BEOSIN
Web3 Security & Compliance

ICPEX

Smart Contract Security Audit

No. 202409261635

Sep 26th, 2024

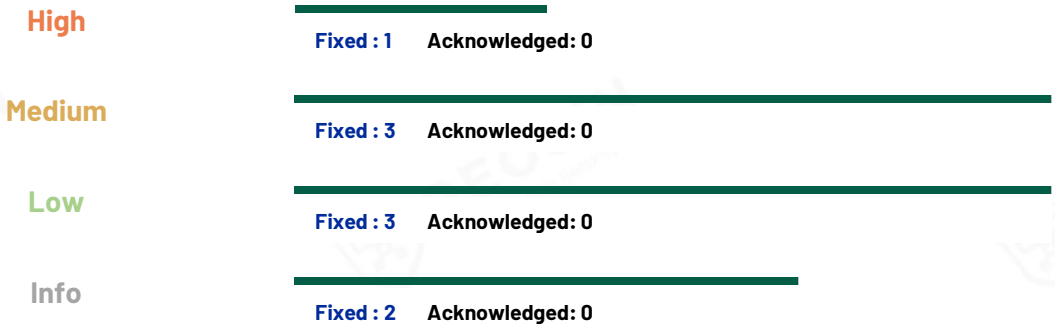
SECURING BLOCKCHAIN ECOSYSTEM

WWW.BEOSIN.COM

Contents

Summary of Audit Results

After auditing, 1 High, 3 Medium, 3 Low risk and 2 Info items were identified in the ICPEX 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:**

ICPEX is a decentralized exchange (DEX) that utilizes an innovative PMM (Proactive Market Maker) algorithm to provide higher capital efficiency and flexibility. Here's a brief overview of its key features:

Private Pool

The private pool allows liquidity providers to fully control the pool's parameters, such as price curves, liquidity amounts, and fee rates. Only the creator can add liquidity or adjust the settings, giving them the flexibility to manage assets according to their personalized needs.

Stable Pool

The stable pool is designed for trading stablecoins or pegged assets (e.g., USDT/USDC), offering low slippage and higher capital efficiency. It features a precise pricing mechanism that reduces price volatility and transaction costs, ideal for assets with stable price ranges.

Common Pool

The common pool, powered by ICPEX's PMM algorithm, is open to all liquidity providers, allowing them to contribute liquidity in a more dynamic and efficient manner. The pool adjusts liquidity distribution automatically to optimize capital usage and reduce impermanent loss. It's designed to function efficiently across different market conditions, making it accessible and beneficial to a wide range of users.

ICRC-2 Token Create

Users can create and manage ICRC-2 compliant tokens through contracts in the backend.

front_run interception detection

Transactions in ICPEX are executed in queues, and it is not possible to preempt transactions by increasing fees, etc.

1 Overview

1.1 Project Overview

Project Name	ICPEX
Project Language	Rust
Platform	Internet Computer
Code base	https://github.com/ICPExchange/icpex-contracts
Commit	8b49d7d040d67c3ea290b7eaed047b5aefb9dec1 d8e174eba55f2d2e2981590290f1dfbecd8aaf37 791cbd95ed9acaf3940c297c093664961ed74bfb e911db76773e555878501978501d13f81bea0258 5290021454185b1c50369caa3ca18f0ed19d00e5

1.2 Audit Overview

Audit work duration: Sep 5, 2024 – Sep 26, 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
ICPEX-01	Private liquidity can be added and withdrawn at will	High	Fixed
ICPEX-02	Sell rollback causes abnormal consumption of funds in the pool	Medium	Fixed
ICPEX-03	Liquidity issue with the createStablePool function	Medium	Fixed
ICPEX-04	The meaning of the parameters before and after is inconsistent	Medium	Fixed
ICPEX-05	The slippage check should use the actual accepted value	Low	Fixed
ICPEX-06	User balance judgment does not take into account transaction fees	Low	Fixed
ICPEX-07	Code implementation does not match the comments	Low	Fixed
ICPEX-08	The error message does not match	Info	Fixed
ICPEX-09	Redundant code	Info	Fixed

