Infinity Swap

Audit

Presented by:

OtterSec contact@osec.io

James Wang james.wang@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Stargaze engaged OtterSec to perform an assessment of the `infinity` program. This assessment was conducted between April 4th and April 12th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 9 findings total.

In particular, we identified an issue with an incorrect quote calculation while handling NFT swaps (OS-ISP-ADV-00, OS-ISP-ADV-01), the possibility of stealing NFTs by abusing missing checks between collections and swap pool IDs (OS-ISP-ADV-02), and improper handling of NFT ownership, allowing attackers to fungibly swap NFTs (OS-ISP-ADV-03).

We also made recommendations around interface function naming (OS-ISP-SUG-00) and internal variable naming (OS-ISP-SUG-01) to avoid confusion and boost code maintainability.

02 | Scope

The source code was delivered to us in a git repository at github.com/public-awesome/infinity. This audit was performed against commit 2f1afb3.

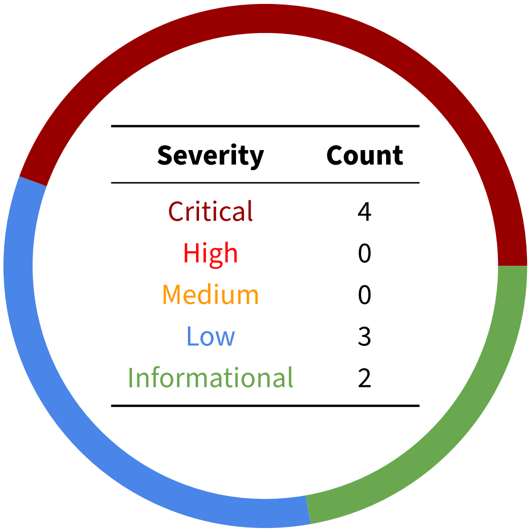
A brief description of the programs is as follows.

Name	Description
infinity	The infinity contract is used for holding `pools`, where users can create NFT buy and sell orders. The following `pool` types are supported. • Token pools, where users deposit native tokens to buy NFTs. • NFT pools, where users deposit NFTs to sell for tokens. • Trade pools, where users supply NFTs and tokens to allow the buying and selling between them. The prices of NFTs are calculated by three kinds of `curves`. • Linear curves, where NFT prices increase linearly with each purchase. • Exponential curves, where NFT prices increase exponentially with each purchase. • Constant product curves, where the product of `nft_count` and `token_count` remains constant.

03 | **Findings**

Overall, we reported 9 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	4
High	0
Medium	0
Low	3
Informational	2

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria can be found in Appendix A.

ID	Severity	Status	Description
OS-ISP-ADV-00	Critical	Resolved	Incorrect pool quote calculation in `get_buy_quote` for linear curves may result in a loss of user funds.
OS-ISP-ADV-01	Critical	Resolved	Incorrect pool quote calculation in `get_buy_quote` and `update_spot_price` for exponential curves may result in a loss of user funds.
OS-ISP-ADV-02	Critical	Resolved	Lack of check between `collection` and `pool_id` within `execute_swap_tokens_for_specific_nfts` allows attackers to steal NFTs.
OS-ISP-ADV-03	Critical	Resolved	NFT owner `pool` is not checked properly, causing all NFTs fungible during withdrawal.
OS-ISP-ADV-04	Low	Resolved	Rounding errors may occur in `ConstantProduct` quote calculations.
OS-ISP-ADV-05	Low	Resolved	Mixed usage of `NATIVE_DENOM` and admin assigned `config.denom`.
OS-ISP-ADV-06	Low	Resolved	`swap_tokens_for_specific_nfts` does not respect the `robust` flag.

OS-ISP-ADV-00 [crit] | Incorrect Calculation For Linear Curves

Description

While setting prices of pools, owners have three curves to choose from: `Linear`, `Exponential` and `ConstantProduct`. Additionally, for `Trade` pools, dealers can buy and sell NFTs to the pool.

