ICPEx Audit Report

Prepared by VelintSec
Version 1.1

Lead Auditors
MaxMa
Vincent

April 1st, 2025 - July 1st ,2025

Table of contents

1 About VelintSec

VelintSec is a full-service security service company based on the blockchain and Web3 ecosystem, focusing on the ICP ecosystem, focusing on providing security audit, protection and consulting services for smart contracts, blockchain projects and centralized systems. Our mission is to establish and maintain the highest security standards in the blockchain industry through rigorous auditing, continuous innovation, and dedicated support. Learn more about us at velintsec.com.

2 Disclaimer

The VelintSec team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the The Internet Computer (ICP) implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

ICPEx is a high-performance decentralized exchange deployed on the Internet Computer (IC), utilizing a single Canister architecture to achieve atomic operations for order matching and fund settlement, eliminating cross-contract communication latency.

All assets and liquidity pool data are centrally stored in one main Canister, with transactions completed through a single Update Call, requiring only 2 seconds to execute trades for any trading pair.

Transaction records are asynchronously written to independent log Canisters, ensuring both system performance and data traceability.

The core contract is governed by SNS (Service Nervous System), achieving complete decentralization for on-chain parameter upgrades and permission control.

Emergency pause mechanisms, anomaly monitoring, and other security protections have been implemented, with multiple rounds of open-source and community testing conducted.

5 Audit Content and Scope

Audit Target: ICPEx Core Backend Contract Code

GitHub Repository: https://github.com/ICPExLabs/swap-canister-next

Audit Scope:

- Core execution paths including order matching, swap execution, fund settlement, fee logic, and logging mechanisms
- Excludes frontend code, indexing services, UI display modules, and other non-critical components

Audit Objectives:

- Verify the correctness and security of core trading execution paths
- Identify potential logical vulnerabilities or asset risks
- Provide security recommendations with risk level classifications
- Generate a formal report suitable for public disclosure

6 Project Information

Official Website: https://next.icpex.org/

Twitter: @ICPExchange

Launch Date: June 1, 2025

Audit Period: April 1, 2025 – July 1, 2025

7 Executive Summary

During a comprehensive 3-month audit period from April 1st to July 1st, 2025, the VelintSec security team conducted an extensive security assessment of the swap-canister-next smart contracts provided by ICPExLabs. Throughout this intensive audit process, the VelintSec team maintained close collaboration with the development team, conducting thorough code reviews and security analysis. The audit resulted in the identification and resolution of a total of 9 security issues, ranging from critical vulnerabilities to minor code improvements.

Summary

Project Name	Swap-canister-next
Repository	https://github.com/ICPExLabs/swap-canister-next
Commit	af129be6e8e31eebb583d3115349581bafefc5a0
Audit Timeline	April 1st, 2025 – July 1st, 2025
Methods	Manual Review

Issues Found

Critical Risk	**0**
High Risk	**2**
Medium Risk	**3**
Low Risk	**4**
Total Issues	**9**

Summary of Findings

Risk	Code	Description	Commit	Status
Low	**001**	**Withdrawal Fee Validation Issue**	**25f25db**	**Fixed & Validated**
Low	**002**	**Custom Fee Implementation**	**ffc4ab8**	**Fixed & Validated**
Medium	**003**	**Trading Pair Pool Lock Enhancement**	**ffc4ab8**	**Fixed & Validated**
Medium	**004**	**Transaction Atomicity Strengthening**	**ffc4ab8**	**Fixed & Validated**
Low	**005**	**Internal Transfer Balance Pre-check**	**154a7d1**	**Fixed & Validated**
High	**006**	**Liquidity Removal Calculation Error**	**59e0717**	**Fixed & Validated**
Medium	**007**	**Fee Account Operation Exemption**	**ff060b2**	**Fixed & Validated**
High	**008**	**Liquidity Removal Account Lock Omission**	**7d526b3**	**Fixed & Validated**
Low	**009**	**Pre-swap Pool Balance Verification**	**a56a62d**	**Fixed & Validated**

8 Findings

8.1 VLS_SCN_001: Withdrawal Fee Validation Issue

Risk Level: Low

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/25f25db

Description:

The withdrawal functionality lacked proper validation to ensure the withdrawal amount exceeded the required transaction fee. This could lead to failed transactions where users attempt to withdraw amounts smaller than the network fee.

```
// canisters/swap/src/business/token/withdraw.rs
// check balance
let balance = with_state(|s| s.business_token_balance_of(self.token, self.from));
let amount = self.withdraw_amount_without_fee.clone() + token.fee.clone();
if balance < amount {
    return Err(BusinessError::insufficient_balance(self.token, balance));
}
```

Impact:

- Failed withdrawal transactions
- Poor user experience
- Potential user fund loss due to gas fees without successful withdrawal

Recommended Mitigation:

Implement comprehensive fee validation checks before processing withdrawal requests, ensuring the withdrawal amount is always greater than the required transaction fee.

