Audited by CRED SHIELDS

CredShields

# Smart Contract Audit

October 6, 2025 • CONFIDENTIAL

**Description**

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Mandala between September 16th, 2025, and September 18th, 2025. A retest was performed on October 3rd, 2025.


**Author**

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

**Reviewers**

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


**Prepared for**

Mandala

# Table of Contents


**Table of Contents** **2**

**1. Executive Summary ——————————-** **4**

State of Security 5

**2. The Methodology ——————————-** **6**

2.1 Preparation Phase 6

2.1.1 Scope 6

2.1.2 Documentation 7

2.1.3 Audit Goals 7

2.2 Retesting Phase 7

2.3 Vulnerability classification and severity 7

2.4 CredShields staff 9

**3. Findings Summary ——————————-** **10**

3.1 Findings Overview 10

3.1.1 Vulnerability Summary 10

**4. Remediation Status ——————————-** **12**

**5. Bug Reports ——————————** **13**

Bug ID #H001 [Fixed] 13

Referrer Can Receive Double Token Rewards 13

Bug ID #L001 [Fixed] 15

Referral Rewards Default To USDT Instead Of Token 15

Bug ID #L002 [Fixed] 16

Referrer Can Lose Claimable Tokens Due To Accounting Error 16

Bug ID #L003 [Fixed] 17

Chainlink Oracle Min/Max price validation 17

Bug ID #L004 [Fixed] 18

Rounding Error Can Leave Dust Tokens Stranded In Contract 18

Bug ID #L005 [Fixed] 19

Use Ownable2Step 19

Bug ID #I001 [Fixed] 20

buyWithETH Contains Redundant Hard Cap Check 20

Bug ID #I002 [Fixed] 21

Owner Can Mistakenly Set Invalid Referee Reward Percent 21

Bug ID #G001 [Fixed] 22

Gas Optimization in Increments 22

Bug ID # G002 [Fixed] 23

Cheaper conditional operators 23
**6. The Disclosure ——————————- 25**

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

Mandala engaged CredShields to perform a smart contract audit from September 16th, 2025, to September 18th, 2025. During this timeframe, ten (10) vulnerabilities were identified. **A retest was performed on October 3rd, 2025, 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 "Mandala" 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 |
|---|---|---|---|---|---|---|---|
| Presale Smart Contract | 0 | 1 | 0 | 5 | 2 | 2 | **10** |
| | **0** | **1** | **0** | **5** | **2** | **2** | **10** |

*Table: Vulnerabilities Per Asset in Scope*

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

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

Mandala 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 September 16th, 2025, to September 18th, 2025, was agreed upon during the preparation phase.

### 2.1.1 Scope

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

| IN SCOPE ASSETS |
| --- |
| **Audited Commit:** https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253<br><br>**Retested Commit:** https://github.com/DDPidhi/ico-smart-contract/blob/e5f32b599a24238adf1fafafd34b98b539213398/mandala-presale.sol |

### 2.1.2 Documentation

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

### 2.1.3 Audit Goals

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

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

## 2.2 Retesting Phase

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

## 2.3 Vulnerability classification and severity

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

| **Overall Risk Severity** | | | | |
|---|---|---|---|---|
| **Impact** | HIGH | Medium | High | Critical |
| | MEDIUM | Low | Medium | High |
| | LOW | None | Low | Medium |
| | | LOW | MEDIUM | HIGH |
| **Likelihood** | | | | |

Overall, the categories can be defined as described below -

## 1. Informational

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

## 2. Low

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

3. **Medium**

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

4. **High**

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

5. **Critical**

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

6. **Gas**

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

## 2.4 CredShields staff

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


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


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

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

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

## 3.1 Findings Overview

### 3.1.1 Vulnerability Summary

During the security assessment, ten (10) security vulnerabilities were identified in the asset.

