Audited by CRED SHIELDS

CredShields

Smart Contract Audit

April 15th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of LogX between August 13th, 2024, and August 21st, 2024. A retest was performed on the 26th and 27th of August, 2024.

Author

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

Reviewers

Aditya Dixit (Research Team Lead), Shreyas Koli(Auditor), Naman Jain (Auditor), Sanket Salavi (Auditor)

Prepared for

LogX

Table of Contents

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

LogX engaged CredShields to perform a smart contract audit from August 13th, 2024, to August 21st, 2024. During this timeframe, 19 vulnerabilities were identified. A retest was performed on the 26th and 27th of August, 2024, and all the bugs have been resolved.

During the audit, 5 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "LogX" and should be prioritized for remediation, and fortunately, none were found.

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
LogX Token Contracts	3	2	2	4	1	7	**19**
Bugs Fixed	**3**	**2**	**2**	**4**	**1**	**7**	**19**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in the LogX Token Contract's scope during the testing window while abiding by the policies set forth by LogX'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 LogX'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 LogX can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, LogX can future-proof its security posture and protect its assets.

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

LogX engaged CredShields to perform a LogX Token Contract's Smart 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 August 13th, 2024, to August 21st, 2024, 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
LogX Token Contracts - https://github.com/eugenix-io/Athena/tree/dd531958156357b47816aebb62d10eca10d6c10b

2.1.2 Documentation

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

2.1.3 Audit Goals

CredShields uses both in-house tools and manual methods for comprehensive smart contract security auditing. The majority of the audit is done by manually reviewing the contract source code, following SWC registry standards, and an extended industry standard self-developed checklist. The team places emphasis on understanding core concepts, preparing test cases, and evaluating business logic for potential vulnerabilities.

2.2 Retesting phase

LogX 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 the asset and SWC classification. Each asset section will include a summary. The table in the executive summary contains the total number of identified security vulnerabilities per asset per risk indication.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 19 security vulnerabilities were identified in the asset. **All these vulnerabilities are fully addressed and resolved by LogX.**

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
A Malicious User Can Claim Other Users' Funds	Critical	Missing Input Validation
Missing Validation For msg.value While Staking	Critical	Missing Input Validation
Missing nonce and block.timestamp Validation While Creating stakeId	Critical	Missing Input Validation
Stake Function Allowing Bypass Of Staking Duration	High	Business Logic
Missing Input Validation can Lead to Loss of Funds while Unskating	High	Missing Input Validation
A Malicious User Can Restake Other User's Token	Medium	Missing Input Validation
Native ETH can be Stuck in the Contract	Medium	Incorrect Funds Management

Missing Zero Address Validations	Low	Missing Input Validation
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Missing Events in Important Functions	Low	Missing Best Practices
Use Ownable2Step	Low	Missing Best Practices
Dead Code	Informational	Code With No Effects (SWC-135)
Cheaper Inequalities in require()	Gas	Gas & Missing Best Practices
Gas Optimization in Require/Revert Statements	Gas	Gas Optimization
Public Constants can be Private	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Large Number Literals	Gas	Gas & Missing Best Practices
Custom Error to Save Gas	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas Optimization

Table: Findings in Smart Contracts

3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	Function Default Visibility	Not Vulnerable	Not applicable after v0.5.X (Currently using solidity v >= 0.8.6)
SWC-101	Integer Overflow and Underflow	Not Vulnerable	The issue persists in versions before v0.8.X.
SWC-102	Outdated Compiler Version	Vulnerable	Bug ID#9
SWC-103	Floating Pragma	Vulnerable	Bug ID#9
SWC-104	Unchecked Call Return Value	Not Vulnerable	call() is not used
SWC-105	Unprotected Ether Withdrawal	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	Unprotected SELFDESTRUCT Instruction	Not Vulnerable	selfdestruct() is not used anywhere
SWC-107	Reentrancy	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	State Variable Default Visibility	Not Vulnerable	Not Vulnerable
SWC-109	Uninitialized Storage Pointer	Not Vulnerable	Not vulnerable after compiler version, v0.5.0
SWC-110	Assert Violation	Not Vulnerable	Asserts are not in use.
SWC-111	Use of Deprecated Solidity Functions	Not Vulnerable	None of the deprecated functions like block.blockhash(), msg.gas, throw, sha3(), callcode(), suicide() are in use
SWC-112	Delegatecall to Untrusted Callee	Not Vulnerable	Not Vulnerable.

