical	Fixed [August 21, 2025]
Oracle Observation Contains Hardcoded Zero Values Leading to Invalid TWAP	Critical	Fixed [August 21, 2025]
Misconfigured Oracle Hook Prevents Observation Recording	Critical	Fixed [August 21, 2025]
Users Can Lose Access To Locked Funds If Price Threshold Is Reached	Critical	Fixed [August 21, 2025]
Malicious Claimer can Claim Both Rewards and Interest After Milestone Trigger	Critical	Fixed [August 21, 2025]

User Can Repeatedly Claim Rewards From Already-Triggered Milestones	Critical	Fixed [August 21, 2025]
Malicious Actor Can Reduce Observation Buffer Size	High	Fixed [August 21, 2025]
Invalid External Call to Missing observe() Function in geomeanOracle	High	Fixed [August 21, 2025]
Unauthorized Pool Manipulation via Missing Access Control in setPool()	High	Fixed [August 21, 2025]
Missing permission declaration can revert PoolManager execution for integrated hook	High	Fixed [August 21, 2025]
Invalid Oracle Implementation Can Permanently Disable Price Checks	High	Fixed [August 21, 2025]
Stale Price Oracle Due to Improper Observation Selection in observe()	High	Fixed [August 21, 2025]
Hook misconfiguration can prevent lifecycle execution in Uniswap V4 pools	High	Fixed [August 21, 2025]
Improper initialization can break oracle functionality for pool participants	Medium	Fixed [August 21, 2025]
Insecure Manual Milestone Triggering Allows Skipping of Reward Levels	Medium	Fixed [August 21, 2025]
Delayed Oracle Update Can Prevent Milestone Triggering for Eligible Users	Medium	Fixed [August 21, 2025]
Missing cardinalityNext Tracking in Observation Recording	Medium	Fixed [August 21, 2025]
getContractBalance() Always Returns Zero Misleading Users about locked token state	Low	Fixed [August 21, 2025]
Admin Inability to Adjust Deadline	Low	Fixed [August 21, 2025]
Missing zero address validations	Low	Fixed [August 21, 2025]
Floating and Outdated Pragma	Low	Fixed [August 21, 2025]

Liquidity Providers Can Compromise Oracle Accuracy Through Partial Range Positions	Low	Fixed [August 21, 2025]
Cheaper conditional operators	Gas	Fixed [August 21, 2025]
Gas Optimization for State Variables	Gas	Fixed [August 21, 2025]

Table: Summary of findings and status of remediation

5. Bug Reports -----------------------

Bug ID #1 [Fixed]

Incorrect price calculation for Identical Oracle Cumulative Values

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The convertUSDTtoFUN() function calculates a time-weighted average tick (TWAT) using two oracle observations: one taken TWAP_WINDOW seconds ago (1 hour) and one at the current time. It then computes the delta. If both observations return identical tickCumulatives values, then. tickCumulativesDelta will be 0 ,TimeWeightedAverageTick will also be 0, The call to _getSqrtRatioAtTick(0) is then made. Inside _getSqrtRatioAtTick(), absTick will be 0. While the code does not revert on zero due to the require check (<= 887272), the zero tick case produces a valid sqrt price. Since there is no logic to handle this condition, this will result in returning the wrong value.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/USDTtoFUNOracle.sol#L109

Impacts
Price check mechanisms that rely on convertUSDTtoFUN() result in returning the wrong value.

Remediation
It is recommended to add a zero Delta Handling Condition in convertUSDTtoFUN().

Retest
This bug has been fixed by reverting when there is zero delta

Bug ID #2 [Fixed]

Incorrect Observation Indexing in _updatePool() Causes Oracle Data Overwrite

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
In the Uniswap V4 hook implementation, the _updatePool() function calculates the next observation index as: uint16 indexUpdated = (state.index + 1) % state.cardinality; . However, during _afterInitialize(), both state.cardinality and state.cardinalityNext are set to 1. This means that indexUpdated will always evaluate to 0, causing every update to overwrite the same observation entry .

Affected Code
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/GeomeanOracle.sol#L104-L128
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/GeomeanOracle.sol#L215

Impacts
Loss of historical observation data for the pool. Potential for market manipulation if price history is assumed valid but is overwritten

Remediation
It is recommended to use cardinalityNext while calculating indexUpdated.

```
uint16 indexUpdated = (state.index + 1) % state.cardinalityNext;
```

Retest
This bug has been fixed by updating the logic so that index gets properly updated

Bug ID #3 [Fixed]

Oracle Users Can Receive Inflated Token Amounts Due to Incorrect Decimal Adjustment

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The USDTtoFUNOracle contract contains a critical decimal adjustment error in the _calculateFUNFromUSDT() function. The function incorrectly multiplies the calculated funAmount by 10^12 under the assumption that decimal adjustment is needed between USDT (6 decimals) and FUN (18 decimals). However, this adjustment is mathematically incorrect given the price calculation logic already employed.

