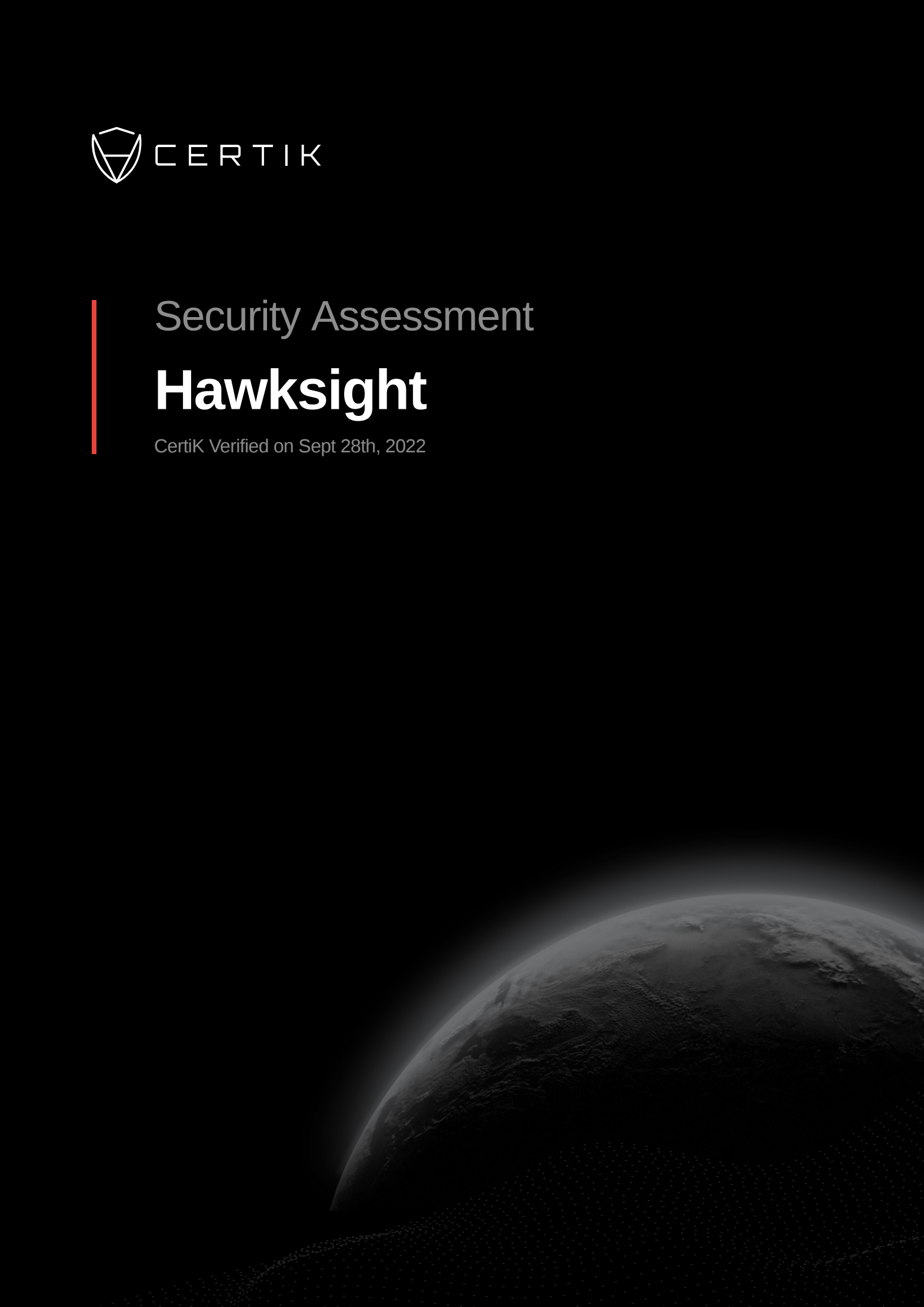
CERTIK

Security Assessment

Hawksight

CertiK Verified on Sept 28th, 2022



CertiK Verified on Sept 28th, 2022

Hawksight

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Staking	Solana	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Rust	Delivered on 09/28/2022	N/A

CODEBASE

https://github.com/hawksightco/hs-dapp/tree/dev/programs/index-yield-farming/src

...View All

COMMITS

36c4577b5ec60ffdec38690ea79d84a940ce5238

...View All

Vulnerability Summary

29	24	2	2	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	2 Resolved, 2 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
3	Medium	3 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
6	Minor	6 Resolved	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
16	Informational	13 Resolved, 2 Partially Resolved, 1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | HAWKSIGHT

CODEBASE | HAWKSIGHT

Repository

https://github.com/hawksightco/hs-dapp/tree/dev/programs/index-yield-farming/src

Commit

36c4577b5ec60ffdec38690ea79d84a940ce5238

AUDIT SCOPE | HAWKSIGHT

8 files audited • 1 file with Acknowledged findings • 5 files with Partially Resolved findings • 1 file with Mitigated findings • 1 file with Resolved findings

ID	File	SHA256 Checksum
PRC	programs/index-yield-farming/src/processors.rs	815589645a3b52e685755cb3e973d4a751db7ddc5538a257e3f3c016d8659d76
COE	programs/index-yield-farming/src/contexts.rs	93266b5735d9e7b77380fe0c798b0e5d25b314944afa625ad24a4504e065643a
ERO	programs/index-yield-farming/src/errors.rs	b62823f442ba3fb07716907966daa5e6f0ebafb178015b393953dff010edff9f
EVN	programs/index-yield-farming/src/events.rs	0cd5be85a232eb763adce63763fe1ae51999b74a1a4c4ac331805abe2c45a0b6
LIR	programs/index-yield-farming/src/lib.rs	961a01b321e608cde60ad84822b80938544606acdfe5455df18590c7ca41437a
STT	programs/index-yield-farming/src/states.rs	8006e4959bc2c75ef2d99430721517ec2f66535c337d9c13c8124e99d0fe7a26
UTL	programs/index-yield-farming/src/utils.rs	6edf9d9eb63b3ec548914493a2fa48a9c1acafc750b21f0b15c3e5f1cd0ec14a
COS	programs/index-yield-farming/src/constants.rs	c1e2d335a73dc7cf04159ddc2d1481d26e1c254bdb083580ec9ee95884aeed7b

APPROACH & METHODS | HAWKSIGHT

This report has been prepared for Hawksight to discover issues and vulnerabilities in the source code of the Hawksight project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | HAWKSIGHT

29	0	4	3	6	16
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Hawksight . Through this audit, we have uncovered 29 issues ranging from different severity levels. Utilizing Static Analysis techniques to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Program Upgrade Centralization Risk**	**Centralization / Privilege**	**Major**	**Mitigated**
GLOBAL-02	Lack Of Tests	Coding Style	Major	Resolved
LIR-01	**Centralization Related Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
PRC-01	Lack Of Authentication For `create_state`	Logical Issue	Major	Resolved
PRC-02	Incorrect `bump` Implementation	Logical Issue	Minor	Resolved
PRC-03	Use `create_miner_v2` Instead Of `create_miner`	Logical Issue	Minor	Resolved
PRC-04	Missing `Action` Check	Logical Issue	Minor	Resolved
PRC-05	Staked And Unstaked Amounts Can Become Inconsistent	Logical Issue	Minor	Resolved
SRI-01	Lack Of Length Check For `weights` And `remaining_accounts`	Volatile Code	Medium	Resolved
SRI-02	Terra's UST Should Not Be Used As A Stable Coin	Logical Issue	Medium	Resolved

