BEOSIN
Blockchain Security

ICPSwap

Smart Contract Security Audit

No. 202412201839

Dec 20th, 2024

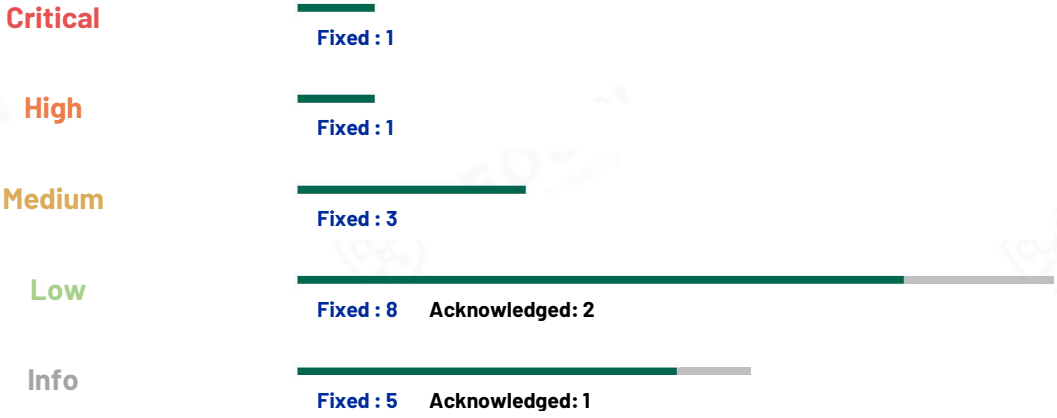
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 Critical-risk, 1 High-risk, 3 Medium-risk ,10 Low-risk and 6 Info items were identified in the ICPSwap project. Specific audit details will be presented in the Findings section. Users should pay attention to the following aspects when interacting with this project:



- **Project Description:**

Business overview

ICPSwap has implemented DEX (Decentralized Exchange) functionality similar to Uniswap-V3. It mainly consists of two functional modules: SwapFactory and SwapPool. Below is a separate explanation of the business logic for each module.

SwapFactory：

This actor is compatible with three token standards on the IC ecosystem: DIP20 (similar to ERC20 with transaction fees for transfers), ICRC1 (has subaccount, transfer method without support for `approve`/`transferFrom`), and ICRC2 (has subaccount, supports `approve`/`transferFrom` for transfers). Three fee standards (0.05%, 0.3%, 1%) have been designed for swaps in the pool. The 1% fee standard can be applied to pools with higher price volatility, the 0.3% standard is suitable for general token types, and the 0.05% standard can be used for pools with lower price volatility (such as stablecoin pools).

To create a corresponding pool, users need to choose the relevant tokens and fee standard. They can then use the `createPool` function to create the pool and store the _poolMap structure locally for recording. If the token standard is upgraded to ICRC2, the Controller of this actor can also call the `upgradePoolTokenStandard` function to modify the records.

The Controller of SwapFactory also has the capability to delete and restore data stored in the _poolMap. Additionally, it can modify the admin permissions for a pool or change the Controller of a specified pool.

SwapPool:

Compared to Uniswap-V3, users do not receive a proof of successful liquidity addition through the ownership of NFT assets after adding liquidity. Instead, they use the positionId to manage subsequent liquidity additions and removals. However, users can still transfer or authorize the position to other users.

When users add liquidity, they need to first deposit the specified tokens into SwapPool using the `deposit` function (designed for ICRC1) or the `depositFrom` function (designed for DIP20 and ICRC2). SwapPool records the user's ledger through the `_tokenHolderService`. After depositing

tokens, users can add liquidity to the pool using the `mint` function, and any excess tokens can be extracted using the `withdraw` function.

If an error occurs during token transfer when a user calls functions like `deposit` or `depositFrom`, an Error type _transferLog record will be generated. Subsequently, the admin or Controller of this actor can delete the Error Log and add a record of the user's deposit.

When calling the `mint` function to add liquidity, users need to specify the price range for addition and the desired quantities of added tokens (amount0, amount1). After a successful liquidity addition, the user's Principal identification will be bound to the corresponding liquidity positionId in the contract. This records the user's liquidity and deducts the corresponding token amounts from the _tokenHolderService user ledger. Additionally, once users hold the corresponding positionId, they can use the `approvePosition` and `transferPosition` functions to send or authorize their positionId to other users. It's important to note that the upper limit for positionId in each pool is 10000.

Users can also deposit tokens into the account with principal as SwapPool and the user as a subaccount, waiting for the administrator to call the `depositAllAndMint` function for liquidity addition. Similarly, users can use the `withdrawMistransferBalance` function later to extract any remaining token balance stored in this subaccount.

After adding liquidity, users can use their held positionId to call the `increaseLiquidity` or `decreaseLiquidity` functions to add or remove liquidity. During these operations, the pool will settle the fees collected in the form of two tokens for the tick intervals corresponding to the positionId in previous swap processes. These fees will be added to the user's tokensOwed0 and tokensOwed1 records (where the user can claim 80% of the total fees, and the remaining 20% will be transferred to the feeReceiverCid specified by the pool). Subsequently, the user's _tokenHolderService ledger state will be appropriately modified. Users can also independently call the claim function for fee settlement.

In the exchange process, unlike UniswapV3, users are not required to perform token transfers when calling the `swap` function. Instead, users need to generate the `_tokenHolderService` ledger record in advance. During the swap exchange, what actually gets updated is the `_tokenHolderService` ledger.

Token Standard Upgrade: When the token standard of the original pool is upgraded to ICRC2, this function can be called to upgrade the recorded standard.

Permission Management: This contract stores arrays for admin and whiteList for permission management. When the caller is in the _admins or Controller arrays, they can call functions such as `depositAllAndMint`, `removeErrorTransferLog`, `setAvailable`, and `setWhiteList`. When the caller is a Controller, they can call functions such as `init`, `setAdmins`, `upgradeTokenStandard`, `resetTokenAmountState`, etc.

Other Contract Modules Explanation:

PositionIndex: Every 30 seconds, it synchronizes records of created pools with SwapFactory. Users can call the `addPoolId` and `removePoolId` functions to modify which pools the user is participating in within the PositionIndex actor. When calling the `removePoolId` function, it's important to ensure that the user has no position in the specified pool for the call to be successful.

SwapFeeReceiver: The Controller of this actor can call the `transfer` function to transfer the fees stored in this actor.

The project underwent an update on Mar 6th, 2024, introducing two new actors: PasscodeManager and TrustedCanisterManager. PasscodeManager is used for users to deposit specified ICRC2 tokens and spend account balances to purchase Passcodes. Once users have a Passcode, they can create corresponding pools in SwapFactory. If users wish to cancel their purchase, they can delete the Passcode and withdraw their pledged capital.

TrustedCanisterManager implements an array whitelist of token addresses, with Controller and governance having modification permissions. Users can withdraw token types from the pool that are listed in the array (even those mistakenly transferred into the pool).

