Cyberscope

Audit Report

TAOlie Staking Contract

August 2024

Repository	https://github.com/taolie-ai/staking-contract
Commit	bae92186e001c40167b16dc53852785414636df3
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	bae92186e001c40167b16dc53852785414636df3
Network	SOL

Audit Updates

Initial Audit	07 Aug 2024

Source Files

Filename	SHA256
errors.rs	c5c9b89985cc9db119dce59075d82fb9c0d3b68b6a160bb7634f6fa28cfe5a53
lib.rs	c7fea72d6b9a370308b4d6c61a82b71a34f51bed54ba58939f4a7928f3af0ea6
utils.rs	114d56d6aee2f8158394380413a8505a15a20553970453c923c883dd6108a23b
instructions/changeapy.rs	0be4d1a466bf2875687be3723493f9e608500caaac903bead19e97027018ca3a
instructions/claim.rs	fee39289dff61bd1c2d20a7439538aeb4bc1f11a155d3674eb3039912537d316
instructions/deposit.rs	10baa8753c6522b3bdd2c5ca80edc5c252d96b187785133b4157bf21922f8a67
instructions/initialize.rs	e3b60976a2e1a07e1b0226c1948552620edadce7807df90a95e08a80f98b8ab9

instructions/mod.rs	8a43f5f15ef3fc05f58463f46b0366d0f25f3e97b75ba82e73141dbbf16cb9b6
instructions/withdraw.rs	a62e9b117ebc2fd7c46399d82040df7fd803a867303045095f936d8342d17d91
state/apyconf.rs	83f1a615ed93b6d27490adb074a9afcd9d9cb919f4e59a66f0ecf7a7265b4a95
state/base.rs	bb6121e8f645966093l0c93e67fe42d7226c338a9262faf579174eef2d2d3505
state/mod.rs	ffd28e8784c77d8170493fd23f50c96ac307f611549f22d33f95fe9222690625
state/reward.rs	ac024d030b12dae12ea7f4d2274bb13670192aad1e6b034e97ed1b253692685f
state/stake.rs	c00426e7c8d50742c34cccc3f1c101551abd17af6989b74f4d052563cc48c79f
state/user.rs	6b07e9180e4fda3ebd849d0440e913c1d925408cfbaa854007bda548c318b8fd

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.

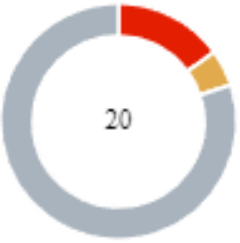
Calculate Rewards Function

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.

Contract Readability Comment

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contracts are missing the fundamental principles of a Rust smart contract regarding code readability, and data structures. According to the previously mentioned issues, the contracts cannot be assumed that are in a production-ready state. Given these issues, it is not advisable to assume that the contracts are in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Rust guidelines to ensure that the contracts adhere to established best practices and security measures. It is recommended that the team review the contracts to improve their efficiency. The code’s readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contracts' auditability and maintenance.

Findings Breakdown



Critical	3
Medium	1
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	1	0	0	0
Minor / Informative	16	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	PPR	Potential Program Reinitialization	Unresolved
●	PUFL	Potential User Fund Lock	Unresolved
●	PWE	Potential Withdraw Exploit	Unresolved
●	IRC	Incorrect Reward Calculation	Unresolved
●	DEMA	Descriptive Error Messages Absence	Unresolved
●	ICEH	Inadequate Clock Error Handing	Unresolved
●	ITH	Inconsistent Timestamps Handling	Unresolved
●	II	Inefficient Initialization	Unresolved
●	LFRV	Large Function Return Variant	Unresolved
●	MC	Misleading Comment	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	PUUP	Potential Unwrap Usage Panic	Unresolved
●	PCR	Program Centralization Risk	Unresolved
●	RAC	Redundant Admin Check	Unresolved

●	RB	Redundant Binding	Unresolved
●	RFN	Redundant Field Names	Unresolved
●	RRS	Redundant Return Statement	Unresolved
●	RCFC	Reward Calculation Formula Concern	Unresolved
●	TNE	Typographical Name Error	Unresolved
●	UTC	Unnecessary Type Conversion	Unresolved