ID	Title	Category	Severity	Status
SRI-03	Missing Validation For `FarmRewardInfo::update` And `ChangeTokenPerSecondMulti::change_token_per_second`	Control Flow	Medium	Resolved
SRI-04	Lack Of `Asset` Length Limit Check	Logical Issue	Minor	Resolved
SRI-05	Lack Of Mint Validation For User Token Accounts	Logical Issue	Minor	Resolved
COE-01	Reduce The Use Of `std:mem:size_of()`	Language Specific	Informational	Resolved
COE-02	Typo	Coding Style	Informational	Resolved
LIR-02	Incorrect Use Of '_' Syntax For Unused Variable	Coding Style	Informational	Partially Resolved
PRC-06	Simplified Implementation Of `index` In Loop	Coding Style	Informational	Resolved
PRC-07	Add Non-Zero Check For `total_weight`	Logical Issue	Informational	Resolved
PRC-08	Third Party Dependencies	Volatile Code	Informational	Acknowledged
PRC-09	`last_amount` Not Reset To Zero	Coding Style	Informational	Resolved
SRI-07	Unnecessary `&` Reference	Coding Style	Informational	Resolved
SRI-08	Unnecessary Conversion To The Same Type	Coding Style	Informational	Resolved
SRI-09	Remove Commented Code	Coding Style	Informational	Partially Resolved
SRI-10	Unnecessary Account	Coding Style	Informational	Resolved

ID	Title	Category	Severity	Status
STT-01	Simplifiable `require` Operation	Coding Style	Informational	Resolved
UTL-01	Unnecessary `return` Statement	Coding Style	Informational	Resolved
UTL-02	Unused Variable	Coding Style	Informational	Resolved
UTL-03	Unnecessary Re-Slicing	Coding Style	Informational	Resolved
UTL-04	Optimize `creator_fee` Calculation For Improved Precision	Logical Issue	Informational	Resolved

GLOBAL-01 | PROGRAM UPGRADE CENTRALIZATION RISK

Category	Severity	Location	Status
Centralization / Privilege	**Major**		**Mitigated**

Description

A Solana program can be deployed on the mainnet as:

- final: the code cannot be updated.
- upgradable: `BPFLoaderUpgradeable` needs to be set as the program owner and an `upgrade authority`, which is a user account, is given.

In case the `Hawksight` program is deployed as upgradable, the `upgrade authority` has the privilege to update the implementation of the program at his/her will.

Any compromise to the `upgrade authority` account may allow a hacker to take advantage of this authority and control the implementation of the program and therefore execute potential malicious functionalities in the program.

Recommendation

Our recommendation depends on the team's intentions that we invite to clarify.

If the `Hawksight` program is going to be deployed as final, no further actions are needed to address the finding.

Otherwise, we recommend that the team make efforts to restrict access to the private key of the `upgrade authority` account. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key being compromised;
  AND

- A medium/blog link for sharing the timelock and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock and multi-signers addresses, and DAO information with the public audience.

Permanent:

Deploying the program as `final` can fully resolve the risk.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

For remediation and mitigated status, please provide the following information:

- Provide the account address with ALL the multi-signer addresses for the verification process.
- Provide a link to the medium/blog with all of the above information included

Alleviation

`[Hawksignt]`: We have just migrated our upgrade authority to a multi-sig, can verify from our program address where the upgrade authority is a multi sig account owned by the governance program. We are currently using `Realms` as our multi sig interface and we've implemented 24 hour time lock for the contract upgrade authority via `tx`.

GLOBAL-02 | LACK OF TESTS

Category	Severity	Location	Status
Coding Style	Major		Resolved

Description

The unit tests here are not enough. Testing programs are a very important aspect of proving program correctness, preventing regressions, and release engineering. Without tests, there is no way to know if the program works as expected.

The fact that code was shipped for review with such major flaw, is considered a signal of some problems with release engineering procedures, which may lead to shipping not intended changes to production.

Recommendation

We recommend adding exhaustive tests to the project, including edge cases, non-happy paths, and error conditions. This can include, but is not limited to:

- Unit tests
- Integration tests
- Behavioral tests
- Stress tests
- CI/CD pipelines various program use cases with unit-tests and integrating tests into CI/CD

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

LIR-01 | CENTRALIZATION RELATED RISKS

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**programs/index-yield-farming/src/lib.rs: 63, 70**	**Mitigated**

Description

In the contract `lib.rs` , the role `authority` has authority over the following function:

- change_multi_index_farm_rate(): change the speed at which farms generate rewards;
- create_saber_farm_strategy(): add a new strategy.

Any compromise to the privileged accounts may allow a hacker to take advantage of this to update project configurations.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Alleviation

`[Hawksignt]`: We are currently using `Realms` as our multi sig interface and we've implemented 24 hour time lock for the contract upgrade authority via `tx`.

PRC-01 | LACK OF AUTHENTICATION FOR `create_state`

Category	Severity	Location	Status
Logical Issue	Major	programs/index-yield-farming/src/processors.rs: 22~35	Resolved

Description

The `create_state` function is designed to initialize a globally unique `state` account. This `state` account will be used to identify the project owner and to ensure that only the owner can create a `strategy` and set the `token_per_second` of the farm.

```
pub struct CreateState<'info> {
    #[account(
        init,
        seeds = [b"state".as_ref()],
        bump,
        payer = authority,
        space = 8 + size_of::<StateAccount>()
    )]
    pub state: Account<'info, StateAccount>,
    ...
}
```

```
pub fn create_state(&mut self, bump: u8) -> Result<()> {
        let state = &mut self.state;
        state.authority = self.authority.key();
        state.bump = bump;
        state.reward_mint = self.reward_mint.key();
        state.reward_vault = self.reward_vault.key();
        emit!(StateCreated {});
        Ok(())
    }
```

Any given program_id can only have one `state` account. This may allow malicious users to pre-empt the creation of `state` by posting `create_state` transactions first, thus making the program unavailable.

Additionally, if a malicious user manages to call `create_state` prior to admin of the program, the malicious attacker would then have a parallel running program who's state is controlled by them. The user could use this to impersonate the real program to exploit users.

Recommendation

We recommend adding authentication to the `create_state` function to ensure the account is being created by the expected entity.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-02 | INCORRECT `bump` IMPLEMENTATION

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/processors.rs: 27, 121, 191, 220	Resolved

Description

By adding an empty `bump` constraint to the `#[account(...)]` macro, you signal Anchor to find the canonical bump on its own for the initialization of the account. The user defined `bump` will be written to storage, but will not be used for account initialization.

```
pub fn create_state(&mut self,
        bump: u8,
    ) -> Result<()> {
        ...
        state.bump = bump;
        ...
    }
```

Anywhere that the storage `state.bump` is used will result in incorrect validation of account addresses.

