Audited by CRED SHIELDS

CredShields

Smart Contract Audit

January 29th, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of BRLA Digital LTDA between December 13th, 2024, and December 20th, 2024. A retest was performed on January 27th, 2025.

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

BRLA Digital LTDA

Table of Contents

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

BRLA Digital LTDA engaged CredShields to perform a smart contract audit from December 13th, 2024, to December 20th, 2024. During this timeframe, 14 vulnerabilities were identified. **A retest was performed on January 27th, 2025, and all the bugs have been addressed.**

During the audit, 2 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "BRLA Digital LTDA" and should be prioritized for remediation.

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

Assets in Scope	Critical	High	Medium	Low	info	Gas	Σ
CentralizedBRLAUSDSwapV2	2	0	3	4	1	4	**14**
	2	**0**	**3**	**4**	**1**	**4**	**14**

Table: Vulnerabilities Per Asset in Scope

The CredShields team conducted the security audit to focus on identifying vulnerabilities in CentralizedBRLAUSDSwapV2's scope during the testing window while abiding by the policies set forth by BRLA Digital LTDA'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 BRLA Digital LTDA'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 BRLA Digital LTDA can implement both manual and automated procedures to eliminate entire classes of vulnerabilities in the future. By taking a proactive approach, BRLA Digital LTDA can future-proof its security posture and protect its assets.

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

BRLA Digital LTDA engaged CredShields to perform a CentralizedBRLAUSDSwapV2 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 20th, 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
• **Token -** https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code • **Operator -** https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code

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

BRLA Digital LTDA 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, 14 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	SWC | Vulnerability Type
Missing access control on initializing functions	Critical	Access Control
Initializer version collision	Critical	Version Collision
WeirdERC20 can prevent approval in some cases	Medium	Improper Token Handling
Transaction DoS via permit() front-running	Medium	Denial of Service
Storage Layout Conflict	Medium	Storage Layout Conflict
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	Missing best practices
Missing Zero Address Validations	Low	Missing Input Validation
Use Ownable2Step	Low	Missing best practices

Floating and Outdated Pragma	Low	Floating Pragma
Require with Empty Message	Informational	Code optimization
Boolean Equality	Gas	Gas Optimization
Gas Optimization in Increments	Gas	Gas Optimization
Cheaper Conditional Operators	Gas	Gas Optimization
Public Constants can be Private	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	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 ----------

BRLA Digital LTDA is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on January 27th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Missing access control on initializing functions	Critical	**Fixed** [Jan 27, 2025]
Initializer version collision	Critical	**Fixed** [Jan 27, 2025]
WeirdERC20 can prevent approval in some cases	Medium	**Fixed** [Jan 27, 2025]
Transaction DoS via permit() front-running	Medium	**Won't Fix** [Jan 27, 2025]
Storage Layout Conflict	Medium	**Not Fixed** [Jan 27, 2025]
Use safeTransfer/safeTransferFrom instead of transfer/transferFrom	Low	**Fixed** [Jan 27, 2025]
Missing Zero Address Validations	Low	**Fixed** [Jan 27, 2025]
Use Ownable2Step	Low	**Not Fixed** [Jan 27, 2025]
Floating and Outdated Pragma	Low	**Fixed** [Jan 27, 2025]
Require with Empty Message	Informational	**Not Fixed** [Jan 27, 2025]
Boolean Equality	Gas	**Fixed** [Jan 27, 2025]

Gas Optimization in Increments	Gas	**Not Fixed** [Jan 27, 2025]
Cheaper Conditional Operators	Gas	**Partially Fixed** [Jan 27, 2025]
Public Constants can be Private	Gas	**Not Fixed** [Jan 27, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Missing access control on initializing functions

Vulnerability Type
Access Control

Severity
Critical

Description
The initializeV1_1 and initializeV2 functions are marked with the reinitializer modifier, allowing them to initialize or reinitialize parts of the contract's state. However, these functions lack access control, permitting any user to invoke them at any time. This oversight exposes the contract to unauthorized calls, enabling malicious actors to modify critical state variables, such as granting themselves administrative privileges or altering the swapRouter address.