When calculating the conversion from USDT to FUN tokens, the contract uses Uniswap V4's tick-based pricing system where the sqrtPriceX96 represents the square root of the price ratio between the two tokens. The price calculation using priceX192 = uint256(sqrtPriceX96) * sqrtPriceX96 already accounts for the inherent decimal relationships encoded in the pool's tick spacing and token configuration. The additional multiplication by 10^12 creates a systematic inflation of the output amount.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/USDTtoFUNOracle.sol#L154

Impacts
Users calling convertUSDTtoFUN() will receive FUN token amounts that are inflated by a factor of one trillion (10^12).

Remediation
The immediate fix requires removing the erroneous decimal adjustment.

Retest
The issue has been fixed.

Bug ID #4 [Fixed]

Oracle Users Can Receive Zero USDT Values Due to Incorrect Decimal Division

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The USDTtoFUNOracle contract contains a critical decimal handling error in the _getUSDTFromFUN() function, which causes the function to return zero for virtually all conversion attempts. The function incorrectly divides the calculated usdtAmount by 10^12 under the false assumption that decimal adjustment is required between FUN (18 decimals) and USDT (6 decimals).

The vulnerability occurs after the price calculation logic correctly computes the USDT equivalent using Uniswap V4's tick-based pricing system. The sqrtPriceX96 value already incorporates the proper decimal relationships as configured in the pool, meaning the calculated usdtAmount is already in the correct scale. The erroneous division by 10^12 on line 146 (usdtAmount = usdtAmount / 10**12) systematically reduces the result by twelve orders of magnitude.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/USDTtoFUNOracle.sol#L190

Impacts
This vulnerability completely breaks the oracle's ability to provide FUN-to-USDT conversions, effectively rendering half of the oracle's functionality inoperative. The getFUNPriceInUSDT() function will consistently return zero, indicating that FUN tokens have no USDT value, which is obviously incorrect and misleading.

Remediation
The immediate fix requires removing the erroneous decimal division.

Retest
This issue has been fixed removing the erroneous decimal division

Bug ID #5 [Fixed]

Improper observation indexing can corrupt time-series data

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The _updatePool function is responsible for writing time-series observations to a circular buffer, used for maintaining historical pricing data. However, the implementation fails to advance the observation index stored in the ObservationState, which determines where the next observation should be written. As a result, each invocation of _updatePool continuously overwrites the same buffer slot, typically at index 0, rather than rotating through the buffer as intended.

Circular buffers are commonly used in oracle systems to store observations over time while limiting memory usage. Proper functioning of such a system depends on maintaining and incrementing a write index, typically modulo the buffer's cardinality. Without updating this index, the system effectively discards all previous observations, storing only the most recent value and destroying historical continuity.

Affected Code
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L105

Impacts
By not updating the index in the `ObservationState`, the buffer loses all prior observations, resulting in an incomplete or entirely broken historical pricing dataset.

Remediation
To fix this issue, the _updatePool function should correctly increment and wrap the observation index using modular arithmetic relative to the cardinality.

```
function _updatePool(PoolKey calldata key) private {
    PoolId id = key.toId();
    (, int24 tick,,) = manager.getSlot0(id);
```

```
        uint128 liquidity = manager.getLiquidity(id);

        (states[id].index, states[id].cardinality) = observations[id].write(
            states[id].index, _blockTimestamp(), tick, liquidity, states[id].cardinality,
states[id].cardinalityNext
        );
    }
```

Retest

This bug has been fixed by updating the logic so that index doesn’t get override

Bug ID #6 [Fixed]

Misconfigured Milestone Triggering Logic Can Delay or Deny Legitimate Reward Distribution to Participants

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The FUNGiveaway contract introduces a reward distribution mechanism based on the progressive achievement of predefined price milestones. These milestones are monitored through a scheduled price-checking system, and once a milestone's threshold is crossed and held for a certain duration, the reward is unlocked for eligible participants. However, a critical logical flaw exists in the _checkAndTriggerMilestones() function, specifically in the reliance on a single currentPriceLevel state variable to sequentially evaluate and trigger milestones.

When the price of the FUN token increases rapidly and crosses multiple milestone thresholds in a short period (e.g., directly jumping from below $0.03 to $0.10), only the milestone corresponding to currentPriceLevel is considered. The logic does not account for the possibility that higher milestones may have already been satisfied. Since currentPriceLevel increments linearly only after each milestone is held for the full PRICE_HOLD_DURATION, subsequent milestones—even if already satisfied—are deferred until earlier ones complete their respective hold periods. This causes the contract to ignore the fact that several milestones might have been simultaneously satisfied, leading to an unintended bottleneck in the milestone-triggering mechanism.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L292

Impacts
This logical misconfiguration directly undermines the time-sensitive nature of the rewards system. If multiple milestones are satisfied but only the lowest is triggered, the contract delays the unlocking of rewards for higher thresholds.