Finding Details:

[ICPEX-01] Private liquidity can be added and withdrawn at will

Severity Level	High
Lines	pmmswap/router/src/pmmswap/router/src/main.rs#L1325-1327
Type	Business Security
Description	1. In the `innerAddPrivateLiquidity` function, there is no error handling for `base_token` transfers like in stable and common. Combined with the fact that anyone can add liquidity to private before, malicious users can add liquidity and profit by not sending `base_token`. 2. In addition, there is no share casting and destruction when adding and removing liquidity in the private pool, and there is no restriction on the `sell_share` function. As a result, anyone can remove any amount of liquidity from the private pool. This is different from another PMM algorithm project DoDo, where only the owner can add and remove liquidity in the private pool.
Recommendation	1. It is recommended to use the throw method for `base_token` transfer. 2. Since anyone is allowed to add liquidity, it is necessary to add code implementation in related functions such as `sell_share` to prevent liquidity from being arbitrarily withdrawn and causing user asset losses.
Status	**Fixed.** Now only the owner of the private pool can add and remove liquidity.

[ICPEX-02] Sell rollback causes abnormal consumption of funds in the pool

Severity Level	Medium
Lines	pmmswap/router/src/pools/lpc_pool.rs#L150-175
Type	Business Security
Description	Taking the sell_base function in lpc_pool as an example, the pool will first transfer the handling fee, and then the user's transfer. When the user transfer fails, the handling fee transfer will be rolled back. Considering the handling fee of token transfer, this will cause the following problems: Assuming a 10% fee rate, the pool sends 4.5 tokens to the router, and the actual pool deducts 5. After the rollback is triggered, the pool receives a refund of 4.05 from the router. The pool's token is 0.95 less. Malicious addresses may consume the quote token in the pool through this method to affect the price.
Recommendation	It is recommended to place the receive_amount transfer logic before the fee transfer and delete fee_rollback.
Status	**Fixed.** Changed transfer order and removed fee_rollback.

```
   pub async fn sell_base(&mut self, to: Principal) -> (Nat,Nat,Nat,
Principal,Nat,Nat) {
...
       if mt_fee_transfer>new_zero() {
           throw(pool_proxy.transfer(self.quote_token.cid.clone(),id
().clone(),mt_fee_transfer.clone()).await.map_err(|e| "MtFee Failed ->
".to_string() + &*e.1));
       }
       if receive_quote_amount_transfer > new_zero() {
           let trans_res =
pool_proxy.transfer(self.quote_token.cid.clone(),to,receive_quote_am
ount_transfer.clone()).await;
           _fee_rollback( trans_res.clone(),self.quote_token.clone()
,self.pool_addr.clone(),mt_fee_transfer.clone()).await;
           throw(trans_res.map_err(|e| "Retrieve Failed ->
".to_string() + &*e.1));
       }
...
}
```

```
    pub async fn sell_base(&mut self, to: Principal) -> (Nat,Nat,Nat,
```

```
Principal,Nat,Nat) {
...
        if receive_quote_amount_transfer > new_zero() {
            let trans_res =
pool_proxy.transfer(self.quote_token.cid.clone(),to,receive_quote_am
ount_transfer.clone()).await;
            throw(trans_res.map_err(|e| "Retrieve Failed ->
".to_string() + &*e.1));
        }
        if mt_fee_transfer>new_zero() {
            throw(pool_proxy.transfer(self.quote_token.cid.clone(),id
().clone(),mt_fee_transfer.clone()).await.map_err(|e| "MtFee Failed ->
".to_string() + &*e.1));
        }
```