PPR - Potential Program Reinitialization

Criticality	Critical
Location	instructions/initialize.rs#L14
Status	Unresolved

Description

The `initialize` function can be called multiple times without any restrictions, leading to potential reinitialization issues. Specifically, if the `initialize` function is called again after users have already staked, it will reset critical state variables such as `stake.total_amount` to 0. When users subsequently attempt to withdraw their staked tokens, the system uses this reset `total_amount`, leading to wrong calculations and potential underflows. This can result in incorrect token balances and a broken withdrawal mechanism, ultimately compromising the integrity and reliability of the staking protocol.

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

require_keys_eq!(ctx.accounts.depin_mint.mint_authority.unwrap(
), ctx.accounts.from.key(), InitializeError::AuthorityError);

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

  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().unwrap().unix_timestamp as u64;

  base_account.admin = admin;

  Ok(())
}
```

Recommendation

To address this issue, implement a mechanism to prevent the `initialize` function from being called multiple times. One approach is to introduce a flag within the contract's state that tracks whether the initialization has already occurred. If the contract is already initialized, the function should prevent further execution and revert any attempts to reinitialize. Another approach involves checking the existing state of critical variables before resetting them. If these variables already hold non-zero values, they should be preserved rather than being reset. By incorporating these safeguards, you ensure that the contract maintains its integrity and prevents unintended or malicious resets of important state variables, thereby protecting the interests of all users interacting with the staking protocol.

PUFL - Potential User Fund Lock

Criticality	Critical
Location	instructions/withdraw.rs#L40
Status	Unresolved

Description

The program allows users to provide a `locktime` value when they deposit their tokens, representing the duration for which their tokens are locked. However, the current implementation does not validate these user-provided `locktime` values to ensure they match the predefined periods used in the withdrawal logic. Specifically, the withdrawal function checks for specific locktime values (such as 1 month, 3 months, 6 months, and 1 year) to determine if a withdrawal is allowed. If a user provides a locktime value that does not match these predefined values, the withdrawal function will fail to recognize it as a valid lock period. Consequently, users with invalid locktime values will be unable to withdraw their funds, potentially losing access to their deposited tokens permanently.