SwapFactory introduces the Passcode feature, allowing users to create pools corresponding to Passcodes. Each time a pool is created, the corresponding Passcode is destroyed. Additionally, a _checkPermission permission check is added, allowing governance and Controller of SwapFactory to modify and upgrade the token standard of pools' Controller and admin.

In SwapPool, a new _isAvailable switch restriction has been added, controlling various key functions in the actor (such as deposit, depositFrom, withdraw, mint, etc.). When the restriction is enabled, only _whiteList, _admins, and Controller have permission to call these functions.

New Features in the First Phase of the Project:

As of December 3, 2024, the project has added new features to its modules. The following sections will introduce these updates in detail.

PasscodeManager: The project has added a new `governanceCid` permission. This permission allows for the retrieval of the tokens spent by users to purchase Passcodes. Additionally, a query interface has been introduced to check the user's deposit balance.

SwapDataBackup: This is a newly added contract in the project, designed for backing up pool data (such as tick information for each pool, user positions, token balances generated from user staking, and limit orders). The backup functionality is restricted and can only be called by users with the `governanceCid`, `factoryCid`, or `Controller` permissions.

SwapFactory: The project has introduced a new installer module, allowing users to deploy pools based on the type of `subnet` (matching the installer array). It also supports bulk management for adding and removing pools, and implements module management for pool containers, including the ability to stop, upgrade, and start them. Additionally, functionality for resetting the Available permissions of pools and managing data backups has been added.

The contract will manage the following tasks for pools corresponding to the `currentUpgradeTask`, in sequential order:

`backup`: Data backup

`turnOffAvailable`: Disable calling permissions

`stop`: Stop container operation

`upgrade`: Upgrade the container

`start`: Start container operation

`turnOnAvailable`: Enable calling permissions

SwapFactoryValidator: This is a newly added contract, designed to validate function calls in `SwapFactory`, ensuring that the call parameters and permissions are correct.

SwapFeeReceiver: This contract accumulates the pool fees generated by user swaps. Every month, the claim function is automatically executed to collect fees from each pool. For pools with no ICP in the token pair, the fees will be claimed every six months and ultimately converted into ICS. ICP and ICS fees are accumulated monthly, and the ICP stored in the contract will eventually be converted into ICS and transferred to governance which means burned.

SwapPool: The contract has introduced an extension for limit orders, allowing users to add one-sided liquidity within a tick range beyond the current price. When the price fluctuation meets the specified settlement conditions, the contract will automatically remove liquidity for the user to settle the order. A user can only open one order per position, and they can also cancel the order before settlement.

SwapPoolInstaller: This is a newly added pool deployment actor, designed to extend the deployment functionality from the SwapFactory.

New Features in the Second Phase of the Project

As of December 20, 2024, ICPSwap has undergone several functional optimizations, outlined below:

While maintaining its core functionalities, the project introduced the `JobService` feature to manage various scheduled tasks. Additionally, the `InstallersValidate` function, previously part of `SwapFactoryValidator`, was optimized and transferred to SwapFactory for practical validation use. Several validations within `SwapFactoryValidator` were also refined.

Here are the detailed updates:

Job： A `JobService` was introduced to the pool for managing scheduled tasks. Each Job has the following properties:

`name`: The name of the job.

`interval`: The interval (in seconds) for the job to repeat its execution.

`job`: A function type used to execute different asynchronous functions.

`timerId`: The identifier for the timer.

`lastRun`: The last time the job was executed.

All jobs have a globally unique expiration variable, `_lastActivity`. Once this variable is updated, the jobs can continue running based on their `interval` for one day. However, if `_lastActivity` is not updated within one day, all jobs will stop.

The `JobService` provides the following functions within the valid `_lastActivity` period:

`restartJobs`: Restarts the specified job.

`stopJobs`: Stops the specified job.

These functions can only be used within the valid `_lastActivity` cycle.

SwapPool: The contract now manages several tasks through jobs, including `syncTokenFee`, `_claimSwapFeeRepurchase`, `_clearExpiredTransferLogs`, and `_syncRecords`. These tasks are handled by the `JobService`. Both the admin and Controller roles in the contract have the ability to stop and restart specific jobs.

Additionally, in the contract's various function calls, the `onActivity` method of `JobService` has been integrated to ensure the validity period of the jobs is continuously extended.

SwapFactory: The contract has introduced a validation feature for installers. There are two key checks for any newly added installer:

Controller: Only two controllers are allowed — `SwapFactory` and `governance`.

Installer's `moduleHash`: The `moduleHash` of the installer must match the one set by the `SwapFactory`.

These validations ensure that only authorized Installer with the correct configuration can be added to the contract.

1 Overview

1.1 Project Overview

Project Name	ICPswap
Project Language	Motoko
Platform	IC
Code Base	https://github.com/ICPSwap-Labs/ICPSwap-service
Commit Id	aa336c3b462dc3f4c2d266c5d46589a32e8edaa9 e0ff1c004dda3a72bd84324aca3740c69e9a64c7 f496c51ad602396af31614ac5e4f45231303fd9a a782a14daecbb2b83a61a18c4a62e7202ee20299 4cb9f5eb4fdc5ed547cd48274ddc416df896e5a2 c812904ba59bd496f46f2bfcfa4eb1338ffe570a 836a29ddc40f35a81b23e9f51eb8a7c632e9b687

1.2 Audit Overview

Audit work duration: Dec 8, 2023 – Dec 29, 2023

Update time: Mar 6, 2024, Dec 3, 2024, Dec 20, 2024

Audit team: Beosin Security Team

1.3 Audit Method

The audit methods are as follows:

1. Formal Verification

Formal verification is a technique that uses property-based approaches for testing and verification. Property specifications define a set of rules using Beosin's library of security expert rules. These rules call into the contracts under analysis and make various assertions about their behavior. The rules of the specification play a crucial role in the analysis. If the rule is violated, a concrete test case is provided to demonstrate the violation.

2. Manual Review

Using manual auditing methods, the code is read line by line to identify potential security issues. This ensures that the contract's execution logic aligns with the client's specifications and intentions, thereby safeguarding the accuracy of the contract's business logic.

The manual audit is divided into three groups to cover the entire auditing process:

The Basic Testing Group is primarily responsible for interpreting the project's code and conducting comprehensive functional testing.

The Simulated Attack Group is responsible for analyzing the audited project based on the collected historical audit vulnerability database and security incident attack models. They identify potential attack vectors and collaborate with the Basic Testing Group to conduct simulated attack tests.

The Expert Analysis Group is responsible for analyzing the overall project design, interactions with third parties, and security risks in the on-chain operational environment. They also conduct a review of the entire audit findings.

3. Static Analysis

Static analysis is a method of examining code during compilation or static analysis to detect issues. Beosin-VaaS can detect more than 100 common smart contract vulnerabilities through static analysis, such as reentrancy and block parameter dependency. It allows early and efficient discovery of problems to improve code quality and security.

