Cyberscope

Audit Report

Solex Launch

January 2024

SHA256 b0d4266355f51c7cf6a1edf098a848c5c69a2602b7190cc1d94ebd90378d2543

Audited by © cyberscope

Table of Contents

Review

Network	SOL

Audit Updates

Initial Audit	25 Jan 2024

Source Files

Filename	SHA256
programs/magnum-contrib/src/lib.rs	b0d4266355f51c7cf6a1edf098a848c5c69a2602b7190cc1d94ebd90378d2543

Overview

This document presents the overview of the audit conducted for the "Solex launch" project and the "magnum_contrib" program. The purpose of this audit is to identify and address security vulnerabilities, provide recommendations for code improvements, and ensure the robustness of the codebase. Recommendations have been provided to enhance security and functionality.

"magnum_contrib" Program Functionality

Presale Management

The program facilitates the creation and management of presale events, enabling the setting of various parameters like total tokens for sale, soft and hard caps, and contribution limits.

Contributor Interaction

Users can participate in the presale by contributing funds, and their contributions are tracked in individual contributor profiles.

Claiming Rewards

Contributors can claim their rewards, dependent on the claiming status controlled by the presale authority.

Withdrawal Functionality

The presale authority has the capability to withdraw accumulated funds from the presale event.

Administrative Controls

Functions for changing claiming status and presale variables provide administrative control over the presale's lifecycle.

Findings Breakdown

Critical	2
Medium	0
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	7	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DPIR	Duplicate Pool ID Risk	Unresolved
●	PLF	Potential Locked Funds	Unresolved
●	CRAC	Claiming Rewards Admin Control	Unresolved
●	IEM	Incomplete Error Messages	Unresolved
●	MC	Missing Check	Unresolved
●	PCR	Program Centralization Risk	Unresolved
●	RVS	Redundant Validation Structure	Unresolved
●	UEC	Unused Error Codes	Unresolved
●	USV	Unused State Variable	Unresolved

DPIR - Duplicate Pool ID Risk

Criticality	Critical
Location	lib.rs#15
Status	Unresolved

Description

`pool_id` is critical for identifying individual presale pools. This ID is provided by the admin in the `create_presale` function. However, the program lacks a mechanism to verify the uniqueness of this `pool_id`. Without such a check, there is a potential risk of creating multiple presale pools with the same `pool_id`. This scenario could lead to operational confusion, data inconsistencies, and challenges in accurately tracking and managing different presale events. The absence of unique identifiers for each pool undermines the reliability of the program and could lead to unintended financial implications.

```
pub fn create_presale(
        ctx: Context<CreatePresalePool>,
        pool_id: u64,
        token_mint: Pubkey,
        token_decimals: u8,
        total_tokens_for_sale: u64,
        soft_cap: u64,
        hard_cap: u64,
        start_timestamp: i64,
        end_timestamp: i64,
        min_contribution: u64,
        max_contribution: u64
    ) -> Result<()> {
        ...

        Ok(())
    }
```

Recommendation

To mitigate this risk, it is recommended to implement a validation process within the create_presale function to ensure the uniqueness of each `pool_id`. This safeguard will prevent the creation of multiple presale pools with identical identifiers, enhancing the program's robustness and reliability.