[ICPEX-03] Liquidity issue with the createStablePool function

Severity Level	Medium
Lines	pmmswap/router/src/lps_pool.rs#L225-239
Type	Business Security
Description	Since the StablePool allows unilateral liquidity initialization, the share will always be zero during the final calculation, leading to a loss of funds for the user.

```
#[update(name = "innerAddStableLiquidity")]
#[candid_method(update, rename = "innerAddStableLiquidity")]
async fn inner_add_lps_liquidity(pool_addr: Principal, base_in_amount:
Nat, quote_in_amount: Nat, slippage: Nat, deadline: u64, mut
pool_change_tx: PoolTxRecord) -> (Nat, Nat, Nat) {
    print("router addStableLiquidity ");
    assert!(deadline >= ic_cdk::api::time(), "EXPIRED");
    let guard = CallerGuard::new(pool_addr.clone());
    if guard.is_err() {
        panic!("{}", guard.err().unwrap());
    }
    ...
    }
```

Recommendation	It is recommended to validate that `quote_input` is greater than 0 in the branch if `self.shares_ledger.total_supply == new_zero()` and modify the logic to require bilateral additions.
Status	**Fixed.** Added relevant judgments when adding liquidity.

```
#[update(name = "innerAddStableLiquidity")]
#[candid_method(update, rename = "innerAddStableLiquidity")]
async fn inner_add_lps_liquidity(pool_addr: Principal, base_in_amount:
Nat, quote_in_amount: Nat, slippage: Nat, deadline: u64, mut
pool_change_tx: PoolTxRecord) -> (Nat, Nat, Nat) {
    print("router addStableLiquidity ");
    assert!(deadline >= ic_cdk::api::time(), "EXPIRED");
    assert!(base_in_amount > new_zero(), "NO_BASE_INPUT");
     assert!(quote_in_amount > new_zero(), "NO_QUOTE_INPUT");
let guard = CallerGuard::new(pool_addr.clone());
    if guard.is_err() {
        panic!("{}", guard.err().unwrap());
```

[ICPEX-04] The meaning of the parameters before and after is inconsistent

Severity Level	Medium
Lines	pmmswap/router/src/main.rs#L1447 pmmswap/router/src/pools/lpc_pool.rs#L306-321
Type	Business Security

Description	In the logic of the `sellShares` function, the value passed into the factory's `sell_shares` function is `share_amount`, which is a clear amount. However, when executed to the corresponding pool, the parameter becomes `share_rate`, which is considered a ratio in the calculation.
	`#[update(name = "sellShares")]`
	`#[candid_method(update, rename = "sellShares")]`
	`async fn sell_shares(share_amount: Nat, pool_addr: Principal, base_min_amount: Nat, quote_min_amount: Nat, slippage: Nat, deadline: u64) -> (Nat, Nat)`
	`...`
	`pub async fn sell_shares(&mut self, share_rate: Nat, to: Principal, base_min_amount: Nat, quote_min_amount: Nat, slippage: Nat, deadline: u64) ->(Nat, Nat,Nat) {`

Recommendation	It is recommended to modify the function implementation or algorithm according to the design requirements to make the parameters consistent.
Status	**Fixed.** It has been standardized to `share_rate`.

[ICPEX-05] The slippage check should use the actual accepted value

Severity Level	Low
Lines	pmmswap/router/src/pools/lpc_pool.rs#L326
Type	Business Security
Description	In the `sell_shares` function, the slippage check checks a quantity (`base_amount`) that is not the actual quantity received by the user (`receive_amount`), and `receive_amount` will be smaller than `base_amount`, which may result in the actual arrival of the user's account being different from what was expected.
Recommendation	It is recommended that the base_amount in slippage_check be changed to receive_amount.
Status	**Fixed.**