`get_buy_quote` and `get_sell_quote` are used to calculate the price where pools buy and sell NFTs. Since all pricing curves support dynamic prices and `Trade` pools essentially allow the price to move in both directions, it is crucial for `infinity` to ensure that `pools` will not suffer a loss from a sequence of buy and sell actions.

To simplify, if a `pool` buys an NFT and then sells it immediately, the number of tokens spent on buying the NFT should not exceed the number of tokens collected from selling the NFT.

This rule is enforced by the following logic

- When a `pool` buys an NFT, the price is calculated with respect to the `pool` state after the swap has finished.
- When a `pool` sells an NFT, the price is calculated with respect to the current `pool` state.

These rules ensure that the `pool` state will always be restored if N sells + N buys occur. However, an error in calculating the NFT buy price causes `pools` using `Linear` curves to buy at a higher price than it should. This potentially leads to a loss of funds for `pool` owners and is shown in the code snippet below:

infinity-swap/src/pool.rs RUST

```rust
pub fn get_buy_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
     ↪  ContractError> {
    // Calculate the buy price with respect to pool types and bonding curves
    let buy_price = match self.pool_type {
        PoolType::Token => Ok(self.spot_price),
        PoolType::Nft => Err(ContractError::InvalidPool(
            "pool cannot buy nfts".to_string(),
        )),
        PoolType::Trade => match self.bonding_curve {
            BondingCurve::Linear => self
                .spot_price
                .checked_add(self.delta)
                .map_err(|e| ContractError::Std(StdError::overflow(e))),
            ...
        },
    }?;
    // If the pool has insufficient tokens to buy the NFT, return None
    if self.total_tokens < buy_price || buy_price < min_quote {
        return Ok(None);
    }
     Ok(Some(buy_price))
}
```

Remediation

Calculate the correct price when acquiring NFTs.

infinity-swap/src/pool.rs DIFF

```
 pub fn get_buy_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
  ↪  ContractError> {
   // Calculate the buy price with respect to pool types and bonding curves
   let buy_price = match self.pool_type {
     ...
     PoolType::Trade => match self.bonding_curve {
       BondingCurve::Linear => self
         .spot_price
-        .checked_add(self.delta)
+        .checked_sub(self.delta)
         .map_err(|e| ContractError::Std(StdError::overflow(e))),
       ...
     },
   }?;
   ...
 }
```

Patch

Resolved in 4453377.

OS-ISP-ADV-01 [crit] | Incorrect Calculation For Exponential Curves

Description

The inverse operation of $V * (1 + delta)$ is $V/(1 + delta)$. The `Exponential` curves incorrectly implement the price decrement step as $V * (1 - delta)$. Since $V * (1 + delta) * (1 - delta) < V$, this exposes the `Exponential pool` to risks of price manipulation and may lead to loss of pool owner funds.

infinity-swap/src/pool.rs RUST

```rust
pub fn get_buy_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
    ↪ ContractError> {
    let buy_price = match self.pool_type {
        ...
        PoolType::Trade => match self.bonding_curve {
            ...
            BondingCurve::Exponential => {
                let product = self
                    .spot_price
                    .checked_mul(self.delta)
                    .map_err(|e| StdError::Overflow { source: e })?
                    .checked_div(Uint128::from(MAX_BASIS_POINTS))
                    .map_err(|e| ContractError::Std(StdError::divide_by_zero(e)))?;
                self.spot_price
                    .checked_add(product)
                    .map_err(|e| ContractError::Std(StdError::overflow(e)))
            }
            ...
        },
    }?;
    ...
}

pub fn update_spot_price(&mut self, tx_type: &TransactionType) -> Result<(),
    ↪ StdError> {
    let result = match tx_type {
        ...
        TransactionType::NftsForTokens => match self.bonding_curve {
            ...
            BondingCurve::Exponential => {
                let product = self
                    .spot_price
                    .checked_mul(self.delta)
                    .map_err(|e| StdError::Overflow { source: e })?
                    .checked_div(Uint128::from(MAX_BASIS_POINTS))
                    .map_err(|e| StdError::DivideByZero { source: e })?;
                self.spot_price
                    .checked_sub(product)
                    .map_err(|e| StdError::Overflow { source: e })
            }
            ...
        },
    };
    ...
}
```