```
let is_possible_withdraw: bool = match user.locktime {
        0 => current_timestamp - user.last_deposit_timestamp > 0, // 1
day
        1 => current_timestamp - user.last_deposit_timestamp > 30 * 24
* 60 * 60, // 1 month
        3 => current_timestamp - user.last_deposit_timestamp > 3 * 30 *
24 * 60 * 60, // 3 months
        6 => current_timestamp - user.last_deposit_timestamp > 6 * 30 *
24 * 60 * 60, // 6 months
        12 => current_timestamp - user.last_deposit_timestamp > 365 *
24 * 60 * 60, // 1 year
        _ => {
            msg!("Unknown timelock");
            false
        }
```

Recommendation

It is crucial to implement validation for the `locktime` values during the deposit process to ensure that users can only provide valid locktime values that correspond to the predefined periods recognized by the withdrawal logic. This validation will prevent users from inadvertently locking their funds with an unsupported locktime value and guarantee that all deposited funds can be withdrawn according to the contract's rules. By enforcing this validation, the contract will provide a more reliable and user-friendly experience, ensuring that users do not lose access to their funds due to invalid locktime inputs.

PWE - Potential Withdraw Exploit

Criticality	Critical
Location	instructions/deposit.rs#L14 instructions/withdraw.rs#L13
Status	Unresolved

Description

The program's current implementation allows users to manipulate the locktime and last deposit timestamp through subsequent deposits. A user can initially deposit a large amount of tokens with a high locktime to accrue significant rewards. Before the high locktime period expires, the user can make another deposit with a minimal amount of tokens and set the locktime to zero. This second deposit recalculates rewards up to the current timestamp and overwrites the user's locktime and last deposit timestamp with the new values provided. Consequently, the locktime is reset, and the user can withdraw their initial large deposit prematurely, circumventing the intended locktime restrictions. This behavior undermines the staking protocol’s locktime mechanism and allows users to exploit the system by gaining rewards meant for longer locktime periods while withdrawing their funds prematurely.

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

  require_gte!(deposit_amount, MINIMUM_DEPOSIT,
TaolieStakeError::MinimumDepositError);

  let destination = &ctx.accounts.stake_ata;
  let source = &ctx.accounts.from_ata;
  let token_program = &ctx.accounts.token_program;
  let authority = &ctx.accounts.from;
  let stake = &mut ctx.accounts.taolie_stake;
  let user = &mut ctx.accounts.taolie_user;
  let apy = &mut ctx.accounts.apy_pda;

  let apy_conf: ApyConf = { ApyConf{
      apy_day_1: apy.apy_day_1,
      apy_month_1: apy.apy_month_1,
      apy_month_3: apy.apy_month_3,
      apy_month_6: apy.apy_month_6,
      apy_year_1: apy.apy_year_1,
      lock_time: apy.lock_time,
      last_apy_day_1: apy.apy_day_1,
      last_apy_month_1: apy.last_apy_month_1,
      last_apy_month_3: apy.last_apy_month_3,
      last_apy_month_6: apy.last_apy_month_6,
      last_apy_year_1: apy.last_apy_year_1,
    }};

  let current_timestamp = Clock::get().unwrap().unix_timestamp;

  let claim_from_past = (current_timestamp -
user.last_claim_timestamp) as u64 * user.claimable_amount /
REWARD_PERIOD;
  let amount: u64 = calculate_rewards(user.deposted_amount,
user.last_claim_timestamp, current_timestamp, user.locktime,
apy_conf);
  user.claimed_amount += (amount / 9) + claim_from_past;
  user.claimable_amount = user.claimable_amount -
claim_from_past + amount;
  user.last_claim_timestamp = current_timestamp;

  stake.total_amount += deposit_amount;
  user.deposted_amount += deposit_amount;
  user.locktime = locktime;
  user.last_deposit_timestamp = current_timestamp;

  token::transfer(
    CpiContext::new(
        token_program.to_account_info(),
        SplTransfer {
```

```
            from: source.to_account_info(),
            to: destination.to_account_info(),
            authority: authority.to_account_info(),
        },
    ),
    deposit_amount,
  )?;
  Ok(())
}

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

    require_gte!(ctx.accounts.taolie_user.deposted_amount,
withdraw_amount, WithdrawError::DepositedAmountError);
    require_gt!(ctx.accounts.taolie_user.deposted_amount, 0,
WithdrawError::DepositedAmountError);

    let destination = &ctx.accounts.to_ata;
    let source = &ctx.accounts.stake_ata;
    let token_program = &ctx.accounts.token_program;
    let stake = &mut ctx.accounts.taolie_stake;
    let user = &mut ctx.accounts.taolie_user;

    let apy = &mut ctx.accounts.apy_pda;
    let apy_conf: ApyConf = { ApyConf{
        apy_day_1: apy.apy_day_1,
        apy_month_1: apy.apy_month_1,
        apy_month_3: apy.apy_month_3,
        apy_month_6: apy.apy_month_6,
        apy_year_1: apy.apy_year_1,
        lock_time: apy.lock_time,
        last_apy_day_1: apy.apy_day_1,
        last_apy_month_1: apy.last_apy_month_1,
        last_apy_month_3: apy.last_apy_month_3,
        last_apy_month_6: apy.last_apy_month_6,
        last_apy_year_1: apy.last_apy_year_1,
    }};

    let current_timestamp =
Clock::get().unwrap().unix_timestamp;
    let is_possible_withdraw: bool = match user.locktime {
        0 => current_timestamp - user.last_deposit_timestamp >
0, // 1 day
        1 => current_timestamp - user.last_deposit_timestamp >
30 * 24 * 60 * 60, // 1 month
        3 => current_timestamp - user.last_deposit_timestamp >
3 * 30 * 24 * 60 * 60, // 3 months
        6 => current_timestamp - user.last_deposit_timestamp >
6 * 30 * 24 * 60 * 60, // 6 months
```

```
        12 => current_timestamp - user.last_deposit_timestamp >
365 * 24 * 60 * 60, // 1 year
        _ => {
            msg!("Unknown timelock");
            false
        }
    };

    if is_possible_withdraw {
        let claim_from_past = (current_timestamp -
user.last_claim_timestamp) as u64 * user.claimable_amount /
REWARD_PERIOD;
        let amount: u64 =
calculate_rewards(user.deposted_amount,
user.last_claim_timestamp, current_timestamp, user.locktime,
apy_conf);
        user.claimed_amount += (amount / 9) + claim_from_past;
        user.claimable_amount = user.claimable_amount -
claim_from_past + amount;
        user.last_claim_timestamp = current_timestamp;

        user.deposted_amount -= withdraw_amount;
        stake.total_amount -= withdraw_amount;

        token::transfer(
            CpiContext::new_with_signer(
                token_program.to_account_info(),
                SplTransfer {
                    from: source.to_account_info(),
                    to: destination.to_account_info(),
                    authority: stake.to_account_info()
                },
                &[&["taolie".as_bytes(), &[stake_bump]]],
            ),
            withdraw_amount
        )?;
        Ok(())
    } else {
        Err(WithdrawError::BeforeLocktimeError.into())
    }
}
```