Remediation
To ensure the milestone progression logic aligns with the intended behavior, the _checkAndTriggerMilestones() function must be revised to handle multiple simultaneously satisfied

milestones. Instead of relying solely on a linear currentPriceLevel, the function should iterate through all untriggered milestones and evaluate whether each one's priceThreshold has been crossed and held for the requisite PRICE_HOLD_DURATION.

Retest

This bug has been fixed by changing the logic of _checkAndTriggerMilestones()

Bug ID #7 [Fixed]

Oracle Observation Contains Hardcoded Zero Values Leading to Invalid TWAP

Vulnerability Type
Hardcoded Constants (SCWE-008)

Severity
Critical

Description
The _updatePool() function is responsible for storing price-related oracle observations for a given Uniswap V4 pool. However, the function hardcodes tickCumulative and secondsPerLiquidityCumulativeX128 to zero when storing the observation. These values are critical for accurate Time-Weighted Average Price (TWAP) calculations. By storing `0` instead of the actual cumulative data. Future oracle queries based on this observation will yield incorrect or meaningless results.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L105

Impacts
Any logic relying on TWAP (e.g., token conversions, vault triggers, reward calculations) will be corrupted. A malicious user may exploit the predictable zero values for arbitrage or to abuse pricing mechanisms.

Remediation
It is suggested to implement correct cumulative calculations. replace hardcoded zeroes with actual values from Uniswap V4's internal cumulative state

Retest
This vulnerability has been fixed by implementing logic related to tickCumulative.

Bug ID #8 [Fixed]

Misconfigured Oracle Hook Prevents Observation Recording

Vulnerability Type
Dead Code (SCWE-062)

Severity
Critical

Description
The GeomeanOracle contract is designed to serve as an on-chain oracle for Uniswap V4 pools by recording observations of price and liquidity over time. However, a critical flaw in the contract's lifecycle flow renders the core functionality inoperative. Specifically, the _updatePool() function, which is responsible for recording updated pool state into the observation buffer, is never invoked by any external or hook-registered function. This function is defined as private and does not appear in any beforeSwap, beforeAddLiquidity, or other hook lifecycle methods that are essential for capturing meaningful changes in pool state.

The oracle relies on hooks being triggered during actions such as swaps or liquidity modifications to write new observations that reflect updated ticks and liquidity. Without integration of _updatePool() into these hook paths, no observations are ever recorded after initialization, and the oracle remains perpetually stale. This misconfiguration breaks the intended purpose of the oracle as no price history or time-weighted average can be derived when updates never occur.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L105

Impacts
Due to the lack of observation updates, the oracle does not accumulate any pricing data over time. As a result, any system depending on this oracle for time-weighted average pricing (TWAP), slippage protection, on-chain volatility measures, or historical reference points will receive invalid or empty data.

Remediation
To restore intended functionality, the _updatePool() function must be invoked from within at least one of the relevant hook lifecycle methods, such as beforeAddLiquidity() or beforeSwap(). These hooks are triggered automatically during meaningful state transitions in the pool and are ideal

entry points for capturing updated tick and liquidity data. Ensuring that _updatePool() is properly integrated into these hooks will allow the oracle to begin recording observations as expected, thereby enabling downstream consumers to reliably access and compute time-weighted pricing and liquidity metrics.

Retest

This bug has been fixed by invoking _updatePool() in various functions like _beforeSwap() , _beforeAddLiquidity() and _beforeRemoveLiquidity().

Bug ID #9 [Fixed]

Users Can Lose Access To Locked Funds If Price Threshold Is Reached

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The FUNGiveaway contract includes a mechanism for users to lock FUN tokens and either (1) withdraw them with interest after a fixed deadline or (2) become eligible for milestone-based rewards if certain price thresholds are reached. The function withdrawWithInterest() is explicitly gated to only execute after the deadline and only if milestone distribution has not been activated. This design assumes that users who locked tokens are either rewarded through price milestones or refunded with interest after the program ends.

However, a critical flaw arises due to the interaction between milestone triggers and the withdrawal logic. Once a price threshold is reached and the price is held for the configured duration (7 days), the _triggerMilestonesUpTo() function marks those milestones as triggered and activates distributionActive = true. At this point, users may claim rewards via claimRewards(), but there is no mechanism to allow withdrawal of the originally locked tokens. This applies even to users who are ineligible or only partially eligible for milestone rewards, as the contract does not track or return the principal once milestones are triggered.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L382

Impacts
Users who lock tokens expecting to retrieve them after the threshold limit reached may permanently lose access to their funds if the price threshold is met

Remediation
Introduce a mechanism for users to withdraw their original locked amount even after milestones are triggered.

