Audited by CRED SHIELDS

CredShields

Smart Contract Audit

February 21st, 2025 • CONFIDENTIAL

Description

This document details the process and result of the Smart Contract audit performed by CredShields Technologies PTE. LTD. on behalf of Artulabs Limited between February 3rd, 2025, and February 13th, 2025. A retest was performed on February 18th, 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

Artulabs Limited

Table of Contents

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

Artulabs Limited engaged CredShields to perform a smart contract audit from February 3rd, 2025, to February 13th, 2025. During this timeframe, 8 vulnerabilities were identified. **A retest was performed on February 18th, 2025, and all the bugs have been addressed.**

During the audit, 4 vulnerabilities were found with a severity rating of either High or Critical. These vulnerabilities represent the greatest immediate risk to "Artulabs Limited" 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
Artu Solana Contracts	1	3	3	1	0	0	**8**
	1	**3**	**3**	**1**	**0**	**0**	**8**

Table: Vulnerabilities Per Asset in Scope

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

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

Artulabs Limited engaged CredShields to perform the Artu Solana 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 February 3rd, 2025, to February 13th, 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
https://github.com/artacle/artu-smartcontract/tree/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana

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 both in-house tools and manual techniques for comprehensive Solana smart contract security auditing. The audit process primarily involves manually reviewing the program's Rust source code, adhering to best practices from the Solana security model, SPL (Solana Program Library) guidelines, and an internally developed security checklist. The team emphasizes understanding core concepts, crafting test cases, and analyzing business logic to identify potential vulnerabilities specific to the Solana runtime and its unique account-based execution model.

2.2 Retesting Phase

Artulabs Limited 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 presents the results of the security assessment. Findings are categorized by severity and grouped based on the affected program and relevant security classification. Each program section includes a summary of identified issues. The table in the executive summary provides an overview of the total number of security vulnerabilities per program, categorized by risk level.

3.1 Findings Overview

3.1.1 Vulnerability Summary

During the security assessment, 8 security vulnerabilities were identified in the asset.

VULNERABILITY TITLE	SEVERITY	Vulnerability Type
Attacker could take control of Treasury Admin	Critical	Access Control
User can exploit signature replay vulnerability to claim tokens multiple times in the Airdrop contract	High	Signature Replay Attack
Double Public Key Signing Function Oracle Attack on ed25519-dalek	High	crypto-failure
Timing Variability in curve25519-dalek Scalar Subtraction Functions (Scalar29::sub and Scalar52::sub)	High	crypto-failure
Incorrect account usage in burn_tokens() function	Medium	Misconfigured Parameters
Burn fee can be bypassed	Medium	Business Logic
Lack of two-step admin transfer in update_token_pda and update_admin	Medium	Ownership Transfer

Lack of Validation for Zero/Empty Admin Address in update_token_pda and update_admin	Low	Incorrect Validation

Table: Findings in Smart Contracts

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

Artulabs Limited is actively partnering with CredShields from this engagement to validate the discovered vulnerabilities' remediations. **A retest was performed on February 18th, 2025, and all the issues have been addressed.**

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

VULNERABILITY TITLE	SEVERITY	REMEDIATION STATUS
Attacker could take control of Treasury Admin	Critical	**Fixed** [Feb 18, 2025]
User can exploit signature replay vulnerability to claim tokens multiple times in the Airdrop contract	High	**Fixed** [Feb 18, 2025]
Double Public Key Signing Function Oracle Attack on ed25519-dalek	High	**Won't Fix** [Feb 18, 2025]
Timing Variability in curve25519-dalek Scalar Subtraction Functions (Scalar29::sub and Scalar52::sub)	High	**Won't Fix** [Feb 18, 2025]
Incorrect account usage in burn_tokens() function	Medium	**Fixed** [Feb 18, 2025]
Burn fee can be bypassed	Medium	**Won't Fix** [Feb 18, 2025]
Lack of two-step admin transfer in update_token_pda and update_admin	Medium	**Won't Fix** [Feb 18, 2025]
Lack of Validation for Zero/Empty Admin Address in update_token_pda and update_admin	Low	**Fixed** [Feb 18, 2025]

Table: Summary of findings and status of remediation

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

Bug ID #1 [Fixed]

Attacker could take control of Treasury Admin

Vulnerability Type
Access Control

Severity
Critical

Description
The initialize_treasury() function allows any user to create the treasury PDA and assign themselves as the admin. The PDA is derived using only a static seed (b"treasury"), meaning any actor can precompute the address and submit a transaction first. Since Solana does not have a traditional mempool, frontrunning occurs through validator transaction monitoring and priority fee manipulation. An attacker can detect an incoming initialization transaction, submit their own with a higher priority fee, and take control of the treasury before the intended admin.