PLF - Potential Locked Funds

Criticality	Critical
Location	lib.rs#199
Status	Unresolved

Description

When users contribute to the pool, the `contribute` function correctly updates the `pool.total_contributions`. However, the `withdraw_sol` function, used by the pool's authority to withdraw funds, does not adjust the withdrawal amount after a withdrawal. This oversight can lead to situations where the pool. `total_contributions` reflects a higher amount than what is actually present in the pool's account. Subsequent contributions add to this incorrect total, worsening this discrepancy. In a scenario where the authority attempts to withdraw based on the inflated total contributions value, it could result in a failed transaction due to insufficient funds, effectively locking the remaining funds in the pool since the program's logic falsely indicates a higher balance.

```
pub fn withdraw_sol(ctx: Context<WithdrawAssets>) -> Result<()>
{
    let pool = &mut ctx.accounts.presale;
    let amount_to_withdraw = pool.total_contributions;

    // Send the SOL from the pool to the authority (owner of
the pool)
    let src = &mut ctx.accounts.presale.to_account_info();
    **src.try_borrow_mut_lamports()? = src
        .lamports()
        .checked_sub(amount_to_withdraw)
        .ok_or(ProgramError::InvalidArgument)?;

    let dst = &mut ctx.accounts.authority.to_account_info();
    **dst.try_borrow_mut_lamports()? = dst
        .lamports()
        .checked_add(amount_to_withdraw)
        .ok_or(ProgramError::InvalidArgument)?;

    Ok(())
}
```

Recommendation

The team is advised to implement a mechanism that correctly allows the admins to withdraw all the contributed SOL funds even if they have already withdrawn them.

CRAC - Claiming Rewards Admin Control

Criticality	Minor / Informative
Location	lib.rs#69,81,181
Status	Unresolved

Description

Claiming of rewards by contributors depends entirely on an administrative setting, the `claiming_enabled` boolean, which is controlled by the program's authority. Initially, when a presale event is created, `claiming_enabled` is set to false, meaning contributors cannot claim their rewards. The ability for contributors to claim their rewards hinges on the admin manually enabling this feature through the `change_claiming_enabled` function. After they enable the claiming of rewards, they can disable it any time. As a result, centralization risk scenario could be present, where the admin disables the claiming and withdraws all the funds contributed, leaving the contributors unable to claim their rewards.

```
pub fn change_claiming_enabled(
        ctx: Context<UpdatePresalePool>,
        claiming_enabled: bool,
    ) -> Result<()> {
        let pool = &mut ctx.accounts.presale;
        pool.claiming_enabled = claiming_enabled;
        Ok(())
    }

pub fn claim_rewards(ctx: Context<ClaimRewards>) -> Result<()>
{
...
    require!(pool.claiming_enabled,
PresaleError::ClaimingNotEnabledYet);
...
```

Recommendation

To mitigate this risk and ensure a fair and transparent process for contributors, it is recommended to implement a "finalization" step that the admin can execute in order to finish the presale process. This step could enable the claiming once, transfer all the contributed SOL funds to the admin, and prevent the user from contributing again.

IEM - Incomplete Error Messages

Criticality	Minor / Informative
Location	lib.rs#413
Status	Unresolved

Description

The `PresaleError` enumeration is used for handling various error conditions within the program. However, there is a notable inconsistency in how error messages are assigned to these error codes. Specifically, only the first error type `SoftcapIsHigherThanHardcap` is associated with a generic placeholder message, and the rest of the error types lack any descriptive messages. This approach leads to a lack of clarity, as the generic message for the first error does not provide specific information about the error.

```
#[error_code]
pub enum PresaleError {
    #[msg("placeholder error message")]
    SoftcapIsHigherThanHardcap,
    EndTimestampIsBeforeStartTimestamp,
    MaxContributionIsLessThanMinContribution,
    SaleNotStartedYet,
    SaleEnded,
    HardcapExceeded,
    NoTokensLeft,
    ContributionIsOutsidePermittedRange,
    ClaimingNotEnabledYet,
}
```

Recommendation

To improve clarity and aid in effective troubleshooting, it is recommended that each error type within the PresaleError enumeration be assigned a unique and descriptive message. These messages should accurately reflect the nature of the error, providing clear and helpful context to the users and developers.

