



CredShields

# Smart Contract Audit

December 20th, 2024 • CONFIDENTIAL

## Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of LYNC WORLD Corporation between December 13th, 2024, and December 18th, 2024. A retest was performed on December 19th, 2024.

## Author

Shashank (Co-founder, CredShields) [shashank@CredShields.com](mailto:shashank@CredShields.com)

## Reviewers

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

## Prepared for

LYNC WORLD Corporation

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>1. Executive Summary</b> -----	<b>3</b>
State of Security	4
<b>2. The Methodology</b> -----	<b>5</b>
2.1 Preparation Phase	5
2.1.1 Scope	5
2.1.2 Documentation	5
2.1.3 Audit Goals	6
2.2 Retesting Phase	6
2.3 Vulnerability classification and severity	6
2.4 CredShields staff	8
<b>3. Findings Summary</b> -----	<b>9</b>
3.1 Findings Overview	9
3.1.1 Vulnerability Summary	9
3.1.2 Findings Summary	11
<b>4. Remediation Status</b> -----	<b>14</b>
<b>5. Bug Reports</b> -----	<b>15</b>
Bug ID #1 [Fixed]	15
Missing Zero Address Validations	15
Bug ID #2 [Fixed]	16
Missing Events in Important Functions	16
Bug ID # 3 [Fixed]	17
Dead Code	17
Bug ID #4 [Fixed]	18
Boolean Equality	18
Bug ID #5 [Fixed]	19
Custom error to save gas	19
Bug ID #6 [Won't fix]	20
Cheaper Inequalities in if()	20
Bug ID #7 [Fixed]	22
Splitting Require/Revert Statements	22
Bug ID #8 [Fixed]	23
Cheaper Conditional Operators	23
Bug ID #9 [Fixed]	24
Public Constants can be Private	24

Bug ID #10 [Fixed]	25
Multiplication/Division by 2 should use Bit-Shifting	25
<b>6. The Disclosure -----</b>	<b>26</b>

# 1. Executive Summary -----

LYNC WORLD Corporation engaged CredShields to perform a smart contract audit from December 13th, 2024, to December 18th, 2024. During this timeframe, 10 vulnerabilities were identified. A retest was performed on December 19th, 2024, and all the bugs have been addressed.

During the audit, 0 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "LYNC WORLD Corporation" 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
Uponly Contracts	0	0	0	2	1	7	<b>10</b>
	<b>0</b>	<b>0</b>	<b>0</b>	<b>2</b>	<b>1</b>	<b>7</b>	<b>10</b>

*Table: Vulnerabilities Per Asset in Scope*

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

## 2. The Methodology -----

LYNC WORLD Corporation engaged CredShields to perform a 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 December 13th, 2024, to December 18th, 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
<a href="https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/tree/7c5654617eafb728d3d2bac972767c0e7177ce2e">https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/tree/7c5654617eafb728d3d2bac972767c0e7177ce2e</a>

#### 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

LYNC WORLD Corporation 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](mailto: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, 10 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC   Vulnerability Type
Missing Zero Address Validations	Low	Missing Validation
Missing Events in Important Functions	Low	Missing Best Practices
Dead Code	Informational	Code With No Effects - <a href="#">SWC-135</a>
Boolean Equality	Gas	Gas Optimization
Custom error to save gas	Gas	Gas Optimization
Cheaper Inequalities in if()	Gas	Gas Optimization
Splitting Require/Revert Statements	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Public Constants can be Private	Gas	Gas Optimization

Multiplication/Division by 2 should use Bit-Shifting	Gas	Gas Optimization
--	-----	------------------

*Table: Findings in Smart Contracts*

### 3.1.2 Findings Summary

SWC ID	SWC Checklist	Test Result	Notes
SWC-100	<a href="#">Function Default Visibility</a>	Not Vulnerable	Not applicable after <b>v0.5.X</b> (Currently using solidity <b>v &gt;= 0.8.6</b> )
SWC-101	<a href="#">Integer Overflow and Underflow</a>	Not Vulnerable	The issue persists in versions before <b>v0.8.X</b> .
SWC-102	<a href="#">Outdated Compiler Version</a>	Not Vulnerable	Version 0^8.0 and above is used
SWC-103	<a href="#">Floating Pragma</a>	Not Vulnerable	Contract uses floating pragma
SWC-104	<a href="#">Unchecked Call Return Value</a>	Not Vulnerable	<b>call()</b> is not used
SWC-105	<a href="#">Unprotected Ether Withdrawal</a>	Not Vulnerable	Appropriate function modifiers and require validations are used on sensitive functions that allow token or ether withdrawal.
SWC-106	<a href="#">Unprotected SELFDESTRUCT Instruction</a>	Not Vulnerable	<b>selfdestruct()</b> is not used anywhere
SWC-107	<a href="#">Reentrancy</a>	Not Vulnerable	No notable functions were vulnerable to it.
SWC-108	<a href="#">State Variable Default Visibility</a>	Not Vulnerable	Not Vulnerable
SWC-109	<a href="#">Uninitialized Storage Pointer</a>	Not Vulnerable	Not vulnerable after compiler version, <b>v0.5.0</b>
SWC-110	<a href="#">Assert Violation</a>	Not Vulnerable	Asserts are not in use.
SWC-111	<a href="#">Use of Deprecated Solidity Functions</a>	Not Vulnerable	None of the deprecated functions like <b>block.blockhash()</b> , <b>msg.gas</b> , <b>throw</b> , <b>sha3()</b> , <b>callcode()</b> , <b>suicide()</b> are in use