Remediation

Use $V/(1 + delta)$ instead of $V * (1 - delta)$.

infinity-swap/src/pool.rs DIFF

```
  pub fn get_buy_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
     ↪  ContractError> {
      ...
          PoolType::Trade => match self.bonding_curve {
              ...
-             BondingCurve::Exponential => {
-                 let product = self
-                     .spot_price
-                     .checked_mul(self.delta)
-                     .map_err(|e| StdError::Overflow { source: e })?
-                     .checked_div(Uint128::from(MAX_BASIS_POINTS))
-                     .map_err(|e|
     ↪  ContractError::Std(StdError::divide_by_zero(e)))?;
-                 self.spot_price
-                     .checked_add(product)
-                     .map_err(|e| ContractError::Std(StdError::overflow(e)))
-             }
+             BondingCurve::Exponential => self
+                 .spot_price
+                 .checked_mul(Uint128::from(MAX_BASIS_POINTS))
+                 .map_err(|e| StdError::Overflow { source: e })?
+                 .checked_div(
+                     self.delta.checked_add(
+                         Uint128::from(MAX_BASIS_POINTS)
+                     )
+                     .map_err(|e| ContractError::Std(StdError::overflow(e)))?
+                 )
+                 .map_err(|e| ContractError::Std(StdError::divide_by_zero(e))),
              ...
          }
      ...
  }

  pub fn update_spot_price(&mut self, tx_type: &TransactionType) -> Result<(),
     ↪  StdError> {
      let result = match tx_type {
          ...
          TransactionType::NftsForTokens => match self.bonding_curve {
              ...
-             BondingCurve::Exponential => {
-                 let product = self
-                     .spot_price
-                     .checked_mul(self.delta)
-                     .map_err(|e| StdError::Overflow { source: e })?
-                     .checked_div(Uint128::from(MAX_BASIS_POINTS))
-                     .map_err(|e| StdError::DivideByZero { source: e })?;
-                 self.spot_price
-                     .checked_sub(product)
-                     .map_err(|e| StdError::Overflow { source: e })
-             }
+             BondingCurve::Exponential => self
+                 .spot_price
+                 .checked_mul(Uint128::from(MAX_BASIS_POINTS))
+                 .map_err(|e| StdError::Overflow { source: e })?
+                 .checked_div(
+                     self.delta.checked_add(
+                         Uint128::from(MAX_BASIS_POINTS)
+                     )
+                     .map_err(|e| ContractError::Std(StdError::overflow(e)))?
+                 )
+                 .map_err(|e| ContractError::Std(StdError::divide_by_zero(e)))
+                 .checked_add(Uint128::one())
+                 .map_err(|e| StdError::Overflow { source: e}),
              ...
          },
      };
      ...
  }
```

Patch

Resolved in 4453377 and ff654a6.

OS-ISP-ADV-02 [crit] | Lack Of Check Between Collection And Swap Pool

Description

The `execute_swap_tokens_for_specific_nfts` action takes a user-provided `collection` and `nfts_to_swap_for`. Each entry of `nfts_to_swap_for` includes a `pool_id` pointing to the target `pool` for a swap to occur. Unfortunately, while performing swaps, the `collection` contract address included in the `pool` configuration is not checked against the user-supplied `collection`.

This potential mismatch between the user-provided and pool `collection` causes issues since the `pool`s are used in processing the majority of the swap logic, except the final NFT `transfer`, which is performed on `collection` instead.

infinity-swap/src/swap_processor.rs RUST

```rust
fn commit_messages(&self, response: &mut Response) -> Result<(), ContractError> {
    ...
    // Iterate over swaps and reduce token payments that need to be made
    for swap in self.swaps.iter() {
        ...
        // Push transfer NFT messages
        transfer_nft(
            &swap.nft_payment.nft_token_id,
            &swap.nft_payment.address,
            self.collection.as_ref(),
            response,
        )?;
        ...
    }
    ...
}
```

The following illustrates how an attacker may abuse this mismatch to steal NFTs from `infinity`.

