Aquarius AMM Security Report by Certora

December 2024

Prepared for
Aquarius

certora

Table of content

certora

Project Summary

Project Scope

Project Name	Repository (link)	Commit Hash	Platform	Comment
Aquarius AMM	https://github.com/AquaToken/soroban-amm	d69959b	Stellar	original
Aquarius AMM	https://github.com/AquaToken/soroban-amm/tree/bugfix/rewards-duplicate-fix	0423af3	Stellar	Duplicate-reward-fix branch
Aquarius AMM	https://github.com/AquaToken/soroban-amm/tree/audit-Q4/final	5f9e1b6	Stellar	Audit fixes branch + some additional features
Aquarius AMM	https://github.com/AquaToken/soroban-amm/tree/V1.4.0	ab09e8a	Stellar	2nd round of audit fixes

Project Overview

This document describes the specification and verification of Aquarius AMM contracts on the Stellar Blockchain using manual code review. The work was undertaken from August 14, 2024 to September 30, 2024 with a second installment for reviewing the fixes from November 18, 2024 to December 13, 2024. The following contract list is included in our scope:

Name	Type	Description
access_control	Library	Memorizes the contract admin, enables call checks that need an admin's signature.
fees_collector	Smart contract	Collects the pool creation fees. Has no distribution logic for now - we will add it later with an upgrade.
liquidity_pool	Smart contract	AMM constant product pool. Works with 2 assets only.

liquidity_pool_events	Library	A set of events emitted by pools.
liquidity_pool_liquidity_calculator	Smart contract	Liquidity calculator for pools. Enables a comparison of different pools with the same assets to determine the amount of rewards.
liquidity_pool_plane	Smart contract	Lightweight storage of pool information. Is used for quick calculations, e.g. when distributing rewards between pools.
liquidity_pool_router	Smart contract	Pool factory. Enables new pool creation if needed. Also serves as a pool address storage and an entry point that distributes incoming calls between pools depending on the assets involved.
liquidity_pool_stableswap	Smart contract	Stable swap pool, can work with any assets.
liquidity_pool_swap_router	Smart contract	Determines the best swap path for a given pair of assets. Currently not in use as the logic was moved off-chain.
liquidity_pool_validation_errors	Library	Stores the pool validation errors.
rewards	Library	Stores and calculates AQUA rewards. Used inside the pools.
token	Smart contract	SEP-41 token interface, is used to store users' shares inside the pools.
token_share	Library	Set of methods for interactions with the `token` contract.
utils	Library	Set of methods to reduce the code size inside the contracts, for calling the smart contract storage and for tests.

During the audit period a community member identified a critical vulnerability in the original commit d69959b. Since the contracts were live at a time, the code was immediately fixed by Aquarius, and the audit proceeded working on the resulting commit . All findings which appear in the audit refer to the later commit (i.e., the "fix-duplicate-rewards") and the two subsequent branches:

- Commit 5f9e1b6 ("audit-q4") includes the fixes to the bugs that were found earlier as well as two additional features (PR #99 and #101) and extra bug fixes (PR #91 and#108).

- Commit ab09e8a ("v1.4.0") is the final pre-release version which incorporates the fixes to the problems found in the 2nd audit round.

Findings Summary

The table below provides a convenient visual summary for the findings of the review, including type and severity details.

Severity	Discovered	Confirmed	Fixed
Critical	0	0	0
High	4	4	4
Medium	4	4	4
Low	6	6	5
Informational	3	3	3
Total	**17**	**17**	**16**

The overall severity of the finding is evaluated based on two criteria - the impact of the vulnerability on the protocol (based on the protocol's stated purpose as well as historical precedents - if exist) and the likelihood of exploitation which takes into account factors such as: the complexity of the attack, the need for specific economic conditions to be met, and the overall control the attacker has over the attack cycle. As a rough guide, we give both factors equal weight and combine the two metrics according to the matrix below:

certora

Severity Matrix

Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium
		Low	Medium	High
		Likelihood		

certora

Detailed Findings

High Severity Issues

H-01 The admin action delay on fee change in the Stableswap pool can be easily bypassed

Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: liquidity_pool_liquidity_calculator/src/stableswap_pool.rs	Category: Logic	Status: Fixed

Description:

