Cyberscope

Audit Report

PLOUTOS

April 2024

SHA256 07e08e629b3e413842964f2c0ae92a968cacfc4fccb1edb15ce43b8cbe86f24b

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/Ploutoslabs/ploutos
Commit	1312e1866e7f2e6ee7be4299d55ebd34b93cfa7b
Network	SOL

Audit Updates

Initial Audit	10 Apr 2024

Source Files

Filename	SHA256
programs/ploutoslabs/src/lib.rs	07e08e629b3e413842964f2c0ae92a968cacfc4fccb1edb15ce43b8cbe86f24b

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.

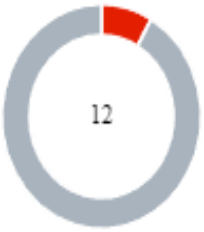
Token Allocation Adjustment

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.

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	1
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	PTAS	Program Token Account Setup	Unresolved
	ESA	Excessive Space Allocation	Unresolved
	IAUC	Inadequate Allocation Unlocking Controls	Unresolved
	ICEH	Inadequate Clock Error Handing	Unresolved
	ICDI	Inappropriate Claim Data Initialization	Unresolved
	IRMI	Incomplete Referral Mechanism Implementation	Unresolved
	MEE	Missing Events Emission	Unresolved
	PCR	Program Centralization Risk	Unresolved
	RACM	Restrictive Airdrop Claim Mechanism	Unresolved
	UB	Unnecessary Borrowing	Unresolved
	UEC	Unused Error Code	Unresolved
	USV	Unused State Variable	Unresolved

PTAS - Program Token Account Setup

Criticality	Critical
Location	lib.rs#12,34,233
Status	Unresolved

Description

The `PloutosData` structure references a `program_token_account` intended to manage token transactions for airdrops and related operations. However, the program does not include explicit instructions or processes to initialize this token account or verify its correct setup. Furthermore, this is an external configuration that is considered untrusted. The lack of these crucial steps could lead to operational failures, as the program assumes the account is correctly set up and possesses the necessary tokens to conduct transactions without verification.

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

pub fn claim_airdrop(ctx: Context<ClaimAirdrop>) -> Result<()> {
    ...
}

#[account]
pub struct PloutosData {
    pub admin_wallet: Pubkey,
    pub fee_amount: u64,
    pub token_mint: Pubkey,
    pub program_token_account: Pubkey,
    pub reserve_amount: u64,
    pub airdrop_amount: u64,
    pub initialized: bool,
}
```

Recommendation

To ensure the program functions as intended and to mitigate potential operational risks, it is crucial to establish and verify the `program_token_account`. The process should include explicitly creating or validating the existence of the token account ensuring that the token account is correctly set up, and confirming that the token account is sufficiently funded to meet the operational demands.

ESA - Excessive Space Allocation

Criticality	Minor / Informative
Location	lib.rs#168,170,187
Status	Unresolved

Description

The program allocates more space than is necessary for the `PloutosData` and `UserData` accounts. Specifically, `PloutosData` is allocated 9000 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=9000,
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.

IAUC - Inadequate Allocation Unlocking Controls

Criticality	Minor / Informative
Location	lib.rs#109
Status	Unresolved

Description

The `unlock_allocation` function allows users to unlock a fraction of their allocated tokens repeatedly without a mechanism to prevent the total unlocked amount from exceeding the total allocated amount. This absence of controls could potentially allow users to claim more than their intended share, leading to discrepancies in token distribution and unintended depletion of the token reserve. Such an oversight can result in implications for the integrity and fairness of the token distribution mechanism.

```
pub fn unlock_allocation(ctx: Context<UnlockAllocation>) ->
Result<()> {
    let user_data = &mut ctx.accounts.user_data;
    let airdrop_data = &ctx.accounts.airdrop_data;
    let clock = Clock::get()?;
    let current_timestamp = clock.unix_timestamp;

    // Ensure the unlock period has been met
    require!(
        current_timestamp - user_data.claim_timestamp >= 30 *
86400,
        ErrorCode::UnlockPeriodNotMet
    );

    ...

    let cpi_program = ctx.accounts.token_program.to_account_info();
    let cpi_ctx = CpiContext::new_with_signer(cpi_program,
cpi_accounts, signer_seeds);
    token::transfer(cpi_ctx, allocation_to_unlock)?;


    // Update the user data
    user_data.claim_timestamp = current_timestamp;
    user_data.total_claimed += allocation_to_unlock;

    Ok(())
}
```

Recommendation

It is recommended to introduce checks to ensure that the total amount unlocked by any user does not surpass their initially allocated amount. Implementing this check will prevent any user from exploiting the system to claim more than what they are entitled to, thus preserving the integrity of the token distribution process. Ensuring that these controls are in place is crucial for maintaining user trust and the overall security of the platform.

ICEH - Inadequate Clock Error Handing

Criticality	Minor / Informative
Location	lib.rs#23,51
Status	Unresolved

Description

Parts of the code employ the `Clock::get().unwrap();` method to retrieve the current blockchain time. This method of handling the system clock is unsafe as it uses `unwrap()`, which forces a panic if the call fails. Panicking can lead to unintended behavior and disrupt the contract's normal operations. It is crucial to ensure that all operations can handle potential errors gracefully to maintain the integrity and reliability of the program.

