Oro Inti

Security Review

Cantina Managed review by:

FrankCastle, Security Researcher

0xHuy0512, Associate Security Researcher

Jdiggidy, Associate Security Researcher

April 17, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Critical Risk . . . 4
  - 3.1.1 Incorrect init Constraint in `toggle_liquid` Causes Fund Locking and DoS . . . 4
  - 3.1.2 Wrong price calculation from Gold to USDC . . . 4
  - 3.1.3 `claim()` and `unstake()` functions will not be useable . . . 5
  - 3.1.4 Missing seed and bump constraints in position account in `claim()` and `unstake()` functions . . . 5
  - 3.1.5 Multiple `unstake()` calls at the same position can drain the vault . . . 6
  - 3.1.6 Incorrect Reward token calculation enables pool drain via arbitrage . . . 6
  - 3.1.7 Incorrect token amount calculation in liquid unstaking leads to asset loss . . . 6
  - 3.1.8 Missing mutable account aonstraint prevents state updates . . . 7
  - 3.1.9 Excessive Payout in unstake and claim Functions Causes Severe Fund Loss . . . 8
  - 3.1.10 Incorrect Time Validation in `unstake` Function leads to permanent lock of the staked funds . . . 8
  - 3.1.11 Fix review Finding: Incorrect Calculation of `amount_to_burn` in `liquid_unstake` . . . 9
- 3.2 High Risk . . . 10
  - 3.2.1 DoS in `claim` Function Due to `init` Constraint on an Already Initialized Account . . . 10
- 3.3 Medium Risk . . . 10
  - 3.3.1 Missing Slippage Parameter Exposes Users to Unintended Prices, Leading to Potential Fund Loss . . . 10
  - 3.3.2 Missing Confidence Validation in Pyth Oracle Price . . . 11
  - 3.3.3 User can unstake anytime instead of getting locked for 1 year . . . 11
  - 3.3.4 Pyth price feed maximum age too high allows stale price data . . . 12
  - 3.3.5 Fix review Finding: Admin Can Reset Critical Configuration by Reinitializing the Contract 12
- 3.4 Low Risk . . . 13
  - 3.4.1 Broken Access Control in Vault Initialization . . . 13
  - 3.4.2 Update price definition from `u64` to `i64` to accommodate for the possibility of a negative price . . . 14
  - 3.4.3 Hard Coded Values leading to maintenance issues . . . 14
- 3.5 Informational . . . 14
  - 3.5.1 Missing Events for Storage-Mutating Functions . . . 14
  - 3.5.2 Reversible Whitelisting Mechanism should be implemented . . . 14
  - 3.5.3 Not Checking for potential overflows . . . 15
  - 3.5.4 The Comment AUX/USD should be XAU/USD . . . 15
  - 3.5.5 Create a robust runnable test suite . . . 15

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

From Feb 27th to Mar 4th the Cantina team conducted a review of oro-inti on commit hash 5ea0803a. The team identified a total of **25** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	11	11	0
High Risk	1	1	0
Medium Risk	5	5	0
Low Risk	3	3	0
Gas Optimizations	0	0	0
Informational	5	5	0
Total	**25**	**25**	**0**

3 Findings

3.1 Critical Risk

3.1.1 Incorrect init Constraint in `toggle_liquid` Causes Fund Locking and DoS

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `toggle_liquid` function is completely frozen because it uses the `init` constraint with the `config` account. Since the `init` constraint is already used in the `initialize_config` function, any attempt to call `toggle_liquid` will always revert.

This results in two critical issues affecting the protocol's functionality:

1. Fund Locking: If `toggle_liquid` is responsible for initialization, the bump seeds used for signing the transfer CPI will not be stored hence the signing will not work permanently. This will lead to a complete lock of funds for all tokens staked by users.
2. Denial of Service: If `initialize_config` is used for initialization, the `liquid_staking` program will be permanently disabled because the `liquid` field in the `config` account will always remain `false`. As a result, the `liquid_stake` function will always revert, preventing staking operations.

Recommendation: Modify the `config` account in the `toggle_liquid` function to avoid reinitialization. The corrected implementation is as follows:

```
**[account(:** mut,
    seeds = [b"config".as_ref()],
    bump = config.bump,
)]
pub config: Account<'info, State>,
```

Oro: Fixed in commit 9b580ede.