Retest

This issue has been fixed by creating a new function withdrawPrincipal()

Bug ID #10 [Fixed]

Malicious Claimer can Claim Both Rewards and Interest After Milestone Trigger

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The vulnerability lies in the withdrawWithInterest() function of the smart contract responsible for handling token locks with milestone-based conditions. The contract is designed to allow users to lock tokens and later withdraw them with an 8% interest but only if the predefined price or time-based milestone has not yet been reached. Once a milestone is triggered, users are expected to claim milestone-specific rewards (such as bonuses or external incentives).

However, this invariant is broken due to the absence of a milestone check in the withdrawWithInterest() function. Specifically, there is no conditional guard to restrict users from calling this function after the milestone condition has been met. As a result, a user may first call claimRewards() after the milestone is triggered, then subsequently call withdrawWithInterest() to recover their initial tokens plus the 8% bonus. This undermines the contract's intended mechanics, where users should be permitted to either claim milestone rewards or withdraw with interest, but not both.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L417

Impacts
This issue enables users to extract more value than intended, violating core economic assumptions of the protocol. A malicious user can effectively double-dip by first triggering a milestone-based reward, and then retrieve their principal along with an additional 8% interest, despite the milestone invalidating such withdrawal.

Remediation

To fix this vulnerability, a milestone check must be introduced at the start of the withdrawWithInterest() function. The function should explicitly verify that no milestone has been reached before proceeding with the withdrawal logic.

Retest

This issue has been fixed by adding a distributionActive check inside withdrawWithInterest()

Bug ID #11 [Fixed]

User Can Repeatedly Claim Rewards From Already-Triggered Milestones

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Critical

Description
The FUNGiveaway contract implements a reward distribution system based on token locking and price milestones. Users lock FUN tokens and become eligible to claim rewards when specific price milestones are triggered. The function claimRewards() calculates a user's share of the reward pool for each triggered milestone and distributes it proportionally based on precomputed weightings.

However, the contract lacks a mechanism to track whether a user has already claimed rewards for a particular milestone. The function aggregates rewards across all triggered milestones up to currentPriceLevel without marking individual claims as completed. This oversight allows malicious users to repeatedly invoke claimRewards() and receive the same rewards multiple times, as long as the milestones remain triggered. The lack of per-user, per-milestone claim tracking leaves the reward pool exposed to repeated drains.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L382

Impacts
This vulnerability allows users to continuously claim rewards for previously triggered milestones, draining the contract's treasury wallet of reward tokens. If left unaddressed, malicious actors can exploit this repeatedly to siphon large amounts of FUN tokens from the treasury, far exceeding their fair share based on locked token weight.

Remediation
In the claimRewards() function, before adding a user's share of a milestone's reward pool to totalRewards, the contract should check if the user has already claimed rewards for that milestone. If not, mark the claim as completed (hasClaimed[msg.sender][i] = true) immediately after computing their share and before performing any external token transfers.

Retest

This issue has been fixed by adding check hasClaimed[msg.sender][i]) in the claimRewards()

Bug ID #12 [Fixed]

Malicious Actor Can Reduce Observation Buffer Size

Vulnerability Type
Access Control

Severity
High

Description
The increaseCardinalityNext function is designed to grow the capacity of the observation buffer used for storing historical data such as tick and liquidity values. This buffer, often implemented as a circular array, determines how many past observations can be retained for computing time-weighted average prices and other oracle metrics. However, the current implementation does not enforce a minimum directional constraint on buffer size changes. Specifically, the function lacks a check ensuring that the new cardinalityNext value is strictly greater than the existing cardinalityNextOld. This allows any external caller to invoke increaseCardinalityNext with a smaller value than the current buffer size, effectively reducing the number of observations that can be stored going forward.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L121

Impacts
A malicious user can submit repeated calls with smaller or equal cardinalityNext values, causing misleading state transitions that imply a smaller observation buffer.

Remediation
To prevent improper updates, the function should enforce an explicit condition that rejects any attempt to set a cardinalityNext value less than or equal to the current one. The logic should be structured to follow the same safety pattern as the grow() method, which internally checks that cardinalityNext > cardinalityOld before allowing changes.

```
function increaseCardinalityNext(PoolKey calldata key, uint16 cardinalityNext)
    external
    returns (uint16 cardinalityNextOld, uint16 cardinalityNextNew)
  {
    PoolId id = PoolId.wrap(keccak256(abi.encode(key)));

    ObservationState storage state = states[ id ];

    cardinalityNextOld = state.cardinalityNext;
    cardinalityNextNew = observations[ id ].grow(cardinalityNextOld, cardinalityNext);
    state.cardinalityNext = cardinalityNextNew;
  }
```

Retest

This bug has been fixed

Bug ID #13 [Fixed]

Invalid External Call to Missing observe( ) Function in geomeanOracle

Vulnerability Type
Broken Functionality

Severity
High

Description
The convertUSDTtoFUN() function is designed to fetch a Time-Weighted Average Price (TWAP) using the observe() method on the geomeanOracle contract. However, the geomeanOracle contract does not implement an observe() function. As a result, the call to geomeanOracle.observe(...) will always revert, making the convertUSDTtoFUN() function unusable.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Oracle.sol#L78

Impacts
Any call to convertUSDTtoFUN() will fail, causing the oracle-based conversion functionality to break

Remediation
It is recommended to ensure that the geomeanOracle address is a deployed contract that implements the observe() function with the correct signature.