SWC-113	DoS with Failed Call	Not Vulnerable	No such function was found.
SWC-114	Transaction Order Dependence	Not Vulnerable	Not Vulnerable.
SWC-115	Authorization through tx.origin	Not Vulnerable	tx.origin is not used anywhere in the code
SWC-116	Block values as a proxy for time	Not Vulnerable	Block.timestamp is not used
SWC-117	Signature Malleability	Not Vulnerable	Not used anywhere
SWC-118	Incorrect Constructor Name	Not Vulnerable	All the constructors are created using the constructor keyword rather than functions.
SWC-119	Shadowing State Variables	Not Vulnerable	Not applicable as this won't work during compile time after version 0.6.0
SWC-120	Weak Sources of Randomness from Chain Attributes	Not Vulnerable	Random generators are not used.
SWC-121	Missing Protection against Signature Replay Attacks	Not Vulnerable	No such scenario was found
SWC-122	Lack of Proper Signature Verification	Not Vulnerable	Not used anywhere
SWC-123	Requirement Violation	Not Vulnerable	Not vulnerable
SWC-124	Write to Arbitrary Storage Location	Not Vulnerable	No such scenario was found
SWC-125	Incorrect Inheritance Order	Not Vulnerable	No such scenario was found
SWC-126	Insufficient Gas Griefing	Not Vulnerable	No such scenario was found
SWC-127	Arbitrary Jump with Function Type Variable	Not Vulnerable	Jump is not used.
SWC-128	DoS With Block Gas Limit	Not Vulnerable	Not Vulnerable.

SWC-129	Typographical Error	Not Vulnerable	No such scenario was found
SWC-130	Right-To-Left-Override control character (U+202E)	Not Vulnerable	No such scenario was found
SWC-131	Presence of unused variables	Vulnerable	Bug ID#12
SWC-132	Unexpected Ether balance	Not Vulnerable	No such scenario was found
SWC-133	Hash Collisions With Multiple Variable Length Arguments	Not Vulnerable	abi.encodePacked() or other functions are not used.
SWC-134	Message call with hardcoded gas amount	Not Vulnerable	Not used anywhere in the code
SWC-135	Code With No Effects	Vulnerable	Bug ID#12
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

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

LogX is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on the 26th and 27th of August, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
A Malicious User Can Claim Other Users' Funds	Critical	**Fixed** [Aug 26th, 2024]
Missing Validation For msg.value While Staking	Critical	**Fixed** [Aug 26th, 2024]
Missing nonce and block.timestamp Validation While Creating stakeId	Critical	**Fixed** [Aug 27th, 2024]
Stake Function Allowing Bypass Of Staking Duration	High	**Fixed** [Aug 26th, 2024]
Missing Input Validation can Lead to Loss of Funds while Unskating	High	**Fixed** [Aug 26th, 2024]
A Malicious User Can Restake Other User's Token	Medium	**Fixed** [Aug 26th, 2024]
Native ETH can be Stuck in the Contract	Medium	**Fixed** [Aug 26th, 2024]
Missing Zero Address Validations	Low	**Fixed** [Aug 26th, 2024]
Floating and Outdated Pragma	Low	**Won't Fix** [Aug 27th, 2024]
Missing Events in Important Functions	Low	**Won't Fix** [Aug 27th, 2024]
Use Ownable2Step	Low	**Fixed** [Aug 27th, 2024]