Recommendation

To prevent this exploit, the program should enforce the locktime specified during the user's initial deposit and prevent it from being shortened by subsequent deposits. One approach is to maintain separate locktime and timestamp records for each deposit, ensuring that each deposit’s locktime must independently be respected during withdrawals. Alternatively, implement a mechanism that verifies the minimum locktime for withdrawals based on the longest locktime of all deposits made by the user. By implementing such measures, the staking protocol can ensure consistent enforcement of the locktime mechanism, preventing users from bypassing the locktime restrictions and maintaining the integrity of the reward system.

IRC - Incorrect Reward Calculation

Criticality	Medium
Location	utils.rs#L11
Status	Unresolved

Description

The function `calculate_rewards` is responsible for calculating user rewards, which does not validate the `locktime` values provided by users. If a user provides a locktime value that does not match the predefined periods (such as 1 day, 1 month, 3 months, 6 months, or 1 year), the function defaults to returning an APY of 0. This results in no rewards being accrued for those users, effectively treating their locktime as invalid. Consequently, users who provide an unsupported locktime value will receive significantly lower or zero rewards, which does not align with their expectations and the intended behavior of the staking protocol.

```
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
as i64;

    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;
            }
        }
    };

    let reward = (((amount * apy_to_use / 100) / 10 * 9 / (365
* 24 * 60 * 60)) * time_diff)
    .try_into()
    .unwrap();

    reward
}
```

Recommendation

It is crucial to implement validation for the `locktime` values during the reward calculation process to ensure that users can only provide valid locktime values corresponding to the predefined periods recognized by the contract. This validation will prevent users from inadvertently using unsupported locktime values, thereby ensuring they receive the appropriate rewards. By enforcing this validation, the contract will provide a more reliable and user-friendly experience, ensuring that users receive the rewards they expect based on their chosen locktime. This approach will also prevent any discrepancies in reward distribution and uphold the integrity of the staking protocol.

DEMA - Descriptive Error Messages Absence

Criticality	Minor / Informative
Location	errors.rs#L5,10,11,16,17,22
Status	Unresolved

Description

The error definitions in the program are currently defined without any accompanying error messages. This practice can lead to confusion and make it more challenging to diagnose and understand the reasons behind transaction failures. Providing descriptive error messages helps quickly identify the specific issues, improving the overall experience and facilitating easier debugging.

```
use anchor_lang::error_code;

#[error_code]
pub enum TaolieStakeError {
  MinimumDepositError,
}

#[error_code]
pub enum WithdrawError {
  BeforeLocktimeError,
  DepositedAmountError,
}

#[error_code]
pub enum ApyChangeError {
  WrongAdminError,
  LockTimeNotExpiredError
}

#[error_code]
pub enum InitializeError {
  AuthorityError,
}
```

Recommendation

It is recommended to enhance the error definitions by including descriptive messages for each error type. These messages should clearly explain the nature of the error and provide context to help understand the cause of the issue. By incorporating descriptive error messages, the program will offer more informative feedback. Consistent and clear error messaging is a best practice that significantly contributes to the reliability and usability of the code.