The liquidity_pool_stableswap contains a delay mechanism designed to give a fair chance to leave the pool in case of a fee/admin fee changes (in particular, against a malicious admin suddenly changing the fee structure)

```
// Sets a new fee to be applied in the future.
//
// # Arguments
//
// * `admin` - The address of the admin.
// * `new_fee` - The new fee to be applied.
// * `new_admin_fee` - The new admin fee to be applied.
```

```
fn commit_new_fee(e: Env, admin: Address, new_fee: u32, new_admin_fee: u32) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);

    if get_admin_actions_deadline(&e) != 0 {
        panic_with_error!(&e, LiquidityPoolError::AnotherActionActive);
    }
    if new_fee > MAX_FEE {
        panic_with_error!(e, LiquidityPoolValidationError::FeeOutOfBounds);
    }
    if new_admin_fee > MAX_ADMIN_FEE {
        panic_with_error!(e, LiquidityPoolValidationError::AdminFeeOutOfBounds);
    }

    let deadline = e.ledger().timestamp() + ADMIN_ACTIONS_DELAY;
    put_admin_actions_deadline(&e, &deadline);
    put_future_fee(&e, &new_fee);
    put_future_admin_fee(&e, &new_admin_fee);
}
// Applies the committed fee.
//
// # Arguments
//
```

```
// * `admin` - The address of the admin.
fn apply_new_fee(e: Env, admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);

    if e.ledger().timestamp() < get_admin_actions_deadline(&e) {
        panic_with_error!(&e, LiquidityPoolError::ActionNotReadyYet);
    }
    if get_admin_actions_deadline(&e) == 0 {
        panic_with_error!(&e, LiquidityPoolError::NoActionActive);
    }

    put_admin_actions_deadline(&e, &0);
    let fee = get_future_fee(&e);
    let admin_fee = get_future_admin_fee(&e);
    put_fee(&e, &fee);
    put_admin_fee(&e, &admin_fee);

    // update plane data for every pool update
    update_plane(&e);
}
```

as well as against ownership transfers:

```
// Commits an ownership transfer.
//
// # Arguments
//
// * `admin` - The address of the admin.
// * `new_admin` - The address of the new admin.
fn commit_transfer_ownership(e: Env, admin: Address, new_admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);

    if get_transfer_ownership_deadline(&e) != 0 {
        panic_with_error!(&e, LiquidityPoolError::AnotherActionActive);
    }

    let deadline = e.ledger().timestamp() + ADMIN_ACTIONS_DELAY;
    put_transfer_ownership_deadline(&e, &deadline);
    access_control.set_future_admin(&new_admin);
}

// Applies the committed ownership transfer.
//
```

```
// # Arguments
//
// * `admin` - The address of the admin.
fn apply_transfer_ownership(e: Env, admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);

    if e.ledger().timestamp() < get_transfer_ownership_deadline(&e) {
        panic_with_error!(&e, LiquidityPoolError::ActionNotReadyYet);
    }
    if get_transfer_ownership_deadline(&e) == 0 {
        panic_with_error!(&e, LiquidityPoolError::NoActionActive);
    }

    put_transfer_ownership_deadline(&e, &0);
    let future_admin = match access_control.get_future_admin() {
        Some(v) => v,
        None => panic_with_error!(&e, StorageError::ValueNotInitialized),
    };
    access_control.set_admin(&future_admin);
}
```

However, as it currently stands, the assurance offered by these mechanisms is actually a false one - since the liquidity pool also has an upgrade function which allows the admin to simply replace the liquidity pool contract WASM without any delay:

```
// Upgrades the contract to a new version.
  //
  // # Arguments
  //
  // * `e` - The environment.
  // * `new_wasm_hash` - The hash of the new contract version.
  fn upgrade(e: Env, new_wasm_hash: BytesN<32>) {
      let access_control = AccessControl::new(&e);
      access_control.require_admin();
      e.deployer().update_current_contract_wasm(new_wasm_hash);
  }
```

Impact: The contract has a security mechanism in place which is in reality ineffective. This is misleading to both the users and to the protocol developers themselves.

Recommendation: First, as mentioned in the appendix we propose to split the admin rule into several rules due to general security considerations.

Customer's response: Upgrade function to be split into apply + commit (with delay) functions to remember new wasm and then apply upgrade, effectively allowing users to react to malicious changes. In the case of system vulnerability fixes, delay may be bypassed by the Emergency Admin role.

Fix Review: Resolved in PR #105, merged into the audit-fixes branch 5f9e1b6.