2 Findings

Index	Risk description	Severity level	Status
ICPSwap-01	The project lacks a rollback mechanism	Critical	Fixed
ICPSwap-02	The incorrect handling of Tick flipping	High	Fixed
ICPSwap-03	Function logic judgment error	Medium	Fixed
ICPSwap-04	The contract lacks a token withdrawal interface	Medium	Fixed
ICPSwap-05	DepositAllAndMint function log records errors	Medium	Fixed
ICPSwap-06	Uint type conversion is not verified	Low	Fixed
ICPSwap-07	The Time.now() function does not perform precision processing	Low	Fixed
ICPSwap-08	DepositAllAndMint function verification error on token1	Low	Fixed
ICPSwap-09	Pools can be created using the same token	Low	Fixed
ICPSwap-10	The cycle of createPool function consumption problem	Low	Fixed
ICPSwap-11	User asset withdrawal is restricted	Low	Acknowledged
ICPSwap-12	Atomicity is essential for order settlement	Low	Acknowledged
ICPSwap-13	The setUpgradePoolList function can utilize a break statement to reduce cycles consumption	Low	Fixed
ICPSwap-14	Inconsistent function calls	Low	Fixed
ICPSwap-15	Orders can be added repeatedly with the same positionId	Low	Fixed
ICPSwap-16	Redundant code	Info	Partially Fixed
ICPSwap-17	Fields of type State do not match	Info	Fixed
ICPSwap-18	The Name specification	Info	Fixed
ICPSwap-19	Redundant functionality design	Info	Fixed
ICPSwap-20	The feeGrowthGlobal values in the backup will remain zero indefinitely	Info	Acknowledged
ICPSwap-21	Query functions are subject to permission restrictions	Info	Fixed

Finding Details:

[ICPSwap-01] The project lacks a rollback mechanism

Severity Level	**Critical**
Type	Business Security
Lines	SwapPool.mo #L988-1043 SwapPool.mo #L54-69
Description	In several functions of this project, there is a pattern of modifying variables first and then making external calls or performing state checks. Moreover, in case of failure, these checks or calls do not trigger a state rollback. 1.For example, in the `mint` function of SwapPool, the `_nextPositionId` is updated first before checking the user's balance status. Therefore, if the user does not have a sufficient balance, an error will be returned directly without rolling back the state modification of `_nextPositionId`. This can lead to a continuous increase in `_nextPositionId`, reaching its upper limit. Subsequently, users with a genuine need for positions may encounter difficulties in adding liquidity.

```
public shared (msg) func mint(args : Types.MintArgs) : async
Result.Result<Nat, Types.Error> {
        if (not _checkUserPositionLimit()) {
            return #err(#InternalError("Number of user position
exceeds limit"));
        };

        _saveAddressPrincipal(msg.caller);

        let positionId = _nextPositionId;
        _nextPositionId := _nextPositionId + 1;

        var amount0Desired =
SafeUint.Uint256(TextUtils.toNat(args.amount0Desired));
        var amount1Desired =
SafeUint.Uint256(TextUtils.toNat(args.amount1Desired));
        if (not _checkAmounts(amount0Desired.val(),
amount1Desired.val(), msg.caller)) {
            var accountBalance : TokenHolder.AccountBalance =
```

```
_tokenHolderService.getBalances(msg.caller);
            return #err(#InternalError("illegal balance in pool. "
                # "amount0Desired=" # debug_show
(amount0Desired.val()) # ", amount1Desired=" # debug_show
(amount1Desired.val())
                # ". amount0Balance=" # debug_show
(accountBalance.balance0) # ", amount1Balance=" # debug_show
(accountBalance.balance1)
            ));
        };
        try {
            _saveBackupData();

            var addResult = switch (_addLiquidity(args.tickLower,
args.tickUpper, amount0Desired, amount1Desired)) {
                case (#ok(result)) { result };
                case (#err(code)) {
                    throw Error.reject("mint " # debug_show (code));
                };
            };
```

2. When creating a pool, if there is a failure in obtaining the tick, it returns 0. In this case, if an error occurs during the tick retrieval, the contract assigns 0 to the tick and completes the creation instead of throwing an exception for rollback. This means that pools with prices outside the expected range can be created. Please confirm if this aligns with the project's business logic.

```
    public shared ({ caller }) func init(
        fee : Nat,
        tickSpacing : Int,
        sqrtPriceX96 : Nat,
    ) : async () {
        if (not _inited) {
            _fee := fee;
            _tickSpacing := tickSpacing;
            _sqrtPriceX96 := sqrtPriceX96;
            _tick := switch
(TickMath.getTickAtSqrtRatio(SafeUint.Uint160(sqrtPriceX96))) { case
(#ok(r)) { r }; case (#err(code)) { 0 }; };
            _maxLiquidityPerTick :=
Tick.tickSpacingToMaxLiquidityPerTick(SafeInt.Int24(tickSpacing));
```

```
            _inited := true;
            _canisterId := ?Principal.fromActor(this);
            await _syncTokenFee();
        };
    };
```

Recommendation

It is recommended to implement a mechanism across the entire project that enables the rollback of critical variable modifications in case of subsequent exceptions. For instance, consider utilizing the `Prim.Trap` function for rollback when handling exceptions.

Status

Fixed

```
   if (not _checkAmounts(amount0Desired.val(), amount1Desired.val(),
msg.caller)) {
            var accountBalance : TokenHolder.AccountBalance =
_tokenHolderService.getBalances(msg.caller);
            return #err(#InternalError("illegal balance in pool. "
                # "amount0Desired=" # debug_show
(amount0Desired.val()) # ", amount1Desired=" # debug_show
(amount1Desired.val())
                # ". amount0Balance=" # debug_show
(accountBalance.balance0) # ", amount1Balance=" # debug_show
(accountBalance.balance1)
            ));
        };
        try {
            let positionId = _nextPositionId;
            _nextPositionId := _nextPositionId + 1;

    public shared ({ caller }) func init(
        fee : Nat,
        tickSpacing : Int,
        sqrtPriceX96 : Nat,
    ) : async () {
        if (not _inited) {
            _tick := switch
(TickMath.getTickAtSqrtRatio(SafeUint.Uint160(sqrtPriceX96))) {
                case (#ok(r)) { r };
                case (#err(code)) { throw Error.reject("init pool
failed: " # code); };
            };
```

```
            _fee := fee;
            _tickSpacing := tickSpacing;
            _sqrtPriceX96 := sqrtPriceX96;
            _maxLiquidityPerTick :=
Tick.tickSpacingToMaxLiquidityPerTick(SafeInt.Int24(tickSpacing));
            _inited := true;
            _canisterId := ?Principal.fromActor(this);
            await _syncTokenFee();
        };
    };
```