Reference material on bump creation can be found here and here

Recommendation

We recommend following the Anchor standard patterns for handling bumps.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-03 | USE `create_miner_v2` INSTEAD OF `create_miner`

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/processors.rs: 254~267	Resolved

Description

The QuarryProtocol provides `create_miner` to initialize the miner. However, since the current `create_miner` still requires `bump` for initialization, QuarryProtocol provides a more standardized `create_miner_v2` to replace `create_miner`.

```
create_miner(CpiContext::new(
    self.quarry_mine_program.to_account_info(),
    CreateMiner{
        authority: self.user.to_account_info(),
        miner: self.miner.to_account_info(),
        quarry: self.quarry.to_account_info(),
        rewarder: self.rewarder.to_account_info(),
        system_program: self.system_program.to_account_info(),
        payer: self.authority.to_account_info(),
        token_mint: self.lp_mint.to_account_info(),
        miner_vault: self.miner_vault.to_account_info(),
        token_program: self.token_program.to_account_info(),
    }).with_signer(&[&authority_seeds[..]]),
    miner_bump)?;
```

Recommendation

The more recent version, `create_miner_v2`, of the `create_miner` function includes additional functionality, in that, there is no need to supply `bump`.

```
/// The V2 variant removes the need for supplying the bump.
    #[access_control(ctx.accounts.validate())]
    pub fn create_miner_v2(ctx: Context<CreateMiner>) -> Result<()> {
        instructions::create_miner::handler(ctx)
    }
```

We recommend considering upgrading to the new version of the function to take advantage of the functionality. Additionally, under any upgrades of third party libraries, it is important to test the program with the upgraded version to ensure it runs as expected.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-04 | MISSING `Action` CHECK

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/processors.rs: 715, 778	Resolved

Description

According to the logic, **Hawksight** program forms a complete process in strict accordance with the `Action` steps. The project strictly controls the process execution through the inspection of `Action` .
For example, in `supply_liquidity` , the function can only be run if the `last_action` was a `SwapAction`

```
pub fn supply_liquidity(&mut self, min_amount_out: u64) -> Result<()> {
        ...
        require!(
            (user_asset_info.last_action == Action::SwapAction as u8)
                || ((user_asset_info.last_action == Action::FundAction as u8)
                    && (strategy.token_mint == USDC_MAINNET)
                    && ((strategy.src_mint == USDC_MAINNET)
                        || (strategy.dst_mint == USDC_MAINNET))),
            ErrorCode::InvalidOperationOrder
        );
        ...
```

However, `Action` is not verified in `stake_to_farm()` and `unstake_from_farm()` function, which may lead to unexpected errors in cross-step operations.

Recommendation

We recommend adding `Action` checks to ensure that the process is executed in the correct order.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-05 STAKED AND UNSTAKED AMOUNTS CAN BECOME INCONSISTENT

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/processors.rs: 755, 767, 768	Resolved

Description

In the `stake_to_farm()` function, the user can stake the LP obtained by adding liquidity to `saber_farm` to obtain rewards.

```
        stake_lp_token_pda(
            self.saber_farm_program.to_account_info(),
            self.token_program.to_account_info(),
            user.to_account_info(),
            &self.saber_farm,
            self.user_pda_lp_token.to_account_info(),
            self.saber_farm_rewarder.to_account_info(),
            // Stake full LP token balance of pass through account
            self.user_pda_lp_token.amount,
            // user.asset_infos[asset_index].last_amount,
            authority_seeds,
        )?;
```

However, the stake amount is `user_pda_lp_token.amount`, but the billed amount is `user_asset.last_amount`.

```
        user_asset.amount = user_asset
            .amount
            .checked_add(user_asset.last_amount)
            .unwrap();
```

If there is an external transaction that transfers the additional LPs to the `user_pda_lp_token`, this will cause the actual stake amount to be inconsistent with the billed amount, users cannot get back all staked LPs when unstaking from the farm.

Recommendation

We recommend computing the unstaked user asset amount such that it is consistent with the amount staked, such that any external transactions to the token liquidity pool won't effect the users account.

Alleviation

`[Hawksight]`: We have decided to deprecate platform rewards coming from Hawksight. The changes have been committed in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-01 LACK OF LENGTH CHECK FOR `weights` AND `remaining_accounts`

Category	Severity	Location	Status
Volatile Code	Medium	programs/index-yield-farming/src/lib.rs: 53~59; programs/index-yield-farming/src/processors.rs: 66~67, 73~76, 125~131	Resolved

Description

In the `farm` account, the `asset_infos` array is initialized by `weights` and `remaining_accounts`.

- `mints` vector is initialized by iterating over `ctx.remaining_accounts`.

```
for mint in _ctx.remaining_accounts.iter(){
            require!(*mint.owner == token::ID, ErrorCode::InvalidMint);
            Mint::unpack(&mint.to_account_info().try_borrow_data()?)?;
            mints.push(mint.key());
        }
```

- `asset_size` is defined by the length of `weights`.

```
let asset_size = weights.len();
let asset_count = asset_size.try_into().unwrap();
```

- `mints` is used to calculate the `farm`'s `seed`.

```
for mint in mints.iter() {
    seeds.push(mint.as_ref());
}
```

- `farm.asset_infos` is filled by iterating over the elements of `mints` and `weights` up to `asset_size`.

```
for i in 0  .. asset_size{
            total_weight += u128::from(weights[i]);
            farm.asset_infos.push(FarmAssetInfo{
                weight: weights[i],
                mint: mints[i]
            })
        }
```

However, the length of `weights` and `remaining_accounts` / `mints` are not checked during this initialization process. This makes it possible that the farm's `seed` and its `asset_infos` to not match, making the service unusable.

Recommendation

We recommend including validations ensuring the correct shape of data being used. Multiple variable length `Vectors` can easily introduce bugs, if the program invariants are not checked.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-02 | TERRA'S UST SHOULD NOT BE USED AS A STABLE COIN

Category	Severity	Location	Status
Logical Issue	Medium	programs/index-yield-farming/src/constants.rs: 27~28; programs/index-yield-farming/src/processors.rs: 458~459, 465~466, 537~538, 544~545	Resolved

Description

Note that with the `UST` price now below $0.05, and no longer has the role of a stable coin. This may easily affect functionality of UST in the project.

```
pub const UST_MAINNET: Pubkey = static_pubkey::static_pubkey!
("9vMJfxuKxXBoEa7rM12mYLMwTacLMLDJqHozw96WQL8i");
pub const STABLE_MINTS: [Pubkey; 2] = [USDC_MAINNET, UST_MAINNET];
```

Recommendation

The current price of UST does not represent the properties of a stable token. We recommend re-confirming UST using UST as a dependency will not cause any regressions or future problems. If it does, it should be removed from as a dependency.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-03 MISSING VALIDATION FOR `FarmRewardInfo::update` AND `ChangeTokenPerSecondMulti::change_token_per_second`

Category	Severity	Location	Status
Control Flow	Medium	programs/index-yield-farming/src/processors.rs: 157, 764, 814, 927; programs/index-yield-farming/src/states.rs: 62	Resolved

