Cyberscope

Audit Report

PLOUTOS

April 2024

SHA256 7a0009a4fd308de75e205dc1600376a5c9fbf0208d7a47d2efb211486a5a6fc0

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/Ploutoslabs/ploutos
Commit	134adee7c0091ef48199e750d43aca9572638bb0
Network	SOL

Audit Updates

Initial Audit	10 Apr 2024 https://github.com/cyberscope-io/audits/blob/main/pltl/v1/audit.pdf
Corrected Phase 2	24 Apr 2024

Source Files

Filename	SHA256
programs/ploutoslabs/src/lib.rs	7a0009a4fd308de75e205dc1600376a5c9fbf0208d7a47d2efb211486a5a6fc0

Overview

The `ploutoslabs` program of the `PLOUTOS` project is designed to manage token distributions, that include mechanisms such as initialization, claiming airdrops, and handling allocations.

Initialization

The `initialize` function sets up the initial state of the program, configuring key parameters such as fee receiver, fee amount, token mint, reserve amount, and airdrop amount. It also marks the program as initialized to prevent re-initialization.

Claim Airdrop

The `claim_airdrop` function allows users who have not previously claimed an airdrop to do so. It checks if the airdrop has already been claimed and verifies the correctness of program-derived addresses before processing the claim. This function also handles the transfer of a fee and a percentage of the claimed tokens to the user's account.

Token Allocation Adjustment

The `increase_allocation` function is designed to adjust the token allocation for a user. By invoking this function, the program increases the recorded total allocation for a user by a specified additional amount.

End Allocation Functionality

The `end_allocation` function allows the program owner to permanently disable the allocation adjustments. Once this function is executed, further increases in allocation are blocked.

Token Unlocking Functionality

The `unlock_allocation` function permits users to unlock a portion of their tokens after meeting a predefined unlock period. It verifies that the necessary time has elapsed since the tokens were claimed before allowing the unlocking process.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ESA	Excessive Space Allocation	Unresolved
●	ICDI	Inappropriate Claim Data Initialization	Unresolved
●	IEM	Incomplete Error Messages	Unresolved
●	IRMI	Incomplete Referral Mechanism Implementation	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	PCR	Program Centralization Risk	Unresolved
●	URU	Unnecessary Reference Usage	Unresolved
●	UEC	Unused Error Code	Unresolved

ESA - Excessive Space Allocation

Criticality	Minor / Informative
Location	lib.rs#251,253,272
Status	Unresolved

Description

The program allocates more space than is necessary for the `PloutosData` and `UserData` accounts. Specifically, `PloutosData` is allocated 64*7 = 448 bytes and `UserData` is allocated 72 bytes (8 + 64), which exceeds the actual storage requirements for the data these accounts are intended to hold. Over-allocation of space leads to increased costs for account creation, as well as inefficient use of blockchain storage resources. Such practices can result in unnecessarily high transaction fees.

```
#[account(init, payer=user, space=64*7,
seeds=[b"PLOUTOS_ROOT".as_ref(), user.key().as_ref()], bump)]
pub data: Account<'info, PloutosData>,
#[account(init, payer = user, space = 8 + 64, seeds =
[b"POUTOS_USER_DATA", user.key().as_ref()], bump)]
pub user_data: Account<'info, UserData>,

#[account(init, payer = user, space = 8 + 64, seeds =
[b"POUTOS_USER_DATA", user.key().as_ref()], bump)]
pub user_data: Account<'info, UserData>,
```

Recommendation

It is recommended to review and adjust the space allocations for both `PloutosData` and `UserData` accounts to closely match their actual data storage needs.

ICDI - Inappropriate Claim Data Initialization

Criticality	Minor / Informative
Location	lib.rs#43,44
Status	Unresolved

Description

The `initialize` function sets the claimed status to true and `claim_timestamp` to the current blockchain timestamp for the user who calls this function. This implementation is unconventional as it automatically marks the initial user as having claimed an airdrop or similar benefit without an actual claim process taking place. Such a practice can lead to confusion, misrepresentation of the user's actual interaction with the program, and potential manipulation of the program's intended use.

```
pub fn initialize(ctx: Context<Initialize>, fee_receiver:
Pubkey, fee_amount: u64, token_mint: Pubkey,
          ...
        data.admin_wallet = fee_receiver;
        data.fee_amount = fee_amount;
        data.token_mint = token_mint;
        data.reserve_amount = reserve_amount;
        data.airdrop_amount = airdrop_amount;
        data.program_token_account =
ctx.accounts.program_token_account.key();

        let clock = Clock::get()?;
        ctx.accounts.user_data.claim_timestamp =
clock.unix_timestamp;
        ctx.accounts.user_data.claimed = true;

        let claim_amount = airdrop_amount;
        ctx.accounts.user_data.total_allocation = claim_amount;

        data.initialized = true;
        data.allocation_enabled = true;
        Ok(())
    }
```

Recommendation

It is recommended to revise the `initialize` function to remove the automatic setting of `claimed` to true and the initialization of `claim_timestamp` unless there is a clear and justifiable reason aligned with the program’s operational requirements. If these fields are necessary, they should be clearly documented. This change will enhance the clarity and integrity of the program's operations, ensuring that all interactions, are explicitly initiated by the users' actions.