[ICPSwap-02] The incorrect handling of Tick flipping

Severity Level	High
Type	Business Security
Lines	PositionTick.mo #L428-441
Description	When a user removes liquidity and the ticks of the specified Position are flipped to 0, the `_updatePosition` function deletes the current tick information instead of the flipped tick information for the two endpoints. This will result in the Position with the current tick as an endpoint becoming invalid, and the contract's business logic will not be able to operate as intended.

```
            var feeGrowthInside0X128:SafeUint.Uint256 =
SafeUint.Uint256(_data.feeGrowthInside0X128);
            var feeGrowthInside1X128:SafeUint.Uint256 =
SafeUint.Uint256(_data.feeGrowthInside1X128);

            let positionKey = "" # Int.toText(tickLower.val()) # "_"
# Int.toText(tickUpper.val()) # "";
            var position = switch (_update(positionKey,
liquidityDelta, feeGrowthInside0X128, feeGrowthInside1X128)) {
                case (#ok(p)) { p; }; case (#err(code)) { return
#err(code); };
            };

            if (liquidityDelta.val() < 0) {
                if (flippedLower)
{ _ticks.delete(Int.toText(tick.val())); };
                if (flippedUpper)
{ _ticks.delete(Int.toText(tick.val())); };
            };
            return #ok(position);
        };
```

Recommendation	It is recommended to appropriately delete tickLower and tickUpper correspondingly.
Status	**Fixed**

```
            var feeGrowthInside0X128:SafeUint.Uint256 =
SafeUint.Uint256(_data.feeGrowthInside0X128);
            var feeGrowthInside1X128:SafeUint.Uint256 =
```

```
SafeUint.Uint256(_data.feeGrowthInside1X128);

            let positionKey = "" # Int.toText(tickLower.val()) # "_"
# Int.toText(tickUpper.val()) # "";
            var position = switch (_update(positionKey,
liquidityDelta, feeGrowthInside0X128, feeGrowthInside1X128)) {
                case (#ok(p)) { p; }; case (#err(code)) { return
#err(code); };
            };

            if (liquidityDelta.val() < 0) {
                if (flippedLower)
{ _ticks.delete(Int.toText(tickLower.val())); };
                if (flippedUpper)
{ _ticks.delete(Int.toText(tickUpper.val())); };
            };
            return #ok(position);
        };
```

[ICPSwap-03] Function logic judgment error

Severity Level	**Medium**
Type	Business Security
Lines	SqrtPriceMath.mo #L55-68
Description	The primary purpose of the getNextSqrtPriceFromAmount0RoundingUp function is to get the next square root price given a delta of token0. However, there is an incomplete validation in the else branch of this function, specifically when removing the amount of token0. The calculation formula is liquidity * sqrtPX96 / (liquidity - amount * sqrtPX96), and the validation fails to simultaneously consider the possibility of the numerator (amount * sqrtPX96) not overflowing and the denominator (liquidity - amount * sqrtPX96) not underflowing. This incomplete validation creates a potential for overflow, leading to the subsequent return of an incorrect sqrtRatioNextX96, thereby affecting the calculation of the amount of tokens exchanged by the user.

```
        } else {
            // if the product overflows, we know the denominator
underflows
            // in addition, we must check that the denominator does not
underflow
            if((product.div(amount).val() != sqrtPX96.val()) and
(numerator1.val() < product.val())){ return #err("SqrtPriceMath
illegal args"); };
            var denominator:SafeUint.Uint256 =
numerator1.sub(product);
            return
switch(FullMath.mulDivRoundingUp(SafeUint.Uint256(numerator1.val())
, SafeUint.Uint256(sqrtPX96.val()),
SafeUint.Uint256(denominator.val()))) {
                case (#ok(result)) {
                    #ok(SafeUint.Uint160(result).val());
                };
                case (#err(err)) {
                    #err(err)
                }
            };
        }
```

Recommendation	It is recommended to use the form if (not (condition1 and condition2)) to check if two parallel conditions are both met.
Status	**Fixed**

```
        } else {
            // if the product overflows, we know the denominator
underflows
            // in addition, we must check that the denominator does not
underflow
            if((product.div(amount).val() != sqrtPX96.val()) and
(numerator1.val() < product.val())){ return #err("SqrtPriceMath
illegal args"); };
            var denominator:SafeUint.Uint256 =
numerator1.sub(product);
            return
switch(FullMath.mulDivRoundingUp(SafeUint.Uint256(numerator1.val())
, SafeUint.Uint256(sqrtPX96.val()),
SafeUint.Uint256(denominator.val()))) {
                case (#ok(result)) {
                    #ok(SafeUint.Uint160(result).val());
                };
                case (#err(err)) {
                    #err(err)
                }
            };
        }
```

[ICPSwap-04] The contract lacks a token withdrawal interface

Severity Level	Medium
Type	Business Security
Lines	PasscodeManager.mo #L201-220
Description	The PasscodeManager actor lacks a fee withdrawal interface. When a user spends ICRC2 tokens to purchase a Passcode and creates a pool using the createPool function of the SwapFactory actor, the Passcode is deleted. At this point, the user's _wallet balance decreases, but the quantity of tokens corresponding to passcodePrice in the PasscodeManager actor is not recorded. When the user attempts to withdraw tokens, they can only withdraw the values stored in the _wallet balance. As a result, the quantity of tokens corresponding to passcodePrice is locked in the actor and cannot be withdrawn.
Recommendation	It is recommended to add a fee record in the requestPasscode function, and an

```
    public shared({caller}) func requestPasscode(token0: Principal,
token1: Principal, fee: Nat) : async Result.Result<Text, Types.Error>
{
        if (Principal.isAnonymous(caller)) return
#err(#InternalError("Illegal anonymous call"));
        if (_walletWithdraw(caller, passcodePrice)) {
            switch(await FACTORY.addPasscode(caller, {
                token0 = token0;
                token1 = token1;
                fee = fee;
            })) {
                case(#ok()) {
                    return #ok("ok");
                };
                case(#err(msg)) {
                    _walletDeposit(caller, passcodePrice);
                    return #err(#InternalError(debug_show (msg)));
                };
            };
        } else {
            return #err(#InsufficientFunds);
        };
    };
```

admin permission-controlled fee withdrawal function.

Status **Fixed.**

```
    public shared ({ caller }) func transfer(token : Types.Token,
recipient : Principal, value : Nat) : async Result.Result<Nat,
Types.Error> {
        _checkPermission(caller);
```

[ICPSwap-05] DepositAllAndMint function log records errors

Severity Level	Medium
Type	Business Security
Lines	SwapPool.mo #L989-1011
Description	In the transfer log of the `DepositAllAndMint` function, the transfer of Token1 triggers a record for Token0 in the log, and the quantity is the transfer amount of token1 (amount1). This leads to incorrect data recording, as the update is erroneously set to amount1. If an exception occurs during the transfer, it will also be recorded in the error handling. In the processing of the `removeErrorTransferLog` function, the user's token balance will be incorrectly increased through the `deposit` function.

```
        if (args.amount1 > 0) {
            if (args.amount1 > args.fee1) {
                var amount1 : Nat = Nat.sub(args.amount1, args.fee1);
                let preTransIndex = _preTransfer(args.positionOwner,
canisterId, subaccount, canisterId, "deposit", token0, amount1,
args.fee1);
                switch (await _token1Act.transfer({
                    from = { owner = canisterId; subaccount =
subaccount }; from_subaccount = subaccount;
                    to = { owner = canisterId; subaccount = null };
                    amount = amount1;
                    fee = ?args.fee1;
                    memo =
Option.make(PoolUtils.natToBlob(preTransIndex));
                    created_at_time = null
                })) {
                    case (#Ok(index)) {
                        ignore
_tokenHolderService.deposit(args.positionOwner, _token1, amount1);
                        _postTransferComplete(preTransIndex);
                    };
                    case (#Err(msg)) {
                        _postTransferComplete(preTransIndex);
                        return #err(#InternalError(debug_show(msg)));
```

```
                    };
                };
            };
        };
```

Recommendation It is recommended to modify the log recording for the transfer of token1 to reflect token1.