Cantina Managed: Fix verified.

3.1.2 Wrong price calculation from Gold to USDC

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `mint_gold()`, `burn_gold()`, `buy()` and `sell()` functions, the USDC amount required or received is calculated based on these formulas:

```
// in buy() and mint_gold()
amount * price / 100 * (10000 + self.price.fee)

// in sell() and buy_gold()
amount * price / 100 * (10000 - self.price.fee)
```

These formulas are incorrect due to several issues:

1. They don't account for the Pyth price feed exponential value of XAU/USD.
2. They fail to handle the decimal differences between Gold token and USDC token properly.
3. They don't divide by `price.fee`'s denominator (10000) after multiplication.
4. They divide first and then multiply, which leads to precision loss.

Recommendation: The formulas should be corrected as follows (in pseudocode):

```
  // in buy() and mint_gold()
- amount * price / 100 * (10000 + self.price.fee)

+ amount * price * (10000 + self.price.fee) * pow(10, mint_b.decimals - mint_a.decimals) * pow(10,
↪  pyth_price_result?.exponent)  / 10000

  // in sell() and buy_gold()
- amount * price / 100 * (10000 - self.price.fee)
+ amount * price * (10000 - self.price.fee) * pow(10,mint_b.decimals - mint_a.decimals) * pow(10,
↪  pyth_price_result?.exponent)  / 10000
```

Oro: The formulas have been changed for multiple different commits (see for instance commit 7658daa7), suggest to compare with current implementations.

Cantina Managed: Fix verified.

3.1.3 `claim()` and `unstake()` functions will not be useable

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The position account in both `claim()` and `unstake()` functions is marked with the `init` constraint, while this same account is already initialized in the `stake()` function:

```
#[account(
    init,
    payer = user,
    space = 8 + Position::INIT_SPACE,
)]
pub position: Box<Account<'info, Position>>,
```

This will cause the `claim()` and `unstake()` functions to revert when called, since an account cannot be initialized if it already exists. As a result, all funds staked are stuck indefinitely, as users cannot claim rewards or withdraw their tokens.

Recommendation: Replace the `init` constraint with the `mut` constraint in the position account definition for both `claim()` and `unstake()` functions.

Oro: Fixed in commit 9b580ede.

Cantina Managed: Fix verified.

3.1.4 Missing seed and bump constraints in position account in `claim()` and `unstake()` functions

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The position account in both `claim()` and `unstake()` functions is missing `seeds` and `bump` constraints, which means a user can include any position account belonging to another user when calling these functions. As a result, a malicious user can claim/unstake others' positions to their own wallet, effectively stealing their funds.

```
    #[account(
    init,
    payer = user,
    space = 8 + Position::INIT_SPACE,
)]
pub position: Box<Account<'info, Position>>,
```

Recommendation: Add `seeds` and `bump` constraints to the position account in both `claim()` and `unstake()` functions, similar to how they are defined in the `stake()` function.

Oro: Fixed in commit 9b580ede.

Cantina Managed: Fix verified.

3.1.5 Multiple `unstake()` calls at the same position can drain the vault

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `unstake()` function, when a user unstakes their position, all staked tokens plus accrued interest are sent back to the original staker. At this point, the staker should not be allowed to unstake from the same position again. However, there is no restriction in the function preventing multiple unstake calls, which allows users to withdraw the same amount of Gold tokens repeatedly from a single position. A malicious user can exploit this vulnerability to drain the vault by calling `unstake()` multiple times on the same position.

Recommendation: To prevent this vulnerability, consider implementing one of these solutions:

- Close the position after the unstake operation is complete.
- Set the position's amount to zero after a successful unstake.
- Add a flag to track whether a position has already been unstaked.

Oro: Fixed in commit 7320cd80.

Cantina Managed: Fix verified.

3.1.6 Incorrect Reward token calculation enables pool drain via arbitrage

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `liquid_stake()` function contains a critical mathematical error in calculating the amount of reward tokens to mint to stakers when the rewards mint supply is non-zero. The current implementation calculates the reward amount as:

```
amount * self.config.liquid_amount / self.rewards_mint.supply
```

The root cause is in the reward calculation logic where the numerator and denominator are swapped. This creates a situation where malicious users can:

1. Perform `liquid_stake()` to receive an inflated amount of reward tokens.
2. Use `liquid_unstake()` to withdraw more underlying tokens than initially deposited.
3. Repeat until the pool is drained.

