BEOSIN
Blockchain Security

ICPSwap

Smart Contract Security Audit

No. 202511121336

Nov 12nd, 2025

SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 3 Low-risks and 1 Info item 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:

Low	Fixed : 1	Acknowledged: 2
Info	Fixed : 1	

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.

New Features in the Third Phase of the Project

An audit was conducted on May 29, 2025, focusing on newly added functionalities. The core improvement introduces the use of _txState to track the transfer status during various pool operations. Additionally, the token swap process has been optimized by simplifying the execution flow. The following sections provide a detailed overview.

transaction Module

The transaction module now implements state update functions for various pool operations. When a user performs a transaction involving transfers, a corresponding transaction entry is created and its status is updated based on the execution outcome.

PositionIndex

Enhancements were made to track user participation and interaction with different markets.

SwapPool

Transaction state tracking has been added to all core operations. Functions such as depositFromAndSwap, depositAndSwap, and depositAllAndMint were introduced to streamline processes and reduce complexity. Additionally, failed transactions are now subject to periodic cleanup.

SwapPoolInstaller

This module is responsible for managing code upgrades for SwapPool.

Update description for Nov 10, 2025:

This update to the project includes iterative verification of project version numbers and permission management for Factory over other Canisters. The number of pools for managing Canister tasks has been optimized from the original upper limit of 500 to 1000.

For the pool, this major optimization focuses on the implementation of the withdrawal function. Each time a user withdraws (the corresponding operation has already made advance changes to the ledger), a corresponding withdrawal transaction ID will be created and the current withdrawal information will be pushed to a dedicated withdrawal queue for centralized processing. This processing will automatically handle each withdrawal transaction. If there is an exception during processing, this transaction ID will be marked as failed and will be handled by subsequent administrators. A new function has been added for withdrawing to token sub-accounts.

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	a5b5eb95c93435d71fef0cc3c0ab5391bd0c6fb4 9a8dc8c7dd5b215780053c70f7f14cda3d84edd2

1.2 Audit Overview

Audit work duration: Nov 3,2025 – Nov 12,2025

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-1	User asset withdrawal is restricted	Low	Acknowledged
ICPSwap-2	Atomicity is essential for order settlement	Low	Acknowledged
ICPSwap-3	The total token supply can be modified	Low	Fixed
ICPSwap-4	Redundant code	Info	Fixed

Finding Details:

[ICPSwap-1] 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-2] 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-3] The total token supply can be modified

Severity Level	Low
Type	Business Security
Lines	SwapPool.mo #L 2285-2291
Description	The token0 and token1 records in the pool can be arbitrarily modified by the admin. If these values are improperly changed, it may cause discrepancies between the actual token balances and the contract's recorded values. Furthermore, if the modified values are set too small, it could lead to overflow issues in other business logic operations.

```
    public shared (msg) func setTokenAmountState(token0Amount : Nat,
token1Amount : Nat) : async Result.Result<(), Types.Error> {
        _checkAdminPermission(msg.caller);
        if (_available) { return #err(#InternalError("Pool should not be
available")); };
        _tokenAmountService.setTokenAmount0(token0Amount);
        _tokenAmountService.setTokenAmount1(token1Amount);
        return #ok();
    };
```

Recommendation	It is recommended to evaluate whether this functionality is necessary based on the specific business requirements.
Status	**Fixed.** The project team has removed this feature.

[ICPSwap-4] Redundant code

Severity Level	Info
Type	Coding Conventions
Lines	lib.mo #L 53 292

Description	1.Since assert(false) halts execution, the subsequent loop statement is never reached, making it redundant.
	2.In the _withdraw function, the token transfer fee check is performed twice, resulting in redundant validation.
	3.In the inspect function, the statement let _ = arg; assigns arg to an anonymous variable, which has no practical effect and serves no functional purpose in the code.
	`case null { assert(false); loop {} };`
	`case(_) { assert(false); (0, null) };`
	
	`system func inspect({`
	`    arg : Blob;`
	`    caller : Principal;`
	`    msg : Types.SwapPoolMsg;`
	`}) : Bool {`
	`    let _ = arg;`

Recommendation	It is recommended to remove redundant code.
Status	**Fixed.** `case(_) { assert(false) };`

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.

Impact / Likelihood	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