1. Attacker aims to steal `collectionV::nftX`, currently held by `poolV` on `infinity`.
2. Attacker deploys mock an NFT contract `collectionM`.
3. Attacker mints `collectionM::nftX` from the mock NFT contract.
4. Attacker creates `poolM` on `infinity` and assigns `collectionM` to the pool.
5. Attacker lists `collectionM::nftX` on `poolM` for sale.
6. Attacker calls `execute_swap_tokens_for_specific_nfts` with `nfts_to_swap_for = poolM::nftX` and `collection = collectionV`.
7. Swap logic is processed for `poolM::nftX`, the attacker pays `poolM` for `collectionM::nftX`. Then, the NFT transfer is completed on `collectionV::nftX`, and the attacker successfully steals the target NFT from `infinity`.

Remediation

Check that each selected `pool` handles the same `collection` provided by the user.

infinity-swap/src/swap_processor.rs DIFF

```diff
  pub fn swap_tokens_for_specific_nfts(
      &mut self,
      storage: &'a dyn Storage,
      nfts_to_swap_for: Vec<PoolNftSwap>,
      swap_params: SwapParams,
  ) -> Result<(), ContractError> {
      ...
      for pool_nfts in nfts_to_swap_for {
          ...
          // If pool is not in pool_map, load it from storage
          if !pool_queue_item_map.contains_key(&pool_nfts.pool_id) {
              let pool_option = pools().may_load(storage, pool_nfts.pool_id)?;
              // If pool is not found, return error
              if pool_option.is_none() {
                  return Err(ContractError::PoolNotFound(format!(
                      "pool {} not found",
                      pool_nfts.pool_id
                  )));
              }
              // Create PoolQueueItem and insert into pool_queue_item_map
              let pool = pool_option.unwrap();
+
+             if pool.collection != self.collection {
+                 return Err(ContractError::InvalidPool(
+                     "pool does not belong to this collection".to_string(),
+                 ));
+             }

              let quote_price = pool.get_sell_quote(self.min_quote)?;
              ...
          }
          ...
      }
      ...
  }
```

Patch

Resolved in f2e7678.

OS-ISP-ADV-03 [crit] | NFTs Treated As Fungible Tokens During Withdrawal

Description

During NFT withdrawals, no checks against the owner `pool` are performed. Combined with `remove_nft_deposit` succeeding regardless of whether the removed `pool_id` and `nft_token_id` storage entry exists, attackers will be able to deposit an NFT (`nftX`) and withdraw another one with a different id (`nftY`) from `infinity`.

infinity-swap/src/helpers.rs RUST

```rust
pub fn remove_nft_deposit(storage: &mut dyn Storage, pool_id: u64, nft_token_id:
    ↪  &str) {
    NFT_DEPOSITS.remove(storage, (pool_id, nft_token_id.to_string()))
}
```

Remediation

Check that each NFT withdrawn belongs to the provided `pool`.

infinity-swap/src/execute.rs DIFF

```diff
  pub fn execute_withdraw_nfts(
      ...
  ) -> Result<Response, ContractError> {
      ...

      // Withdraw NFTs to the asset recipient if specified, otherwise to the sender
      let recipient = asset_recipient.unwrap_or(info.sender);
      for nft_token_id in &nft_token_ids {
          transfer_nft(
              nft_token_id,
              recipient.as_ref(),
              pool.collection.as_ref(),
              &mut response,
          )?;
+         verify_nft_deposit(deps.storage, pool_id, nft_token_id);
          remove_nft_deposit(deps.storage, pool_id, nft_token_id);
      }
      // Track the NFTs that have been withdrawn from the pool
      pool.withdraw_nfts(&nft_token_ids)?;

      ...
  }
```

Patch

Resolved in 152bd02.