IEM - Incomplete Error Messages

Criticality	Minor / Informative
Location	lib.rs#368
Status	Unresolved

Description

The `ErrorCode` 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, some 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 ErrorCode {
    #[msg("The program has already been initialized")]
    AlreadyInitialized,
    #[msg("Invalid token mint")]
    MintMismatch,
    #[msg("PDA mismatch")]
    PdaMismatch,
    #[msg("The airdrop has already been claimed by this user")]
    AirdropAlreadyClaimed,
    #[msg("The unlock period has not yet been met")]
    UnlockPeriodNotMet,
    #[msg("All allocation has been claimed")]
    ClaimCompleted,
    #[msg("Allocation has ended")]
    AllocationNotEnabled,
    #[msg("You don' the right to perform this zaction")]
    Unauthorized,
    InvalidTokenAccount,
    InvalidTokenAccountOwner,
    InsufficientFunds,
}
```

Recommendation

To improve clarity and aid in effective troubleshooting, it is recommended that each error type within the `ErrorCode` 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.

IRMI - Incomplete Referral Mechanism Implementation

Criticality	Minor / Informative
Location	lib.rs#128,129,275
Status	Unresolved

Description

The program contains provisions for tracking referrals through `referral_count` and `upline_data` fields within the `UserData` struct. However, the program does not provide a clear or explicit method for initializing or updating `upline_data`, nor does it define how users are linked in referral relationships. This omission could lead to inconsistencies and potential misuse of the referral system, as there is no safeguarded process to ensure that referrals are tracked correctly and that the `referral_count` is used effectively within the platform's operational logic.

```
// update upline
ctx.accounts.upline_data.referral_count += 1;
ctx.accounts.upline_data.total_allocation += claim_amount/10;


#[account(mut)]
pub upline_data: Account<'info, UserData>,
```

Recommendation

It is recommended to clearly define the referral system. Additionally, reviewing and ensuring that the referral system's implementation aligns with the platform's business goals will be crucial. These steps will help to enhance the functionality and reliability of the referral system, providing clear benefits and incentives for users to engage with the platform's referral program.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	lib.rs#12,147
Status	Unresolved

Description

The program performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, to track and monitor the activity on the program. Without these events, it may be difficult for external parties to accurately determine the current state of the program.

```
pub fn initialize(ctx: Context<Initialize>, fee_receiver:
Pubkey, fee_amount: u64, token_mint: Pubkey,
        ...

    data.admin_wallet = fee_receiver;
    data.fee_amount = fee_amount;
    data.token_mint = token_mint;
    data.reserve_amount = reserve_amount;
    data.airdrop_amount = airdrop_amount;
    data.program_token_account =
ctx.accounts.program_token_account.key();

    let clock = Clock::get()?;
    ctx.accounts.user_data.claim_timestamp =
clock.unix_timestamp;
    ctx.accounts.user_data.claimed = true;

    let claim_amount = airdrop_amount;
    ctx.accounts.user_data.total_allocation = claim_amount;

    data.initialized = true;
    data.allocation_enabled = true;
    Ok(())
}

...
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the program. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the program will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	lib.rs#12,147,157
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. Specifically, the admin has to properly set these key configurations and can make changes to them, that are crucial for the smooth functionality of the program.

```
pub fn initialize(ctx: Context<Initialize>, fee_receiver:
Pubkey, fee_amount: u64, token_mint: Pubkey,
    reserve_amount: u64, airdrop_amount: u64) -> Result<()> {
    ...
}

pub fn increase_allocation(ctx: Context<IncreaseAllocation>,
additional_amount: u64) -> Result<()> {
    let user_data = &mut ctx.accounts.user_data;
    user_data.total_allocation += additional_amount;
    Ok(())
}

pub fn end_allocation(ctx: Context<EndAllocation>) ->
Result<()> {
```

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.

URU - Unnecessary Reference Usage

Criticality	Minor / Informative
Location	lib.rs#27
Status	Unresolved

Description

In the `initialize` function, there is an instance where the code inefficiently handles the comparison of two references. Specifically, the comparison between `program_token_account.owner` and `data.to_account_info().key()` is performed using direct references for both operands. This method involves needlessly taken references of both operands, which can be optimized. Such practices, while minor, can affect the clarity and perceived quality of the codebase.

```
require!(
    &ctx.accounts.program_token_account.owner ==
&data.to_account_info().key(),
    ErrorCode::InvalidTokenAccountOwner
);
```

Recommendation

It is recommended to simplify the comparison operations by removing the unnecessary references. This change will not only adhere to Rust's idiomatic practices but also enhance the readability and maintainability of the code. Adjusting this would involve using the actual values directly in the comparison rather than their references, which aligns with Rust's efficiency guidelines.

UEC - Unused Error Code

Criticality	Minor / Informative
Location	lib.rs#372
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.

```
#[msg("Invalid token mint")]
MintMismatch,
```

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.

Summary

The Ploutoslabs program facilitates token allocation, claims, and distribution management on the Solana blockchain. This audit investigates security issues, business logic concerns and potential improvements.

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