CredShields

Smart Contract Audit

April 15th, 2024 • CONFIDENTIAL

Description

This document details the process and result of the staking smart contract audit performed by CredShields Technologies PTE. LTD. on behalf of Arcana between April 10th, 2024, and April 13th, 2024. And a retest was performed on April 15th, 2024.

Author

Shashank (Co-founder, CredShields)

shashank@CredShields.com

Reviewers

Aditya Dixit (Research Team Lead)

aditya@CredShields.com

Prepared for

Arcana

Table of Contents

CRED SHIELDS

1. Executive Summary

Arcana engaged CredShields to perform a smart contract audit from April 10th, 2024, to April 13th, 2024. During this timeframe, Eleven (11) vulnerabilities were identified. **A retest was performed on April 15th, 2024, and all the bugs have been addressed.**

During the audit, One (1) vulnerability was found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Arcana" 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
staking smart contract	0	1	0	3	2	5	**11**
	0	**1**	**0**	**3**	**2**	**5**	**11**

Table: Vulnerabilities Per Asset in Scope

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

2. Methodology

Arcana engaged CredShields to perform an Arcana 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 April 10th, 2024, to April 13th, 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
Phase 1 **https://github.com/arcana-network/staking-platform-fixed-apy/tree/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts** **Phase 2** **https://github.com/arcana-network/staking-platform-fixed-apy/tree/ad8b026d4c7664df75174a82e004a8bd6dc39cea/contracts**

Table: List of Files in Scope

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

Arcana 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, 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	Note	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 around the engagement or this document.

3. Findings

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, Eleven (11) security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Bypass the lockupDuration period in withdraw() function	High	Business Logic Issue
Floating and Outdated Pragma	Low	Floating Pragma (SWC-103)
Use Ownable2Step	Low	Missing Best Practices
Missing Events in Important Functions	Low	Missing Best Practices
Functions should be declared External	Informational	Best Practices
Incorrect Documentation	Informational	Improper Documentation
Gas Optimization in Require Statements	Gas	Gas Optimization

Code Optimization by using max and min	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Gas Optimization for State Variables	Gas	Gas Optimization
Dead Code	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	Not Vulnerable	Version 0^.8.0 and above is used
SWC-103	Floating Pragma	Not Vulnerable	Contract uses floating pragma
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	Not Vulnerable	No such scenario was found
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	Not Vulnerable	No such scenario was found
SWC-136	Unencrypted Private Data On-Chain	Not Vulnerable	No such scenario was found

4. Remediation Status

Arcana is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on April 15th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Bypass the lockupDuration period in withdraw() function	High	**Fixed [15/04/2024]**
Floating and Outdated Pragma	Low	**Fixed [15/04/2024]**
Use Ownable2Step	Low	**Fixed [15/04/2024]**
Missing Events in Important Functions	Low	**Fixed [15/04/2024]**
Functions should be declared External	Informational	**Fixed [15/04/2024]**
Incorrect Documentation	Informational	**Fixed [15/04/2024]**
Gas Optimization in Require Statements	Gas	**Fixed [15/04/2024]**
Code Optimization by using max and min	Gas	**Fixed**

		[15/04/2024]
Cheaper Conditional Operators	Gas	**Fixed [15/04/2024]**
Gas Optimization for State Variables	Gas	**Fixed [15/04/2024]**
Dead Code	Gas	**Fixed [15/04/2024]**

Table: Summary of findings and status of remediation

5. Bug Reports

Bug ID #1

Bypass the lockupDuration period in withdraw() function

Vulnerability Type

Business Logic Issue

Severity

High

Description:

In the provided contract, the withdraw() function allows users to withdraw their staked tokens before the lockup period ends. In the _updateRewards() function the user's start time (_userStartTime) is initialized to zero if they stake tokens before the staking period starts. This results in the calculation of rewards based on the startTime to the current time. Consequently, users can withdraw their tokens before the lockup duration because _userStartTime[_msgSender()] will be set to zero because user staked tokens before staking start then (block.timestamp - _userStartTime[_msgSender()]) is greater than lockupDuration that's why user can withdraw tokens using withdraw() before lockupDuration ends.