OS-ISP-ADV-04 [low] | Rounding Errors In Quote Calculation

Description

Quote calculation for `ConstantProduct` curves contain division, leading to potential rounding errors. `pools` may suffer a marginal loss due to prices rounding down when selling NFTs.

infinity-swap/src/pool.rs RUST

```rust
pub fn get_sell_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
    ↪ ContractError> {
    ...
    let sell_price = match self.bonding_curve {
        BondingCurve::Linear | BondingCurve::Exponential => self.spot_price,
        BondingCurve::ConstantProduct => {
            if self.total_nfts < 2 {
                return Ok(None);
            }
            self.total_tokens
                .checked_div(Uint128::from(self.total_nfts - 1))
                .unwrap()
        }
    };
    ...
}
```

Remediation

Ensure that `pool` always sells NFTs with rounded-up prices.

infinity-swap/src/pool.rs DIFF

```diff
  pub fn get_sell_quote(&self, min_quote: Uint128) -> Result<Option<Uint128>,
    ↪ ContractError> {
      ...
              self.total_tokens
                  .checked_div(Uint128::from(self.total_nfts - 1))
+                 .unwrap()
+                 .checked_add(Uint128::one())
                  .unwrap()
          }
      };
      ...
  }
```

Patch

Resolved in 4453377 and ff654a6.

OS-ISP-ADV-05 [low] | Mixed Usage Of Different Coin

Description

infinity mixes the usage of NATIVE_DENOM and the admin assigned config.denom throughout the contract. While stargaze currently only supports native coins, it is not guaranteed that no other coins will be introduced in the future. Therefore, code relying on such behaviour may expose users to security risks.

Remediation

Replace all usage of config.denom with NATIVE_DENOM.

infinity-swap/src/execute.rs DIFF

```diff
  pub fn execute_deposit_tokens(
      deps: DepsMut,
      info: MessageInfo,
      pool_id: u64,
  ) -> Result<Response, ContractError> {
-     let config = CONFIG.load(deps.storage)?;
-     let received_amount = must_pay(&info, &config.denom)?;
+     let received_amount = must_pay(&info, &NATIVE_DENOM)?;
      ...
  }

  ...

  pub fn execute_withdraw_tokens(
      deps: DepsMut,
      info: MessageInfo,
      pool_id: u64,
      amount: Uint128,
      asset_recipient: Option<Addr>,
  ) -> Result<Response, ContractError> {
      ...

      let mut response = Response::new();

-     let config = CONFIG.load(deps.storage)?;
      // Withdraw tokens to the asset recipient if specified, otherwise to the
    ↪  sender
      let recipient = asset_recipient.unwrap_or(info.sender);
      transfer_token(
-         coin(amount.u128(), config.denom),
+         coin(amount.u128(), NATIVE_DENOM),
          recipient.as_ref(),
          &mut response,
      )?;
      ...
  }
  ...

  pub fn execute_remove_pool(
      deps: DepsMut,
      info: MessageInfo,
      pool_id: u64,
      asset_recipient: Option<Addr>,
  ) -> Result<Response, ContractError> {
      ...

      // If the pool has tokens, transfer them to the asset recipient
      if pool.total_tokens > Uint128::zero() {
          let recipient = asset_recipient.unwrap_or(info.sender);
          transfer_token(
-               coin(pool.total_tokens.u128(), config.denom),
+               coin(pool.total_tokens.u128(), NATIVE_DENOM),
              recipient.as_ref(),
              &mut response,
          )?;
      }
      ...
  }

  ...

  pub fn execute_swap_tokens_for_specific_nfts(
      deps: DepsMut,
      env: Env,
      info: MessageInfo,
      collection: Addr,
      nfts_to_swap_for: Vec<PoolNftSwap>,
      swap_params: SwapParams,
  ) -> Result<Response, ContractError> {
      let swap_prep_result = prep_for_swap(
          deps.as_ref(),
          &Some(env.block),
          &info.sender,
          &collection,
          &swap_params,
      )?;

-     let received_amount =
-         validate_nft_swaps_for_buy(&info, &swap_prep_result.denom,
    ↪   &nfts_to_swap_for)?;
+     let received_amount = validate_nft_swaps_for_buy(&info, &nfts_to_swap_for)?;
      ...
  }
```