Description

The function `change_token_per_second` updates the farms `token_per_second` parameter. The function provides no validation for when the function should be called.

```
impl<'info> ChangeTokenPerSecondMulti<'info>{
    pub fn change_token_per_second(&mut self,
        token_per_seconds: Vec<u128>
    ) -> Result<()> {
        let farm = &mut self.farm;
        let farm_key = farm.key();

        farm.validate_address(farm_key)?;

        let asset_count:u8 = token_per_seconds.len().try_into().unwrap();
        require!( asset_count == farm.asset_count, ErrorCode::InvalidAssetCount);

        let mut index:usize = 0;
        for token_per_second in token_per_seconds.iter() {
            farm.reward_infos[index].token_per_second = *token_per_second;
            index += 1;
        }

        Ok(())
    }
}
```

In the `state.rs`, the `update()` function will calculate the rewards that have been generated for a farm.

```
pub fn update<'info>(&mut self, clock: &Sysvar<'info, Clock>) -> Result<()> {
      let seconds = u128::try_from(
          clock
              .unix_timestamp
              .checked_sub(self.last_reward_time)
              .unwrap(),
      )
      .unwrap();
      let mut reward_per_share: u128 = 0;
      if self.amount > 0 && seconds > 0 {
          reward_per_share = u128::from(self.token_per_second)
              .checked_mul(seconds)
              .unwrap()
              .checked_mul(ACC_PRECISION)
              .unwrap()
              .checked_div(u128::from(self.amount))
              .unwrap();
      }
      self.acc_reward_per_share = self
          .acc_reward_per_share
          .checked_add(reward_per_share)
          .unwrap();

      self.last_reward_time = clock.unix_timestamp;

      Ok(())
  }
```

Neither `update` nor `change_token_per_second` have any validation for *when* `token_per_second` should be changed. In practice, this could mean someone calling `change_token_per_second` prior to when they are supposed to, which would result in unexpected results when calling `update`. The logic of when `token_per_second` should change is totally off-chain and in the hands of the owner, without any validation. Both human error and malicious intent are risks with the current architecture of this function.

Recommendation

We recommend adding validation logic to the functions to ensure all changes to `tokens_per_second` happen as expected. Ideally, reward structures should be encoded on-chain.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-04 | LACK OF `Asset` LENGTH LIMIT CHECK

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/constants.rs: 13~14; programs/index-yield-farming/src/processors.rs: 125~131	Resolved

Description

`MAX_ASSET_COUNT` is used to determine the total size that is to be allocated for accounts in `create_or_allocate_account_raw`.

```
pub const MAX_ASSET_COUNT:usize = 10;
```

However, when this upper limit is not checked when allocating `asset_infos` in `farm`. This may result in the allocated space of the account being depleted.

```
for i in 0..asset_size {
            total_weight += u128::from(weights[i]);
            farm.asset_infos.push(FarmAssetInfo {
                weight: weights[i],
                mint: mints[i],
            })
        }
```

Recommendation

We recommend ensuring that the allocated account space cannot be unexpectedly used up.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-05 | LACK OF MINT VALIDATION FOR USER TOKEN ACCOUNTS

Category	Severity	Location	Status
Logical Issue	Minor	programs/index-yield-farming/src/contexts.rs: 461~468; programs/index-yield-farming/src/processors.rs: 372, 458, 465, 473, 622~624, 680~682, 727~728, 792, 854; programs/index-yield-farming/src/utils.rs: 582~597, 601~616	Resolved

Description

`Mint` is the only basis for identifying different types of tokens in Solana, and it is important to make sure that the mint in the a given Token Accounts data, which is input by the user, is correct.

The current program only checks whether the address of the Token Account is constructed by the right seeds and program id, via the `check_token_account` function.

It is best practice to include mint and ownership checks when interacting with the SPL Token program.

For example, a validation that guarantees the Token Accounts mint address is the same as the one expected in the `strategy`.

```
require!(token_account.mint == strategy.src_mint, ErrorCode::CustomErrorCode);
```

This would guarantee the token account was not created incorrectly.

Recommendation

We recommend adding a check for `mint` and `owner` in token_account's data, instead of just checking the `Pubkey` generated by `seeds`, `bump` and `program_id`.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

COE-01 | REDUCE THE USE OF `std:mem:size_of()`

Category	Severity	Location	Status
Language Specific	Informational	programs/index-yield-farming/src/contexts.rs: 71, 206, 256~257	Resolved

Description

`mem::size_of<T>()` should be used for size calculations with caution. Borsch will always serialize an option as 1 byte for the variant identifier and then additional x bytes for the content if it's Some. However, Rust uses null-pointer optimization to make Option's variant identifier 0 bytes when it can, so an option is sometimes just as big as its contents, such as with `Sign`.

Recommendation

We recommend calculating the space in such situations manually, as this can prevent future problems in the future.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

COE-02 | TYPO

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/contexts.rs: 103, 105, 108, 111, 114, 117, 123, 127, 166, 218, 221, 327, 508	Resolved

Description

The comment includes a spelling mistake of `further`.

Recommendation

We recommend fixing typos in comments.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

LIR-02 | INCORRECT USE OF '_' SYNTAX FOR UNUSED VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/lib.rs: 33, 40, 48, 64, 71, 78, 87, 94, 104, 111, 117, 125, 133, 139, 146, 151, 158, 164	Partially Resolved

Description

In Rust, it is standard practice to prefix unused parameters with `_`. However `_ctx` is used in all the referenced functions.

Recommendation

We recommend following standard Rust coding style.

Alleviation

`[Certik]`: The team heeded the advice and partially resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7 >.

PRC-06 | SIMPLIFIED IMPLEMENTATION OF `index` IN LOOP

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/processors.rs: 170~173, 341~351	Resolved

Description

The code in the following loop uses `index`, but it can be further simplified.

```
341         for asset_info in farm.asset_infos.iter(){
342             user.asset_infos[index].last_amount = u128::from(new_amount)
343                 .checked_mul(u128::from(asset_info.weight)).unwrap()
344                 .checked_div(farm.total_weight).unwrap()
345                 .try_into().unwrap();
346             user.asset_infos[index].last_action = Action::FundAction as u8;
347
348             require!(rest_amount >= user.asset_infos[index].last_amount,
ErrorCode::IntegerUnderflow);
349             rest_amount =
rest_amount.checked_sub(user.asset_infos[index].last_amount).unwrap();
350             index += 1;
351         }
```

```
        for token_per_second in token_per_seconds.iter() {
            farm.reward_infos[index].token_per_second = *token_per_second;
            index += 1;
        }
```

Recommendation

Consider using Rusts enumerate functionality to reduce the need for an additional mutable variable.

```
for (index, asset_info) in farm.asset_infos.iter().enumerate()
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-07 | ADD NON-ZERO CHECK FOR `total_weight`

Category	Severity	Location	Status
Logical Issue	Informational	programs/index-yield-farming/src/processors.rs: 133~134	Resolved

Description

`total_weight` will be used in the future to assign `amount` to different `asset_info.last_amount`.

```
user.asset_infos[index].last_amount = u128::from(new_amount)
                .checked_mul(u128::from(asset_info.weight))
                .unwrap()
                .checked_div(farm.total_weight)
                .unwrap()
                .try_into()
                .unwrap();
```

But the instruction doesn't have a validation over the weights and the total weight. It is important to make sure that `total_weight` is not equal to 0 in order to prevent the construction of unavailable `farm` accounts.