Retest
This issue has been fixed by introducing a new observe() function in geomeanOracle.

Bug ID #14 [Fixed]

Unauthorized Pool Manipulation via Missing Access Control in setPool()

Vulnerability Type
Access Control

Severity
High

Description
The setPool(PoolKey calldata poolKey) function allows setting the usdtFunPoolKey value, which is later used in the convertUSDTtoFUN() function to fetch price data from the geomeanOracle.observe() call. However, this function lacks any access control, allowing any external user to invoke it and override the usdtFunPoolKey. This creates a critical vulnerability because convertUSDTtoFUN() relies on usdtFunPoolKey to fetch price data from the geomeanOracle.If a malicious user changes the usdtFunPoolKey to point to a manipulated or irrelevant pool, the result from the oracle call becomes invalid or misleading.This could affect financial conversions or reward systems based on the FUN/USDT price.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Oracle.sol#L61-L64

Impacts
Malicious actors can set usdtFunPoolKey to a pool they control or an incorrect pool, corrupting TWAP-based price conversions.

Remediation
It is recommended to restrict access to setPool(). Only privileged roles should be able to modify the pool key.

Retest
This bug has been fixed by implementing access control.

Bug ID #15 [Fixed]

Missing permission declaration can revert PoolManager execution for integrated hook

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
High

Description
The GeomeanOracle contract is intended to function as a Uniswap V4 hook, designed to interface with the PoolManager contract during lifecycle events such as swaps, liquidity updates, and initialization. However, the contract does not implement the required getHookPermissions() function, which is a mandatory part of the hook interface as defined by the BaseHook abstract contract. The getHookPermissions() function is expected to return a Hooks.Permissions struct, indicating to the PoolManager which callbacks the hook intends to receive, such as beforeSwap, afterInitialize, and others.

When the PoolManager attempts to interact with the hook, it performs an initial permissions query by calling getHookPermissions() on the hook contract. If this function is not implemented, as is the case in GeomeanOracle, the call reverts unconditionally due to the missing selector. This causes the entire pool initialization or swap lifecycle action to fail, as the PoolManager cannot safely determine what operations the hook supports or intends to intercept. The result is a critical compatibility failure that blocks all meaningful interaction between the hook and the pool infrastructure.

Affected Code
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L16

Impacts
The omission of the getHookPermissions() function causes any call from the PoolManager to the hook to revert when attempting to resolve permissions. This results in complete functional failure of the hook integration. Pools attempting to use the GeomeanOracle hook will fail during creation or operation, including during initialize, swap, addLiquidity, and other actions.

Remediation

To resolve this issue, the GeomeanOracle contract must implement the getHookPermissions() function as defined in the BaseHook interface. This function must return a properly constructed Hooks.Permissions struct that specifies which lifecycle methods the hook intends to use. This declaration enables the PoolManager to safely route callback logic and validate hook behavior at runtime.

Retest

This bug has been fixed by implementing getHookPermissions()

Bug ID #16 [Fixed]

Invalid Oracle Implementation Can Permanently Disable Price Checks

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
High

Description
The FUNGiveaway contract relies on an external oracle to fetch the current FUN/USDT price via the getPriceInUSDT function. This is implemented in the getCurrentPriceFromOracle() function, which uses Solidity's try-catch syntax to handle potential failures from the oracle call. However, the declared interface IFUNOracle incorrectly specifies the getPriceInUSDT function as view, implying it reads only local state and does not perform any complex computation or external call. But the actual oracle contract does not implement the getPriceInUSDT(uint256) function with the correct signature or at all, the low-level call inside the try block will always fail and revert.

As a result, any function depending on getCurrentPriceFromOracle() such as performScheduledPriceCheck() and emergencyPriceCheck() will fail every time due to a revert from the catch block.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L491

Impacts
The primary impact of this vulnerability is a system-wide denial of service for all users of the FUNGiveaway contract. Since price milestones are never evaluated due to failed oracle reads, the contract becomes permanently stuck in a non-progressing state.

Remediation
To resolve this issue, the contract must ensure that the getPriceInUSDT() should be properly implemented in the Oracle contract.

Retest
This issue has been fixed by properly implementing getPriceInUSDT() in the USDTtoFUNOracle contract.