Dead Code	Informational	**Fixed** [Aug 26th, 2024]
Cheaper Inequalities in require()	Gas	**Won't Fix** [Aug 27th, 2024]
Gas Optimization in Require/Revert Statements	Gas	**Fixed** [Aug 27th, 2024]
Public Constants can be Private	Gas	**Won't Fix** [Aug 27th, 2024]
Cheaper Conditional Operators	Gas	**Fixed** [Aug 26th, 2024]
Large Number Literals	Gas	**Won't Fix** [Aug 27th, 2024]
Custom Error to Save Gas	Gas	**Won't Fix** [Aug 27th, 2024]
Gas Optimization in Increments	Gas	**Fixed** [Aug 26th, 2024]

Table: Summary of findings and status of remediation

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

Bug ID#1 [Fixed]

A Malicious User Can Claim Other Users' Funds

Vulnerability Type
Missing Input Validation

Severity
Critical

Description
The claimTokens function allows a malicious user to claim rewards belonging to other users. This vulnerability arises because the claimTokens function accepts an arbitrary _account as input and passes it to the internal _claimTokens function, while automatically setting the _receiver to msg.sender. This allows a malicious user to specify any user's account as the _account parameter and effectively claim that user's token rewards for themselves.

Affected Code
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L317-L319

Impact
A malicious user can exploit this vulnerability to steal rewards from multiple users, leading to financial losses for the affected users.

Remediation
To remediate this issue, the contract should enforce strict access control by ensuring that only the actual owner (msg.sender) can claim their rewards.

Retest
This vulnerability has been fixed by removing claimTokens() function. However, without the claimTokens() function, a new issue arises where users won't be able to claim their tokens without a handler.

Bug ID#2 [Fixed]

Missing Validation For msg.value While Staking

Vulnerability Type
Missing Input Validation

Severity
Critical

Description
The _stake function lacks validation for msg.value. While the function requires that the _amount parameter be greater than zero, it fails to verify that the actual ether sent (msg.value) matches the specified _amount. This allows a malicious user to call the _stake function with a mismatched msg.value, resulting in an inaccurate recording of staked amounts within the contract.
The malicious user could exploit this vulnerability by sending a very low or zero msg.value while specifying a higher _amount. The contract would then record the higher _amount as staked, even though the funds were never transferred.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L218-L228

Impact
A malicious user can exploit it by sending zero native tokens (msg.value) while specifying a high _amount value. This causes the contract to record a large stake without actual funds being transferred incorrectly.

Remediation
To mitigate this issue, the _stake function should include a validation check that ensures the msg.value matches the _amount specified by the user.

Retest
This vulnerability has been fixed by removing "_stake" function.

Bug ID#3 [Fixed]

Missing nonce and block.timestamp Validation While Creating stakeId

Vulnerability Type
Missing Input Validation

Severity
Critical

Description
The _addStake function is responsible for generating a unique stakeId based on the combination of an account address, amount, duration, APY (Annual Percentage Yield), and start time. However, the current implementation is vulnerable because the same set of inputs can produce identical stakeId, leading to duplicate IDs. This occurs when a malicious user submits the same inputs (i.e., account, amount, duration, APY, and start time), resulting in the same stakeId being generated twice. As a consequence, the system will treat both stakes as the same, potentially overwriting or confusing the state of each stake.

Affected Code
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L373-L381

Impact
If a duplicate stakeId is created, it may result in the overwriting of previous staking records. This can lead to inaccurate tracking of stakes and rewards, ultimately affecting users. Users may experience discrepancies in their balances or lose their staking history.

Remediation
To resolve this issue, it is recommended to incorporate a nonce or the current block's timestamp into the stakeId generation process. By including an additional unique value like a nonce or block.timestamp in the keccak256 hash, each stakeId will be uniquely tied to its specific transaction, even if the other inputs are identical. This will effectively prevent the generation of duplicate stakeIds, ensuring that each stake is uniquely identified and managed correctly in the system.

Retest
This vulnerability has been fixed by adding Nonce and blocktime in the _stake() function.

Bug ID#4 [Fixed]

Stake Function Allowing Bypass Of Staking Duration

Vulnerability Type
Business Logic

Severity
High

Description
The stake function in the contract allows users to specify a _startTime parameter, which represents the starting time of their staking period. This _startTime is later used in the unstake function to check if the staking duration is complete. However, since the _startTime is provided by the user, a malicious user can manipulate this value by setting it to a past timestamp. By doing so, the users can effectively bypass the required staking duration, allowing them to unstake their assets before the stake duration.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L201-L204

Impact
A user can specify a _startTime in the past, thereby tricking the contract into thinking that the staking period has already elapsed. This allows the user to unstake their assets without waiting for the intended duration.