Affected Variables and Line Numbers

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/main/contracts/staking/StakingPlatform.sol#L114-L135
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/main/contracts/staking/StakingPlatform.sol#L289-L294

Impacts

Users can withdraw their staked tokens before the lockup duration ends, thereby bypassing the intended lockup period requirement. Allowing early withdrawals undermines the lockup mechanism's purpose, resulting in users receiving rewards without fulfilling the lockup duration requirement. This can lead to a loss of incentives for users to stake their tokens for the intended duration.

Remediation

Ensure that the user's start time (_userStartTime) is correctly initialized to the beginning of the staking period when they stake tokens, regardless of whether they stake before the staking period starts. This ensures that the reward calculation is based on the actual start time of the user's stake.

Test Case

```
describe("Withdraw Bypass:", () => {
	it("Should bypasss the lockupDuration check and withdraw the amount after
staking is started", async function () {
	// User deposits tokens
	await token
	.connect(user)
	.approve(stakingPlatform.address, depositAmount);
	await stakingPlatform.connect(user).deposit(depositAmount);

	const blockTimestamp = (await ethers.provider.getBlock("latest"))
	.timestamp;
	console.log("time while depositing the token: ", blockTimestamp);

	// Start staking period
	await stakingPlatform.connect(deployer).startStaking();

	// Fast forward to 8 days
	const timeToPass = 8 * 24 * 60 * 60;
	await ethers.provider.send("evm_increaseTime", [timeToPass]);
	await ethers.provider.send("evm_mine");

	const blockTimestampAfter = (await ethers.provider.getBlock("latest"))
	.timestamp;
	console.log("time after staking started: ", blockTimestampAfter);
```

```
        // Record balances before withdrawal
        const initialUserBalance = await token.balanceOf(user.address);

        const rewardsToClaim = await stakingPlatform.rewardOf(user.address);

        // User withdraws their stake
        await stakingPlatform.connect(user).withdraw(depositAmount);

        // Ensure user's token balance increased by the withdrawn amount
        const finalUserBalance = await token.balanceOf(user.address);
        expect(finalUserBalance).to.equal(
        initialUserBalance.add(depositAmount).add(rewardsToClaim)
        );
        });
});
```

Retest

This vulnerability has been fixed by introducing _getStartTime() function in the contract.
Ref: [b82b301bf1333e6165fac33384bc27b3043b3a17](b82b301bf1333e6165fac33384bc27b3043b3a17)

Bug ID #2 [Fixed]

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., 0.8.10. 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/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L2
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/IStakingPlatform.sol#L2
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/token/Token.sol#L2-L3

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 really low therefore this is only informational.

Remediation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.23 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest
The pragma is now fixed and updated to 0.8.23.
Ref: f3ac98a1899202549d0ea61dc1fe2a5e684389cc

Bug ID #3 [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/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L17

Impacts

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 behaviour.

Remediation

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

Retest

The contracts are now using Ownable2Step instead of Ownable.

Ref: 9e29524fe4eac8ade28257c8fc821882e1448979

Bug ID #4 [Fixed]

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
The following functions were affected -

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L303-L307
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L313-L315
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L321-L325

Impacts
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

Important functions are now emitting events.

Ref: ce4c5afb0d14f16f1279a3be001882a8a941f24d

Bug ID #5 [Fixed]

Functions should be declared External

Vulnerability Type

Best Practices

Severity

Informational

Description

Public functions that are never called by a contract should be declared **external** in order to conserve gas.

The following functions were declared as public but were not called anywhere in the contract, making public visibility useless.

Affected Code

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L331-L333
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L339-L341

Impacts

Smart Contracts are required to have effective Gas usage as they cost real money and each function should be monitored for the amount of gas it costs to make it gas efficient.

"**public**" functions cost more Gas than "**external**" functions.

Remediation

Use the "**external**" state visibility for functions that are never called from inside the contract.

Retest

The functions are updated to external visibility.

Ref: f833642f468bc3496a3b4059645a4ac694fd622a

Bug ID #6 [Fixed]

Incorrect Documentation

Vulnerability Type
Improper Documentation

Severity
Informational

Description
The solidity code for the withdrawResidualBalance() function contained a discrepancy between the documented behavior and the actual implementation. The documentation incorrectly stated that the function could only be called "one year after the end of the staking period" and specified that "initial stakeholders' deposits cannot be claimed." However, the implemented logic allowed withdrawal after only 90 days following the end of the staking period. This inconsistency led to misleading information about the timing constraints for withdrawing residual balances.