Bug ID #17 [Fixed]

Stale Price Oracle Due to Improper Observation Selection in observe( )

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
High

Description
The observe() function is responsible for fetching oracle prices by selecting observations from the pool's historical data. In its current implementation, if no observation is found where obs.blockTimestamp <= targetTime, the function defaults to using the oldest observation in the queue. This approach introduces a risk of returning stale price data because the oldest observation could be significantly outdated compared to the target timestamp. In scenarios where the oracle is queried during critical trading periods, returning stale data can lead to inaccurate pricing, manipulation opportunities, and loss of funds.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/GeomeanOracle.sol#L281-L283

Impacts
Malicious users can exploit this to force outdated prices into TWAP-based trades. Arbitrage traders can profit from the discrepancy between stale and current prices.

Remediation
It is recommended to modify the observation selection logic to ensure the returned observation is the most recent one before or equal to targetTime.

Retest
This bug has been fixed.

Bug ID #18 [Fixed]

Hook misconfiguration can prevent lifecycle execution in Uniswap V4 pools

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
High

Description
The GeomeanOracle contract is intended to serve as a Uniswap V4 hook, capturing and recording pool observations such as tick and liquidity over time. However, the contract does not inherit from the BaseHook abstract contract provided by Uniswap V4. This omission results in a critical misconfiguration, rendering the contract incompatible with the hook architecture of the Uniswap V4 protocol. The BaseHook contract defines the standard interface and mechanisms by which the PoolManager communicates with hook contracts during pool lifecycle events, including beforeSwap, beforeAddLiquidity, beforeInitialize, and others.

By failing to inherit from BaseHook, GeomeanOracle is not formally recognized by the PoolManager as a valid hook, meaning it will not receive any of the callback invocations that are essential to its operation. As a result, even if the contract address is registered as a hook during pool initialization, the Uniswap V4 infrastructure will not interact with it during state transitions. This severs the core mechanism by which the contract is supposed to observe and respond to pool activity, such as swaps and liquidity modifications.

Affected Code
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L16

Impacts
The absence of inheritance from BaseHook causes the contract to be excluded from the lifecycle invocation path of the Uniswap V4 PoolManager. Consequently, no observation updates, validations, or oracle-related logic will ever execute during relevant pool events.

Remediation
To ensure proper integration with the Uniswap V4 ecosystem, the GeomeanOracle contract must inherit from the BaseHook abstract contract. This inheritance establishes the formal relationship between the hook and the PoolManager, enabling the protocol to invoke the appropriate lifecycle callbacks

Retest

This bug has been fixed by inheriting from the base hook contract.

Bug ID #19 [Fixed]

Improper initialization can break oracle functionality for pool participants

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Medium

Description
The afterInitialize() function within the hook contract is responsible for setting up observation-related state variables necessary for recording historical pool data. However, the current implementation hardcodes the cardinality and cardinalityNext values to 1 without invoking any logic to properly initialize an associated observation buffer or timestamp.

This bypasses the proper initialization pattern expected in observation-based hooks, such as invoking observations[id].initialize(_blockTimestamp()). Without this call, the associated observation data structure may remain uninitialized, and any future logic relying on historical data (e.g., TWAP oracles) could behave unpredictably or revert during access.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L92

Impacts
This improper initialization could cause failures in downstream logic that relies on historical price or liquidity data. Pool participants relying on oracle readings may receive incorrect or missing data, leading to degraded trading conditions or economic inefficiencies.

Remediation
Replace the hardcoded initialization with the appropriate pattern that initializes the observation structure using the current block timestamp.

```
PoolId id = key.toId();
    (states[ id ].cardinality, states[ id ].cardinalityNext) =
observations[ id ].initialize(_blockTimestamp());
    return GeomeanOracle.afterInitialize.selector;
```

Retest

This bug has been fixed by proper initializing values.

Bug ID #20 [Fixed]

Insecure Manual Milestone Triggering Allows Skipping of Reward Levels

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Medium

Description
The manuallyTriggerMilestone(uint256 _level) function enables an account with the OPERATOR_ROLE to manually trigger a specific milestone. However, the implementation does not enforce sequential milestone progression. This means an operator can directly trigger a higher milestone (e.g., level 3) without first triggering lower milestones (e.g., levels 0, 1, and 2).
The private _triggerPriceMilestone() function doesn't contain logic to backfill or validate the status of previous milestones. This means skipped milestones will never be triggered, and users depending on those milestones for reward distribution will permanently lose access to those rewards.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L355-L376

Impacts
If an operator skips a milestone, the associated rewards will never be distributed.

Remediation
It is suggested to add a check in manuallyTriggerMilestone() to only allow the next untriggered milestone to be called

Retest
This issue has been fixed by changing the logic.

Bug ID #21 [Fixed]

Delayed Oracle Update Can Prevent Milestone Triggering for Eligible Users

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Medium

Description
The FUNGiveaway contract relies on a price oracle (IFUNOracle) to track the FUN/USDT price and determine whether milestones have been met. This logic is executed in the performScheduledPriceCheck() function, which can only be called every 6 hours as enforced by the PRICE_CHECK_INTERVAL constant. However, this relatively large interval introduces a critical timing vulnerability.

