Cyberscope

Audit Report

TAOlie Staking Contract

August 2024

Repository	https://github.com/taolie-ai/staking-contract
Commit	537aebecd1086ee94654a94b6ae5c4c14cf55572
Audited by	© cyberscope

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/taolie-ai/staking-contract
Commit	537aebecd1086ee94654a94b6ae5c4c14cf55572
Network	SOL

Audit Updates

Initial Audit	07 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/taoliedepin/v1/audit.pdf
Corrected Phase 2	13 Aug 2024

Source Files

Filename	SHA256
errors.rs	eb846da50494e874b31ce909ab84e8f1470ecb9aa8d59bf22b3657c0a1d64834
lib.rs	470ca65d8d7a58c5f1c445f52e9a545f1c3ce5b43e6bd6b8d2e1b0af45e38c69
utils.rs	c31aad815ee9737cfd4b3593d89a7b41d13cfe2160435109cccdb872921b4f0c
instructions/changeapy.rs	8c2a6d09baf74a03a7cec5e80186cf421958ad08c3f87d782f7e24bd47264989
instructions/claim.rs	2cf52b042802e68c2f777150e6bb6ad37f607c48ed1f457b18b27768553f78aa
instructions/deposit.rs	07d93c76b8e9d7b186817a3bf6ec7435adc531997d12e9a61ee3a1f11f5bc693

instructions/initialize.rs	0746ef34936227a6155acea00abff8db9e3bc1a1304a6c57317ecc3c424c9e48
instructions/mod.rs	8a43f5f15ef3fc05f58463f46b0366d0f25f3e97b75ba82e73141dbbf16cb9b6
instructions/withdraw.rs	753ca8564a8a0170bde12916ed1d2a1f52d6734b6863fae6315fe773382c305a
state/apyconf.rs	978f2d939ebbd092fc8cb13716c98bc3f7c2ad7d917c3ad057330df81b4990f8
state/base.rs	bb6121e8f64596609310c93e67fe42d7226c338a9262faf579174eef2d2d3505
state/mod.rs	ffd28e8784c77d8170493fd23f50c96ac307f611549f22d33f95fe9222690625
state/reward.rs	ac024d030b12dae12ea7f4d2274bb13670192aad1e6b034e97ed1b253692685f
state/stake.rs	33b5e0dbce29fe9ec3b450402d5bd1c08b7360daa9d5541746972de62eb06373
state/user.rs	c116b4d5433a820f91568dafecc178fc9358976c748e4396d4a8d2604abdcd43

Overview

The TAOlie Staking Contract implements a staking and rewards system for two specific tokens: Taolie and Depin. This project provides functionalities for initializing the staking system, allowing users to deposit tokens for staking, claim rewards based on the amount of staked tokens and the duration of staking, withdraw staked tokens, and adjust the annual percentage yield (APY) configurations.

Initialization Function

The initialize function sets up the initial state for the staking system. It verifies that the depin_mint authority matches the signer of the transaction, ensuring proper authorization. The function then initializes various program accounts, including the taolie_stake, apy_pda, reward_pda, and base_account. Initial values for APY configurations are set, and the lock time for the APY is recorded. The base_account is assigned the provided admin address, establishing administrative control over future changes to the APY settings.

Deposit Function

The deposit function allows users to deposit their Taolie tokens into the staking system. It first ensures that the deposit amount meets the minimum required threshold. The function calculates any pending rewards based on the user's previous deposits and updates their claimable and claimed amounts accordingly. The user's staked amount and the total staked amount in the system are then updated to reflect the new deposit. Additionally, the user's lock time and last deposit timestamp are recorded. Finally, the function transfers the deposited tokens from the user's account to the staking account.

Withdraw Function

The withdraw function enables users to withdraw their staked tokens, provided they meet the lock time requirements. It checks if the user has sufficient staked tokens to cover the withdrawal amount and ensures the requested withdrawal does not exceed the staked amount. The function also calculates and updates any pending rewards before proceeding with the withdrawal. If the withdrawal is allowed, it updates the user's staked amount and

the total staked amount in the system. The tokens are then transferred from the staking account back to the user's account.

Claim Function

The claim function allows users to claim their accrued rewards based on the amount of tokens they have staked and the duration of staking. The function calculates the total claimable rewards and verifies that there are sufficient tokens in the reward account to cover the claim. Once verified, the function updates the user's claimed and claimable amounts and resets their last claim timestamp. The claimed rewards are then transferred from the reward account to the user's account.