```
    pub fn create_farm(&mut self,
      ...
    ) -> Result<()> {
        for i in 0  .. asset_size{
            total_weight += u128::from(weights[i]);
            ...
        }
      farm.total_weight = total_weight;
    }
```

Recommendation

We suggest adding a check in `create_farm` that `total_weight` is not equal to zero.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

PRC-08 | THIRD PARTY DEPENDENCIES

Category	Severity	Location	Status
Volatile Code	Informational	programs/index-yield-farming/src/processors.rs: 486~500	Acknowledged

Description

The contract is serving as the underlying entity to interact with third party `QuarryProtocol`, `spl-swap` and `stable-swap` protocols. The scope of the audit treats 3rd party entities as black boxes and assume their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Recommendation

We understand that the business logic of Hawksight requires interaction with `QuarryProtocol`, `spl-swap` and `stable-swap` etc. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed. We recommend including rigorous tests. This will help identify when there are breaking changes in third party libraries. See `Lack of Tests` finding.

Alleviation

`[Hawksight]`: we acknowledged and we'll continue to monitor 3rd party contract integrations.

PRC-09 | `last_amount` NOT RESET TO ZERO

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/processors.rs: 424, 925, 925	Resolved

Description

After withdrawing reward, the user's `last_action` is set as `FinishAction`, however the users `last_amount` is not reset to 0 after processing. For comparison, `stake_to_farm()` and `redeem_stable_token()`, after a `FinishAction` reset the `last_amount` to zero. This could cause undefined behavior in the future.

For example

```
919 pub fn withdraw_reward(&mut self) -> Result<()> {
920     ...
921     for asset_index in 0..usize::from(farm.asset_count) {
922         ...
923         user_asset.calculate_reward_debt(&farm_reward)?;
924         user_asset.last_action = Action::FinishAction as u8;
925         user_asset.last_amount = 0; // reset as zero before FinishAction
926     }
927 ...
928 }
929
```

Recommendation

We recommend confirming the `last_amount` should be set to 0 after the `FinishAction` in `withdraw_reward()`.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-07 | UNNECESSARY & REFERENCE

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/processors.rs: 83, 96, 765, 770, 815, 820, 928, 933; programs/index-yield-farming/src/states.rs: 121; programs/index-yield-farming/src/utils.rs: 118~119, 130, 136, 619	Resolved

Description

The references on the linked lines would be dereferenced immediately by the compiler, so the borrow operations are unnecessary.

Recommendation

We suggest that the receiver of the expression borrows the expression.

For example:

```
Pubkey::find_program_address(&seeds, &program_id);
```

could be written as

```
Pubkey::find_program_address(seeds, &program_id);
```

Alleviation

`[CertiK]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-08 | UNNECESSARY CONVERSION TO THE SAME TYPE

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/processors.rs: 319, 328, 337, 403, 412, 421, 949, 959, 970; programs/index-yield-farming/src/states.rs: 72, 169	Resolved

Description

The references on the linked lines make unnecessary conversions to the same type.

Recommendation

Statements such as `*.try_into()` and `u128::from()` are only necessary when doing type conversions. The linked functions will compile without the conversions.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-09 | REMOVE COMMENTED CODE

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/contexts.rs: 54~56, 59, 505; programs/index-yield-farming/src/errors.rs: 18~19, 33~46; programs/index-yield-farming/src/events.rs: 15~32, 44~88; programs/index-yield-farming/src/processors.rs: 63, 109~110, 183~187, 297, 479, 527~531, 558~560, 567, 579, 590, 611, 676, 756, 845~847, 868~870; programs/index-yield-farming/src/states.rs: 198~202; programs/index-yield-farming/src/utils.rs: 25, 38~45, 230~272, 321, 326, 356, 499~501, 540~541, 609~610	Partially Resolved

Description

The linked statements do not affect the functionality of the codebase and appear to be either leftovers from test code or older functionality.

Recommendation

We advise all code comments are removed in production code before deployment.

Alleviation

`[CertiK]`: The team heeded the advice and partially resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

SRI-10 | UNNECESSARY ACCOUNT

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/contexts.rs: 355~437; programs/index-yield-farming/src/processors.rs: 281~282	Resolved

Description

The `CreateUserToken` struct declares a `strategy` account, however the only field used in that account is the token mint. Passing around unnecessary data can increase complexity of the program, as well as computational requirements.

```
pub struct CreateUserToken<'info> {
    #[account(
        seeds = [
            b"saber-farm-strategy".as_ref(),
            strategy.token_mint.as_ref(),
            strategy.src_mint.as_ref(),
            strategy.dst_mint.as_ref(),
            strategy.lp_mint.as_ref(),
            strategy.spl_swap.as_ref(),
            strategy.saber_swap.as_ref(),
            strategy.quarry.as_ref(),
        ],
        bump,
    )]
    pub strategy: Account<'info, SaberFarmStrategy>,

  // ...

}
```

```
  pub fn create_user_token(&mut self) -> Result<()> {
      let farm = &mut self.farm;
      let strategy = &mut self.strategy;

      let farm_key = farm.key();

      farm.validate_address(farm_key)?;
      farm.find_asset(strategy.token_mint)?; // Used here

      Ok(())
  }
```

Recommendation

We recommend reviewing the design of this code path. Reducing code complexity reduces the risk of bugs and improves compute utilization.

Alleviation

`[Certik]` : The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

STT-01 | SIMPLIFIABLE `require` OPERATION

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/states.rs: 137~138	Resolved

Description

In `require!` macro, checking the `x == true` expression is redundant.

```
require!(found == true, ErrorCode::AssetNotFound);
```

Recommendation

We suggest using the following example to simplify the code.

```
require!(found, ErrorCode::AssetNotFound);
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

UTL-01 | UNNECESSARY `return` STATEMENT

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/utils.rs: 592~593, 621	Resolved

Description

There is an unnecessary `return` statement in the current code base.

```
if exp_token_address != address {
  return
Err(Error::ProgramError(ProgramErrorWithOrigin::from(ProgramError::InvalidArgument))
);
}
```

Recommendation

The Rust standard syntax for returning from a function is to not add a ';' to the end of the line. We recommend following standard Rust code styles.

In this case, the line could be simplified to:

```
if exp_token_address != address {

Err(Error::ProgramError(ProgramErrorWithOrigin::from(ProgramError::InvalidArgument))
)
}
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

UTL-02 | UNUSED VARIABLE

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/utils.rs: 550	Resolved

Description

The variable `_res` is declared but is not used in the code logic.