Specifically, if the price of FUN briefly crosses the threshold and sustains the price level for several hours, but subsequently dips below the threshold before the next scheduled price check, this milestone crossing will never be acknowledged. As a result, the priceFirstHit and lastValidPriceTime variables will not be updated, and the milestone will not be triggered even if the price met the defined conditions between checks. And the same opposite case can also happen that the price was below every time but during the check it went up so the milestone gets reached.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L57

Impacts
This vulnerability introduces a scenario where genuine milestone conditions go undetected due to infrequent oracle updates.

Remediation
The contract should reduce the PRICE_CHECK_INTERVAL from 6 hours to a shorter period such as 1 hour, to allow more accurate and responsive milestone detection

Retest
This issue has been fixed by changing PRICE_CHECK_INTERVAL to 1 hr.

Bug ID #22 [Fixed]

Missing cardinalityNext Tracking in Observation Recording

Vulnerability Type
Incorrect Accounting

Severity
Medium

Description

The _updatePool() function records an Observation struct for a given pool, intending to track oracle-related data such as tick and liquidity over time. However, the implementation omits tracking the cardinalityNext field, which is critical in Uniswap V4-style oracle systems for defining the desired number of observations that can be stored in the circular buffer. The cardinalityNext field indicates how many total slots the observation ring buffer can eventually expand to and helps maintain smooth oracle observation growth. Not recording or updating this field means that the observation history won't grow as expected, severely limiting the accuracy and reliability of time-weighted price calculations.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L112-L117

Impacts
With a fixed or untracked observation capacity, time-weighted average price (TWAP) queries may rely on a very narrow window or revert due to insufficient data.

Remediation
It is recommended to include cardinalityNext in the Observation Struct Initialization.

Retest
This bug has been fixed by including cardinalityNext in the Observation Struct.

Bug ID #23 [Fixed]

getContractBalance() Always Returns Zero Misleading Users about locked token state

Vulnerability Type
Business Logic Issue (SCWE-001)

Severity
Low

Description
The function getContractBalance() in the FUNGiveaway contract is intended to return the total amount of FUN tokens locked in the contract by users. The accompanying comment describes it as returning the "Current FUN token balance of contract (locked user tokens)." However, this description is misleading and the function itself is fundamentally flawed in the context of this contract's logic.

When users lock their FUN tokens via the lockTokens() function, the tokens are not held in the contract's own balance. Instead, they are immediately transferred to the treasuryWallet, an external address defined during contract deployment and modifiable by the admin. As a result, the contract's own token balance, returned by funToken.balanceOf(address(this)), will always remain zero unless tokens are mistakenly or arbitrarily sent directly to the contract.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L733

Impacts
Although this vulnerability does not result in direct financial loss, unauthorized access, or control compromise, it undermines contract observability and trustworthiness, particularly for users or developers integrating the contract with interfaces or analytics systems.

Remediation
If the intent is to track tokens currently locked by users, the function should instead return the value of totalLockedTokens, which is incremented during each lockTokens() call and reflects the actual total of user deposits.

Retest
This issue has been fixed by returning totalLockedTokens.

Bug ID #24 [Fixed]

Admin Inability to Adjust Deadline

Vulnerability Type
Hardcoded Constants (SCWE-008)

Severity
Low

Description
The FUNGiveaway contract hardcodes a fixed timestamp as the deadline for user participation and reward eligibility in the _initializePriceMilestones() function. Specifically, each milestone is initialized with a deadline of 1735689600 (31 December 2025, 00:00:00 UTC), which is embedded directly into the contract without any setter mechanism to adjust it. This immutability creates an operational inflexibility where any change in campaign timelines such as delays in token launch, market disruptions, or user onboarding periods cannot be reflected on-chain.

Affected Code
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L149

Impacts
If market conditions delay the achievement of price milestones past the arbitrary cutoff date, the entire incentive structure breaks down, rendering locked tokens ineligible for milestone rewards.

Remediation
To resolve this vulnerability, the contract should introduce an administrative setter function allowing updates to the deadline.

Retest
This issue has been fixed by adding updateDeadline. This function should be used such that it doesn't affect other things.

Bug ID #25 [Fixed]

Missing zero address validations

Vulnerability Type
Missing Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description:
The contracts were found to be setting new addresses without proper validations for zero addresses.
Address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever.
Depending on the logic of the contract, this could prove fatal and the users or the contracts could lose their funds, or the ownership of the contract could be lost forever.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Oracle.sol#L49-L55

Impacts
If address type parameters do not include a zero-address check, contract functionality may become unavailable or tokens may be burned permanently.

Remediation
Add a zero address validation to all the functions where addresses are being set.