Change APY Function

The changeapy function permits the admin to update the APY configurations for the staking system. It checks if the current time is past the lock time for the APY settings and ensures that the caller is the designated admin. If these conditions are met, the function updates the APY settings and sets a new lock time to prevent further changes until the lock time has expired.

Calculate Rewards Function

The calculate_rewards function is responsible for calculating the rewards based on the amount of tokens staked, the duration of staking, and the specified lock time. It determines the applicable APY rate based on whether the current time is past the APY lock time and the user's lock time. The function then calculates the rewards proportionally, considering the time elapsed since the last claim.

Findings Breakdown

Critical	0
Medium	1
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	5	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	PRR	Potential Reduced Rewards	Unresolved
Minor / Informative	CDI	Constants Definition Inefficiency	Unresolved
Minor / Informative	LFRV	Large Function Return Variant	Unresolved
Minor / Informative	PCR	Program Centralization Risk	Unresolved
Minor / Informative	SPMI	Suboptimal Pattern Matching Implementation	Unresolved
Minor / Informative	UTC	Unnecessary Type Conversion	Unresolved

PRR - Potential Reduced Rewards

Criticality	Medium
Location	instructions/deposit.rs#L50 instructions/claim.rs#L37 utils.rs#L18
Status	Unresolved

Description

The current implementation allows the `locktime` to be reset each time a user makes a deposit. While the system prevents the actual unlock timestamp from being shortened by comparing and setting the maximum value, the `locktime` used for reward calculation is still updated with each deposit. This means that if a user sets a lower locktime during a subsequent deposit, their rewards will be calculated based on this lower value. This behavior can lead to users receiving fewer rewards than they are entitled to.

```
let amount = calculate_rewards(user.deposted_amount,
user.last_claim_timestamp, current_timestamp, user.locktime, apy_conf);

let apy_to_use = if apy_available {
    match locktime {
        0 => apy.apy_day_1,
        1 => apy.apy_month_1,
        3 => apy.apy_month_3,
        6 => apy.apy_month_6,
        12 => apy.apy_year_1,
        _ => {
            msg!("Unknown timelock");
            return 0;
        }
    }
}
...
```

Recommendation

It is recommended to ensure that the locktime used in reward calculations reflects the longest commitment period made by the user. This could be achieved by either retaining the maximum locktime across all deposits or tracking the locktime for each deposit separately to maintain consistency and fairness in reward calculations. Additionally, the system should clearly communicate to users how the locktime affects their rewards to avoid unintended consequences. This approach would prevent users from unintentionally reducing their rewards and ensure the integrity of the staking protocol.

CDI - Constants Definition Inefficiency

Criticality	Minor / Informative
Location	utils.rs#L9
Status	Unresolved

Description

The code contains an unnecessary mathematical operation in the definition of the `MINIMUM_DEPOSIT` constant, where the value is multiplied by one. This operation does not affect the final value but introduces unnecessary complexity and may lead to confusion. Such operations do not contribute to the functionality of the code and can be considered redundant.

```
pub const MINIMUM_DEPOSIT: u64 = 1 * 1_000_000;
```

Recommendation

To enhance code clarity and efficiency, it is recommended to remove unnecessary operations in constant definitions. The value should be defined directly without any extraneous operations. This approach simplifies the code and aligns with best practices for maintaining clean and understandable codebases. By ensuring that constants are defined in the most straightforward manner, the code will be more maintainable and less prone to misunderstandings.

LFRV - Large Function Return Variant

Criticality	Minor / Informative
Location	instructions/initialize.rs#L14 instructions/deposit.rs#L15 instructions/claim.rs#L11 instructions/withdraw.rs#L13 instructions/changeapy.rs#L7
Status	Unresolved