Status **Fixed**

```
            if (args.amount1 > args.fee1) {
                var amount1 : Nat = Nat.sub(args.amount1, args.fee1);
                let preTransIndex = _preTransfer(args.positionOwner,
canisterId, subaccount, canisterId, "deposit", token1, amount1,
args.fee1);
                switch (await _token1Act.transfer({
                    from = { owner = canisterId; subaccount =
subaccount }; from_subaccount = subaccount;
                    to = { owner = canisterId; subaccount = null };
                    amount = amount1;
                    fee = ?args.fee1;
                    memo =
Option.make(PoolUtils.natToBlob(preTransIndex));
                    created_at_time = null
                })) {
                    case (#Ok(index)) {
                        ignore
_tokenHolderService.deposit(args.positionOwner, _token1, amount1);
                        _postTransferComplete(preTransIndex);
                    };
                    case (#Err(msg)) {
                        _postTransferComplete(preTransIndex);
                        return #err(#InternalError(debug_show(msg)));
                    };
                };
            };
```

[ICPSwap-06] Uint type conversion is not verified

Severity Level	Low
Type	Business Security
Lines	LiquidityAmounts.mo #L13-32
Description	In the LiquidityAmounts' `getLiquidityForAmount0` function, as the return value of `FullMath.mulDiv` is of type Uint256 with a range larger than the return type Uint128, it is necessary to check whether the data exceeds the upper limit of Uint128 during the type conversion process. If it exceeds the upper limit of Uint128, data truncation will occur during processing, leading to inaccurate results.
Recommendation	It is recommended to perform a value equality check both before and after type conversion to avoid data truncation.
Status	**Fixed**

```
    public func getLiquidityForAmount0(
        sqrtRatioAX96: SafeUint.Uint160,
        sqrtRatioBX96: SafeUint.Uint160,
        amount0: SafeUint.Uint256
    ): Uint128 {
        var _sqrtRatioAX96 = if(sqrtRatioAX96.val() > 
sqrtRatioBX96.val()){ sqrtRatioBX96 } else{ sqrtRatioAX96 };
        var _sqrtRatioBX96 = if(sqrtRatioAX96.val() > 
sqrtRatioBX96.val()){ sqrtRatioAX96 } else{ sqrtRatioBX96 };

        var _intermediate = FullMath.mulDiv(
            SafeUint.Uint256(_sqrtRatioAX96.val()),
            SafeUint.Uint256(_sqrtRatioBX96.val()),
            SafeUint.Uint256(FixedPoint96.Q96)
        );

        return SafeUint.Uint128(FullMath.mulDiv(
            amount0,
            SafeUint.Uint256(_intermediate),
            SafeUint.Uint256(_sqrtRatioBX96.sub(_sqrtRatioAX96).val
())
        )).val();
    };
```

```
    public func toUint128(x: Uint256) : Uint128 {
        var y = SafeUint.Uint128(x).val();
        if (not (y == x)) {
            Prim.trap("Liquidity amount overflows");
        };
        return y;
    };

    public func getLiquidityForAmount0(
        sqrtRatioAX96: SafeUint.Uint160,
        sqrtRatioBX96: SafeUint.Uint160,
        amount0: SafeUint.Uint256
    ): Uint128 {
        var _sqrtRatioAX96 = if(sqrtRatioAX96.val() >
sqrtRatioBX96.val()){ sqrtRatioBX96 } else{ sqrtRatioAX96 };
        var _sqrtRatioBX96 = if(sqrtRatioAX96.val() >
sqrtRatioBX96.val()){ sqrtRatioAX96 } else{ sqrtRatioBX96 };

        var _intermediate = FullMath.mulDiv(
            SafeUint.Uint256(_sqrtRatioAX96.val()),
            SafeUint.Uint256(_sqrtRatioBX96.val()),
            SafeUint.Uint256(FixedPoint96.Q96)
        );

        return toUint128(FullMath.mulDiv(
            amount0,
            SafeUint.Uint256(_intermediate),
            SafeUint.Uint256(_sqrtRatioBX96.sub(_sqrtRatioAX96).val
())
        ));
    };
```

[ICPSwap-07] DepositAllAndMint function verification error on token1

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L855-868
Description	In the `depositAllAndMint` function of SwapPool.mo, there is an error in the standard check for token1 where _token0's standard is incorrectly used. As long as the standard of token0 is ICRC3, the standard of token1 of any type can pass the condition check.

```
    public shared ({ caller }) func depositAllAndMint(args :
Types.DepositAndMintArgs) : async Result.Result<Nat, Types.Error> {
        if (not _checkUserPositionLimit()) {
            return #err(#InternalError("Number of user position
exceeds limit"));
        };
        _saveAddressPrincipal(caller);
        if ((args.tickLower >= args.tickUpper) or (args.tickLower <
Tick.MIN_TICK) or (args.tickUpper > Tick.MAX_TICK)) {
            return #err(#InternalError("Illegal tick number"));
        };
        if (Text.notEqual(_token0.standard, "ICP") and
Text.notEqual(_token0.standard, "ICRC1") and
Text.notEqual(_token0.standard, "ICRC2") and
Text.notEqual(_token0.standard, "ICRC3")) {
            return #err(#InternalError("Illegal token0 standard: " #
debug_show (_token0.standard)));
        };
        if (Text.notEqual(_token1.standard, "ICP") and
Text.notEqual(_token1.standard, "ICRC1") and
Text.notEqual(_token1.standard, "ICRC2") and
Text.notEqual(_token0.standard, "ICRC3")) {
            return #err(#InternalError("Illegal token1 standard: " #
debug_show (_token1.standard)));
        };
```

Recommendation	It is recommended to modify the check for the ICRC3 standard of token0 to be a check for token1.