Recommendation: The reward token calculation should be corrected to maintain proper proportions between deposits and rewards. Update the calculation to:

```
- amount * self.config.liquid_amount / self.rewards_mint.supply
+ amount * self.rewards_mint.supply / self.config.liquid_amount
```

Oro: The reward token calculation has been fixed across multiple commits (see for instance commit 7df8758f), recommend to check the current implementation of the instruction.

Cantina Managed: Fix verified.

3.1.7 Incorrect token amount calculation in liquid unstaking leads to asset loss

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `liquid_unstake()` function incorrectly handles token amounts during the unstaking process, leading to potential asset loss. The root cause lies in two key areas:

1. Token Amount Mismatch: While `liquid_stake()` accepts GOLD token amount as user input, `liquid_unstake()` accepts reward tokens amount as user input. This inconsistency creates an asymmetric relationship between staking and unstaking operations.

2. Incorrect State Update: The function decrease `config.liquid_amount` by the reward token amount instead of the GOLD token amount. Since `liquid_amount` tracks the total GOLD tokens in the vault, this leads to incorrect accounting of the protocol's assets.

This vulnerability can be exploited by users to manipulate the exchange rate between GOLD and reward tokens, potentially draining the vault.

Recommendation: The function should be modified to:

1. Use the input amount as GOLD currency to maintain consistency with the `liquid_stake()` function, adjusting calculations accordingly.
2. Update `config.liquid_amount` with the correct GOLD amount.

Here's the proposed fix:

```
  pub fn liquid_unstake(&mut self, amount: u64) -> Result<()> {
      let (canonical_bump_pda, _canonical_bump) =
          Pubkey::find_program_address(&[b"whitelist", &self.user.key.to_bytes()], &ID);
      assert_eq!(canonical_bump_pda, self.whitelist.key());

      let cpi_program = self.token_program.to_account_info();

      let seeds = &[b"config".as_ref(), &[self.config.bump]];
      let signer_seeds = &[&seeds[..]];

      let cpi_accounts = Burn {
          mint: self.rewards_mint.to_account_info(),
          from: self.rewards_ata.to_account_info(),
          authority: self.user.to_account_info(),
      };

      let cpi_context: CpiContext<'_, '_, '_, '_, Burn<'_>> =
          CpiContext::new_with_signer(cpi_program, cpi_accounts, signer_seeds);

-     let amount_to_burn = amount;
-     let amount_to_transfer = amount * self.config.liquid_amount / self.rewards_mint.supply;
+     let amount_to_burn = amount * self.rewards_mint.supply / self.config.liquid_amount;
+     let amount_to_transfer = amount;

      burn(cpi_context, amount_to_burn)?;

      let transfer_accounts = TransferChecked {
          from: self.vault.to_account_info(),
          mint: self.mint_a.to_account_info(),
          to: self.user_ata_a.to_account_info(),
          authority: self.config.to_account_info(),
      };

      let cpi_ctx = CpiContext::new_with_signer(
          self.token_program.to_account_info(),
          transfer_accounts,
          signer_seeds,
      );

      let _ = transfer_checked(cpi_ctx, amount_to_transfer, self.mint_a.decimals);
      self.config.liquid_amount -= amount;

      Ok(())
  }
```

Oro: Fixed in commit 9b580ede.

Cantina Managed: Fix verified.

3.1.8 Missing mutable account aonstraint prevents state updates

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `config` account in `liquid_stake()`, `liquid_unstake()`, and `reward()` functions lacks the required `mut` constraint in the account validation, preventing critical state updates from being persisted.

In Solana, accounts that need to be modified during instruction execution must be explicitly marked as mutable using the `mut` constraint. Without this constraint, any attempts to modify the account's data will not be persisted at runtime, even though the account is successfully loaded and the modification logic executes correctly.

The `config` account, which holds critical protocol state including liquid staking amounts and configuration parameters, is defined without the `mut` constraint in its account validation struct. As a result, while the code successfully executes state variable updates within this account (e.g., `self.config.liquid_amount += amount` in `liquid_stake()`), these changes are not persisted to the blockchain.