H-02 No checks for numerical stability may lead to loss of funds

Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: liquidity_pool_liquidity_calculator/src/stableswap_pool.rs	Category: arithmetic	Status: Fixed

Impact: Aquarius's rust implementation of the pool for stablecoins is based on Curve's Solidity Stableswap implementation. Both use the Newton-Rapson method to iteratively converge to a solution for the AMM invariant equation. However there is a key difference: since we have no proof of numerical stability of this process, Curve's newer implementation of the get_d() reverts if it failed to converge fast enough:

```
@external

@pure

def get_D(

    _xp: DynArray[uint256, MAX_COINS],

    _amp: uint256,

    _n_coins: uint256

) -> uint256:

    """

    D invariant calculation in non-overflowing integer operations

    iteratively

    A * sum(x_i) * n**n + D = A * D * n**n + D**(n+1) / (n**n * prod(x_i))

    Converging solution:

    D[j+1] = (A * n**n * sum(x_i) - D[j]**(n+1) / (n**n prod(x_i))) / (A * n**n - 1)
```

```
    """

    S: uint256 = 0

    for x in _xp:

        S += x

    if S == 0:

        return 0

    D: uint256 = S

    Ann: uint256 = _amp * _n_coins

    for i in range(255):

        D_P: uint256 = D

        for x in _xp:

            D_P = D_P * D / x  # If division by 0, this will be borked: only withdrawal will work.
And that is good

        D_P /= pow_mod256(_n_coins, _n_coins)

        Dprev: uint256 = D

        # (Ann * S / A_PRECISION + D_P * _n_coins) * D / ((Ann - A_PRECISION) * D / A_PRECISION +
(_n_coins + 1) * D_P)

        D = (

            (unsafe_div(Ann * S, A_PRECISION) + D_P * _n_coins) *

            D / (

                unsafe_div((Ann - A_PRECISION) * D, A_PRECISION) +

                unsafe_add(_n_coins, 1) * D_P

            )

        )
```

```
        # Equality with the precision of 1
        if D > Dprev:
            if D - Dprev <= 1:
                return D
        else:
            if Dprev - D <= 1:
                return D
    # convergence typically occurs in 4 rounds or less, this should be unreachable!
    # if it does happen the pool is borked and LPs can withdraw via `remove_liquidity`
    raise
```

while Aquarius's get_d() implementation (see L.#33-68) *does not*, creating a potential attack vector:

```
// xp size = N_COINS
fn get_d(e: &Env, n_coins: u32, xp: &Vec<u128>, amp: u128) -> u128 {
    let mut s = 0;
    for x in xp.clone() {
        s += x;
    }
    if s == 0 {
        return 0;
    }
    let mut d_prev;
    let mut d = s;
    let ann = amp * n_coins as u128;
    for _i in 0..255 {
```

```
        let mut d_p = d.clone();
        for x1 in xp.clone() {
            d_p = d_p.fixed_mul_floor(&e, &d, &(x1 * n_coins as u128));
        }
        d_prev = d.clone();
        d = (ann * s + d_p * n_coins as u128).fixed_mul_floor(
            &e,
            &d,
            &((ann - 1) * d + (n_coins as u128 + 1) * d_p),
        );
        // // Equality with the precision of 1
        if d > d_prev {
            if d - d_prev <= 1 {
                break;
            }
        } else if d_prev - d <= 1 {
            break;
        }
    }
    d
}
```