ICEH - Inadequate Clock Error Handing

Criticality	Minor / Informative
Location	instructions/initialize.rs#L34 instructions/change_apy.rs#L9 instructions/claim.rs#L34 instructions/deposit.rs#L34 instructions/withdraw.rs#L39
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.

```
apy.lock_time = Clock::get().unwrap().unix_timestamp as u64;

let current_time = Clock::get().unwrap().unix_timestamp as u64;

let current_timestamp = Clock::get().unwrap().unix_timestamp;
```

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.

ITH - Inconsistent Timestamps Handling

Criticality	Minor / Informative
Location	instructions/change_apy.rs#L9 instructions/claim.rs#L34 instructions/deposit.rs#L34 instructions/withdraw.rs#L39
Status	Unresolved

Description

The program code exhibits inconsistency in handling blockchain timestamps, with some parts of the code casting the `unix_timestamp` to u64 while others use it directly as i64. Specifically, there are instances where the current blockchain time is retrieved and stored directly as i64, while in other cases, it is explicitly cast to u64.Maintaining consistency in handling and storing similar values is crucial to avoid such issues. The differing types can lead to compilation errors or logic errors if the code that utilizes these timestamps expects a specific type. Ensuring that timestamps are consistently handled as either i64 or u64 based on the application's requirements will prevent such errors and enhance the codebase's reliability.

```
let current_time = Clock::get().unwrap().unix_timestamp as u64;

let current_timestamp = Clock::get().unwrap().unix_timestamp;
```

Recommendation

It is recommended to standardize the handling of timestamps throughout the codebase. Decide whether timestamps should be stored and used as i64 or u64 based on the application's needs and ensure that all instances where timestamps are retrieved and used follow this standard. By adopting a consistent approach to handling timestamps, the program will be more robust and less prone to errors related to type mismatches and unexpected conversions.

II - Inefficient Initialization

Criticality	Minor / Informative
Location	instructions/deposit.rs#L26 instructions/claim.rs#L21 instructions/withdraw.rs#L25
Status	Unresolved

Description

The current implementation of the `ApyConf` struct initialization in different parts of the codebase uses redundant braces. This style of initialization does not offer any functional advantage and unnecessarily complicates the code. Simplifying the initialization process improves code readability and maintainability.

```
let apy_conf: ApyConf = { ApyConf{
      apy_day_1: apy.apy_day_1,
      apy_month_1: apy.apy_month_1,
      apy_month_3: apy.apy_month_3,
      apy_month_6: apy.apy_month_6,
      apy_year_1: apy.apy_year_1,
      lock_time: apy.lock_time,
      last_apy_day_1: apy.apy_day_1,
      last_apy_month_1: apy.last_apy_month_1,
      last_apy_month_3: apy.last_apy_month_3,
      last_apy_month_6: apy.last_apy_month_6,
      last_apy_year_1: apy.last_apy_year_1,
    }};
```

Recommendation

It is recommended to remove the redundant braces in the initialization of the `ApyConf` struct. Simplify the code to directly assign values to the struct fields without using unnecessary braces. This will make the code cleaner and easier to understand, enhancing overall code quality.

LFRV - Large Function Return Variant

Criticality	Minor / Informative
Location	instructions/initialize.rs#L14 instructions/deposit.rs#L14 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.