Recommendation: Add the `mut` constraint to the `config` account validation in `liquid_stake()`, `liquid_unstake()`, and `reward()` functions. The fix should be applied as follows:

```
  **[account(:**

+     mut,
      seeds = [b"config".as_ref()],
      bump = config.bump,
  )]
  pub config: Account<'info, State>,
```

Oro: Fixed in commit 7320cd80.

Cantina Managed: Fix verified.

3.1.9 Excessive Payout in unstake and claim Functions Causes Severe Fund Loss

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `unstake` and `claim` functions incorrectly calculate the amount to transfer, resulting in users receiving 12 times the amount they originally staked. This leads to a significant loss of funds for the protocol.

The flawed calculation is shown here:

```
let amount_to_transfer = self.position.amount
    + (self.position.amount * (12 - self.position.claimed as u64) * 10025 / 10000);
```

This miscalculation causes users to receive an excessive reward, far exceeding the intended staking rewards.

Recommendation: The correct implementation should ensure that only a small portion of the staked amount is transferred per claim. Corrected Claim Function Calculation::

```
let amount_to_transfer = self.position.amount * 25 / 10000;
```

Corrected Unstake Function Calculation:: To properly transfer the unclaimed amount after the staking duration has passed:

```
let amount_to_transfer =
    self.position.amount + ((12 - self.position.claimed as u64) * self.position.amount * 25 / 10000);
```

Oro: Fixed in commit 603555b6.

Cantina Managed: Fix verified.

3.1.10 Incorrect Time Validation in `unstake` Function leads to permanent lock of the staked funds

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: The `unstake` function is designed to allow unstaking only after 12 months from the stake start time. However, the current time validation logic incorrectly multiplies the seconds in a year by the number of claimed months, leading to an excessive required time delay.

```
let enlapsed_time = Clock::get()?.unix_timestamp - self.position.start_time;

// < SECONDS_IN_A_YEAR
if enlapsed_time
    < SECONDS_IN_A_YEAR
        .checked_mul(
            (self.position.claimed as i64)
                .checked_add(1)
                .ok_or(Overflow)?,
        )
        .ok_or(Overflow)?
{
    return Err(ErrorCode::NotEnoughTimeElapsed.into());
}
```

If a user claims rewards for 6 months and later tries to unstake after 12 months, the function would require over 7 years to elapse, resulting in a significant loss of funds.

Recommendation: Modify the validation logic to check only for a minimum of 12 months (1 year) elapsed time:

```
let enlapsed_time = Clock::get()?.unix_timestamp - self.position.start_time;

// < SECONDS_IN_A_YEAR
if enlapsed_time < SECONDS_IN_A_YEAR {
    return Err(ErrorCode::NotEnoughTimeElapsed.into());
}
```

Oro: Fixed in commit a5585707.

Cantina Managed: Fix verified.

3.1.11 Fix review Finding: Incorrect Calculation of `amount_to_burn` in `liquid_unstake`

Severity: Critical Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `liquid_unstake` function, the calculation of `amount_to_burn` is incorrect:

- `amount` represents the amount of gold to be unstaked.
- `config.liquid_amount` represents the total amount of gold in the pool.
- `rewards_mint.supply` represents the total minted supply of reward tokens.

The current formula incorrectly calculates `amount_to_burn` as:

```
let amount_to_burn = (amount as u128)
    .checked_mul(self.config.liquid_amount.into())
    .ok_or(Overflow)?
    .checked_div(self.rewards_mint.supply.into())
    .ok_or(Overflow)?;
```

This leads to an incorrect token burn amount, affecting the protocol's balance and potentially causing loss of funds.

Recommendation: Use the correct formula, which ensures `amount_to_burn` is proportional to the unstaked gold relative to the total gold in the protocol:

```
let amount_to_burn = (amount as u128)
    .checked_mul(self.rewards_mint.supply.into())
    .ok_or(Overflow)?
    .checked_div(self.config.liquid_amount.into())
    .ok_or(Overflow)?;
```

Oro: Fixed in commit 2c19a74f.

Cantina Managed: Fix verified.

3.2 High Risk

3.2.1 DoS in `claim` Function Due to `init` Constraint on an Already Initialized Account

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `claim` function, the `init` constraint is incorrectly applied to the PDA `position`. However, this account is already initialized in the `stake` function. As a result, the `claim` function will fail to execute after the initial staking process, leading to a permanent denial of service (DoS) for claiming rewards. This renders the protocol's claiming functionality unusable.