```
550  let _res = redeem_all_tokens_from_mint_proxy(CpiContext::new(...));
```

Recommendation

All unused variables should be removed from production code before deploying.

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

UTL-03 | UNNECESSARY RE-SLICING

Category	Severity	Location	Status
Coding Style	Informational	programs/index-yield-farming/src/utils.rs: 222, 333, 393, 435, 479, 546, 570	Resolved

Description

In the `utils.rs`, `authority_seeds` was re-sliced, which was unnecessary:

```
with_signer(&[&authority_seeds[..]])
```

Recommendation

Since the `authority_seeds` value is already a slice, we recommended passing it by value instead of re-slicing it for the entire range:

```
with_signer(&[authority_seeds])
```

Alleviation

`[CertiK]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

UTL-04 OPTIMIZE `creator_fee` CALCULATION FOR IMPROVED PRECISION

Category	Severity	Location	Status
Logical Issue	Informational	programs/index-yield-farming/src/utils.rs: 79~99	Resolved

Description

The current computation of `creator_fee` does not conform to the multiply first, divide later specification. This will result in a loss of precision in the final value.

```
    let fee_amount = u128::from(input_amount)
        .checked_mul(fee_pct)
        .unwrap()
        .checked_div(FEE_DENOMINATOR)
        .unwrap();

    let creator_fee = fee_amount
        .checked_mul(CREATOR_FEE_WEIGHT)
        .unwrap()
        .checked_div(...).unwrap()
        .unwrap();
```

Recommendation

In order to improve the accuracy of the calculation of the `creator_fee`, we suggest to calculate creator_fee directly.

```
    let creator_fee = u128::from(input_amount)
        .checked_mul(fee_pct)
        .unwrap()
        .checked_mul(CREATOR_FEE_WEIGHT)
        .unwrap()
        .checked_div(FEE_DENOMINATOR)
        .unwrap()
        .checked_div(...).unwrap())
        .unwrap();
```

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

OPTIMIZATIONS | HAWKSIGHT

ID	Title	Category	Severity	Status
SRI-06	Removal Of Unnecessary Checks For Computing Budget Optimization	Gas Optimization	Optimization	Resolved

SRI-06 REMOVAL OF UNNECESSARY CHECKS FOR COMPUTING BUDGET OPTIMIZATION

Category	Severity	Location	Status
Gas Optimization	Optimization	programs/index-yield-farming/src/contexts.rs: 84, 97, 453, 501; programs/index-yield-farming/src/processors.rs: 342~350, 372~373	Resolved

Description

There are some unnecessary checks in the current code. For example

- **Already checked by** `check_sub`

```
user.asset_infos[index].last_amount = u128::from(new_amount)
                .checked_mul(u128::from(asset_info.weight))
                .unwrap()
                .checked_div(farm.total_weight)
                .unwrap()
                .try_into()
                .unwrap();
            user.asset_infos[index].last_action = Action::FundAction as u8;
            require!(
                rest_amount >= user.asset_infos[index].last_amount,
                ErrorCode::IntegerUnderflow
            );//Duplicate check
            rest_amount = rest_amount
                .checked_sub(user.asset_infos[index].last_amount)
                .unwrap();
            index += 1;
```

- **Already checked by** `#[account(mut,seeds = [..], bump)]`

```
check_token_account(
            self.user_pda_stable_token.key(),
            farm.stable_mint,
            farm_key,
            user_key,
        )?;
```

- **Already checked by** `Program<'info, Token>`

```
#[account(constraint = token_program.key == &token::ID)]
```

Recommendation

We recommend removing duplicate checks.

Alleviation

`[Certik]`: The team heeded the advice and resolved the finding in the commit hash <b70e6a8cc80807ac5af92700da6600447242bce7>.

APPENDIX | HAWKSIGHT

Details on Formal Verification

Technical description

Some Solidity smart contracts from this project have been formally verified using symbolic model checking. Each such contract was compiled into a mathematical model which reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The model also formalizes a simplified execution environment of the Ethereum blockchain and a verification harness that performs the initialization of the contract and all possible interactions with the contract. Initially, the contract state is initialized non-deterministically (i.e. by arbitrary values) and over-approximates the reachable state space of the contract throughout any actual deployment on chain. All valid results thus carry over to the contract's behavior in arbitrary states after it has been deployed.

Assumptions and simplifications

The following assumptions and simplifications apply to our model:

- Gas consumption is not taken into account, i.e. we assume that executions do not terminate prematurely because they run out of gas.
- The contract's state variables are non-deterministically initialized before invocation of any of those functions. That ignores contract invariants and may lead to false positives. It is, however, a safe over-approximation.
- The verification engine reasons about unbounded integers. Machine arithmetic is modeled as operations on the congruence classes arising from the bit-width of the underlying numeric type. This ensures that over- and underflow characteristics are faithfully represented.
- Certain low-level calls and inline assembly are not supported and may lead to an ERC-20 token contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property definitions

All properties are expressed in linear temporal logic (LTL). For that matter, we treat each invocation of and each return from a public or an external function as a discrete time steps. Our analysis reasons about the contract's state upon entering and upon leaving public or external functions.

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `started(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond`.

- `willSucceed(f, [cond])` Indicates an invocation of contract function `f` within a state satisfying formula `cond` and considers only those executions that do not revert.
- `finished(f, [cond])` Indicates that execution returns from contract function `f` in a state satisfying formula `cond`. Here, formula `cond` may refer to the contract's state variables and to the value they had upon entering the function (using the `old` function).
- `reverted(f, [cond])` Indicates that execution of contract function `f` was interrupted by an exception in a contract state satisfying formula `cond`.

The verification performed in this audit operates on a harness that non-deterministically invokes a function of the contract's public or external interface. All formulas are analyzed w.r.t. the trace that corresponds to this function invocation.

Description of ERC-20 Properties

The specifications are designed such that they capture the desired and admissible behaviors of the ERC-20 functions `transfer`, `transferFrom`, `approve`, `allowance`, `balanceOf`, and `totalSupply`.

In the following, we list those property specifications.

Properties for ERC-20 function `transfer`

erc20-transfer-revert-zero

Function `transfer` Prevents Transfers to the Zero Address.

Any call of the form `transfer(recipient, amount)` must fail if the recipient address is the zero address.

Specification:

```
[](started(contract.transfer(to, value), to == address(0))
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-succeed-normal

Function `transfer` Succeeds on Admissible Non-self Transfers.

All invocations of the form `transfer(recipient, amount)` must succeed and return `true` if

- the `recipient` address is not the zero address,
- `amount` does not exceed the balance of address `msg.sender`,
- transferring `amount` to the `recipient` address does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to != msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[to] + value <= type(uint256).max && _balances[to] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-succeed-self

Function `transfer` Succeeds on Admissible Self Transfers.