MC - Misleading Comment

Criticality	Minor / Informative
Location	instructions/withdraw.rs#L41
Status	Unresolved

Description

The `withdraw` function contains a misleading comment related to the locktime scenario when it is set to 0. The current comment suggests that users should lock their tokens for at least one day. However, the actual behavior indicates that a locktime of 0 means there is no lock period, and users can withdraw their tokens immediately.

```
let is_possible_withdraw: bool = match user.locktime {
        0 => current_timestamp - user.last_deposit_timestamp >
0, // 1 day
        1 => current_timestamp - user.last_deposit_timestamp >
30 * 24 * 60 * 60, // 1 month
        3 => current_timestamp - user.last_deposit_timestamp >
3 * 30 * 24 * 60 * 60, // 3 months
        6 => current_timestamp - user.last_deposit_timestamp >
6 * 30 * 24 * 60 * 60, // 6 months
        12 => current_timestamp - user.last_deposit_timestamp >
365 * 24 * 60 * 60, // 1 year
        _ => {
            msg!("Unknown timelock");
            false
        }
```

Recommendation

It is recommended to update the comment to accurately reflect the behavior of the locktime scenario when it is set to 0. The revised comment should clearly state that a locktime of 0 implies no lock period, allowing immediate withdrawals. This ensures that the documentation is consistent with the code logic, improving readability and reducing the risk of misunderstandings.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	instructions/deposit.rs#L44 instructions/withdraw.rs#L55 instructions/withdraw.rs#L43
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand. Specifically, the variable `claimed_amount` is reset to zero each time a user claims their rewards, which contradicts the expected behavior suggested by its name. Typically, `claimed_amount` should represent the total amount of tokens the user has claimed over time.

```
user.claimed_amount += (amount / 9) + claim_from_past;

user.claimed_amount = 0;
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

PUUP - Potential Unwrap Usage Panic

Criticality	Minor / Informative
Location	instructions/initialize.rs#L15
Status	Unresolved

Description

The program uses `unwrap()` on the mint_authority field of the `depin_mint` account to verify that the caller is the mint authority. Using `unwrap()` on an option type without handling the possibility of a `None` value introduces a risk of panicking if the value is `None`. This can lead to unexpected program behavior and potential crashes, which can compromise the reliability and security of the program.

```
require_keys_eq!(ctx.accounts.depin_mint.mint_authority.unwrap(
), ctx.accounts.from.key(), InitializeError::AuthorityError);
```

Recommendation

To enhance the reliability and security of the program, it is recommended to explicitly handle the possibility of the `mint_authority` being `None`. Instead of using `unwrap()`, the code should include a check to ensure that the `mint_authority` is present before proceeding with the comparison. By incorporating proper error handling, the program can gracefully handle cases where the `mint_authority` is absent, preventing unnecessary panics and maintaining smooth program execution. This approach ensures that the program only proceeds with authorized actions and enhances overall robustness.

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<()> {

require_keys_eq!(ctx.accounts.depin_mint.mint_authority.unwrap(
), ctx.accounts.from.key(), InitializeError::AuthorityError);

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

  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().unwrap().unix_timestamp as u64;

  base_account.admin = admin;

  Ok(())
}

pub fn changeapy(ctx: Context<Apy>, apy_new: ApyConf) -> Result
<()> {
    let apy = &mut ctx.accounts.apy_pda;
    let current_time = Clock::get().unwrap().unix_timestamp as
u64;
    let base_account = &mut ctx.accounts.base_account;
    let admin = &mut ctx.accounts.admin;

    if base_account.admin == *admin.key {
        if current_time < apy.lock_time {
            return
Err(ApyChangeError::LockTimeNotExpiredError.into());
        }
        apy.last_apy_day_1 = apy.apy_day_1;
        apy.last_apy_month_1 = apy.apy_month_1;
        apy.last_apy_month_3 = apy.apy_month_3;
        apy.last_apy_month_6 = apy.apy_month_6;
        apy.last_apy_year_1 = apy.apy_year_1;

        apy.apy_day_1 = apy_new.apy_day_1;
```

```
            apy.apy_month_1 = apy_new.apy_month_1;
            apy.apy_month_3 = apy_new.apy_month_3;
            apy.apy_month_6 = apy_new.apy_month_6;
            apy.apy_year_1 = apy_new.apy_year_1;

            apy.lock_time = current_time + APY_LOCK_TIME_PERIOD;
            Ok(())
        } else {
            return Err(ApyChangeError::WrongAdminError.into());
        }
    }
```