Description

The program includes multiple functions where the `Err` -variant returned is very large, specifically in the `Result` type. When the `Err` -variant is too large, it can lead to inefficiencies in memory usage and potentially impact the performance of the program. This can be particularly problematic in a program environment, where efficient resource utilization is crucial for maintaining optimal performance and minimizing transaction costs. The large size of the `Err` -variant results from the way errors are structured, possibly including large elements that could be more efficiently managed.

```
pub fn initialize(ctx: Context<Initialize>, admin: Pubkey) ->
Result<()> {

pub fn deposit(ctx: Context<Deposit>, deposit_amount: u64,
locktime: u64) -> Result<()> {

pub fn claim(ctx: Context<Claim>, reward_bump:u8) -> Result<()>
{

pub fn withdraw(ctx: Context<Withdraw>, stake_bump: u8,
withdraw_amount: u64) -> Result<()> {

pub fn changeapy(ctx: Context<Apy>, apy_new: ApyConf) -> Result
<()> {
```

Recommendation

To address this issue, it is advisable to reduce the size of the Err-variant returned by these functions. This can be achieved by restructuring the error types to be more compact. For instance, consider boxing large elements within the error type or replacing the current error type with a boxed version to minimize the memory footprint. This approach will ensure that the program remains efficient in terms of memory usage and performance. By optimizing the error handling mechanism, the program can maintain high performance and reliability, even when handling errors, thereby improving the overall robustness and efficiency of the program.

PCR - Program Centralization Risk

Criticality	Minor / Informative
Location	instructions/initialize.rs#L14,81 instructions/changeapy.rs#L7 instructions/claim.rs#L50
Status	Unresolved

Description

The program'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 program's functionality and behavior are heavily dependent on external parameters or configurations. Specifically, the administrative control over the `initialize` and `changeapy` functions introduces a significant centralization risk. These functions must be executed by a specific authorized account to set and update critical parameters within the protocol. If this account is compromised, it could lead to unauthorized changes in the APY settings or reinitialization of the contract, potentially affecting all users.

```
pub fn initialize(ctx: Context<Initialize>, admin: Pubkey) ->
Result<()> {

  if let COption::Some(mint_authority) =
ctx.accounts.depin_mint.mint_authority {
    require_keys_eq!(mint_authority, ctx.accounts.from.key(),
InitializeError::AuthorityError);
  } else {
      return Err(InitializeError::AuthorityError.into());
  }

  let stake = &mut ctx.accounts.taolie_stake;
  let apy = &mut ctx.accounts.apy_pda;
  let base_account = &mut ctx.accounts.base_account;

  if stake.is_initialized {
    return Err(InitializeError::AlreadyInitializedError.into());
  }

  stake.total_amount = 0;
  apy.apy_day_1 = 6;
  apy.apy_month_1 = 12;
  apy.apy_month_3 = 20;
  apy.apy_month_6 = 35;
  apy.apy_year_1 = 100;

  apy.last_apy_day_1 = 6;
  apy.last_apy_month_1 = 12;
  apy.last_apy_month_3 = 20;
  apy.last_apy_month_6 = 35;
  apy.last_apy_year_1 = 100;

  apy.lock_time = Clock::get()?.unix_timestamp;

  base_account.admin = admin;

  stake.is_initialized = true;

  Ok(())
}

pub fn calculate_rewards(
    amount: u64,
    last_claim_timestamp: i64,
    current_timestamp: i64,
    locktime: u64,
    apy: ApyConf,
) -> u64 {
    let time_diff: u64 = (current_timestamp - last_claim_timestamp)
        .try_into()
```

```
            .unwrap();
        let apy_available: bool = current_timestamp > apy.lock_time;

        let apy_to_use = if apy_available {
            match locktime {
                0 => apy.apy_day_1,
                1 => apy.apy_month_1,
                3 => apy.apy_month_3,
                6 => apy.apy_month_6,
                12 => apy.apy_year_1,
                _ => {
                    msg!("Unknown timelock");
                    return 0;
                }
            }
        } else {
            match locktime {
                0 => apy.last_apy_day_1,
                1 => apy.last_apy_month_1,
                3 => apy.last_apy_month_3,
                6 => apy.last_apy_month_6,
                12 => apy.last_apy_year_1,
                _ => {
                    msg!("Unknown timelock");
                    return 0;
                }
            }
        };

        (((amount * apy_to_use / 100) / 10 / (365 * 24 * 60 * 60)) *
time_diff)  // Get 10% of Reward
            .try_into()
            .unwrap()
}
```