Affected Code
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3549
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3558

Impacts
This vulnerability allows unauthorized users to perform reinitialization, granting themselves roles like DEFAULT_ADMIN_ROLE or UPGRADER_ROLE, which can lead to complete control over the contract.

Remediation
To address this vulnerability, both initializeV1_1 and initializeV2 should implement strict access control. This can be achieved by adding modifiers like onlyOwner() or checking for a specific role using hasRole.

Retest
This issue has been fixed.

Bug ID #2 [Fixed]

Initializer version collision

Vulnerability Type
Version Collision

Severity
Critical

Description
The initializeV1_1 and initializeV2 functions utilize the OpenZeppelin reinitializer modifier, which ensures that each initializer is called only once and in the correct sequence. However, these functions lack proper access control, allowing any user to invoke them. This makes it possible for an unauthorized actor to deliberately call a higher version initializer (e.g., initializeV2) before a lower version (e.g., initializeV1_1). Doing so causes a version collision, as the reinitializer modifier prevents a lower version from being executed after a higher version has been initialized. This misstep renders the lower version permanently uncallable, potentially leaving the contract in an inconsistent or incomplete state.

Affected Code

- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3549
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3558

Impacts
By exploiting this vulnerability, an attacker/admin can permanently block the execution of lower version initializers.

Remediation
It is recommended to add checks in each initializer to verify that all dependent initializations from previous versions have been successfully completed.

Retest
This issue has been fixed.

Bug ID #3 [Fixed]

WeirdERC20 can prevent approval in some cases

Vulnerability Type
Improper Token Handling

Severity
Medium

Description
The safeIncreaseAllowance() function in the contract attempts to increase approval for a specified amount for a spender without considering the approval behavior of certain tokens. Specifically, tokens like USDT and KNC enforce a restriction that prevents approving a non-zero amount if a prior approval already exists. This behavior is a safeguard against an ERC20 attack vector, where an attacker exploits race conditions to drain funds by front-running approval changes. The implementation in this contract does not address such token-specific restrictions, as it directly attempts to set the approval amount without revoking the previous approval or resetting it to zero first.

Affected Code

- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4013
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4038
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4064
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4093
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4130
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4167
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4193

Impacts
The failure to handle tokens with restricted approval mechanisms will result in transaction reverts when trying to approve amounts for such tokens.

Remediation
To ensure compatibility with tokens enforcing such restrictions, the contract must first revoke any existing allowance by setting it to zero before setting the desired approval amount.

Retest
This issue has been fixed.

Bug ID #4 [Won't Fix]

Transaction DoS via permit() front-running

Vulnerability Type
Denial of Service

Severity
Medium

Description
The contract allows users to approve token transfers using EIP-2612 permits. This functionality enables gasless approvals by submitting a signed message to the smart contract, which processes the permit and subsequent token transfer in a single transaction. However, the mempool visibility of the signed message introduces a front-running vulnerability. An attacker can monitor pending transactions for valid permit parameters and signatures. Once identified, the attacker can preemptively execute the permit function on the underlying token contract using the same signature. This action increments the sender's nonce, invalidating the original permit signature and causing the user's transaction to revert.

Affected Code
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3277
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3672
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3766
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3919
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4009
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4060
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4229

Impacts
This vulnerability allows attackers to grief users by invalidating their permit signatures before their transactions are processed. Although this attack does not enable theft of funds or direct financial loss.

Remediation

To mitigate this vulnerability, it is recommended to replace all instances of permit() with the trustlessPermit() function provided below. This approach ensures that even if a frontrunning attempt occurs, the function can proceed based on existing allowances, thereby preventing transaction reversion.

```
pragma solidity ^0.8.17;

import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Permit.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

library TrustlessPermit {
  function trustlessPermit(
    address token,
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
  ) internal {
    // Attempt to execute permit() to advance the nonce if possible
    try IERC20Permit(token).permit(owner, spender, value, deadline, v, r, s) {
      return;
    } catch {
      // If permit was frontrun, verify that the allowance is sufficient
      if (IERC20(token).allowance(owner, spender) >= value) {
        return;
      }
    }
    revert("Permit failure");
  }
}
```

Retest

Client Comment:

Our backend relies on the permit revert to work, because a user can approve our contract to spend his tokens. When this happens, the user's wallet has allowance.