[ICPEX-06] User balance judgment does not take into account transaction fees

Severity Level	Low
Lines	pmmswap/router/src/main.rs#L885-909
Type	Business Security
Description	The `get_icp_platform_fee` function only reduces the authorization value by `pool_token_amount` without checking the user's `total_balance`. This may result in insufficient `pool_token_amount` of tokens to add liquidity after the user pays the handling fee.
Recommendation	It is recommended to subtract `pool_token_amount` from total_balance before determining total_balance < min_fee.
Status	**Fixed.**

[ICPEX-07] Code implementation does not match the comments

Severity Level	Low
Lines	backend/src/main.rs#L258-277 pmmswap/router/src/main.rs#L947-957
Type	Coding Conventions
Description	The `is_base64_image_smaller_than_80kb` function name is 80kb, but the actual value used in the judgment is 100kb. The i = 10^5 * 10^(quote decimal - base decimal) in the `inner_create_common_pool` function comment actually uses 10000 (10^4).
Recommendation	It is recommended to modify the code implementation or comments according to actual needs.
Status	**Fixed.** The function name has been changed to `is_base64_image_smaller_than_100kb` and the error comment for `i` has been removed.

```
fn is_base64_image_smaller_than_80kb(logo: &str) -> Result<(), String>
{
...{
        return Err("The logo must be in base64(png,jepg,jpg) format and
smaller than 100KB.".to_string());
    }
    let baseLogo = logo.split(",").nth(1).unwrap_or("");
    match base64::decode(baseLogo) {
        Ok(decoded_data) => {
            if(decoded_data.len() < 100 * 1024){
                Ok(())
            }else{
                Err("The logo must be in base64 format and smaller than
100KB.".to_string())
...
 //i = 10^5 * 10^(quote decimal - base decimal)
        let i_u128 = BigDecimal::from(10000 as
u32).mul(10u128.pow(quote_decimal.clone() as
u32)).div(10u128.pow(base_decimal.clone() as
u32)).to_u128().unwrap();
        print(format!("adjust i u128:{}",i));
```

[ICPEX-08] The error message does not match

Severity Level	Info
Lines	pmmswap/router/src/main.rs#L1641-1650
Type	Coding Conventions

Description	The parameter check does not match the comments. The error message returned by the token type cannot be the same is the value 0.
	`async fn _swap_token_to_token(order_hash: u64, swap_caller: Principal, base_from_token: Principal, base_to_token: Principal, base_from_amount: Nat, base_min_return_amount: Nat, pairs: Vec<Principal>, mut directions: u64, deadline: u64) -> Nat {`
	`...`
	`assert!(!&base_to_token.eq(&base_from_token), "RETURN_AMOUNT_ZERO");`

Recommendation	It is recommended to modify the error message to ensure consistency with the logic.

Status	**Fixed.**
	`async fn _swap_token_to_token(order_hash: u64, swap_caller: Principal, base_from_token: Principal, base_to_token: Principal, base_from_amount: Nat, base_min_return_amount: Nat, pairs: Vec<Principal>, mut directions: u64, deadline: u64) -> Nat {`
	`...`
	`assert!(!&base_to_token.eq(&base_from_token), "The tokens for both parties in the transaction cannot be the same.");`

[ICPEX-09] Redundant code

Severity Level	Info
Lines	backend/src/main.rs#L281-504
Type	Coding Conventions

Description	The `createToken` function already requires that the token type must be ICRC-2, so the subsequent judgment of DIP20 is redundant code.
	`if &token_type != "ICRC-2" {`
	`return Err("Currently, only ICRC-2 is supported.".to_string());`
	`}`
	`...`
	`if TokenType::ICRC2.to_string() == token_type {`
	`...`
	`} else if TokenType::DIP20.to_string() == token_type {`
	`...`

Recommendation	It is recommended to delete redundant code.
Status	**Fixed.** Redundant code about DIP20 has been removed.

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.

3.1.3 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.

3.1.4 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
Web3 Security & Compliance

Official Website
https://www.beosin.com

Telegram
https://t.me/beosin

Twitter
https://twitter.com/Beosin_com

Email
service@beosin.com