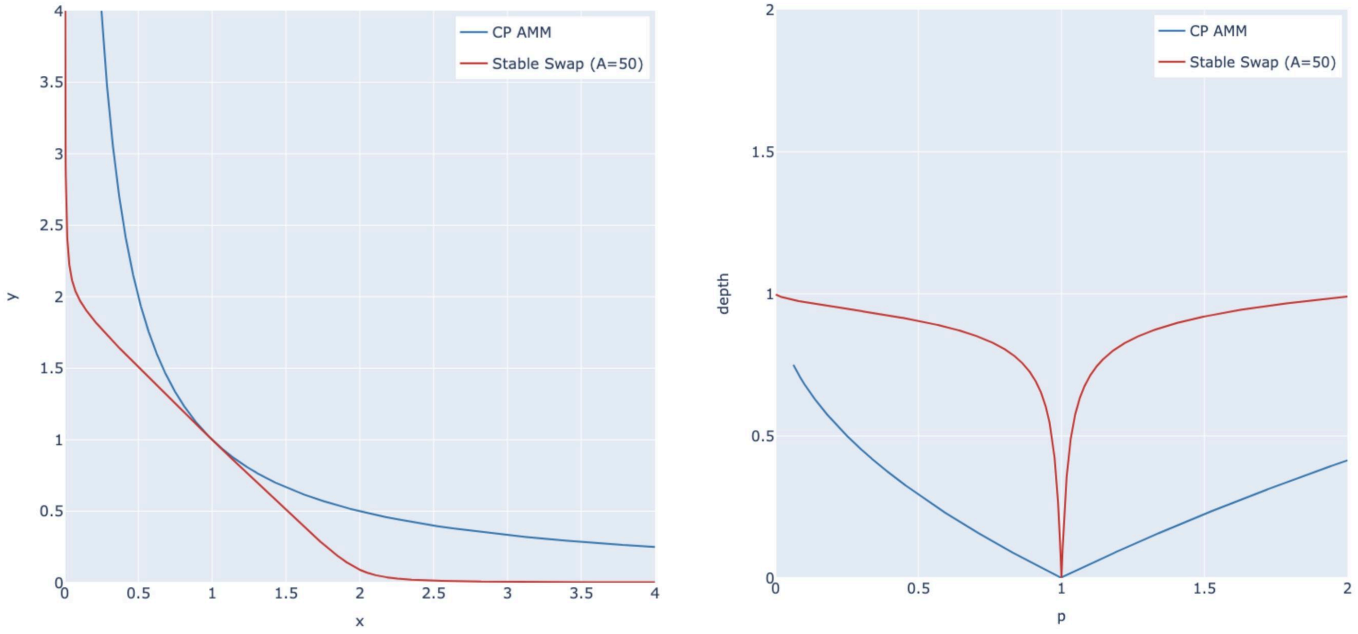
Recommendation: Change the function to revert if it needs too many rounds, so that it is compatible with Curve's implementation.

Customer's response: confirmed, to be updated.

Fix Review: Resolved in PR #94 and subsequently merged into branch 5f9e1b6.

H-03 No support for SEP-41 decimals in the Stableswap pool leads to capital inefficiency		
Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: Multiple locations	Category: Design, Economical	Status: Fixed

Description: The stableswap pool treats all input tokens as if they had the same number (seven) decimals. However, this is highly problematic since the advantage of the stableswap constant function market maker (CFMM) formula in terms of capital efficiency assumes all prices are the same, i.e., that we are scaling all the stablecoins with respect to their decimals.



Impact: A decrease in one of the key AMM utility metrics (capital efficiency) which leads to worse rates for liquidity providers and traders.

Recommendation: We suggest adding support for token decimals.

Customer's response: confirmed, stableswap decimals support to be added.

Fix Review: the issue is resolved in PR#90 and PR#104, and merged into 5f9e1b6.

certora

H-04 Incorrect amplification value in the StableSwap pool		
Severity: **High**	Impact: **Medium**	Likelihood: **High**
Files: Multiple locations	Category: Design, Economical	Status: Fixed

Description: There is a subtle contradiction in StableSwap between the formulation of the whitepaper, some of the Vyper implementations, and the comments in the codebase regarding the actual meaning of the quantities _A, Ann, and amp: for StableSwap to work optimally with the correct leverage we should initialize the value of 'a' not to be the *amplification factor* but rather the amplification factor times N_{coins} ** (N_{coins} - 1). However, different implementations of Curve's Stableswap pool throughout the years state that the value of 'a' is either the actual amplification factor or the amplification factor times N_{coins} * (N_{coins} - 1). See e.g., p.29-31 of George Weatherill's "DeFi Automated Market Makers" for a detailed explanation.

In Aquarius's case, the documentation in liquidity_pool_stableswap states that 'a' should be set to be the actual amplification factor, which is incorrect:

```
// Initializes the liquidity pool with the given parameters.
//
// # Arguments
//
// * `admin` - The address of the admin.
// * `privileged_addrs` - (
//      emergency admin,
//      rewards admin,
//      operations admin,
//      pause admin,
//      emergency pause admins
//  ).
// * `router` - The address of the router.
// * `token_wasm_hash` - The hash of the token's WASM code.
```

```
// * `tokens` - The addresses of the coins.
// * `a` - The amplification coefficient.
// * `fee` - The fee to be applied.
fn initialize(
    e: Env,
    admin: Address,
    privileged_addrs: (Address, Address, Address, Address, Vec<Address>),
    router: Address,
    token_wasm_hash: BytesN<32>,
    tokens: Vec<Address>,
    a: u128, //@audit is this a BUG?
    fee: u32,
) {
```

Further checking with the Aquarius team showed that this is indeed the case, which means that the amplification factor is actually being set incorrectly in Aquarius's StableSwap pools.