Status **Fixed**

```
    public shared ({ caller }) func depositAllAndMint(args :
Types.DepositAndMintArgs) : async Result.Result<Nat, Types.Error> {
        if (not _checkUserPositionLimit()) {
            return #err(#InternalError("Number of user position
exceeds limit"));
        };
        _saveAddressPrincipal(caller);
        if ((args.tickLower >= args.tickUpper) or (args.tickLower <
Tick.MIN_TICK) or (args.tickUpper > Tick.MAX_TICK)) {
            return #err(#InternalError("Illegal tick number"));
        };
        if (Text.notEqual(_token0.standard, "ICP") and
Text.notEqual(_token0.standard, "ICRC1") and
Text.notEqual(_token0.standard, "ICRC2") and
Text.notEqual(_token0.standard, "ICRC3")) {
            return #err(#InternalError("Illegal token0 standard: " #
debug_show (_token0.standard)));
        };
        if (Text.notEqual(_token1.standard, "ICP") and
Text.notEqual(_token1.standard, "ICRC1") and
Text.notEqual(_token1.standard, "ICRC2") and
Text.notEqual(_token1.standard, "ICRC3")) {
            return #err(#InternalError("Illegal token1 standard: " #
debug_show (_token1.standard)));
        }
```

[ICPSwap-08] The Time.now() function does not perform precision processing

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L161-182
Description	In the `_preTransfer` function of SwapPool.mo, there is a lack of precision handling when converting the data obtained from `Time.now()` to seconds while calculating the number of days. This leads to an unusually large number of days in the calculation. During the execution of the `_clearExpiredTransferLogsJob`, due to the abnormal number of days, the judgment always passes.

```
    private func _preTransfer(owner: Principal, from: Principal,
fromSubaccount: ?Blob, to: Principal, action: Text, token: Types.Token,
amount: Nat, fee: Nat): Nat {
        let time: Nat = Int.abs(Time.now());
        let ind: Nat = _transferIndex;
        let transferLog: TransferLog = {
            index = ind;
            owner = owner;
            from = from;
            fromSubaccount = fromSubaccount;
            to = to;
            action = action;
            amount = amount;
            fee = fee;
            token = token;
            result = "processing";
            errorMsg = "";
            daysFrom19700101 = time / 86400;
            timestamp = time;
        };

    let _clearExpiredTransferLogsJob =
Timer.recurringTimer(#seconds(43200), func (): async () {
        let today: Nat = Int.abs(Time.now()) / 86400;
        for ((index, log) in _transferLog.entries()) {
            if (Nat.sub(today, log.daysFrom19700101) > 60) {
                _postTransferComplete(index);
```

```
            };
        };
    });
```

Recommendation It is recommended to perform precision handling when converting the obtained time variable to seconds to avoid an abnormal calculation of the number of days.

Status **Fixed**

```
    private func _preTransfer(owner: Principal, from: Principal,
fromSubaccount: ?Blob, to: Principal, action: Text, token: Types.Token,
amount: Nat, fee: Nat): Nat {
        let time: Nat = Int.abs(Time.now());
        let ind: Nat = _transferIndex;
        let transferLog: TransferLog = {
            index = ind;
            owner = owner;
            from = from;
            fromSubaccount = fromSubaccount;
            to = to;
            action = action;
            amount = amount;
            fee = fee;
            token = token;
            result = "processing";
            errorMsg = "";
            daysFrom19700101 = time / NANOSECONDS_PER_SECOND /
SECOND_PER_DAY;
            timestamp = time;
        };
        _transferLog.put(ind, transferLog);
        _transferIndex := _transferIndex + 1;
        return ind;
    };
```

[ICPSwap-09] Pools can be created using the same token

Severity Level	Low
Type	Business Security
Lines	SwapFactory.mo #L102-125
Description	In the `createPool` function of SwapFactory, it is allowed to create a pool with the same token. However, creating a pool with identical tokens is meaningless and results in a certain amount of resource wastage.

```
    public shared (msg) func createPool(args : Types.CreatePoolArgs) :
async Result.Result<Types.PoolData, Types.Error> {
        var tickSpacing = switch (_feeTickSpacingMap.get(args.fee)) {
            case (?feeAmountTickSpacingFee)
{ feeAmountTickSpacingFee };
            case (_) { 0 };
        };
        if (tickSpacing == 0) {
            return #err(#InternalError("TickSpacing cannot be 0"));
        };
        let poolKey : Text = PoolUtils.getPoolKey(args.token0,
args.token1, args.fee);

        if (not _lock()) {
            return #err(#InternalError("Please wait for previous
creating job finished"));
        };
        var poolData = switch
(_poolDataService.getPools().get(poolKey)) {
            case (?pool) { pool };
            case (_) {
                let pool = await _createPool(args.token0, args.token1,
args.fee, args.sqrtPriceX96, tickSpacing);
                pool
            };
        };
        _unlock();

        return #ok(poolData);
    };
```

Recommendation	It is recommended to add a check for non-identical token addresses in the `createPool` function.
Status	**Fixed**

```
    public shared (msg) func createPool(args : Types.CreatePoolArgs) :
async Result.Result<Types.PoolData, Types.Error> {
        if (Text.equal(args.token0.address, args.token1.address)) {
            return #err(#InternalError("Can not use the same token"));
        };
        var tickSpacing = switch (_feeTickSpacingMap.get(args.fee)) {
            case (?feeAmountTickSpacingFee)
{ feeAmountTickSpacingFee };
            case (_) { 0 };
        };
```

[ICPSwap-10] The cycle of createPool function consumption problem

Severity Level	Low
Type	Business Security
Lines	SwapFactory.mo #L102-125
Description	In the SwapFactory.mo file, the `createPool` function can be called arbitrarily, potentially allowing malicious users to consume cycles stored in the contract. When the stored cycles are too low, it may impact the normal business functionality of the SwapFactory.

```
    public shared (msg) func createPool(args : Types.CreatePoolArgs) :
async Result.Result<Types.PoolData, Types.Error> {
        var tickSpacing = switch (_feeTickSpacingMap.get(args.fee)) {
            case (?feeAmountTickSpacingFee)
{ feeAmountTickSpacingFee };
            case (_) { 0 };
        };
        if (tickSpacing == 0) {
            return #err(#InternalError("TickSpacing cannot be 0"));
        };
        let poolKey : Text = PoolUtils.getPoolKey(args.token0,
args.token1, args.fee);

        if (not _lock()) {
            return #err(#InternalError("Please wait for previous
creating job finished"));
        };
        var poolData = switch
(_poolDataService.getPools().get(poolKey)) {
            case (?pool) { pool };
            case (_) {
                let pool = await _createPool(args.token0, args.token1,
args.fee, args.sqrtPriceX96, tickSpacing);
                pool
            };
        };
        _unlock();
```

```
        return #ok(poolData);
    };
```

Recommendation It is recommended for users to transfer the corresponding quantity of cycles when calling the `createPool` function or to add calling permissions to the `createPool` function.