Retest
This issue has been fixed by adding zero address validation.

Bug ID #26 [Fixed]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SCWE-060)

Severity
Low

Description
Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities.
The contract allowed floating or unlocked pragma to be used, i.e., >= 0.8.19. This allows the contracts to be compiled with all the solidity compiler versions above the limit specified. The following contracts were found to be affected -

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/GeomeanOracle.sol#L2
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Oracle.sol#L4
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L2

Impacts
If the smart contract gets compiled and deployed with an older or too recent version of the solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions.
Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic.
The likelihood of exploitation is low.

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.30 pragma version
Reference: https://scs.owasp.org/SCWE/SCSVS-CODE/SCWE-060/

Retest

This issue has been fixed.

Bug ID #27 [Fixed]

Liquidity Providers Can Compromise Oracle Accuracy Through Partial Range Positions

Vulnerability Type
Missing Input Validation (SC04-Lack Of Input Validation)

Severity
Low

Description
The USDTtoFUNOracle contract lacks critical validation in its liquidity management functionality, specifically missing tick spacing constraints in the _beforeAddLiquidity hook. While the provided code excerpt shows the oracle's price calculation logic, the vulnerability exists in the underlying hook implementation that should enforce full-range liquidity positions for accurate TWAP oracle functionality.

Oracle-based hooks in Uniswap V4 require liquidity to be distributed across the full tick range to ensure that Time-Weighted Average Price (TWAP) calculations remain representative of true market conditions. The missing validation allows liquidity providers to add concentrated positions within narrow tick ranges, which can create artificial price movements and skewed TWAP data. The absent check should validate that liquidity positions span from TickMath.minUsableTick(maxTickSpacing) to TickMath.maxUsableTick(maxTickSpacing), ensuring that all liquidity contributions cover the maximum possible price range.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/1fd931f12921f1c8eb35c3e534e5d180ec0fd3d5/contracts/GeomeanOracle.sol#L154

Impacts
This vulnerability enables sophisticated oracle manipulation attacks that can have severe consequences for any protocol depending on the USDTtoFUNOracle for pricing decisions.

Remediation
The immediate fix requires implementing strict tick range validation in the _beforeAddLiquidity hook to ensure all liquidity positions cover the full usable tick range.

```
int24 maxTickSpacing = manager.MAX_TICK_SPACING();
    if (
      params.tickLower != TickMath.minUsableTick(maxTickSpacing)
        || params.tickUpper != TickMath.maxUsableTick(maxTickSpacing)
    ) revert OraclePositionsMustBeFullRange();
_updatePool(key);
```

Retest

This bug has been fixed by adding validation.

Bug ID #28 [Fixed]

Cheaper conditional operators

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Upon reviewing the code, it has been observed that the contract uses conditional statements involving comparisons with unsigned integer variables. Specifically, the contract employs the conditional operators x != 0 and x > 0 interchangeably. However, it's important to note that during compilation, x != 0 is generally more cost-effective than x > 0 for unsigned integers within conditional statements.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L407
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L440
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L539
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L698

Impacts
Employing x != 0 in conditional statements can result in reduced gas consumption compared to using x > 0. This optimization contributes to cost-effectiveness in contract interactions.

Remediation
Whenever possible, use the x != 0 conditional operator instead of x > 0 for unsigned integer variables in conditional statements.

Retest
This issue has been fixed.

Bug ID #29 [Fixed]

Gas Optimization for State Variables

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Plus equals (+=) costs more gas than the addition operator. The same thing happens with minus equals (-=). Therefore, x +=y costs more gas than x = x + y.

Affected Code

- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L258
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L259
- https://github.com/FunTokenHubs/5m-staking-contracts/blob/766a67344487baafdc384352f6e72f1eb5992ca3/contracts/Staking.sol#L260

Impacts
Writing the arithmetic operations in x = x + y format will save some gas.

Remediation
It is suggested to use the format x = x + y in all the instances mentioned above.

Retest
This issue has been fixed.

6. The Disclosure -----------------------

The Reports provided by CredShields are not an endorsement or condemnation of any specific project or team and do not guarantee the security of any specific project. The contents of this report are not intended to be used to make decisions about buying or selling tokens, products, services, or any other assets and should not be interpreted as such.

Emerging technologies such as Smart Contracts and Solidity carry a high level of technical risk and uncertainty. CredShields does not provide any warranty or representation about the quality of code, the business model or the proprietors of any such business model, or the legal compliance of any business. The report is not intended to be used as investment advice and should not be relied upon as such.

CredShields Audit team is not responsible for any decisions or actions taken by any third party based on the report.

YOUR SECURE FUTURE STARTS HERE

At CredShields, we're more than just auditors. We're your strategic partner in ensuring a secure Web3 future. Our commitment to your success extends beyond the report, offering ongoing support and guidance to protect your digital assets

Audited by