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!("5jaCSTPi6fhEwNDR138qj3f8JP2StXg13r2T5JZuM1KV");
pub const DEPIN_MINT_ADDRESS: Pubkey =
pubkey!("GY5B3APcopfPv6Trnw7UgCUJ35YfQA7HWEpPtoVemrCi");
```

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.

RAC - Redundant Admin Check

Criticality	Minor / Informative
Location	instructions/changeapy.rs#L13
Status	Unresolved

Description

The `changeapy` function includes a manual runtime check to verify that the admin initiating the APY change matches the admin recorded in the `base_account`. This check is performed by comparing the admin's public key in the function itself. Additionally, the `Apy` struct, which defines the accounts for the `changeapy` function, uses the `#[account(has_one = admin)]` attribute provided by the Anchor framework. This attribute enforces at compile-time that the base_account's admin field must match the provided admin account. While both the compile-time attribute and the runtime check aim to ensure that the correct admin is interacting with the `base_account`, they serve similar purposes and thus, the runtime check is redundant. The compile-time constraint already guarantees that the relationship between the `base_account` and the admin is correctly enforced before the function logic is executed.

```
if base_account.admin == *admin.key {
```

Recommendation

To streamline the code and avoid redundancy, it is recommended to rely on the compile-time verification provided by the Anchor framework's `#[account(has_one = admin)]` attribute. This attribute ensures the integrity of the account relationships and is sufficient to enforce the correct admin interaction. Removing the redundant runtime check will simplify the code and maintain clarity while still providing robust security. By eliminating unnecessary checks, the code becomes more maintainable and easier to understand, while still ensuring that only the correct admin can perform APY changes.

RB - Redundant Binding

Criticality	Minor / Informative
Location	utils.rs#L44
Status	Unresolved

Description

The program code contains a redundant `let` binding in the `calculate_rewards` function. Specifically, the result of an expression is assigned to a variable and then immediately returned. This practice is unnecessary and leads to less concise code. Instead, the expression can be returned directly without storing it in an intermediate variable. Removing such redundant bindings can improve code readability and maintainability, making the logic clearer and the function more streamlined.

```
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
as i64;

    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;
            }
        }
    };

    let reward = (((amount * apy_to_use / 100) / 10 * 9 / (365
* 24 * 60 * 60)) * time_diff)
    .try_into()
    .unwrap();

    reward
}
```

Recommendation

To improve the readability and maintainability of the code, it is recommended to eliminate the redundant `let` binding by returning the expression directly. This approach will result in cleaner and more concise code, adhering to best practices for writing efficient and readable Rust code. By making this adjustment, the function's logic becomes clearer, which helps in maintaining and understanding the codebase.

RFN - Redundant Field Names

Criticality	Minor / Informative
Location	onstructions/claim.rs#L55
Status	Unresolved

Description

The program code includes instances of redundant field names during struct initialization. Specifically, there is a case where the field name and the variable name are the same, resulting in redundancy. This occurs when the `mint` field is assigned the value of the `mint` variable, which unnecessarily repeats the field name. This redundancy does not cause functional issues but leads to less readable and maintainable code. Cleaner and more concise code is generally preferred as it enhances readability and reduces potential confusion.

```
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 improve the readability and maintainability of the code, it is recommended to remove redundant field names during struct initialization. Use the shorthand syntax provided by Rust for struct initialization when the field name and the variable name are identical. This will make the code cleaner and easier to understand, contributing to better coding practices and overall code quality. Ensuring that the codebase follows best practices in struct initialization will enhance the maintainability and clarity of the program code.

RRS - Redundant Return Statement

Criticality	Minor / Informative
Location	instructions/changeapy.rs#L32
Status	Unresolved