In the `claim` function, the `init` constraint is incorrectly applied to the PDA `position`. However, this account is already initialized in the `stake` function. As a result, the `claim` function will fail to execute after the initial staking process, leading to a permanent denial of service (DoS) for claiming rewards. This renders the protocol's claiming functionality unusable.

```
// in claim function

#[account(
    init,
    payer = user,
    space = 8 + Position::INIT_SPACE,
)]
pub position: Box<Account<'info, Position>>,
```

Recommendation: Replace the `init` constraint with `mut` to ensure the `position` account can be modified instead of being reinitialized. Additionally, verify the account's existence before performing operations to prevent unintended reinitialization attempts.

```
#[account(mut)]
pub position: Box<Account<'info, Position>>,
```

Oro: Fixed in commit 9b580eded.

Cantina Managed: Fix verified.

3.3 Medium Risk

3.3.1 Missing Slippage Parameter Exposes Users to Unintended Prices, Leading to Potential Fund Loss

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In the trading functions `buy` and `sell`, trades are executed based on either the `oracle price` or a stored constant price, with the higher price being used. While there is a check to ensure price freshness, there is no validation of the price value itself against a reasonable threshold.

In the trading functions `buy` and `sell` , trades are executed based on either the `oracle price` or a stored constant price, with the higher price being used. While there is a check to ensure price freshness, there is no validation of the price value itself against a reasonable threshold. This lack of validation can result in the use of an inflated price from the Pyth oracle due to market volatility, leading to users unknowingly purchasing assets at significantly higher prices. For example, if the stored price and the intended trade price are 100, but the Pyth oracle returns an inflated price of 1000, the system will use 1000, causing a significant loss to the user.

Recommendation: Introduce a slippage parameter to allow users to specify an acceptable price deviation, ensuring they are protected from extreme price fluctuations. Updated `buy` Function:

```
impl<'info> Buy<'info> {.
pub fn buy(&mut self, amount: u64, max_amount_in: u64) -> Result<()> {
let total_cost = amount * price / 100 * (10000 + self.price.fee)
// Ensure the total cost does not exceed the user's maximum acceptable amount
        require!(total_cost <= max_amount_in, CustomError::SlippageExceeded);
Ok(())
}
}
```

Oro: Fixed in commit f7923435.

Cantina Managed: Fix verified.

3.3.2 Missing Confidence Validation in Pyth Oracle Price

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The trading functions that use the Pyth oracle price do not validate confidence values (`conf` and `ema_conf`). They only use `get_price_no_older_than` to check staleness but do not ensure that the confidence percentage remains within an acceptable threshold. This omission can lead to:

- Using prices during high uncertainty periods, which may not reflect accurate market conditions.
- Missing market anomaly signals, potentially overlooking extreme price deviations.
- Potential incorrect liquidations during low market confidence, leading to unfair liquidations or improper risk calculations.

As per Pyth's best practices, confidence intervals should be considered when evaluating price validity.

Recommendation: Introduce a configurable `max_confidence_pct` parameter in the price account and validate the confidence percentage before using the price.

- Updated `buy` Function with Confidence Validation:

```
impl<'info> Buy<'info> {
pub fn buy(&mut self, amount: u64, max_amount_in: u64, max_confidence_pct: u64) -> Result<()> {
// Fetch price and confidence from Pyth oracle
let pyth_price_result =
            price_update.get_price_no_older_than(&Clock::get()?, maximum_age, &feed_id);
let (pyth_price, conf) = match pyth_price_result {
            Ok(price) => (price.price, price.conf),
            Err(e) => Err(e)?,
        };

// Validate confidence percentage.
let confidence_pct = (conf as u128)
            .checked_mul(100)?
            .checked_div(pyth_price.abs() as u128)?;
require!(
            confidence_pct <= max_confidence_pct as u128,
CustomError::PriceConfidenceTooHigh
        );


Ok(())
}
}
```

Oro: Fixed in commit 5df656a9.

Cantina Managed: Fix verified.

3.3.3 User can unstake anytime instead of getting locked for 1 year

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In the `unstake()` function, there is no time restriction for any positions. This means stakers can unstake at any time instead of being locked for 1 year as specified in the requirements.

Recommendation: Consider implementing a time check in `unstake()` function that only allows stakers to unstake after 1 year has passed since their initial stake.

```
+ assert!(Clock::get()?.unix_timestamp >= self.position.start_time + (366 * 24 * 60 * 60));
```

Oro: Fixed in commit b7f0aeb7.