Impact: As in H-03 this is a bug, not a vulnerability. However, setting incorrect amplification value leads to worse rates for liquidity providers and traders **(we further note that** this issue worsens as the number of tokens increases).

Recommendation: We suggest fixing the comment above, ensuring that all references to a , Ann, etc in the code are aligned with the assumption that 'a' is now the actual amplification factor from the StableSwap whitepaper, and resetting the value of the 'a' parameter for any pool already deployed to its intended value.

Customer's response: confirmed. Updated comments to document amplification factor better. Calculate amp properly based on tokens number on stableswap pool creation.
PR:

Fix Review: Fixed in PR#109 and merged into ab09e8a.

Medium Severity Issues

M-01 Integration with fee-on-transfer, rebasing, or deflationary tokens can cause the protocol to be exploited		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files: Multiple locations	Category: Unusual Tokens	Status: Fixed

Description: from discussion with the development team it seems that the Aquarius AMM is meant to support any reasonable, non-malicious SEP-41 token. In particular, the Aquarius AMM code base contains code and comments alluding to support for fee-on-transfer and rebasing tokens (see e.g., the comment in L.#1324 of the swap function).

However, it is important to note that the Aquarius AMM is based almost everywhere on *internal accounting* (which makes sense due to Soroban's current heavy restrictions on I/O operations), almost any DeFi operation involving such a token would break the basic AMM invariants, wreaking havoc on the logic of the system.

Impact: Integration of fee-on-transfer, rebasing, or deflationary tokens would create a discrepancy between the protocol internal accounting and the actual balances, making the protocol exploitable.

Recommendation: We suggest documenting this restriction clearly in the code and comments and to avoid integration with such tokens.

Customer's response: confirmed. remove mentions of fee-on-transfer/rebasing tokens; document system restrictions on which tokens are supported by pools.

Fix Review: Resolved in PR#95, merged into the audit-fixes branch 5f9e1b6.

certora

M-02 Lack of scam protection for AMM Users		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Multiple locations	Category: Design, The Soroban Framework	Status: Confirmed

Description: This is a general known problem with DeFi in Stellar that was first discovered by OtterSec in their audit of Soroswap: one of the key features of the Soroban is its powerful authentication and authorization framework. However, there are some known issues (see the open issue #1092) with this framework which could have an adverse effect on inexperienced users, especially in regard to interaction with unknown tokens. Of course, scamming can be an issue in any blockchain (e.g., Solidity) but it is precisely the powerful nature of Soroban's require_auth() which makes it possible for scammers to create highly opaque authorization requests which would not only steal your balance in the specific scam token but also in any other token (e.g., XLM). See the discussion thread in the Stellar Discord Channel for a detailed explanation as well as an illustration of the attack.

Impact: Inexperienced users may loss *all* their funds in a single moment, if they are not being careful.

Recommendation: We propose that Aquarius would add token-list protection at the front-end and curate a default list of safe tokens.

Customer's response: confirmed. manual transaction authorizations were added on the front-end to protect users.

Frontend PR: https://github.com/AquaToken/dao-aquarius/pull/8

M-03 Security rules do not adhere to the 'least privilege' principle		
Severity: **Medium**	Impact: **Medium**	Likelihood: **Medium**
Files: Multiple locations	Category: Design, Key Management, Access Control	Status: Fixed

Description: The admin role in Aquarius currently design is essentially all-powerful (can upgrade the WASM code of the smart contracts in the system) while also being required for basic day to day standard operational procedures like adjusting liquidity pool parameters. The admin key also needs to be 'hot' since it is required for emergency response situations (e.g., in order to freeze withdraw/deposits in a certain pool). This is in sharp contrast to standard 'best practice' in terms of key management and the principles of role based security policy.

Impact: If the admin key is ever compromised, the entire protocol and the LP's funds deposited within it are at risk.

Recommendation: We suggest that the current admin function would be split into several less powerful roles. One possible hierarchy of such roles has been included in Appendix A.