Status **Fixed.** The project charges fees for pools created using the Passcode feature.

```
    public shared (msg) func createPool(args : Types.CreatePoolArgs) :
async Result.Result<Types.PoolData, Types.Error> {
        if (not _validatePasscode(msg.caller, args)) { return
#err(#InternalError("Please pay the fee for creating SwapPool.")); };
        if (Text.equal(args.token0.address, args.token1.address))
{ return #err(#InternalError("Can not use the same token")); };
        var tickSpacing = switch (_feeTickSpacingMap.get(args.fee)) {
            case (?feeAmountTickSpacingFee)
{ feeAmountTickSpacingFee };
            case (_) { return #err(#InternalError("TickSpacing cannot
be 0")); };
        };

        if (not _lock()) { return #err(#InternalError("Please wait for
previous creating job finished")); };

        let (token0, token1) = PoolUtils.sort(args.token0,
args.token1);
        let poolKey : Text = PoolUtils.getPoolKey(token0, token1,
args.fee);
        var poolData = switch
(_poolDataService.getPools().get(poolKey)) {
            case (?pool) { pool };
            case (_) {
                try {
                    if(not _deletePasscode(msg.caller, { token0 =
Principal.fromText(token0.address); token1 =
Principal.fromText(token1.address); fee = args.fee; })) {
                        return #err(#InternalError("Passcode is not
existed."));
                    };
```

[ICPSwap-11] User asset withdrawal is restricted

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L2009-2022
Description	The `withdrawMistransferBalance` and `withdraw` functions of the SwapPool actor are both subject to the `_isAvailable` restriction, which may prevent users from withdrawing assets in a timely manner. `private func _isAvailable(caller: Principal) : Bool { if (_available and _transferLog.size() < 2000) { return true; }; if (CollectionUtils.arrayContains<Principal>(_whiteList, caller, Principal.equal)) { return true; }; if (CollectionUtils.arrayContains<Principal>(_admins, caller, Principal.equal)) { return true; }; if (Prim.isController(caller)) { return true; }; return false; };`
Recommendation	It is recommended to remove the `_isAvailable` restriction from the `withdrawMistransferBalance` and `withdraw` functions.
Status	**Acknowledged.** According to the project team, this is a brake switch used for quick response to some emergencies and does not need to be removed.

[ICPSwap-12] Atomicity is essential for order settlement

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L178-190
Description	Since the order settlement condition is triggered using `setTimer` for liquidity removal, this approach does not ensure the atomicity of the order status and may lead to inconsistent results.

```
    private func _checkLimitOrder() : async () {
        if (not _isLimitOrderAvailable) { return; };
        // backward iteration
        label lt {
            for ((key, value) in
RBTree.iter(_lowerLimitOrders.share(), #bwd)) {
                if (_tick <= key.tickLimit) {
                    _lowerLimitOrders.delete({ timestamp =
key.timestamp; tickLimit = key.tickLimit; });
                    _pushLimitOrderStack((key, value));
                    ignore Timer.setTimer<system>(#nanoseconds (0),
_autoDecrease);
```

Recommendation	It is recommended to ensure the atomicity of order settlement.
Status	**Acknowledged.**

[ICPSwap-13] The setUpgradePoolList function can utilize a break statement to reduce cycles consumption

Severity Level	Low
Type	Business Security
Lines	SwapFactory.mo #L368

Description	The `setUpgradePoolList` function contains three nested loops during iteration. In the innermost for loop, a break statement can be introduced after matching pooldata.canisterId with the parameters and modifying `_pendingUpgradePoolList`. This prevents unnecessary traversal of the remaining pooldata.canisterId, reducing cycles consumption.
	`for (poolId in args.poolIds.vals()) {`
	`label poolLoop {`
	`for ((poolKey, pooldata) in _poolDataService.getPools().entries()) {`
	`if (Principal.equal(poolId, pooldata.canisterId)) {`

Recommendation	It is recommended to use a break statement to exit the loop and reduce cycle consumption.

Status	**Fixed.**
	`_pendingUpgradePoolList := List.push(newTask, _pendingUpgradePoolList);`
	`break poolLoop; // Break after finding and processing the matching pool`

[ICPSwap-14] Inconsistent function calls

Severity Level	Low
Type	Business Security
Lines	SwapFactory.mo #L424

Description	clearRemovedPool and batchClearRemovedPool have the same functionality, but clearRemovedPool includes an additional _addPoolControllers operation, which is missing in batchClearRemovedPool.
	`public shared (msg) func clearRemovedPool(canisterId : Principal) : async Text {`
	`_checkPermission(msg.caller);`
	`await _addPoolControllers(canisterId, [feeReceiverCid]);`
	`_poolDataService.deletePool(Principal.toText(canisterId));`
	`};`
	
	`public shared (msg) func batchClearRemovedPool(poolCids : [Principal]) : async () {`

Recommendation	It is recommended to confirm whether the design aligns with business requirements.

Status	**Fixed.**
	`public shared (msg) func batchClearRemovedPool(poolCids : [Principal]) : async () {`
	`_checkPermission(msg.caller);`
	`for (poolCid in poolCids.vals()) { await _addPoolControllers(poolCid, [feeReceiverCid]); };`

[ICPSwap-15] Orders can be added repeatedly with the same positionId

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L1317

Description	A user with the same `positionId` can create multiple orders in either `_upperLimitOrders` or `_lowerLimitOrders` because the timestamp generates different keys.
	`public shared (msg) func addLimitOrder(args : Types.LimitOrderArgs) : async Result.Result<Bool, Types.Error> {`
	`assert(_isAvailable(msg.caller) and _isLimitOrderAvailable);`

Recommendation	It is recommended to add a validation ensuring that a user can only create one order per `positionId`.

Status	**Fixed.**
	`// Check if position already has an active order`
	`if (_hasActiveLimitOrder(args.positionId)) {`
	`return #err(#InternalError("Position already has an active limit order"));`
	`};`

[ICPSwap-16] Redundant code

Severity Level	Info
Type	Business Security
Lines	SwapPool.mo #L1686-1700 SwapFactory.mo #L175
Description	In the `metadata` function of SwapPool, there is redundant code. Due to the absence of field definitions for feeGrowthGlobal0X128 and feeGrowthGlobal1X128 in the type definition of PoolMetadata, the values of these two variables will not be displayed when returned. In the `_pushSwapInfoCache` function SwapPool, there is a local variable named timestamp that, after obtaining the current time, is not utilized, resulting in redundant code. The upgradePoolTokenStandard function in the SwapFactory actor does not utilize the result return value, resulting in redundant code.

```
    public query func metadata() : async
Result.Result<Types.PoolMetadata, Types.Error> {
        var metadata = {
            key = PoolUtils.getPoolKey(_token0, _token1, _fee);
            token0 = _token0;
            token1 = _token1;
            fee = _fee;
            tick = _tick;
            liquidity = _liquidity;
            sqrtPriceX96 = _sqrtPriceX96;
            maxLiquidityPerTick = _maxLiquidityPerTick;
            feeGrowthGlobal0X128 = _feeGrowthGlobal0X128;
            feeGrowthGlobal1X128 = _feeGrowthGlobal1X128;
        };
        #ok(metadata);
    };

    private func _pushSwapInfoCache(
        action : Types.TransactionType,
        from : Text,
        to : Text,
        recipient : Text,
```

```
        liquidityChange : Nat,
        token0ChangeAmount : Nat,
        token1ChangeAmount : Nat,
        zeroForOne : Bool,
    ) : () {
        var timestamp = Time.now();
        var poolCid : Text =
Principal.toText(Principal.fromActor(this));
        let (token0Id, token1Id, token0Standard, token1Standard,
token0Amount, token1Amount) = if (zeroForOne) {
            (_token0.address, _token1.address, _token0.standard,
_token1.standard, _tokenAmountService.getTokenAmount0(),
_tokenAmountService.getTokenAmount1());
        } else {
            (_token1.address, _token0.address, _token1.standard,
_token0.standard, _tokenAmountService.getTokenAmount1(),
_tokenAmountService.getTokenAmount0());
        };

 let poolKey : Text = PoolUtils.getPoolKey(metadata.token0,
metadata.token1, metadata.fee);
                    if (isSupportedICRC2) {
                        let result = await
poolAct.upgradeTokenStandard(tokenCid);
                        switch (await poolAct.metadata()) {
                            case (#ok(verifiedMetadata)) {
```

Recommendation It is recommended to evaluate the result return value.