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/airdrop/programs/airdrop/src/lib.rs#L15-L22
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L23-L75

Impacts
The attacker becomes the treasury admin, gaining full control over treasury funds and operations. This can lead to asset theft, protocol compromise, and irreversible financial loss. Since the function is only called once, there is no recovery mechanism if an unauthorized party gains control.

Remediation
Enforce access control by requiring a predefined admin public key for initialization:

```
#[account(
  init,
  payer = treasury,
```

```
  space = 8 + TreasuryPda::INIT_SPACE,
  seeds = [b"treasury"],
  bump
)]
pub treasury_pda: Account<'info, TreasuryPda>,

#[account(
  mut,
  constraint = treasury.key() == Pubkey::from_str("YOUR_ADMIN_PUBKEY").unwrap()
)]
pub treasury: Signer<'info>,
```

Alternatively, modify the PDA derivation to include the expected admin's public key in the seed:

```
seeds = [b"treasury", treasury.key().as_ref()]
```

This ensures only the intended admin can create the treasury, preventing unauthorized initialization.

Retest

This issue is fixed by validating admin public key when initializing.

Bug ID #2 [Fixed]

User can exploit signature replay vulnerability to claim tokens multiple times in the Airdrop contract

Vulnerability Type
Signature Replay Attack

Severity
High

Description
In the current implementation of the `claim_tokens` function, the smart contract verifies a signature to authenticate a claim request. The `nonce` is passed by the user as an argument, and there is no check to prevent reusing the same nonce in future claims. This lack of tracking allows an attacker to reuse a valid signature to submit the same claim multiple times, leading to potential unauthorized token claims. Due to this replay attacks can occur by re-using the same signature to claim tokens multiple times.

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/airdrop/programs/airdrop/src/lib.rs#L36

Impacts
The replay attack allows an attacker to submit multiple claims using the same signature, leading to the unauthorized minting and transfer of tokens.

Remediation
To mitigate the signature replay attack vulnerability, the following steps should be taken:

1. **Nonce Tracking**: The contract should track the `nonce` on-chain for each user to ensure that each claim is unique. The nonce should be stored per user (or transaction) and updated after each valid claim. This would prevent the reuse of the same nonce for multiple claims.

Retest
This issue has been fixed by validating if the nonce is previously used or not.

Bug ID #3 [Won't Fix]

Double Public Key Signing Function Oracle Attack on ed25519-dalek

Vulnerability Type
crypto-failure

Severity
High

Description
The Rust crate ed25519-dalek versions before 2.0 had a vulnerability that allowed an attacker to exploit the signing function as an oracle, ultimately leading to private key recovery.

The issue stems from how ed25519-dalek previously handled private and public keys as separate types, allowing them to be combined into a key pair. This design flaw permitted an attacker to request signatures on the same message with different public keys while maintaining the same random nonce (R-value). As a result, two distinct signatures could be generated where only the S value differed.

Due to the deterministic nature of Ed25519 signatures, this enabled a straightforward algebraic attack to extract the private key.

Ref: https://rustsec.org/advisories/RUSTSEC-2022-0093

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/Cargo.lock#L916-L928

Impacts
If an attacker can obtain two signatures with the same R-value, they can compute the private key mathematically.

Remediation
It is recommended to upgrade to ed25519-dalek v2.0+

Retest
Client's Comments: This is installed by Solana. Solana 2.0 is relatively new and is not that stable at the moment, so we suggest not updating that.

Bug ID #4 [Won't Fix]

Timing Variability in curve25519-dalek Scalar Subtraction Functions (Scalar29::sub and Scalar52::sub)

Vulnerability Type
crypto-failure

Severity
High

Description
The curve25519-dalek cryptographic library, widely used for secure elliptic curve operations, contains a timing vulnerability in its Scalar29::sub (32-bit) and Scalar52::sub (64-bit) functions. These functions exhibit timing variability due to a compiler-optimized branch instruction (jns on x86).

LLVM optimizes the subtraction logic by conditionally bypassing a section of code when a mask value is zero. This introduces timing differences, which could be exploited in side-channel attacks to leak sensitive scalar values, including private keys in cryptographic operations.

This vulnerability is similar to a previously discovered timing attack in the Kyber post-quantum cryptography implementation.