All self-transfers, i.e. invocations of the form `transfer(recipient, amount)` where the `recipient` address equals the address in `msg.sender` must succeed and return `true` if

- the value in `amount` does not exceed the balance of `msg.sender` and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transfer(to, value), to != address(0)
          && to == msg.sender && value >= 0 && value <= _balances[msg.sender]
          && _balances[msg.sender] >= 0
          && _balances[msg.sender] <= type(uint256).max)
          ==> <>(finished(contract.transfer(to, value), return)))
```

erc20-transfer-correct-amount

Function `transfer` Transfers the Correct Amount in Non-self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must subtract the value in `amount` from the balance of `msg.sender` and add the same value to the balance of the `recipient` address.

Specification:

```
[](willSucceed(contract.transfer(to, value), to != msg.sender
   && _balances[to] >= 0 && value >= 0
   && _balances[to] + value <= type(uint256).max
   && _balances[msg.sender] >= 0 && _balances[msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transfer(to, value), return
             ==> _balances[msg.sender] == old(_balances[msg.sender]) - value
                 && _balances[to] == old(_balances[to]) + value)))
```

erc20-transfer-correct-amount-self

Function `transfer` Transfers the Correct Amount in Self Transfers.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` and where the `recipient` address equals `msg.sender` (i.e. self-transfers) must not change the balance of address `msg.sender`.

Specification:

```
[](willSucceed(contract.transfer(to, value), to == msg.sender
  && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
     ==> <>(finished(contract.transfer(to, value), return
         ==> _balances[to] == old(_balances[to]))))
```

erc20-transfer-change-state

Function `transfer` Has No Unexpected State Changes.

All non-reverting invocations of `transfer(recipient, amount)` that return `true` must only modify the balance entries of the `msg.sender` and the `recipient` addresses.

Specification:

```
[](willSucceed(contract.transfer(to, value), p1 != msg.sender && p1 != to)
  ==> <>(finished(contract.transfer(to, value), return
      ==> (_totalSupply == old(_totalSupply) && _allowances == old(_allowances)
           && _balances[p1] == old(_balances[p1])  ))))
```

erc20-transfer-exceed-balance

Function `transfer` Fails if Requested Amount Exceeds Available Balance.

Any transfer of an amount of tokens that exceeds the balance of `msg.sender` must fail.

Specification:

```
[](started(contract.transfer(to, value), value > _balances[msg.sender]
   && _balances[msg.sender] >= 0 && value <= type(uint256).max)
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return)))
```

erc20-transfer-recipient-overflow

Function `transfer` Prevents Overflows in the Recipient's Balance.

Any invocation of `transfer(recipient, amount)` must fail if it causes the balance of the `recipient` address to overflow.

Specification:

```
[](started(contract.transfer(to, value), to != msg.sender
   && _balances[to] + value > type(uint256).max
   && _balances[to] >= 0 && _balances[to] <= type(uint256).max
   && _balances[msg.sender] <= type(uint256).max
   && value > 0 && value <= _balances[msg.sender])
   ==> <>(reverted(contract.transfer) || finished(contract.transfer(to, value),
          !return) || finished(contract.transfer(to, value), _balances[to]
                      > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transfer-false

If Function `transfer` Returns `false`, the Contract State Has Not Been Changed.

If the `transfer` function in contract `contract` fails by returning `false`, it must undo all state changes it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.transfer(to, value))
   ==> <>(finished(contract.transfer(to, value), !return]
   ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                    && _allowances == old(_allowances)  ))))
```

erc20-transfer-never-return-false

Function `transfe` Never Returns `false`.

The transfer function must never return `false` to signal a failure.

Specification:

```
[](!(finished(contract.transfer, !return)))
```

Properties for ERC-20 function `transferFrom`

erc20-transferfrom-revert-from-zero

Function `transferFrom` Fails for Transfers From the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `from` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), from == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-revert-to-zero

Function `transferFrom` Fails for Transfers To the Zero Address.

All calls of the form `transferFrom(from, dest, amount)` where the `dest` address is zero, must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), to == address(0))
   ==> <>(reverted(contract.transferFrom) || finished(contract.transferFrom,
          !return)))
```

erc20-transferfrom-succeed-normal

Function `transferFrom` Succeeds on Admissible Non-self Transfers. All invocations of `transferFrom(from, dest, amount)` must succeed and return `true` if

- the value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`,
- transferring a value of `amount` to the address in `dest` does not lead to an overflow of the recipient's balance, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && to != address(0) && from != to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && _balances[to] + value <= type(uint256).max
   && value >= 0 && _balances[to] >= 0 && _balances[from] >= 0
   && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-succeed-self

Function `transferFrom` Succeeds on Admissible Self Transfers.

All invocations of `transferFrom(from, dest, amount)` where the `dest` address equals the `from` address (i.e. self-transfers) must succeed and return `true` if:

- The value of `amount` does not exceed the balance of address `from`,
- the value of `amount` does not exceed the allowance of `msg.sender` for address `from`, and
- the supplied gas suffices to complete the call.

Specification:

```
[](started(contract.transferFrom(from, to, value), from != address(0)
   && from == to && value <= _balances[from]
   && value <= _allowances[from][msg.sender]
   && value >= 0 && _balances[from] <= type(uint256).max
   && _allowances[from][msg.sender] <= type(uint256).max)
     ==> <>(finished(contract.transferFrom(from, to, value), return)))
```

erc20-transferfrom-correct-amount

Function `transferFrom` Transfers the Correct Amount in Non-self Transfers.

All invocations of `transferFrom(from, dest, amount)` that succeed and that return `true` subtract the value in `amount` from the balance of address `from` and add the same value to the balance of address `dest`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from != to && value >= 0
&& _balances[from] >= 0 && _balances[from] <= type(uint256).max
&& _balances[to] >= 0 && _balances[to] + value <= type(uint256).max)
   ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> _balances[from] == old(_balances[from]) - value
              && _balances[to] == old(_balances[to] + value))))
```

erc20-transferfrom-correct-amount-self

Function `transferFrom` Performs Self Transfers Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` and where the address in `from` equals the address in `dest` (i.e. self-transfers) do not change the balance entry of the `from` address (which equals `dest`).

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), from == to
   && value >= 0 && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
             ==> _balances[from] == old(_balances[from]))))
```

erc20-transferfrom-correct-allowance

Function `transferFrom` Updated the Allowance Correctly.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` must decrease the allowance for address `msg.sender` over address `from` by the value in `amount`.

Specification:

```
[](willSucceed(contract.transferFrom(from, to, value), value >= 0
   && value <= type(uint256).max && _balances[from] >= 0
   && _balances[from] <= type(uint256).max && _balances[to] >= 0
   && _balances[to] <= type(uint256).max && _allowances[from][msg.sender] >= 0
   && _allowances[from][msg.sender] <= type(uint256).max)
      ==> <>(finished(contract.transferFrom(from, to, value), return
          ==> ((_allowances[from][msg.sender]
                == old(_allowances[from][msg.sender]) - value)
               || (_allowances[from][msg.sender]
                   == old(_allowances[from][msg.sender])
                      && (from == msg.sender
                           || old(_allowances[from][msg.sender])
                              == type(uint256).max))))))
```

erc20-transferfrom-change-state

Function `transferFrom` Has No Unexpected State Changes.

All non-reverting invocations of `transferFrom(from, dest, amount)` that return `true` may only modify the following state variables:

- The balance entry for the address in `dest`,
- The balance entry for the address in `from`,
- The allowance for the address in `msg.sender` for the address in `from`. Specification:

```
[](willSucceed(contract.transferFrom(from, to, amount), p1 != from && p1 != to
   && (p2 != from || p3 != msg.sender))
    ==> <>(finished(contract.transferFrom(from, to, amount), return
     ==> (_totalSupply == old(_totalSupply) && _balances[p1] == old(_balances[p1])
         && _allowances[p2][p3] == old(_allowances[p2][p3])  ))))
```

erc20-transferfrom-fail-exceed-balance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the balance of address `from` must fail.