infinity-swap/src/helpers.rs DIFF

```
  ...
  use sg_marketplace::msg::{ParamsResponse, QueryMsg as MarketplaceQueryMsg};
```

```diff
- use sg_std::Response;
+ use sg_std::{Response, NATIVE_DENOM};
  ...

  pub struct SwapPrepResult {
-     pub denom: String,
      pub marketplace_params: ParamsResponse,
      pub collection_royalties: Option<RoyaltyInfoResponse>,
      pub asset_recipient: Addr,
      pub finder: Option<Addr>,
      pub developer: Option<Addr>,
  }

  pub fn prep_for_swap(
      deps: Deps,
      block_info: &Option<BlockInfo>,
      sender: &Addr,
      collection: &Addr,
      swap_params: &SwapParams,
  ) -> Result<SwapPrepResult, ContractError> {
      ...

      Ok(SwapPrepResult {
-         denom: config.denom.clone(),
          marketplace_params,
          collection_royalties,
          asset_recipient: seller_recipient,
          finder,
          developer: config.developer,
      })
  }

  pub fn validate_nft_swaps_for_buy(
      info: &MessageInfo,
-     denom: &str,
      pool_nft_swaps: &Vec<PoolNftSwap>,
  ) -> Result<Uint128, ContractError> {
      ...
-     let received_amount = must_pay(info, denom)?;
+     let received_amount = must_pay(info, NATIVE_DENOM)?;
      if received_amount != expected_amount {
          return Err(ContractError::InsufficientFunds(format!(
              "expected {} but received {}",
              expected_amount, received_amount
          )));
      }
      Ok(received_amount)
  }
```

infinity-swap/src/instantiate.rs DIFF

```diff
  pub fn instantiate(
      deps: DepsMut,
      _env: Env,
      _info: MessageInfo,
      msg: InstantiateMsg,
  ) -> Result<Response, ContractError> {
      ...
      CONFIG.save(
          deps.storage,
          &Config {
-             denom: msg.denom.clone(),
              marketplace_addr: deps.api.addr_validate(&msg.marketplace_addr)?,
              developer: maybe_addr(deps.api, msg.developer)?,
          },
      )?;
      ...
  }
```

infinity-swap/src/state.rs DIFF

```
  pub struct Config {
-     /// The fungible token used in the pools
-     pub denom: String,
      /// The address of the marketplace contract
      pub marketplace_addr: Addr,
      /// The address of the developer who will receive a portion of the fair burn
      pub developer: Option<Addr>,
  }
```

Patch

Resolved in 8e717e9.

OS-ISP-ADV-06 [low] | Robust Flag Not Respected When Buying Specific NFTs

Description

When the `robust` flag is enabled, a single swap failure should not revert successfully processed swaps. However, in `swap_tokens_for_specifc_nfts`, when a `pool_queue_item` becomes unusable and inserted into `pools_to_save`, the loop around it continues to run. Hence, unwrapping `pool_queue_item` fetched from `pool_queue_item_map` in the next iteration will `panic`.

infinity-swap/src/swap_processor.rs RUST

```rust
pub fn swap_tokens_for_specific_nfts(
    &mut self,
    storage: &'a dyn Storage,
    nfts_to_swap_for: Vec<PoolNftSwap>,
    swap_params: SwapParams,
) -> Result<(), ContractError> {
    // Create a pool_queue_item map for tracking swap pools
    let mut pool_queue_item_map: BTreeMap<u64, PoolQueueItem> = BTreeMap::new();

    for pool_nfts in nfts_to_swap_for {
        ...
        // Iterate over all NFTs selected for the given pool
        for nft_swap in pool_nfts.nft_swaps {
            let pool_queue_item =
    ↪   pool_queue_item_map.remove(&pool_nfts.pool_id).unwrap();
            ...
            if pool_queue_item.usable {
                // If the swap was a success, and the quote price was updated, save
    ↪   into pool_queue
                pool_queue_item_map.insert(pool_queue_item.pool.id,
    ↪   pool_queue_item);
            } else {
                // If the swap was a success, but the quote price was not updated,
                // withdraw from circulation by inserting into pools_to_save
                self.pools_to_save
                    .insert(pool_queue_item.pool.id, pool_queue_item.pool);
            }
        }
    }
    ...
}
```