Vulnerable Code

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L174

Impacts
Even though there's no actual impact to user funds since only the residual balance is transferred, it may give the owners a false pretext on the number of days they have to wait before transferring all the residual tokens.

Remediation
It is recommended to update the documentation to show the actual behavior of the contracts.

Retest
The contract and documentation are updated to 15 days instead of 90 days and a year.

Ref: fe557cfd3fbf5b0aabfe654c2e4fa6542c81afad

Bug ID #7 [Fixed]

Gas Optimization in Require Statements

Vulnerability Type
Gas Optimization

Severity
Gas

Description
The **require()** 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. Once such example is given below:

Affected Code

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L93-L96

Impacts
Having longer require strings than 32 bytes cost a significant amount of gas.

Remediation
It is recommended to go through all the **require()** statements present in the contract and shorten the strings passed inside them 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
The require statement is shortened to less than 32 bytes to save gas.
Ref: ac941a08bbd52b288a03b88f8ff88c11429581f1

Bug ID #8 [Fixed]

Code Optimization by using max and min

Vulnerability Type
Gas Optimization

Severity
Gas

Description:
In Solidity contract code, optimizing expressions involving powers of 2, such as 2**256, by using the built-in type(uint256).max. Max constants can lead to improved code readability and gas efficiency. The original code utilizes 2**256 to calculate the maximum storage capacity of a uint256 data type, but this expression can be replaced with more expressive and gas-efficient alternatives.

Affected Code:

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L56

Impacts:
Using 2**256 in code can hinder readability and result in higher gas costs. Gas consumption is a crucial factor in determining the cost of executing smart contracts on the Ethereum blockchain. Optimizing such expressions contributes to more concise and understandable code, while also potentially reducing the gas fees associated with contract deployment and execution.

Remediation:
To optimize code involving powers of 2, developers should replace expressions like 2**256 with type(uint256).max for maximum values is essential to note that type(uint256).max is equivalent to 2**256 - 1.

Retest

This is fixed by using the updated type(uint256).max syntax.
Ref: [ac941a08bbd52b288a03b88f8ff88c11429581f1]

Bug ID #9 [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/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L92
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L124
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L131
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L154
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L182
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L279

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 is fixed by using != instead of >.

Ref: ac941a08bbd52b288a03b88f8ff88c11429581f1

Bug ID #10 [Fixed]

Gas Optimization for State Variables

Vulnerability Type

Gas Optimization

Severity

Gas

Description

In Solidity, the compound assignment operators '+=' and '-=' tend to consume more gas compared to the basic addition and subtraction operators ('+' and '-', respectively). As a result, when you use 'x += y', it typically incurs a higher gas cost than using 'x = x + y'.

Affected Code

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L105
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L134
- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L159

Impacts

By using basic operators or optimizing code, you can decrease the gas costs associated with smart contract transactions. This can make your application more cost-effective.

Remediation

Replace += and -= with the basic + and - operators whenever feasible. This can help reduce gas consumption, especially when working with large-scale operations.

Retest

This is fixed.

Ref: ef616214835be2bcdddc1f2f6ffc9f7a5a4ffd14

Bug ID #11 [Fixed]

Dead Code

Vulnerability Type
Code With No Effects - SWC-135

Severity
Gas

Description
Solidity is a Gas-constrained language. Having unused code incurs extra gas usage when deploying the contract.
The contract was found to be importing the file hardhat/console.sol which is not used anywhere in the code.

Vulnerable Code

- https://github.com/arcana-network/staking-platform-fixed-apy/blob/3c5f2987df27a30cfac6d746b5515a3aee9db9d7/contracts/staking/StakingPlatform.sol#L10

Impacts
Having dead and unused code in the contract leads to excessive gas usage when deploying on production chains.

Remediation
It is recommended to remove the import statement of hardhat/console.log.

Retest
This is updated and the dead code is removed.

6. Disclosure

The Reports provided by CredShields is 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.