Customer's response: confirmed, access_control module to be updated with new roles

Fix Review: Resolved in PR#100, merged into the 5f9e1b6.

certora

M-04 Inflation attack vector		
Severity: **Medium**	Impact: **High**	Likelihood: **Low**
Files:	Category: Economical	Status: Fixed

Description: As is well known, in an inflation attack, an attacker manipulates the exchange rate of an AMM (or any "ERC4626-esque" DeFi protocol) to benefit themselves at the expense of other users. By exploiting the rounding down of shares during deposit, the attacker can dilute the value of other users' deposits, see this overview by OpenZeppelin for general context. Apriori, one would assume that Aquarius's liquidity pools are unaffected by such an attack since they keep track of internal reserves which is one of the basic ways to circumvent such an attack. However, we note that this is not quite true - an admin (or any other privileged user with operational access to "admin fees") can force the external token balances and internal reserves to sync via the function donate_admin_fees:

```
fn donate_admin_fees(e: Env, admin: Address) {

    admin.require_auth();

    let access_control = AccessControl::new(&e);

    access_control.check_admin(&admin);



    let coins = get_tokens(&e);

    let mut reserves = get_reserves(&e);



    for i in 0..coins.len() {

        let token_client = SorobanTokenClient::new(&e, &coins.get(i).unwrap());

        let balance = token_client.balance(&e.current_contract_address());

        reserves.set(i, balance as u128);
```

```
        }

        put_reserves(&e, &reserves);

        // update plane data for every pool update

        update_plane(&e);

    }
```

Impact: A privileged user may carry out an inflation attack, stealing liquidity providers' deposits.

Recommendation: In the short term, we recommend to either document this option clearly (and instruct users on the importance of specifying a non-trivial slippage parameter for protection) or to make an initial deposit a mandatory part of liquidity pool creation. In the longer term, we recommend considering the use of one of the more robust methods of inflation attack protection such as virtual shares.

Customer's response: Confirmed. Remove admin fees from StableSwap pool, getting rid of vulnerable functionality.

Fix Review: Resolved in PR#96, and merged into 5f9e1b6.

Low Severity Issues

L-01 Assets could be lost in case of token address migration		
Severity: **Low**	Impact: **Medium**	Likelihood: **Low**
Files: Multiple locations	Category: Unspecified Behavior	Status: Confirmed

Description: Once a liquidity pool has been initialized, no token can be added or removed. Thus, if one of these tokens migrates to a new address, the pool has no way to follow suit. In this situation, the pool's liquidity providers are likely to withdraw their deposits as soon as possible, causing the slowest ones to lose funds.

Apriori one might assume that this can be solved by pausing the pool and then performing a simple upgrade to its WASM code. However, this is not possible since the router contract explicitly relies on the address of the pool as being derived from a process which includes the hash of its token addresses, see e.g.:

```
// Returns a map of pools for given set of tokens.
//
// # Arguments
//
// * `tokens` - A vector of token addresses that the pair consists of.
//
// # Returns
//
// A map of pool index hashes to pool addresses.
fn get_pools(e: Env, tokens: Vec<Address>) -> Map<BytesN<32>, Address> {
    let salt = get_tokens_salt(&e, tokens);
    get_pools_plain(&e, &salt)
}

// Returns a map of pools for given set of tokens.
//
```

```
// # Arguments
//
// * `tokens` - A vector of token addresses that the pair consists of.
//
// # Returns
//
// A map of pool index hashes to pool addresses.
fn remove_pool(e: Env, user: Address, tokens: Vec<Address>, pool_hash: BytesN<32>) {
    let access_control = AccessControl::new(&e);
    user.require_auth();
    access_control.check_admin(&user); //@audit isn't this just require_admin?
    let salt = get_tokens_salt(&e, tokens.clone());
    if has_pool(&e, &salt, pool_hash.clone()) {
        remove_pool(&e, &salt, pool_hash)
    }
}
```

Recommendation: In the short term, we recommend that the Aquarius team would properly document this limitation. Long term, we suggest that Aquarius study which solutions can be implemented and document what happens in the case of token migration.

Remark: this is one place where an on-chain solution might not be best since it can potentially increase the attack surface and decrease the overall trust of users in the system. But if no on-chain solution exists, the documentation should be clear and explain the off-chain mitigation strategy. For instance, if the pool's deposited value in said token is large, the Aquarius token might pause the pool and reach out to the token owner to enable migration to the new pool - if possible).

Customer's response: Confirmed. Limitation to be documented & potential solution investigation to be performed.