Description

The program code contains an unnecessary `return` statement within the `changeapy` function when handling an error case. Specifically, when the provided admin is not authorized, the function uses `return` to return an error. This is redundant in Rust, as simply returning the error expression without the `return` keyword is sufficient and more idiomatic. The presence of unnecessary `return` statements can lead to less readable and maintainable code. Adhering to Rust's idiomatic practices helps ensure the codebase is clean, concise, and easy to understand.

```
if base_account.admin == *admin.key {
        if current_time < apy.lock_time {
            return Err(ApyChangeError::LockTimeNotExpiredError.into());
        }
        apy.last_apy_day_1 = apy.apy_day_1;
        apy.last_apy_month_1 = apy.apy_month_1;
        apy.last_apy_month_3 = apy.apy_month_3;
        apy.last_apy_month_6 = apy.apy_month_6;
        apy.last_apy_year_1 = apy.apy_year_1;

        apy.apy_day_1 = apy_new.apy_day_1;
        apy.apy_month_1 = apy_new.apy_month_1;
        apy.apy_month_3 = apy_new.apy_month_3;
        apy.apy_month_6 = apy_new.apy_month_6;
        apy.apy_year_1 = apy_new.apy_year_1;

        apy.lock_time = current_time + APY_LOCK_TIME_PERIOD;
        Ok(())
    } else {
        return Err(ApyChangeError::WrongAdminError.into());
    }
}
```

Recommendation

To improve the readability and maintainability of the code, it is recommended to remove the unneeded `return` statement in the `changeapy` function. Instead, directly return the error expression without the `return` keyword. This change will make the code more idiomatic and consistent with Rust's best practices, enhancing overall code quality. Ensuring that the codebase follows idiomatic Rust practices will contribute to better maintainability and readability of the program code.

RCFC - Reward Calculation Formula Concern

Criticality	Minor / Informative
Location	utils.rs#L44
Status	Unresolved

Description

The current implementation of the reward calculation formula in the program includes a step where the calculated reward is multiplied by 9 and then divided by 10 within the formula, followed by dividing the final reward by 9 when it is returned. This sequence of operations does not provide a clear rationale within the context of the formula and a concern arises around the business logic of the calculationof rewards. The rest of the calculations within the formula, such as the application of the APY based on the locktime and the time difference, are logically sound.

```
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
as i64;

    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;
            }
        }
    };

    let reward = (((amount * apy_to_use / 100) / 10 * 9 / (365
* 24 * 60 * 60)) * time_diff)
    .try_into()
    .unwrap();

    reward
}
```

Recommendation

It is recommended to review and clarify the rationale behind the reward calculation formula. If these operations are intended to serve a specific purpose, it should be explicitly documented to avoid confusion and ensure that the formula accurately reflects the intended business logic. If these operations are found to be redundant or incorrect, they should be removed or adjusted to ensure the reward calculation is straightforward and logically sound. This will enhance the transparency and correctness of the reward calculation, thereby maintaining the integrity of the staking protocol.

TNE - Typographical Name Error

Criticality	Minor / Informative
Location	state/user.rs#L14
Status	Unresolved

Description

The `TaolieUser` struct in the program contains a typographical error in one of its field names. The field `deposted_amount` is incorrectly named and should be `deposited_amount`. This typo can lead to confusion users interacting with the code, as well as potential issues in maintaining and understanding the program. Consistent and correct naming conventions are essential for code clarity and maintainability.

```
use anchor_lang::prelude::*;

#[account]
pub struct TaolieUser {
  pub deposted_amount: u64,
  pub claimable_amount: u64,
  pub claimed_amount: u64,
  pub last_deposit_timestamp: i64,
  pub last_claim_timestamp: i64,
  pub locktime: u64
}

impl TaolieUser {
  pub const LEN: usize = 8 + 8 * 6;
}
```

Recommendation

It is recommended to correct the typographical error by renaming the field from `deposted_amount` to `deposited_amount`. This change will improve the readability and accuracy of the code, ensuring that the field name correctly represents the amount deposited by the user. Additionally, any references to this field throughout the codebase should be updated to reflect the corrected name. This practice will help in maintaining a high standard of code quality and reduce potential misunderstandings or errors during development and future updates.

UTC - Unnecessary Type Conversion

Criticality	Minor / Informative
Location	utils.rs#L44
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
as i64;

    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;
            }
        }
    };

    let reward = (((amount * apy_to_use / 100) / 10 * 9 / (365
* 24 * 60 * 60)) * time_diff)
    .try_into()
    .unwrap();

    reward
}
```

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