Ref : https://rustsec.org/advisories/RUSTSEC-2024-0344

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/Cargo.lock#L821-L833

Impacts
Malicious users can analyze execution times to infer secret scalars, potentially leading to key leakage in cryptographic applications.

Remediation
Update dependency to >=4.1.3 to mitigate the vulnerability

Retest

Client's Comments: This is installed by Solana. Solana 2.0 is relatively new and is not that stable at the moment, so we suggest not updating that.

Bug ID #5 [Fixed]

Incorrect account usage in burn_tokens() function

Vulnerability Type
Misconfigured Parameters

Severity
Medium

Description
The burn_tokens() function in the Solana program is incorrectly using the BridgeBurnTokens context struct instead of the BurnTokens context. Each function should use the appropriate account context struct that defines the correct accounts required for execution. The function is passing BridgeBurnTokens as the context, even though a separate BurnTokens struct is defined and should be used. This mistake can cause unintended behavior, as the function might not receive the correct accounts expected for token burning.

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L254

Impacts
The function may not work correctly due to differences between BridgeBurnTokens and BurnTokens. Potential security risks if BridgeBurnTokens has fields that are not intended to be accessed or modified by burn_tokens.

Remediation
It is recommended to use a BurnTokens account for the burn_tokens() function

Retest
This issue has been fixed by updating the **ctx** struct in burn_tokens() function.

Bug ID #6 [Won't Fix]

Burn fee can be bypassed

Vulnerability Type
Business Logic

Severity
Medium

Description
The Solana program provides two functions for burning tokens burn_bridge_tokens and burn_tokens both functions use the same account context (BridgeBurnTokens), meaning users can bypass the fee by calling burn_tokens directly instead of burn_bridge_tokens. This creates an inconsistency in the fee enforcement mechanism.

Affected Code
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L254-L278
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L199-L251

Impacts
Some users can exploit this loophole to burn tokens without incurring costs, gaining an advantage over those who follow the intended fee mechanism

Remediation
It is suggested to modify burn_tokens to also charge the burn fee, similar to burn_bridge_tokens.

Retest
Client's Comment: Intentionally allowed admin to burn tokens without paying any fee.

Bug ID #7 [Won't Fix]

Lack of two-step admin transfer in update_token_pda and update_admin

Vulnerability Type
Ownership Transfer

Severity
Medium

Description
The update_token_pda function allows updating the admin field of the token_pda account directly without following a two-step admin transfer process. The existing admin can be changed instantly without confirmation from the new admin.

A secure admin transfer should follow a two-step process:

1. Initiation: The current admin sets a new pending admin.
2. Confirmation: The new admin must explicitly accept the role before the change takes effect.

Affected Code

- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L304-L327
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/airdrop/programs/airdrop/src/lib.rs#L25-L33

Impacts
Accidental admin changes could lock out the intended owner. The rightful owner loses authority if a malicious or incorrect address is set as the new admin.

Remediation
To fix this issue it is recommended to modify update_token_pda to first set a pending_admin. Introduce a confirm_admin_transfer function that requires the pending_admin to confirm the transfer. Ensure that only the new admin can call confirm_admin_transfer to finalize the role change

Retest
Client's comment: Not needed as per business logic.

Bug ID #8 [Fixed]

Lack of Validation for Zero/Empty Admin Address in update_token_pda and update_admin

Vulnerability Type
Incorrect Validation

Severity
Low

Description
The update_token_pda function allows setting the admin field in the token_pda account. However, there is no validation to prevent setting the admin to a zero or empty address. This can lead to situations where:

1. Accidental Admin Lockout: If the admin is set to Pubkey::default() (which is all zeros), no one will have admin privileges, potentially breaking the contract.
2. Malicious Disruptions: An attacker with admin privileges could intentionally set the admin to an invalid address, preventing future updates or management of the contract.

Affected Code
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/token_contract/programs/token_program/src/lib.rs#L304-L327
- https://github.com/artacle/artu-smartcontract/blob/0bad32dbdd0f2d2effabb2ee264e9f40fcb80e1b/solana/airdrop/programs/airdrop/src/lib.rs#L25-L33

Impacts
If the admin address is set to an empty or zero address, no further administrative actions can be performed. If the contract relies on admin functions for upgrades, fee adjustments, or access control, it may become permanently unmanageable.

Remediation
Validate the new_admin address before updating it :

```
if let Some(admin) = new_admin {
        require!(admin != Pubkey::default(), ErrorCode::InvalidAdminAddress);
        msg!("Updating admin to: {}", admin);
```

```
        token_pda.admin = admin;
}
```

Retest

This issue has been fixed by adding a default address validation.

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