Remediation
It is recommended to remove the _startTime parameter from the stake function and instead use the current block's timestamp (block.timestamp) as the start time for the staking period. This ensures that the start time is always accurate and cannot be manipulated by the user.

Retest
This vulnerability has been fixed by removing _stake function.

Bug ID#5 [Fixed]

Missing Input Validation can Lead to Loss of Funds while Unskating

Vulnerability Type
Missing Input Validation

Severity
High

Description
The unstake function allows a user to unstake tokens based on the provided account and stake ID. However, it permits a user to specify any account, including that of another user whose staking duration has ended. If a user (User A) provides the account data of another user (User B), the _unstake function will incorrectly process the request. As a result, User B's staking data will be deleted through the _removeStake() function, and User A's tokens will be burned. This leads to a significant accounting issue, as both users effectively lose their funds: User B's staked tokens are removed without their consent, and User A loses their tokens due to the unintended burn.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L249

Impact
The impact of this vulnerability is severe. It results in the loss of funds for both users involved. User A may lose their staked tokens without receiving any corresponding rewards, while User B is deprived of their staked assets as the native token is not transferred.

Remediation
To remediate this issue, the contract should enforce strict access control by ensuring that only the actual owner of the staked ETH (msg.sender) can initiate the unstake process for their account.

```
  function unstake(bytes32 account, bytes32 _stakeId) external nonReentrant returns(uint256){
    if(inPrivateStakingMode) { revert("LogxStaker: action not enabled"); }
-    return _unstake(account, msg.sender, _stakeId);
+    return _unstake(msg.sender, msg.sender, _stakeId);
  }
```

Retest

This vulnerability has been fixed by removing the unstake() function. However, without the unstake() function, a new issue arises where users won't be able to unstake their tokens without a handler.

Bug ID#6 [Fixed]

A Malicious User Can Restake Other User's Token

Vulnerability Type
Missing Input Validation

Severity
Medium

Description
The restake function allows a user to restake their tokens once the staking duration has ended. However, the function lacks user validation which allows any user to specify an arbitrary _account as input. A malicious user can use this flaw to restake tokens belonging to another user, forcing the original user into a new staking period without their consent.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L295-L298

Impact
a malicious user to deny the original user the ability to withdraw their staked tokens when the staking duration ends. The victim is forced to wait for the new staking period to expire before they can access their funds.

Remediation
To mitigate this issue, the restake function should be modified to ensure that only the original owner of the staked tokens (msg.sender) can initiate the restaking process for their account.

Retest
This vulnerability has been fixed by removing the restake() function. However, by removing the restake() function, a new issue arises where users won't be able to claim their tokens without a handler.

Bug ID#7 [Fixed]

Native ETH can be Stuck in the Contract

Vulnerability Type
Incorrect Funds Management

Severity
Medium

Description
The current implementation of the smart contract lacks a function to withdraw native funds that are sent to it. This creates a critical issue where any native tokens directly sent to the contract are permanently locked, with no available mechanism for their retrieval. These funds become inaccessible, except for potential use in paying for future transactions, leading to a scenario where users cannot recover their deposited assets.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L451

Impact
Without the ability to withdraw, users are left with funds trapped within the contract, rendering them useless for any other purpose outside the contract's intended functionality. This could lead to significant financial loss.

Remediation
To address this issue, it is essential to implement a native token withdrawal function within the contract. This function should allow users to retrieve any funds they have deposited, ensuring that their assets remain accessible and under their control. By adding this withdrawal capability, the contract will safeguard user funds from being irretrievably lost.

Retest
This vulnerability has been fixed by introducing the withdrawLogX() function.

Bug ID#8 [Fixed]

Missing Zero Address Validations

Vulnerability Type
Missing 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/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L46-L48
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L55-L57
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L96-L98

Impact
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 vulnerability has been fixed by implementing zero address validation.