| VULNERABILITY TITLE | SEVERITY | SCWE \| Vulnerability Type |
|---|---|---|
| Referrer Can Receive Double Token Rewards | High | Business Logic (SC03-LogicErrors) |
| Referral Rewards Default To USDT Instead Of Token | Low | Business Logic (SC03-LogicErrors) |
| Referrer Can Lose Claimable Tokens Due To Accounting Error | Low | Business Logic (SC03-LogicErrors) |
| Chainlink Oracle Min/Max price validation | Low | Input Validation (SC04-Lack Of Input Validation) |
| Rounding Error Can Leave Dust Tokens Stranded In Contract | Low | Rounding Error |
| Use Ownable2Step | Low | Missing Best Practices |

| | | |
|---|---|---|
| buyWithETH Contains Redundant Hard Cap Check | Informational | Code Quality |
| Owner Can Mistakenly Set Invalid Referee Reward Percent | Informational | Improper Input Validation (SC04-Lack Of Input Validation) |
| Gas Optimization in Increments | Gas Optimization | Gas Optimization (SCWE-082) |
| Cheaper conditional operators | Gas Optimization | Gas Optimization (SCWE-082) |

*Table: Findings in Smart Contracts*

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

Mandala is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on October 3rd, 2025, and all the issues have been addressed.**

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

| VULNERABILITY TITLE | SEVERITY | REMEDIATION STATUS |
|---|---|---|
| Referrer Can Receive Double Token Rewards | High | Fixed [October 3, 2025] |
| Referral Rewards Default To USDT Instead Of Token | Low | Fixed [October 3, 2025] |
| Referrer Can Lose Claimable Tokens Due To Accounting Error | Low | Fixed [October 3, 2025] |
| Chainlink Oracle Min/Max price validation | Low | Fixed [October 3, 2025] |
| Rounding Error Can Leave Dust Tokens Stranded In Contract | Low | Fixed [October 3, 2025] |
| Use Ownable2Step | Low | Fixed [October 3, 2025] |
| buyWithETH Contains Redundant Hard Cap Check | Informational | Fixed [October 3, 2025] |
| Owner Can Mistakenly Set Invalid Referee Reward Percent | Informational | Fixed [October 3, 2025] |
| Gas Optimization in Increments | Gas Optimization | Fixed [October 3, 2025] |
| Cheaper conditional operators | Gas Optimization | Fixed [October 3, 2025] |

*Table: Summary of findings and status of remediation*

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

Bug ID #H001 [Fixed]

## Referrer Can Receive Double Token Rewards

**Vulnerability Type**
Business Logic (SC03-LogicErrors)

**Severity**
High

**Description**
The _recordReferral(...) function is used to attribute referral rewards whenever a buyer participates. When the reward type is USDT, the contract only updates referralRewardsUSDT[_referrer], which aligns with claim-based reward distribution. However, when the reward type is TOKEN, the contract both increments referralRewardsTOKEN[_referrer] (which is later claimable via claimReferralRewardsToken) and also converts the reward amount into tokens that are directly credited to tokensPurchased[_referrer]. This creates two parallel accounting systems for the same reward, effectively granting both a claimable balance and an immediate ownership balance. The root cause is the redundant accounting line referralRewardsTOKEN[_referrer] += referralReward; that mirrors the USDT reward flow but is not applicable for the TOKEN reward type.

**Affected Code**
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L679-L690

**Impacts**
A referrer who should only receive tokens via tokensPurchased will also accumulate a separate claimable TOKEN reward, effectively doubling the intended reward. This inflates the referral cost for the project, allows unintended extra claims through claimReferralRewardsToken, and causes discrepancies between reported allocations and actual user entitlements.

**Remediation**
Remove redundant accumulation into referralRewardsTOKEN and keep only the tokensPurchased update for TOKEN reward type.

**Retest**

This issue has been fixed by removing the token reward claim functionality. When rewardType is TOKEN, token credit is added.