Status **Partially Fixed**

```
    public query func metadata() : async
Result.Result<Types.PoolMetadata, Types.Error> {
        var metadata = {
            key = PoolUtils.getPoolKey(_token0, _token1, _fee);
            token0 = _token0;
            token1 = _token1;
            fee = _fee;
            tick = _tick;
            liquidity = _liquidity;
            sqrtPriceX96 = _sqrtPriceX96;
```

```
            maxLiquidityPerTick = _maxLiquidityPerTick;
            nextPositionId = _nextPositionId;
        };
        #ok(metadata);
    };

    private func _pushSwapInfoCache(
        action : Types.TransactionType,
        from : Text,
        to : Text,
        recipient : Text,
        liquidityChange : Nat,
        token0ChangeAmount : Nat,
        token1ChangeAmount : Nat,
        zeroForOne : Bool,
    ) : () {
        var poolCid : Text = Principal.toText(Principal.fromActor(this));
        let (token0Id, token1Id, token0Standard, token1Standard, token0Amount, token1Amount) = if (zeroForOne) {
            (_token0.address, _token1.address, _token0.standard, _token1.standard, _tokenAmountService.getTokenAmount0(), _tokenAmountService.getTokenAmount1());
        } else {
            (_token1.address, _token0.address, _token1.standard, _token0.standard, _tokenAmountService.getTokenAmount1(), _tokenAmountService.getTokenAmount0());
        };
```

[ICPSwap-17] Fields of type State do not match

Severity Level	Info
Type	Business Security
Lines	SwapPool.mo #L107-111
Description	The definition of the state in SwapRecord includes only three fields (records, retryCount, errors). However, during the initialization of this state in SwapPool, four fields are utilized, resulting in a mismatch of fields.

```
public type State = {
    // infoCid : Text;
    records : [Types.SwapRecordInfo];
    retryCount : Nat;
    errors : [Types.PushError];
};

private stable var _recordState : SwapRecord.State = {
    records = [];
    retryCount = 0;
    errors = [];
    infoCanisterAvailable = true;
};
```

Recommendation	It is recommended to ensure that the initialization parameters match the structure type.
Status	**Fixed**

```
public type State = {
    records : [Types.SwapRecordInfo];
    retryCount : Nat;
    errors : [Types.PushError];
};

private stable var _recordState : SwapRecord.State = {
    records = [];
    retryCount = 0;
    errors = [];
};
```

[ICPSwap-18] The Name specification

Severity Level	Info
Type	Business Security
Lines	SwapPool.mo #L536-546
Description	In the `_computeSwap` function of SwapPool, the names of the parameters returned are identical to the locally defined variable names within the function, leading to a naming convention issue.

```
        var amount0 = if (args.zeroForOne) {
            SafeInt.Int256(amountIn).sub(SafeInt.Int256(state.amoun
tSpecifiedRemaining)).val();
        } else { state.amountCalculated };
        var amount1 = if (args.zeroForOne) { state.amountCalculated }
else {
            SafeInt.Int256(amountIn).sub(SafeInt.Int256(state.amoun
tSpecifiedRemaining)).val();
        };
        return #ok({
            amount0 = amount0;
            amount1 = amount1;
        });
```

Recommendation	It is recommended to modify variable names to comply with naming conventions.
Status	**Fixed**

```
    public type State = {
        records : [Types.SwapRecordInfo];
        retryCount : Nat;
        errors : [Types.PushError];
    };

    private stable var _recordState : SwapRecord.State = {
        records = [];
        retryCount = 0;
        errors = [];
    };
```

[ICPSwap-19] Redundant functionality design

Severity Level	Info
Type	Business Security
Lines	SwapDataBackup.mo #L42 216

Description	`getBackupData` and `getPoolBackup` have identical functionality. It is important to verify whether both functions are necessary and whether they align with business needs.
	`public query func getPoolBackup(poolCid: Principal) : async Result.Result<PoolBackupData, Types.Error> {`
	`switch(_poolBackupMap.get(poolCid)) {`
	`public query func getBackupData(poolCid : Principal) : async Result.Result<PoolBackupData, Types.Error> {`
	`switch (_poolBackupMap.get(poolCid)) {`

Recommendation	It is recommended to remove redundant utility functions.
Status	**Fixed**

[ICPSwap-20] The feeGrowthGlobal values in the backup will remain zero indefinitely

Severity Level	Info
Type	Business Security
Lines	SwapDataBackup.mo #L176
Description	During the backup process, the values of `feeGrowthGlobal0X128` and `feeGrowthGlobal1X128` will remain zero, and will not reflect the accumulated fee situation.
Recommendation	It is recommended to confirm whether the design meets the business requirements.
Status	**Acknowledged**

```
        var amount0 = if (args.zeroForOne) {
            SafeInt.Int256(amountIn).sub(SafeInt.Int256(state.amoun
tSpecifiedRemaining)).val();
        } else { state.amountCalculated };
        var amount1 = if (args.zeroForOne) { state.amountCalculated }
else {
            SafeInt.Int256(amountIn).sub(SafeInt.Int256(state.amoun
tSpecifiedRemaining)).val();
        };
        return #ok({
            amount0 = amount0;
            amount1 = amount1;
        });
```

[ICPSwap-21] Query functions are subject to permission restrictions

Severity Level	Info
Type	Business Security
Lines	SwapPool.mo #L536-546
Description	The query function is restricted by permissions, which may hinder users from accessing asset information. `public shared (msg) func getTokenBalance() : async { token0 : Nat; token1 : Nat; } { assert(_isAvailable(msg.caller));`
Recommendation	It is recommended to remove the permission restrictions on the query function.

Status	**Fixed**
	`public query func getUserPosition(positionId : Nat) : async Result.Result<Types.UserPositionInfo, Types.Error> { let refreshResult = switch (_refreshIncome(positionId)) {`
	`public query func getUserPositionIds() : async Result.Result<[(Text, [Nat])], Types.Error> { return #ok(Iter.toArray(_positionTickService.getUserPositionIds().entries()));`

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1(Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Likelihood \ Impact	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	Medium	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		assert Usage
		Cycles Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		Returned Value Security
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

X
https://x.com/Beosin_com

Email
service@beosin.com