Bug ID#9 [Won't Fix]

Floating and Outdated Pragma

Vulnerability Type
Floating Pragma (SWC-103)

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.

Affected Code
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L3
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L3

Impact
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 0.8.25 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest
This vulnerability is not fixed; it is recommended to use the above remediation.
Client's Comment: We couldn't compile OpenZeppelin contracts if using a fixed version.

Bug ID#10 [Won't Fix]

Missing Events in Important Functions

Vulnerability Type
Missing Best Practices

Severity
Low

Description
Events are inheritable members of contracts. When you call them, they cause the arguments to be stored in the transaction's log—a special data structure in the blockchain. These logs are associated with the address of the contract which can then be used by developers and auditors to keep track of the transactions.
The contract was found to be missing these events on certain critical functions which would make it difficult or impossible to track these transactions off-chain.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L46-L48
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L50-L53
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L55-L57
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L60-L62
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L84-L86
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L88-L90
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L92-L94
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L96-L98
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L103-L105

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L111-L113
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L115-L117
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L124-L127
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L130-L132
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L295-L298
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L300-L303
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L305-L312
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L370-L392
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L395-L417
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L419-L430
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L433-L452

Impact
Events are used to track the transactions off-chain and missing these events on critical functions makes it difficult to audit these logs if they're needed at a later stage.

Remediation
Consider emitting events for important functions to keep track of them.

Retest
Client's Comment: The important events are emitted in our core trading contracts. As all of the flows undergo our Endpoint (our entry contract for trading contracts) and the events are emitted there.

Bug ID#11 [Fixed]

Use Ownable2Step

Vulnerability Type
Missing Best Practices

Severity
Low

Description
The "Ownable2Step" pattern is an improvement over the traditional "Ownable" pattern, designed to enhance the security of ownership transfer functionality in a smart contract. Unlike the original "Ownable" pattern, where ownership can be transferred directly to a specified address, the "Ownable2Step" pattern introduces an additional step in the ownership transfer process. Ownership transfer only completes when the proposed new owner explicitly accepts the ownership, mitigating the risk of accidental or unintended ownership transfers to mistyped addresses.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L18

Impact
Without the "Ownable2Step" pattern, the contract owner might inadvertently transfer ownership to an unintended or mistyped address, potentially leading to a loss of control over the contract. By adopting the "Ownable2Step" pattern, the smart contract becomes more resilient against external attacks aimed at seizing ownership or manipulating the contract's behavior.

Remediation
It is recommended to use either Ownable2Step or Ownable2StepUpgradeable depending on the smart contract.

Retest
This vulnerability is fixed by adding Ownable2StepUpgradeable in the contract.

Bug ID#12 [Fixed]

Dead Code

Vulnerability Type
Code With No Effects - SWC-135

Severity
Informational

Description
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters, contracts, and interfaces that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L84-L86
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L92-L94

Impact
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement.
This reduces the overall size of the contracts and also helps in saving gas.

Remediation
If the library functions are not supposed to be used anywhere, consider removing them from the contract.

Retest
This vulnerability has been fixed by removing the function.

Bug ID#13 [Won't Fix]

Cheaper Inequalities in require()

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract was found to be performing comparisons using inequalities inside the require statement. When inside the require statements, non-strict inequalities (>=, <=) are usually costlier than strict equalities (>, <).

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L88
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L114
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L126
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L284

Impact
Using non-strict inequalities inside "require" statements costs more gas.

Remediation
It is recommended to go through the code logic, and, if possible, modify the non-strict inequalities with the strict ones to save gas as long as the logic of the code is not affected.

Retest
Client's Comment: We require loose inequalities wherever they are not changed.

Bug ID#14 [Fixed]

Gas Optimization in Require/Revert Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The require/revert statement takes an input string to show errors if the validation fails.
The strings inside these functions that are longer than 32 bytes require at least one additional MSTORE, along with additional overhead for computing memory offset and other parameters. For this purpose, having strings lesser than 32 bytes saves a significant amount of gas.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L88
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L114
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L122
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L123
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L126
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L135
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L136
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L104
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L108
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L112

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L116
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L202
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L231
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L264
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L266
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L283
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L284
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L307
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L308
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L345
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L415

Impact
Having longer require/revert strings than 32 bytes cost a significant amount of gas.