This would mean that, if he sent a permit to our backend, even if this permit fails, with the trustlessPermit, the permit would not revert, which is a security issue, since the following would be possible:

User A approve our contract to operate on his tokens.

User B sees that our contract now has allowance over User A
User B sends a fake permit which would revert on the blockchain using our backend, asking to burn the stable coin and make a money withdraw. Since the function is trustlessPermit, it would go through.
We'd need to make some internal changes on our backend as well to support the trustlessPermit.

Bug ID #5 [Not Fixed]

Storage Layout Conflict

Vulnerability Type
Storage Layout Conflict

Severity
Medium

Description
The contracts are upgradeable but lack a storage gap. Without a storage gap, future versions of the contract may introduce new variables that overwrite storage slots in unexpected ways, potentially corrupting the contract's state. This issue arises from Solidity's use of sequential storage slots, where new variables can be written into slots reserved for other purposes in the original version of the contract.

Affected Code
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3521
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L2571

Impacts
Without a storage gap, new storage variables introduced in the Vault contract can overwrite the beginning of the storage layout, causing unexpected behavior and potentially severe vulnerabilities.

Remediation
Introduce a storage gap in the Vault contract to reserve space for future storage variables without affecting the contract's storage layout. Or you can use namespace variables.

Retest
-

Bug ID #6 [Fixed]

Use safeTransfer/safeTransferFrom instead of transfer/transferFrom

Vulnerability Type
Missing best practices

Severity
Low

Description
The transfer() and transferFrom() method is used instead of safeTransfer() and safeTransferFrom(), presumably to save gas however OpenZeppelin's documentation discourages the use of transferFrom(), use safeTransferFrom() whenever possible because safeTransferFrom auto-handles boolean return values whenever there's an error.

Affected Code
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3278

Impacts
Using safeTransferFrom has the following benefits -
- It checks the boolean return values of ERC20 operations and reverts the transaction if they fail,
- at the same time allowing you to support some non-standard ERC20 tokens that don't have boolean return values.
- It additionally provides helpers to increase or decrease an allowance, to mitigate an attack possible with vanilla approve.

Remediation
Consider using safeTransfer() and safeTransferFrom() instead of transfer() and transferFrom().

Retest
This won't be applicable since it's the token's own transferFrom function

Bug ID #7 [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://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3541
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3561

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.

Bug ID #8 [Not 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://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3521

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 behavior.

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

Retest:
-

Bug ID #9 [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., ^0.8.20, ^0.8.9. 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://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3106
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3262
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3508

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

Remediation
Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the 0.8.28 pragma version
Reference: https://swcregistry.io/docs/SWC-103

Retest

This is fixed.

Bug ID #10 [Not Fixed]

Require with Empty Message

Vulnerability Type
Code optimization

Severity
Informational

Description
During analysis; multiple **require** statements were detected with empty messages. The statement takes two parameters, and the message part is optional. This is shown to the user when and if the **require** statement evaluates to false. This message gives more information about the conditional and why it gave a false response.

Affected Code

- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3184
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3238

Impacts
Having a short descriptive message in the **require** statement gives users and developers more details as to why the conditional statement failed and helps in debugging the transactions.

Remediation
It is recommended to add a descriptive message, no longer than 32 bytes, inside the **require** statement to give more detail to the user about why the condition failed.

Retest
-

Bug ID #11 [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://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3632
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3650
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3668
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3687
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3703

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 functionality has been removed.

Bug ID#12 [Not 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**".
The contract 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.

Vulnerable Code

- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4228
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L4232

Impacts
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
-

Bug ID #13 [Partially 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://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3638
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3656
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3675
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3694
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3709
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3725
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3745
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3769
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3793
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3810

- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3841

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
Some functionality has been removed.

Bug ID #14 [Not 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://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3125
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3126
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3127
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3128
- https://polygonscan.com/address/0xe91916515def4c2f5756b8d75741cdf2bb3a3244#code#L3131
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3553
- https://polygonscan.com/address/0x81510e7bd0ca747bbde758ad1e4999a55111a497#code#L3554

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

-

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