Remediation

Break out of the loop if `pool_queue_item` is flagged as unusable.

infinity-swap/src/swap_processor.rs DIFF

```diff
  if pool_queue_item.usable {
      ...
  } else {
      // If the swap was a success, but the quote price was not updated,
      // withdraw from circulation by inserting into pools_to_save
      self.pools_to_save
          .insert(pool_queue_item.pool.id, pool_queue_item.pool);
+     break;
  }
```

Patch

Resolved in 1ee941e.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent antipatterns and could lead to security issues in the future.

ID	Description
OS-ISP-SUG-00	Mismatching function names and behaviour may mislead users.
OS-ISP-SUG-01	Semantically incorrect variable names may cause confusion in future development and maintenance of code.

OS-ISP-SUG-00 | Avoid Mismatching Function Name And Behavior

Description

`execute_withdraw_all_nfts` is capped to withdraw at most 10 NFTs at once. The chosen function name may mislead users into believing all NFTs are withdrawn. In the worst case, further actions such as reconfiguring `pool` of unaware users may result in loss of funds.

infinity-swap/src/execute.rs RUST

```rust
pub fn execute_withdraw_all_nfts(
    deps: DepsMut,
    info: MessageInfo,
    pool_id: u64,
    asset_recipient: Option<Addr>,
) -> Result<Response, ContractError> {
    let withdrawal_batch_size: u8 = 10;
    let token_id_response = query_pool_nft_token_ids(
        deps.as_ref(),
        pool_id,
        QueryOptions {
            descending: None,
            start_after: None,
            limit: Some(withdrawal_batch_size as u32),
        },
    )?;

    execute_withdraw_nfts(
        deps,
        info,
        pool_id,
        token_id_response.nft_token_ids,
        asset_recipient,
    )
}
```

Remediation

Use function names that match function behavior.

infinity-swap/src/execute.rs DIFF

```diff
-  pub fn execute_withdraw_all_nfts(
+  pub fn execute_withdraw_batch_nfts(
       deps: DepsMut,
       info: MessageInfo,
```

OS-ISP-SUG-01 | Avoid Semantically Incorrect Variable Names

Description

For `execute_swap_tokens_for_any_nfts`, users provide the maximum slippage acceptable through `max_expected_token_input`. However, upon passing this to `swap_tokens_for_any_nfts`, it is renamed to `min_expected_token_input`. Avoiding semantically incorrect variable naming would boost code maintainability.

Remediation

Use a more appropriate variable name.

infinity-swap/src/swap_processor.rs DIFF

```
  pub fn swap_tokens_for_any_nfts(
      &mut self,
      storage: &'a dyn Storage,
-     min_expected_token_input: Vec<Uint128>,
+     max_expected_token_input: Vec<Uint128>,
      swap_params: SwapParams,
  ) -> Result<(), ContractError> {
-     for token_amount in min_expected_token_input {
+     for token_amount in max_expected_token_input {
          ...
      }
      Ok(())
  }
```

A Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical

Vulnerabilities that immediately lead to loss of user funds with minimal preconditions

Examples:

- Misconfigured authority or access control validation
- Improperly designed economic incentives leading to loss of funds

High

Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit.

Examples:

- Loss of funds requiring specific victim interactions
- Exploitation involving high capital requirement with respect to payout

Medium

Vulnerabilities that could lead to denial of service scenarios or degraded usability.

Examples:

- Malicious input that causes computational limit exhaustion
- Forced exceptions in normal user flow

Low

Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk.

Examples:

- Oracle manipulation with large capital requirements and multiple transactions

Informational

Best practices to mitigate future security risks. These are classified as general findings.

Examples:

- Explicit assertion of critical internal invariants
- Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.