Specification:

```
[](started(contract.transferFrom(from, to, value), value > _balances[from]
   && _balances[from] >= 0 && _balances[from] <= type(uint256).max)
       ==> <>(reverted(contract.transferFrom)
              || finished(contract.transferFrom, !return)))
```

erc20-transferfrom-fail-exceed-allowance

Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance.

Any call of the form `transferFrom(from, dest, amount)` with a value for `amount` that exceeds the allowance of address `msg.sender` must fail.

Specification:

```
  [](started(contract.transferFrom(from, to, value), value > _allowances[from]
[msg.sender]
     && _allowances[from][msg.sender] >= 0 && value <= type(uint256).max)
     ==> <>(reverted(contract.transferFrom)
           || finished(contract.transferFrom(from, to, value), !return)
           || finished(contract.transferFrom(from, to, value), return
              && (msg.sender == from
                  || _allowances[from][msg.sender] == type(uint256).max))))
```

erc20-transferfrom-fail-recipient-overflow

Function `transferFrom` Prevents Overflows in the Recipient's Balance.

Any call of `transferFrom(from, dest, amount)` with a value in `amount` whose transfer would cause an overflow of the balance of address `dest` must fail.

Specification:

```
  [](started(contract.transferFrom(from, to, value), from != to
     && _balances[to] + value > type(uint256).max && value <= type(uint256).max
     && _balances[to] >= 0 && _balances[to] <= type(uint256).max)
        ==> <>(reverted(contract.transferFrom)
            || finished(contract.transferFrom(from, to, value), !return)
            || finished(contract.transferFrom(from, to, value), _balances[to]
                > old(_balances[to]) + value - type(uint256).max - 1)))
```

erc20-transferfrom-false

If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed.

If `transferFrom` returns `false` to signal a failure, it must undo all incurred state changes before returning to the caller.

Specification:

```
  [](willSucceed(contract.transfer(to, value))
     ==> <>(finished(contract.transfer(to, value), !return
     ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
                                      && _allowances == old(_allowances)  ))))
```

erc20-transferfrom-never-return-false

Function `transferFrom` Never Returns `false`.

The `transferFrom` function must never return `false`.

Specification:

```
[](!(finished(contract.transferFrom, !return)))
```

Properties related to function `totalSupply`

erc20-totalsupply-succeed-always

Function `totalSupply` Always Succeeds.

The function `totalSupply` must always succeeds, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.totalSupply) ==> <>(finished(contract.totalSupply)))
```

erc20-totalsupply-correct-value

Function `totalSupply` Returns the Value of the Corresponding State Variable.

The `totalSupply` function must return the value that is held in the corresponding state variable of contract contract.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, return == _totalSupply)))
```

erc20-totalsupply-change-state

Function `totalSupply` Does Not Change the Contract's State.

The `totalSupply` function in contract contract must not change any state variables.

Specification:

```
[](willSucceed(contract.totalSupply)
   ==> <>(finished(contract.totalSupply, _totalSupply == old(_totalSupply)
          && _balances == old(_balances) && _allowances == old(_allowances)  )))
```

Properties related to function `balanceOf`

erc20-balanceof-succeed-always

Function `balanceOf` Always Succeeds.

Function `balanceOf` must always succeed if it does not run out of gas.

Specification:

```
[](started(contract.balanceOf) ==> <>(finished(contract.balanceOf)))
```

erc20-balanceof-correct-value

Function `balanceOf` Returns the Correct Value.

Invocations of `balanceOf(owner)` must return the value that is held in the contract's balance mapping for address `owner`.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), return == _balances[owner])))
```

erc20-balanceof-change-state

Function `balanceOf` Does Not Change the Contract's State.

Function `balanceOf` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.balanceOf)
   ==> <>(finished(contract.balanceOf(owner), _totalSupply == old(_totalSupply)
          && _balances == old(_balances)
          && _allowances == old(_allowances)  )))
```

Properties related to function `allowance`

erc20-allowance-succeed-always

Function `allowance` Always Succeeds.

Function `allowance` must always succeed, assuming that its execution does not run out of gas.

Specification:

```
[](started(contract.allowance) ==> <>(finished(contract.allowance)))
```

erc20-allowance-correct-value

Function `allowance` Returns Correct Value.

Invocations of `allowance(owner, spender)` must return the allowance that address `spender` has over tokens held by address `owner`.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
           return == _allowances[owner][spender])))
```

erc20-allowance-change-state

Function `allowance` Does Not Change the Contract's State.

Function `allowance` must not change any of the contract's state variables.

Specification:

```
[](willSucceed(contract.allowance(owner, spender))
   ==> <>(finished(contract.allowance(owner, spender),
   _totalSupply == old(_totalSupply) && _balances == old(_balances)
               && _allowances == old(_allowances)  )))
```

Properties related to function `approve`

erc20-approve-revert-zero

Function `approve` Prevents Giving Approvals For the Zero Address.

All calls of the form `approve(spender, amount)` must fail if the address in `spender` is the zero address.

Specification:

```
[](started(contract.approve(spender, value), spender == address(0))
   ==> <>(reverted(contract.approve)
          || finished(contract.approve(spender, value), !return)))
```

erc20-approve-succeed-normal

Function `approve` Succeeds for Admissible Inputs.

All calls of the form `approve(spender, amount)` must succeed, if

- the address in `spender` is not the zero address and
- the execution does not run out of gas.

Specification:

```
[](started(contract.approve(spender, value), spender != address(0))
   ==> <>(finished(contract.approve(spender, value), return)))
```

erc20-approve-correct-amount

Function `approve` Updates the Approval Mapping Correctly.

All non-reverting calls of the form `approve(spender, amount)` that return `true` must correctly update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount`.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && value >= 0 && value <= type(uint256).max)
       ==> <>(finished(contract.approve(spender, value), return
              ==> _allowances[msg.sender][spender] == value)))
```

erc20-approve-change-state

Function `approve` Has No Unexpected State Changes.

All calls of the form `approve(spender, amount)` must only update the allowance mapping according to the address `msg.sender` and the values of `spender` and `amount` and incur no other state changes.

Specification:

```
[](willSucceed(contract.approve(spender, value), spender != address(0)
   && (p1 != msg.sender || p2 != spender))
       ==> <>(finished(contract.approve(spender, value), return
              ==> _totalSupply == old(_totalSupply) && _balances == old(_balances)
                  && _allowances[p1][p2] == old(_allowances[p1][p2])  )))
```

erc20-approve-false

If Function `approve` Returns `false`, the Contract's State Has Not Been Changed.

If function `approve` returns `false` to signal a failure, it must undo all state changes that it incurred before returning to the caller.

Specification:

```
[](willSucceed(contract.approve(spender, value))
   ==> <>(finished(contract.approve(spender, value), !return
          ==> (_balances == old(_balances) && _totalSupply == old(_totalSupply)
               && _allowances == old(_allowances)  ))))
```

erc20-approve-never-return-false

Function `approve` Never Returns `false`.

The function `approve` must never returns `false`.

Specification:

```
[](!(finished(contract.approve, !return)))
```

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

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 considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report 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 in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing 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. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK'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 CertiK is subject to dependencies and 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.