Cantina Managed: Fix verified.

3.3.4 Pyth price feed maximum age too high allows stale price data

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: When using Pyth price feed, the `maximum_age` parameter for Pyth price feed validation is set to 99,999 seconds (approximately 27.8 hours), which is significantly higher than recommended for secure price oracle implementations. This excessive timeout allows the use of stale price data that could be exploited during periods of high price volatility.

The issue occurs in the price validation logic where `get_price_no_older_than()` is called with this maximum age parameter. While the function does properly validate that the price update is not older than the specified maximum age, setting such a high threshold effectively weakens this security check.

Recommendation: The maximum age for price feed data should be set to a more conservative value that aligns with market dynamics and security best practices. For most DeFi applications, price feeds should not be older than 5-15 minutes. Implement a more appropriate maximum age limit:

```
- let maximum_age: u64 = 99999;
+ let maximum_age: u64 = 900; // 15 minutes in seconds
```

Oro: Fixed in commit 7320cd80.

Cantina Managed: Fix verified.

3.3.5 Fix review Finding: Admin Can Reset Critical Configuration by Reinitializing the Contract

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: In `liquid_staking/src/contexts/init.rs`, there is a vulnerability in the `Initialize` context and its `initialize_config()` function. The root cause is that the `config` account is created with `init_if_needed` instead of `init`, allowing the admin to call the initialization function multiple times. The initialization code creates a `config` account as follows:

```
**account(:** init_if_needed,
    payer = admin,
    seeds = [CONFIG_SEED.as_ref()],
    bump,
    space = 8 + State::INIT_SPACE,
)
```

When `initialize_config()` is called, it sets the `State` structure with crucial values including `bump`, `liquid`, `liquid_amount`, and `monthly_rewards`. Because the function can be called multiple times due to the `init_if_needed` constraint, the admin can reset these properties to arbitrary values at any time. This is particularly dangerous because:

1. The `monthly_rewards` parameter can be changed to manipulate rewards.
2. The `liquid` flag and `liquid_amount` fields can be reset, causing funds to be locked.

Recommendation: Change the constraint from `init_if_needed` to `init` to ensure the initialization can only happen once:

```
  **[account(:**

-     init_if_needed,
+     init,
      payer = admin,
      seeds = [CONFIG_SEED.as_ref()],
      bump,
      space = 8 + State::INIT_SPACE,
  )]
  pub config: Account<'info, State>,
```

Oro: Fixed in commit 04f2ce61.

Cantina Managed: Fix verified.

3.4 Low Risk

3.4.1 Broken Access Control in Vault Initialization

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In the function `initialize_vault` , the vault account can be pre-initialized by anyone, leading to this function to revert so this breaks access control mechanism in this function. Additionally, the function itself is redundant, as its sole purpose is to create the vault. Instead of keeping it separate, it can be merged with the `initialize_config` instruction to streamline initialization.

Recommendation: Merge the vault creation logic into the `initialize_config` instruction, as shown in the following implementation:

```
pub struct Initialize<'info> {
    #[account(mut, address = ADMIN)]
    pub admin: Signer<'info>,
    #[account(
        init,
        payer = admin,
        seeds = [b"config".as_ref()],
        bump,
        space = 8 + State::INIT_SPACE,
    )]
    pub config: Account<'info, State>,
    #[account(
        init_if_needed,
        payer = admin,
        seeds = [b"rewards".as_ref(), config.key().as_ref()],
        bump,
        mint::decimals = 6,
        mint::authority = config,
    )]
    pub rewards_mint: Account<'info, Mint>,
    #[account(
        init_if_needed,
        payer = admin,
        associated_token::mint = mint,
        associated_token::authority = config,
        associated_token::token_program = token_program,
    )]
    pub vault: Box<InterfaceAccount<'info, TokenAccount>>,
    pub system_program: Program<'info, System>,
    pub token_program: Program<'info, Token>,
}

impl<'info> Initialize<'info> {
    pub fn initialize_config(&mut self, bumps: &InitializeBumps) -> Result<()> {
        self.config.set_inner(State {
            rewards_bump: bumps.rewards_mint,
            bump: bumps.config,
            liquid: false,
            liquid_amount: 0,
        });

        Ok(())
```

```
    }
}
```

Oro: Fixed in commit 9b580eded.