Bug ID #L001 [Fixed]

## Referral Rewards Default To USDT Instead Of Token

**Vulnerability Type**
Business Logic (SC03-LogicErrors)

**Severity**
Low

**Description**
The contract defines an enum RewardType { USDT, TOKEN }, where the default zero value corresponds to USDT. In _getReferralRewardType, rounds greater than 2 default to TOKEN unless explicitly overridden in referralRewardType.
However, because Solidity assigns the first enum value (USDT) as the default, any unset mapping entry for rounds beyond 2 technically defaults to USDT. While the function tries to treat unset rounds as TOKEN, this creates an inconsistency between the declared enum ordering and the intended reward policy. The root cause is that USDT was placed as the first enum entry, making it the default, while the contract's logic expects TOKEN to be the fallback.

**Affected Code**
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L135

**Impacts**
If the owner forgets to explicitly set the reward type for a round > 2, the system may interpret it inconsistently.
For example, in round 3, a missing mapping entry evaluates to USDT by default, which may contradict the intended design of rewarding in tokens. This can cause unexpected referral reward allocations and disputes from referrers.

**Remediation**
Reorder the enum so that TOKEN is the default value, or explicitly initialize future rounds in initialize to avoid reliance on enum defaults.

**Retest**
This bug has been fixed by repositioning the enum variables.

Bug ID #L002 [Fixed]

## Referrer Can Lose Claimable Tokens Due To Accounting Error

**Vulnerability Type**
Business Logic (SC03-LogicErrors)

**Severity**
Low

**Description**
The contract tracks token allocations in two mappings: tokensPurchased and claimableTokens. In _processPurchase(...), both are updated consistently to reflect the buyer's purchased and claimable balance. However, in _recordReferral(...), when referral rewards are distributed as tokens (i.e., in rounds where RewardType.TOKEN applies), the function only increases tokensPurchased[_referrer] by the calculated bonus but does not update claimableTokens[_referrer].
As a result, the referrer's purchased tokens are recorded, but their claimable balance used for later distribution remains incomplete. The root cause is the missing update of claimableTokens during referral reward handling.

**Affected Code**

- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L656-L702

**Impacts**
If the referrer gets rewards in Tokens, his tokensPurchased increases, but his claimableTokens does not. Since tokens will be distributed after TGE based on claimableTokens, he may not receive his entitled tokens, causing a permanent loss of rewards.

**Remediation**
Ensure claimableTokens[_referrer] is updated consistently with tokensPurchased[_referrer] when token rewards are allocated in _recordReferral function.

**Retest**
This bug has been fixed by adding rewards in claimableTokens[_referrer] mapping.

Bug ID #L003 [Fixed]

## Chainlink Oracle Min/Max price validation

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

**Severity**
Low

**Description**
Chainlink has a library AggregatorV3Interface with a function called latestRoundData(). This function returns the price feed among other details for the latest round.
Chainlink aggregators have a built-in circuit breaker if the price of an asset goes outside of a predetermined price band. The result is that if an asset experiences a huge drop in value, the price of the oracle will continue to return the minPrice instead of the actual price of the asset.

**Affected Code**

- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L949-L962
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L967-L992

**Impacts**
This would allow users to store their allocations with the asset but at the wrong price.

**Remediation**
The contract should check the returned answer/price against the minPrice/maxPrice and revert if the answer is outside of the bounds.

```
if (price >= maxPrice or price <= minPrice) revert(); // eg
```

**Retest**
This bug has been fixed by validating the minPrice and maxPrice.

Bug ID #L004 [Fixed]

## Rounding Error Can Leave Dust Tokens Stranded In Contract

**Vulnerability Type**
Rounding Error

**Severity**
Low

**Description**
The withdrawRaisedFunds(...) function distributes stablecoin balances to designated wallets according to percentage allocations. The distribution is computed using integer division, meaning any fractional remainder from tokenBalance * percent / 10000 is truncated. As a result, the sum of toWalletA, toWalletB, and toTreasury may be slightly less than the total contract balance. The difference, typically 1–2 tokens at most depending on decimals, remains locked in the contract with no subsequent mechanism to recover it. The root cause is the lack of handling for rounding remainders during percentage-based division.