Documentation:
https://docs.aqua.network/aquarius-amms/what-are-aquarius-amms/system-limitations/aquarius-amm-token-address-migration-limitations-and-mitigation-strategy

L-02 Privileged Address can be set without confirmation, even to invalid values

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: Multiple locations	Category: Input Validation	Status: Fixed

Description: Everywhere in the code base (except the stableswap variant) transferring critical privileged rule is done without a challenge-response mechanism. For example in the stablecoin liquidity pool we have the functions:

```
// Commits an ownership transfer.
//
// # Arguments
//
// * `admin` - The address of the admin.
// * `new_admin` - The address of the new admin.
fn commit_transfer_ownership(e: Env, admin: Address, new_admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);
    if get_transfer_ownership_deadline(&e) != 0 {
        panic_with_error!(&e, LiquidityPoolError::AnotherActionActive);
    }
    let deadline = e.ledger().timestamp() + ADMIN_ACTIONS_DELAY;
```

```
    put_transfer_ownership_deadline(&e, &deadline);
    access_control.set_future_admin(&new_admin);
}
// Applies the committed ownership transfer.
//
// # Arguments
//
// * `admin` - The address of the admin.
fn apply_transfer_ownership(e: Env, admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);
    if e.ledger().timestamp() < get_transfer_ownership_deadline(&e) {
        panic_with_error!(&e, LiquidityPoolError::ActionNotReadyYet);
    }
    if get_transfer_ownership_deadline(&e) == 0 {
        panic_with_error!(&e, LiquidityPoolError::NoActionActive);
    }
    put_transfer_ownership_deadline(&e, &0);
    let future_admin = match access_control.get_future_admin() {
        Some(v) => v,
        None => panic_with_error!(&e, StorageError::ValueNotInitialized),
    };
    access_control.set_admin(&future_admin);
}
```

```
// Reverts the committed ownership transfer.
```

```
//
// # Arguments
//
// * `admin` - The address of the admin.
fn revert_transfer_ownership(e: Env, admin: Address) {
    admin.require_auth();
    let access_control = AccessControl::new(&e);
    access_control.check_admin(&admin);
    put_transfer_ownership_deadline(&e, &0);
}
```

But no similar mechanism exists in the “ordinary” constant product liquidity pool.

Impact: in case that a mistake happens, Aquarius would suffer a critical loss of admin (or other privileged rule) control over the pool in the case ownership was transferred to an invalid address

Recommendation: We suggest splitting this functionality into multiple parts, as was done for the Stableswap pool, and extending the challenge-response mechanism to the rest of the code base.

Customer’s response: Confirmed. Functionality to be split into apply + commit functions.

Fix Review: Resolved in PR#100 and merged into the audit-fixes branch.

certora

L-03 Empty salt used in token creation

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: liquidity_pool_stableswap/src/token.rs	Category:Logic	Status: Fixed

Description: The salt used in create contract is actually empty and does not serve any purpose

```
pub fn create_contract(e: &Env, token_wasm_hash: BytesN<32>) -> Address {
    let mut salt = Bytes::new(e); //@audit this is an empty salt.
    let salt = e.crypto().sha256(&salt);
    e.deployer()
        .with_current_contract(salt)
        .deploy(token_wasm_hash)
}
```

Recommendation: we suggest adding the relevant salt parameter.

Customer's response: Confirmed. Salt to be generated using pool tokens.

Fix Review: Resolved in PR#92 and merged into 5f9e1b6.

certora

L-04 Fee fraction check should not be hard-coded

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: liquidity_pool/src/contract.rs	Category:Input Validation	Status: Fixed

Description: In constants.rs, the FEE_MULTIPLIER for the standard constant product liquidity pool is defined as

```
pub(crate) const FEE_MULTIPLIER: u128 = 10_000;
```

This is the value that ought to be used throughout the contract. However, in L.# 145 of contract.rs, the input validation for the fee fraction during initialization has the hard-coded value of 9999 instead of being FEE_MULTIPLIER - 1:

```
        // 0.01% = 1; 1% = 100; 0.3% = 30

        if fee_fraction > 9999 {

            panic_with_error!(&e, LiquidityPoolValidationError::FeeOutOfBounds);

        }
```

Impact: if the code gets copied, refactored, or updated and the value of FEE_MULTIPLIER is changed, this could render the input check invalid or wrong.

Recommendation: change to checking if fee_fraction >= FEE_MULTIPLIER

Customer's response: Confirmed.