Cantina Managed: Fix verified.

3.4.2 Update price definition from `u64` to `i64` to accommodate for the possibility of a negative price

Severity: Low Risk

Context: price.rs#L23

Description: Update price definition from u64 to i64 to accommodate for a "below zero price" as referenced in the kickoff call.

Recommendation:

```
- pub fn price(&mut self, bumps: &UpdatePriceBumps, price: u64, fee: u64) -> Result<()> {
+ pub fn price(&mut self, bumps: &UpdatePriceBumps, price: i64, fee: u64) -> Result<()> {
```

Oro: Fixed in commit f7923435.

Cantina Managed: Fix verified.

3.4.3 Hard Coded Values leading to maintenance issues

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The use of hardcoded values like 305 * 24 * 6 * 60 for seconds in a month and 10025 / 10000 for the transfer amount calculation can lead to maintenance issues and potential errors if these values need to be updated or changed.

Recommendation: Avoid Hardcoded Values. Replace hardcoded values with constants or configuration parameters that can be easily updated and maintained.

Oro: Fixed in our main branch.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Missing Events for Storage-Mutating Functions

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Functions that modify storage, such as `price` , `buy`,`sell`,`deposit`,and`withdraw` do not emit events. Emitting events is essential for off-chain indexing, transparency, and debugging. Without them, it is harder for users and external applications to track state changes efficiently.

Recommendation: Emit an event after modifying storage to ensure state changes can be tracked.

Oro: Fixed in commit 603555b6.

Cantina Managed: Fix verified.

3.5.2 Reversible Whitelisting Mechanism should be implemented

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `inti` program can only whitelist users, but the reverse operation (removing a user from the whitelist) is not possible. This is because the validation of a whitelisted user is done by checking whether the account has been initialized. Once initialized, the account remains permanently whitelisted.

Recommendation: To allow for the removal of users from the whitelist, add an `is_whitelisted` flag to the account structure. This will enable dynamic updates to the user's whitelist status without relying solely on account initialization.

```
pub struct Whitelist {
    pub bump: u8,
    pub is_whitelisted: bool,
}
```

Oro: Fixed in commit 9b580ede.

Cantina Managed: Fix verified.

3.5.3 Not Checking for potential overflows

Severity: Informational

Context: claim.rs#L63

Description: The code performs arithmetic operations without checking for potential overflows or underflows. Rust's default behavior is to panic on overflow in debug mode, but in release mode, it wraps around silently.

Recommendation: Use Rust's checked arithmetic methods (e.g., `checked_add`, `checked_mul`) to ensure that arithmetic operations do not overflow or underflow.

Oro: This recommendation has been implemented across both smart contracts and in diferent commits (see for instance commit 027baab4), recommend to verify that current implementation operates only with checked math.

Cantina Managed: Fix verified.

3.5.4 The Comment AUX/USD should be XAU/USD

Severity: Informational

Context: burn.rs#L71

Description: AUX/USD should be XAU/USD. While this has low impact, it can cause maintenance issues down the road and confusion for future auditors.

Recommendation: Change AUX to XAU.

Oro: Fixed in our main branch.

Cantina Managed: Fix verified.

3.5.5 Create a robust runnable test suite

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The existing test suite sets up a series of tests to interact with the Inti and liquid_staking Solana program using the Anchor framework. It initializes the necessary accounts, defines utility functions for transaction confirmation and logging, and performs various operations such as updating prices, depositing tokens, whitelisting users, buying, selling, withdrawing, minting, burning tokens, initializing a vault, staking, and unstaking tokens. However, several verification tests are needed to ensure critical vulnerabilities are not presented in operation.

Recommendation: Ensure test cases are created for critical vulnerabilities identified during this assessment:

- Testing for incorrect initializations and reinitializations.
- Testing price, reward, and staking / unstaking, and transfer calculations to ensure accuracy.
- Authorization of functions such as attempting claim/unstake other users' positions to their own wallet.

- Replay attacks such as attempting to unstake from the same position after a successful initial unstaking.
- Ensuring expected data persists throughout state updates.

In addition, in accordance with security best practices, it is recommend to engage in a follow up assessment to ensure critical and high vulnerabilities are mitigated.

Oro: Fixed in our main branch.

Cantina Managed: Fix verified.