Remediation
It is recommended to shorten the strings passed inside require/revert statements to fit under 32 bytes. This will decrease the gas usage at the time of deployment and at runtime when the validation condition is met.

Retest
This issue is fixed by shortening the String passed inside require/revert statements.

Bug ID#15 [Won't Fix]

Public Constants can be Private

Vulnerability Type
Gas Optimization

Severity
Gas

Description
Public constant variables cost more gas because the EVM automatically creates getter functions for them and adds entries to the method ID table. The values can be read from the source code instead.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogX.sol#L18
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L22
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L23
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L26

Impact
Public constants are more costly due to the default getter functions created for them, increasing the overall gas cost.

Remediation
If reading the values for the constants is not necessary, consider changing the public visibility to private.

Retest
Client's Comment: We would like to keep decimals public. Already made BPD and Precision internal.

Bug ID#16 [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/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L125
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L219

Impact
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 vulnerability has been fixed.

Bug ID#17 [Won't Fix]

Large Number Literals

Vulnerability Type
Gas Optimization & Missing Best Practices

Severity
Gas

Description
Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L76-L81

Impact
Having a large number literals in the code increases the gas usage of the contract during its deployment and when the functions are used or called from the contract.
It also makes the code harder to read and audit and increases the chances of introducing code errors.

Remediation
Scientific notation in the form of 2e10 is also supported, where the mantissa can be fractional, but the exponent has to be an integer. The literal MeE is equivalent to M * 10**E. Examples include 2e10, 2e10, 2e-10, 2.5e1, as suggested in official solidity documentation. https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals
It is recommended to use numbers in the form "35 * 1e7 * 1e18" or "35 * 1e25".
The numbers can also be represented by using underscores between them to make them more readable such as "35_00_00_000"

Retest
Client's Comment: We would like to go with the current representation since it's a gas optimization to maintain a unified architecture with our trading contracts.

Bug ID#18 [Won't Fix]

Custom Error to Save Gas

Vulnerability Type
Gas Optimization

Severity
Gas

Description
During code analysis, it was observed that the smart contract is using the revert() statements for error handling. However, since Solidity version 0.8.4, custom errors have been introduced, providing a better alternative to the traditional revert(). Custom errors allow developers to pass dynamic data along with the revert, making error handling more informative and efficient. Furthermore, using custom errors can result in lower gas costs compared to the revert() statements.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L104
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L108
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L112
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L116
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L202
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L248
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L296
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L415

Impact

Custom errors allow developers to provide more descriptive error messages with dynamic data. This provides better insights into the cause of the error, making it easier for users and developers to understand and address issues.

Remediation
It is recommended to replace all the instances of revert() statements with error() to save gas.

Retest
Client's Comment: We would like to go with string errors since it's a gas optimization to maintain a unified architecture with our trading contracts.

Bug ID#19 [Fixed]

Gas Optimization in Increments

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The contract uses two for loops, which use post increments for the variable "i".
It can save some gas by changing this to ++i.
++i costs less gas compared to i++ or i += 1 for unsigned integers. In i++, the compiler has to create a temporary variable to store the initial value. This is not the case with ++i in which the value is directly incremented and returned, thus, making it a cheaper alternative.

Affected Code

- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L169
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L328
- https://github.com/eugenix-io/Athena/blob/dd531958156357b47816aebb62d10eca10d6c10b/src/LogxStaker.sol#L403

Impact
Using i++ instead of ++i costs the contract deployment around 600 more gas units.

Remediation
It is recommended to switch to ++i and change the code accordingly so the function logic remains the same and meanwhile saves some gas.

Retest
This vulnerability has been fixed by updating as recommended.

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

CRED SHIELDS

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 CRED SHIELDS

20A TANJONG PAGAR ROAD, SINGAPORE (088443)

info@credshields.com