Fix Review: Resolved in PR#91 and merged into 5f9e1b6.

certora

L-05 Input Validation for the standard liquidity pool is scattered across multiple functions

Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files: liquidity_pool_router/src/contract.rs liquidity_pool/src/contract.rs	Category: Input Validation	Status: Fixed

Description: Part of the input validation for the creation of a standard constant product pool includes validating the tokens and that the fee_fraction fits into a reasonable pattern, see the function init_standard_pool of the liquidity pool router:

```
validate_tokens(&e, &tokens);

if !CONSTANT_PRODUCT_FEE_AVAILABLE.contains(&fee_fraction) {

    panic_with_error!(&e, LiquidityPoolRouterError::BadFee);

}
```

However, it is not obvious why the input validation should only take place in case we are creating the pool "in the right way" via the liquidity router and not as part of the initialization process of the pool itself along with the other types of input validation that are already occurring in L. # 117-134 of the initialize function:

```
if tokens.len() != 2 {

    panic_with_error!(&e, LiquidityPoolValidationError::WrongInputVecSize);

}

let token_a = tokens.get(0).unwrap();

let token_b = tokens.get(1).unwrap();

if token_a >= token_b {

    panic_with_error!(&e, LiquidityPoolValidationError::TokensNotSorted);
```

```
        }
```

Recommendation: We suggest making input validation more uniform and coherent to make the code more manageable.

Customer's response: Confirmed. Everything except sanity checks to be removed from pools.

Fix Review: Resolved in PR#103 and merged into branch 5f9e1b6.

certora

L-06 Some Admin-level operations do not emit events		
Severity: **Low**	Impact: **Low**	Likelihood: **Low**
Files:	Category: Events	Status: Fixed

Description: The following functions make substantial changes to the contract state but do not emit events:

- donate_admin_fees
- ramp_a
- stop_ramp_a
- commit_new_fee
- apply_new_fee
- revert_new_parameters
- commit_transfer_ownership
- apply_transfer_ownership
- revert_transfer_ownership

Impact: makes it harder to track the protocol using off-chain monitoring tools.

Recommendation: we propose adding the missing events.

Customer's response: Confirmed. New events to be implemented.

Fix Review: Resolved in PR#102 and PR#107, merged into audit-fixes branch 5f9e1b6.

Informational Severity Issues

I-01. Incorrect Documentation

Description: The list of the different security roles in the readme.md file of the project does not include the “emergency admin” - consider changing it and adding it.

Customer’s response: Confirmed and will be fixed.

Fix Review: Resolved in PR#110 and merged into the final version ab09e8a.

I-02. Redundant Code

Description: In a few places (e.g., lines 1004-1008 from PR#91 and the get_unused_rewards function from PR#101) the if-else subtraction structure can be replaced with Rust’s built-in saturating_sub, which would make for clearer and more concise code.

Customer’s response: Confirmed, and will be fixed.

Fix Review: resolved in PR#111 and merged into the final version ab09e8a.

I-03. Some computations can be streamlined and improved

Description: In a few places in the code we call get_rates (which multiplies everything by get_precision) and then immediately divide by get_precision - this can be simplified by simply calling get_precision_mul instead which would also prevent the possibility of overflow in rare cases where the Stableswap pool utilizes tokens with a very large decimal difference between them.

Customer’s response: Confirmed and will be fixed.

Fix Review: resolved in PR#112 and merged into the final version ab09e8a.

Appendix A – Security Roles Recommendation

The following is one possible suggestion for a security role design which adheres to the 'least privilege' principle and separates day-to-day operational and "first responders" roles from the more dangerous and powerful ones (like the pause admin or owner).

Admin Roles

Owner - can upgrade the code of the underlying smart contracts, remove or add Addresses for any of the other roles, etc. Has to be a multisig (or DAO). Not a "live" (or on-chain) role - the key does not exist in any computer connected to an outside network and is kept secure offline.

Pause Admin - can pause/unpause the execution of the entire protocol, or of a specific component (router, liquidity pool) within it.This role is not live and has to be a multisig or DAO.

Operations Admin - can add/remove pools, adjust certain parameters (e.g., like A in a stablecoin pool), and collect/donate fees. There can be multiple instances of this role (perhaps even divided by pool). It is a live role.

Rewards Admin - configure and distribute global rewards (currently called "operator" in the code). It is a live role.

Emergency Pauser - can stop the execution of the entire protocol, or of a specific component (router, liquidity pool) within it. More selectively, it can also shutdown specific functions like deposit/withdrawal or swapping. This role is live, there can be multiple Addresses which have it, and can also be given to off-chain monitoring tools or firms.

User Roles

Liquidity Providers: Liquidity Providers (a.k.a. Lenders) provide capital to the pool and earn a proportional share of the pool income in the form of LP-tokens as well as rewards. They can only deposit and withdraw capital from the pool.

Traders: Traders (or swap-users) exchange currency in a liquidity pool (or several) and pay an exchange fee in return.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.