```
let clock = Clock::get().unwrap();
```

Recommendation

It is advised to replace the `unwrap()` usage with proper error handling mechanisms. A more robust approach would be to utilize Rust's error propagation feature by replacing `unwrap()` with the ? operator. This change would allow the function to return an error in a controlled manner if the clock data cannot be fetched, rather than causing the program to panic.

ICDI - Inappropriate Claim Data Initialization

Criticality	Minor / Informative
Location	lib.rs#52,53
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, reserve_amount: u64,
airdrop_amount: u64) -> Result<()> {
    let data = &mut ctx.accounts.data;
    if data.initialized {
        return err!(ErrorCode::AlreadyInitialized);
    }
    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;

    let clock = Clock::get().unwrap();
    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;
    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.

IRMI - Incomplete Referral Mechanism Implementation

Criticality	Minor / Informative
Location	lib.rs#97,98,190
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#12,34,103,109
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, reserve_amount: u64,
airdrop_amount: u64) -> Result<()> {
    let data = &mut ctx.accounts.data;
    if data.initialized {
        return err!(ErrorCode::AlreadyInitialized);
    }
    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;

    let clock = Clock::get().unwrap();
    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;
    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#12,103
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, 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(())
}
```

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.

RACM - Restrictive Airdrop Claim Mechanism

Criticality	Minor / Informative
Location	lib.rs#179
Status	Unresolved

Description

The `claim_airdrop` function restricts users to claim only 1% of their allocated airdrop amount. The function is designed such that once a user claims this 1%, the claimed flag in their `UserData` is set to true, effectively preventing any further claims by this user. This logic is enforced by a check at the start of the function that exits the function if the claimed flag is already set to true.

```
pub fn claim_airdrop(ctx: Context<ClaimAirdrop>) -> Result<()> {
    if ctx.accounts.user_data.claimed {
        return Err(ErrorCode::AirdropAlreadyClaimed.into());
    }
    ...
}
```

Recommendation

It is recommended to revise the logic of `claim_airdrop` function to determine if users should be allowed to claim only that specific amount once. Furthermore, if this logic is intended, it would be prudent to clearly document the intention behind the claim limits. This change will align the airdrop functionality more closely with standard practices, enhancing user engagement and satisfaction while ensuring fair and complete distribution of the tokens as intended.

UB - Unnecessary Borrowing

Criticality	Minor / Informative
Location	lib.rs#130
Status	Unresolved

Description

In the `unlock_allocation` function, there is an instance of unnecessary borrowing. This occurs during the derivation of the program-derived address (PDA) where the program ID is unnecessarily borrowed. This represents a non-optimal coding practice that could affect code clarity and maintainability. Moreover, excessive unnecessary borrowing could potentially impact the performance of the contract, albeit minimally, due to inefficiencies in memory usage and handling.

```
let (data_account_pda, bump_seed) = Pubkey::find_program_address(
    &[
        b"PLOUTOS_ROOT".as_ref(),
        airdrop_data.admin_wallet.as_ref()
    ],
    &ctx.program_id,
);
```

Recommendation

It is advisable to review the code to ensure that borrowing is used appropriately and only when required. Reducing unnecessary borrowing can enhance the readability and efficiency of the code. Conducting a thorough review of the program with a focus on optimizing Rust specific language features such as borrowing and ownership will ensure that the program adheres to best coding practices.

UEC - Unused Error Code

Criticality	Minor / Informative
Location	lib.rs#253
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.

```
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,
}
```

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#20,234
Status	Unresolved

Description

In the `PloutosData` struct, the `reserve_amount` variable is initialized but never used within the program's functions. This redundant variable does not contribute to any program operations, which could lead to confusion and unnecessary data storage without providing any functional benefit.

```
#[account]
pub struct PloutosData {
    pub admin_wallet: Pubkey,
    pub fee_amount: u64,
    pub token_mint: Pubkey,
    pub program_token_account: Pubkey,
    pub reserve_amount: u64,
    pub airdrop_amount: u64,
    pub initialized: bool,
}
```

Recommendation

It is advisable to review the necessity of the `reserve_amount` within the `PloutosData` struct. If the variable is not required, removing it could simplify the program's structure and reduce the storage costs. Streamlining the program to eliminate unused variables will enhance clarity and efficiency, ensuring that all components of the program serve a clear purpose.

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