SWC-112	<a href="#">Delegatecall to Untrusted Callee</a>	Not Vulnerable	Not Vulnerable.
SWC-113	<a href="#">DoS with Failed Call</a>	Not Vulnerable	No such function was found.
SWC-114	<a href="#">Transaction Order Dependence</a>	Not Vulnerable	Not Vulnerable.
SWC-115	<a href="#">Authorization through tx.origin</a>	Not Vulnerable	<code>tx.origin</code> is not used anywhere in the code
SWC-116	<a href="#">Block values as a proxy for time</a>	Not Vulnerable	<code>Block.timestamp</code> is not used
SWC-117	<a href="#">Signature Malleability</a>	Not Vulnerable	Not used anywhere
SWC-118	<a href="#">Incorrect Constructor Name</a>	Not Vulnerable	All the constructors are created using the <code>constructor</code> keyword rather than functions.
SWC-119	<a href="#">Shadowing State Variables</a>	Not Vulnerable	Not applicable as this won't work during compile time after version <code>0.6.0</code>
SWC-120	<a href="#">Weak Sources of Randomness from Chain Attributes</a>	Not Vulnerable	Random generators are not used.
SWC-121	<a href="#">Missing Protection against Signature Replay Attacks</a>	Not Vulnerable	No such scenario was found
SWC-122	<a href="#">Lack of Proper Signature Verification</a>	Not Vulnerable	Not used anywhere
SWC-123	<a href="#">Requirement Violation</a>	Not Vulnerable	Not vulnerable
SWC-124	<a href="#">Write to Arbitrary Storage Location</a>	Not Vulnerable	No such scenario was found
SWC-125	<a href="#">Incorrect Inheritance Order</a>	Not Vulnerable	No such scenario was found
SWC-126	<a href="#">Insufficient Gas Griefing</a>	Not Vulnerable	No such scenario was found
SWC-127	<a href="#">Arbitrary Jump with Function Type Variable</a>	Not Vulnerable	<code>Jump</code> is not used.

SWC-128	<a href="#">DoS With Block Gas Limit</a>	Not Vulnerable	Not Vulnerable.
SWC-129	<a href="#">Typographical Error</a>	Not Vulnerable	No such scenario was found
SWC-130	<a href="#">Right-To-Left-Override control character (U+202E)</a>	Not Vulnerable	No such scenario was found
SWC-131	<a href="#">Presence of unused variables</a>	Not Vulnerable	No such scenario was found
SWC-132	<a href="#">Unexpected Ether balance</a>	Not Vulnerable	No such scenario was found
SWC-133	<a href="#">Hash Collisions With Multiple Variable Length Arguments</a>	Not Vulnerable	<code>abi.encodePacked()</code> or other functions are not used.
SWC-134	<a href="#">Message call with hardcoded gas amount</a>	Not Vulnerable	Not used anywhere in the code
SWC-135	<a href="#">Code With No Effects</a>	Vulnerable	Bug ID #3
SWC-136	<a href="#">Unencrypted Private Data On-Chain</a>	Not Vulnerable	No such scenario was found

## 4. Remediation Status -----

LYNC WORLD Corporation is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on December 19th, 2024, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDICATION STATUS
Missing Zero Address Validations	Low	<b>Fixed</b> [ Dec 19, 2024 ]
Missing Events in Important Functions	Low	<b>Fixed</b> [ Dec 19, 2024 ]
Dead Code	Informational	<b>Fixed</b> [ Dec 19, 2024 ]
Boolean Equality	Gas	<b>Fixed</b> [ Dec 19, 2024 ]
Custom error to save gas	Gas	<b>Fixed</b> [ Dec 19, 2024 ]
Cheaper Inequalities in if()	Gas	<b>Won't Fix</b> [ Dec 19, 2024 ]
Splitting Require/Revert Statements	Gas	<b>Fixed</b> [ Dec 19, 2024 ]
Cheaper Conditional Operators	Gas	<b>Fixed</b> [ Dec 19, 2024 ]
Public Constants can be Private	Gas	<b>Fixed</b> [ Dec 19, 2024 ]
Multiplication/Division by 2 should use Bit-Shifting	Gas	<b>Fixed</b> [ Dec 19, 2024 ]