Additionally, the program interacts with two tokens: `taolie_mint` and `depin_mint`. The configuration and management of these tokens add another layer of centralization risk. Any issues with these tokens' configuration or availability could impact the entire staking process and rewards distribution.

```
pub const TAOLIE_MINT_ADDRESS: Pubkey =
pubkey!("2zE5rJ2ctXMz9hVbk1AvJa78X7mh3kuR728SNzGXTEeu");
pub const DEPIN_MINT_ADDRESS: Pubkey =
pubkey!("C1ZLCjaw9T79VBUKUhzXj4RLXEMFkp39oNNp2hoqDSq2");
```

Moreover, the rewards are distributed from the `reward_ata` account. The proper functioning of the rewards mechanism is dependent on this account having a sufficient balance of tokens. If this account runs out of tokens or is otherwise compromised, users will not be able to claim their earned rewards, disrupting the entire rewards system.

```
#[account(
    mint::token_program = token_program2022
  )]
  pub reward_ata: Box<InterfaceAccount<'info, TokenAccount2022>>,

token_2022::transfer_checked(
        CpiContext::new_with_signer(
            token_program.to_account_info(),
            TransferChecked {
                from: source.to_account_info(),
                mint: mint,
                to: destination.to_account_info(),
                authority: reward.to_account_info(),
            },
            &[&[b"reward", &[reward_bump]]],
        ),
        total_claim_amount, ctx.accounts.depin_mint.decimals
    )?;
```

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.

SPMI - Suboptimal Pattern Matching Implementation

Criticality	Minor / Informative
Location	instructions/deposit.rs#L19
Status	Unresolved

Description

The current implementation of the pattern matching used to validate the `locktime` value is functionally correct but is not written in the most concise or idiomatic way. The pattern matching could be streamlined to enhance readability and adherence to Rust's best practices. While the existing code correctly identifies valid `locktime` values, its structure could be simplified, making the code more maintainable and easier to understand. This is particularly important in codebases where clarity and conciseness are valued.

```
let is_valid_locktime = match locktime {
    0 | 1 | 3 | 6 | 12 => true,
    _ => false,
};
```

Recommendation

It is recommended to refactor the current pattern matching implementation by using the `matches!` macro, which provides a more concise and idiomatic approach to validating values in Rust. By adopting this method, the code will align better with Rust's standards, improving both readability and maintainability. This adjustment ensures that the code remains clear and efficient, facilitating easier updates and reducing the potential for errors.

UTC - Unnecessary Type Conversion

Criticality	Minor / Informative
Location	utils.rs#L50
Status	Unresolved

Description

The program code contains an unnecessary type conversion in the `calculate_rewards` function. Specifically, the result of an expression, already of type `u64`, is converted to the same type using `.try_into()`. This conversion is redundant and does not provide any functional benefit. Removing such unnecessary operations can improve code clarity and efficiency, as redundant conversions can confuse the code's intent and lead to minor inefficiencies.

```
pub fn calculate_rewards(
    amount: u64,
    last_claim_timestamp: i64,
    current_timestamp: i64,
    locktime: u64,
    apy: ApyConf,
) -> u64 {
    let time_diff: u64 = (current_timestamp -
last_claim_timestamp)
        .try_into()
        .unwrap();
    let apy_available: bool = current_timestamp >
apy.lock_time;

    let apy_to_use = if apy_available {
        match locktime {
            0 => apy.apy_day_1,
            1 => apy.apy_month_1,
            3 => apy.apy_month_3,
            6 => apy.apy_month_6,
            12 => apy.apy_year_1,
            _ => {
                msg!("Unknown timelock");
                return 0;
            }
        }
    } else {
        match locktime {
            0 => apy.last_apy_day_1,
            1 => apy.last_apy_month_1,
            3 => apy.last_apy_month_3,
            6 => apy.last_apy_month_6,
            12 => apy.last_apy_year_1,
            _ => {
                msg!("Unknown timelock");
                return 0;
            }
        }
    };

    (((amount * apy_to_use / 100) / 10 / (365 * 24 * 60 * 60))
* time_diff)  // Get 10% of Reward
        .try_into()
        .unwrap()
}
```

Recommendation

To improve the clarity and efficiency of the code, it is recommended to remove the redundant type conversion. Ensure that expressions are only converted when necessary and that type conversions add meaningful value to the code's logic. By eliminating unnecessary type conversions, the code becomes cleaner and more straightforward, adhering to best practices for writing efficient and readable Rust code. This adjustment will enhance the maintainability and understanding of the codebase.

Summary

The TAOlie Staking Contract implements a staking and rewards mechanism. 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

cyberscope.io