MC - Missing Check

Criticality	Minor / Informative
Location	lib.rs#46,142
Status	Unresolved

Description

The program is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues. Specifically, the current implementation does not safeguard against scenarios where the `token_decimals` value is greater than 9.

```
let normalise_multiplier = (10u64).pow(9 - (token_decimals as u32));

let normalise_multiplier = (10u64).pow(9 - (pool.token_decimals as u32));
```

Recommendation

The team is advised to properly check the variables according to the required specifications. It is recommended to implement a validation check to ensure that the `token_decimals` value does not exceed 9.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	lib.rs#15,81,90
Status	Unresolved

Description

The programs's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

Furthermore, after the admin sets these key configurations, they have the authority to change some of these initial values that are crucial for the smooth functionality of the program.

```
pub fn create_presale(...)->Result<()>{...}

pub fn change_claiming_enabled(
    ctx: Context<UpdatePresalePool>,
    claiming_enabled: bool
) -> Result<()> {
    let pool = &mut ctx.accounts.presale;
    pool.claiming_enabled = claiming_enabled;
    Ok(())
}

pub fn change_sale_dates(
    ctx: Context<UpdatePresalePool>,
    start_timestamp: i64,
    end_timestamp: i64
) -> Result<()> {
    let pool = &mut ctx.accounts.presale;

    // Check that the start_timestamp is less than the
end_timestamp
    require!(start_timestamp < end_timestamp,
PresaleError::EndTimestampIsBeforeStartTimestamp);

    pool.start_timestamp = start_timestamp;
    pool.end_timestamp = end_timestamp;

    Ok(())
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the program's codebase itself. This approach would reduce external dependencies and enhance the program's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RVS - Redundant Validation Structure

Criticality	Minor / Informative
Location	lib.rs#388
Status	Unresolved

Description

The validation structure `PoolEconomyManagement` is defined but not utilized in any of the program's functions. This unused structure raises concerns about potential unfinished or omitted functionality within the program. The presence of this structure implies that there might have been intended features or checks related to pool economy management that are not currently implemented. This can lead to confusion.

```
#[derive(Accounts)]
pub struct PoolEconomyManagement<'info> {
    // Check that the authority is the same as the one that
initialized the pool
    #[account(mut, constraint = presale.authority.key() ==
*authority.key)]
    pub authority: Signer<'info>,
    #[account(mut)]
    pub presale: Account<'info, PresalePool>,
}
```

Recommendation

The development team should review the program to determine if the `PoolEconomyManagement` structure was intended for a specific feature that has not been implemented yet. If `PoolEconomyManagement` is not required for the current version of the program, removing it would streamline the code and eliminate any confusion regarding its purpose.

UEC - Unused Error Codes

Criticality	Minor / Informative
Location	lib.rs#408,409,421
Status	Unresolved

Description

Error codes are present that are not referenced or used anywhere in the program's logic. The presence of these unused error codes can be misleading, suggesting potential authentication checks that are not actually implemented. This could lead to a misunderstanding of the program's security features and possibly overlook actual authentication mechanisms that are in place.

```
#[error_code]
pub enum AuthenticationError {
    MismatchedContributorProfile,
    MismatchedAuthority,
}

NoTokensLeft
```

Recommendation

The development team should review the program to determine if these error codes were intended for specific authentication checks that have not been implemented.

USV - Unused State Variable

Criticality	Minor / Informative
Location	lib.rs#226
Status	Unresolved

Description

The variable `soft_cap` within the `PresalePool` struct is defined but not effectively utilized in the program's core logic, apart from the initial validation to ensure that `soft_cap` is less than `hard_cap`. This lack of use may lead to misunderstandings about the variable's role and its impact on the program's functionality. Specifically, `soft_cap` is traditionally understood in presale contexts to represent the minimum amount of funds the presale aims to raise, and its underutilization could imply missed checks or underdeveloped features related to the presale's financial goals.

```
soft_cap: u64
```

Recommendation

The development team should reassess the intended functionality associated with `soft_cap`. If it is meant to play a significant role in the presale logic, such as affecting the distribution of tokens or influencing the continuation of the presale, appropriate logic should be implemented to reflect this. Alternatively, if `soft_cap` does not serve a purpose in the current iteration of the program, it should be removed to avoid confusion and reduce unnecessary complexity in the program's state.

Summary

Solex Launch establishes a solid foundation for managing presale events within the Solana ecosystem. The audit analysis presented some critical issues.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io