Table: Summary of findings and status of remediation

## 5. Bug Reports -----

Bug ID #1[Fixed]

### Missing Zero Address Validations

#### Vulnerability Type

Missing 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/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L41-L42>

#### 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 implementing zero address validation.



Bug ID #2 [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

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L676-L681>

### 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

This vulnerability has been fixed

Bug ID # 3 [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/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L11>

### Impacts

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 issue has been fixed.

Bug ID #4 [Fixed]

## Boolean Equality

### Vulnerability Type

Gas Optimization

### Severity

Gas

### Description

The contract was found to be equating variables with a boolean constant inside a "require()" statement which is not recommended and is unnecessary. Boolean constants can be used directly in conditionals.

### Affected Code

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L623>

### Impacts

Equating the values to boolean constants in conditions cost gas and can be used directly.

### Remediation

It is recommended to use boolean constants directly. It is not required to equate them to true or false.

### Retest

This issue is fixed by using boolean constants directly in the if condition.

Bug ID #5 [Fixed]

## 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/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpToken.sol#L51>

### Impacts

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

This issue is fixed by using custom errors.

Bug ID #6 [Won't fix]

## Cheaper Inequalities in if()

### Vulnerability Type

Gas Optimization

### Severity

Gas

### Description

The contract was found to be doing comparisons using inequalities inside the "if" statement. When inside the "if" statements, non-strict inequalities ( $\geq$ ,  $\leq$ ) are usually cheaper than the strict equalities ( $>$ ,  $<$ ).

### Affected Code

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L37>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L111-L115>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L133>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L196-L199>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L217>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L290>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L612>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L667>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L671>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L677>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L37>

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L64>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L86>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L108>

### **Impacts**

Using strict inequalities inside “if” statements costs more gas.

### **Remediation**

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

### **Retest**

Acknowledged by client.

Bug ID #7[Fixed]

## Splitting Require/Revert Statements

### Vulnerability Type

Gas Optimization

### Severity

Gas

### Description

Require/Revert statements when combined using operators in a single statement usually lead to a larger deployment gas cost but with each runtime calls, the whole thing ends up being cheaper by some gas units.

### Affected Code

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L22-L23>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L40-42>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L59-61>

### Impacts

The multiple conditions in one **require/revert** statement combine require/revert statements in a single line, increasing deployment costs and hindering code readability.

### Remediation

It is recommended to separate the **require/revert** statements with one statement/validation per line.

### Retest

This issue has been fixed.

Bug ID #8 [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/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpAMM.sol#L677>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L97>

### 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 #9 [**Fixed**]

## 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/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpToken.sol#L15-L16>
- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/PumpToken.sol#L17-18>

### Impacts

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

This issue has been fixed.

Bug ID #10 [**Fixed**]

## Multiplication/Division by 2 should use Bit-Shifting

### Vulnerability Type

Gas Optimization

### Severity

Gas

### Description

In Solidity, the EVM (Ethereum Virtual Machine) executes operations in terms of gas consumption, where gas represents the computational cost of executing smart contract functions. Multiplication and division by two can be achieved using either traditional multiplication and division operations or bitwise left shift (<<) and right shift (>>) operations, respectively. However, using bit-shifting operations is more gas-efficient than using traditional multiplication and division operations.

- $x * 2$  can be replaced with  $x \ll 1$ .
- $x / 2$  can be replaced with  $x \gg 1$ .

### Affected Code

- <https://github.com/LYNC-WORLD/uponly-lol-smart-contracts/blob/7c5654617eafb728d3d2bac972767c0e7177ce2e/contracts/libraries/PumpAMMLibrary.sol#L110>

### Impacts

Gas consumption directly affects the cost of executing smart contracts on the Ethereum blockchain. Using bit-shifting operations for multiplication and division by two reduces the gas cost from 5 to 3, leading to more cost-effective and efficient smart contract execution. This optimization is particularly relevant in scenarios where gas efficiency is crucial, such as high-frequency operations or resource-intensive contracts.

### Remediation

It is recommended to use left and right shift instead of multiplying and dividing by 2 to save some gas.

### 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

Q Audited by