**Affected Code**

- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L1186-L1204

**Impacts**
Small residual token amounts ("dust") may remain in the contract, leading to accounting mismatches and minor fund loss that cannot be withdrawn for a long time.

**Remediation**
Assign the remainder to one recipient (e.g., the treasury wallet) to ensure the full balance is distributed.

Eg:

```
- uint256 toTreasury = (tokenBalance * treasuryPercent) / 10000;
+ uint256 toTreasury = tokenBalance - toWalletA - toWalletB;
```

**Retest**
This bug has been fixed by implementing the suggested fix.

Bug ID #L005 [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/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L42

**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**:
This bug has been fixed by using Ownable2StepUpgradeable instead of OwnableUpgradeable.

Bug ID #I001 [Fixed]

## buyWithETH Contains Redundant Hard Cap Check

**Vulnerability Type**
Code Quality

**Severity**
Informational

**Description**
The buyWithETH(...) function validates the presale hard cap in two places. First, it calls _processPurchase(...), which is protected by the withinHardCap modifier. This modifier ensures totalRaisedUSD + usdAmount <= hardCapUSD. In addition, buyWithETH itself includes an explicit require(totalRaisedUSD + usdAmount < hardCapUSD + 1, "Hard cap would be exceeded").
Both checks enforce the same constraint, leading to duplicated validation logic. The root cause is performing the hard cap check both in the external entry function and in the internal purchase processor.

**Affected Code**

- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L444-L447
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L558-L561

**Impacts**
There is no direct security risk. However, the redundant check increases code complexity, makes maintenance harder, and may confuse auditors or developers reviewing the logic.

**Remediation**
Remove the redundant hard cap check from buyWithETH and rely on the withinHardCap modifier applied in _processPurchase.

**Retest**
This issue has been fixed by removing the redundant checks from the functions.

Bug ID #I002 [Fixed]

## Owner Can Mistakenly Set Invalid Referee Reward Percent

**Vulnerability Type**
Improper Input Validation (SC04-Lack Of Input Validation)

**Severity**
Informational

**Description**
The initialize(...) function validates critical parameters via _validateInitParams(...) before setting state. This helper enforces constraints such as non-zero token price, valid presale timing, and referral reward percent being less than 100%. However, it does not validate the refereeRewardPercent argument, which is used in _processPurchase(...) to calculate bonus tokens for buyers when a referral is provided. As a result, an excessively large value can be set (e.g., 10001 or higher), allowing allocations that exceed the intended bounds. The root cause is the missing validation check for refereeRewardPercent in _validateInitParams(...).

**Affected Code**
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L368

**Impacts**
If the owner mistakenly sets refereeRewardPercent above 100%, buyers receive more bonus tokens than intended. For example, if Alice purchases tokens worth $100 and the percentage is set to 20000 (200%), she will receive triple the normal allocation. This inflates circulating supply and undermines presale tokenomics.

**Remediation**
Validate refereeRewardPercent in _validateInitParams to ensure it cannot exceed 100%.

**Retest**
This issue has been fixed.

Bug ID #G001 [Fixed]

## Gas Optimization in Increments

**Vulnerability Type**
Gas optimization (SCWE-082)

**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://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L909

**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**
This issue has been fixed by using ++i instead of i++.

Bug ID # G002 [Fixed]

## Cheaper conditional operators

**Vulnerability Type**
Gas Optimization (SCWE-082)

**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/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L890
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L942
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L953
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L974
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L1150
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L1266
- https://github.com/DDPidhi/ico-smart-contract/blob/73c0f8ce4ac6b53244243781fbb913461e9eb253/mandala-presale.sol#L625

**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**
This bug has been fixed by using x!=0 instead x>0.

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

**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