Solution:

```
// canisters/swap/src/business/token/withdraw.rs
// check balance
let balance = with_state(|s| s.business_token_balance_of(self.token, self.from));
let amount = self.withdraw_amount_without_fee.clone() + token.fee.clone();

// Added: Validate withdrawal amount against fee
if self.withdraw_amount_without_fee < token.fee {
    return Err(BusinessError::WithdrawAmountTooSmall {
        withdraw_amount: self.withdraw_amount_without_fee,
        fee: token.fee,
    });
}
```

```
if balance < amount {
    return Err(BusinessError::insufficient_balance(self.token, balance));
}
```

8.2 VLS_SCN_002: Custom Fee Implementation

Risk Level: Low

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/ffc4ab8

Description:

The system lacked support for customizable fee structures, limiting flexibility in fee management and preventing dynamic fee adjustments based on market conditions.

```
// canisters/swap/src/business/config/custom.rs
#[ic_cdk::update(guard = "has_business_config_custom_token")]
async fn config_token_custom_put(token: TokenInfo) {
    // Original code only supported fixed fees
    if token.fee != fee {
        ic_cdk::trap("token standard not match");
    }
}
```

Impact:

- Limited fee management flexibility
- Inability to adjust fees based on market conditions
- Reduced competitive advantage

Recommended Mitigation:

Limited fee management flexibility

Inability to adjust fees based on market conditions

Solution:

```
// canisters/swap/src/business/config/custom.rs
#[ic_cdk::update(guard = "has_business_config_custom_token")]
async fn config_token_custom_put(token: TokenInfo) {
    // Added: Support custom fee ranges
    let min_fee = Nat::from(1000); // Minimum fee
    let max_fee = Nat::from(1000000); // Maximum fee

    if token.fee < min_fee || token.fee > max_fee {
        ic_cdk::trap("fee out of allowed range");
    }
    // Additional validation logic...
}
```

8.3 VLS_SCN_003: Trading Pair Pool Lock Enhancement

Risk Level: Medium

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/ffc4ab8

Description:

Insufficient locking mechanisms for trading pair pools could lead to race conditions during high-frequency trading operations, potentially causing inconsistent state updates.

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// Simple pool validation
for pool in &args.path {
    let (pa, _fee_tokens, _required) = check_pool(pool, &self_canister, None)?;
    pas.push(pa);
    fee_tokens.extend(_fee_tokens);
    required.extend(_required);
}
```

Impact:

- Potential race conditions during high-frequency trading
- Inconsistent pool state updates
- Risk of arbitrage exploitation

Recommended Mitigation:

Enhance pool locking mechanisms to ensure proper concurrency control and prevent race conditions during trading operations.

Solution:

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// Enhanced pool locking mechanism
for pool in &args.path {
    let (pa, _fee_tokens, _required) = check_pool(pool, &self_canister, None)?;

    // Added: Pool status validation
    with_state(|s| s.business_pool_status_check(&pa))?;

    // Added: Pool locking
    with_mut_state(|s| s.business_pool_lock(&pa))?;
    pas.push(pa);
    fee_tokens.extend(_fee_tokens);
    required.extend(_required);
}
```

8.4 VLS_SCN_004: Transaction Atomicity Strengthening

Risk Level: Medium

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/ffc4ab8

Description:

Weak atomicity guarantees in transaction execution could lead to partial transaction failures, where some operations complete while others fail, leaving the system in an inconsistent state.

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// Simple transaction execution
with_mut_state(|s| {
    s.business_token_pair_swap_exact_tokens_for_tokens(
        &locks,
        ArgWithMeta {
            now,
            caller,
            arg,
            memo: args.memo,
            created: args.created,
        },
    )
})?
```

Impact:

- Partial transaction failures
- Inconsistent system state
- Potential fund loss or stuck transactions

Recommended Mitigation:

Implement robust atomic transaction protocols that ensure all-or-nothing execution of complex trading operations.

Solution:

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// Enhanced atomic transaction
let transaction_result = with_mut_state(|s| {
    // Added: Transaction begin marker
    s.business_transaction_begin()?;

    let result = s.business_token_pair_swap_exact_tokens_for_tokens(
        &locks,
        ArgWithMeta {
            now,
            caller,
            arg,
            memo: args.memo,
            created: args.created,
        },
    );

    // Added: Transaction commit or rollback
    match result {
        Ok(success) => {
            s.business_transaction_commit()?;
            Ok(success)
        }
        Err(error) => {
            s.business_transaction_rollback()?;
            Err(error)
        }
    }
})?;
```

8.5 VLS_SCN_005: Internal Transfer Balance Pre-check

Risk Level: Low

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/154a7d1

Description:

Internal transfers lacked pre-execution balance verification, potentially allowing transfers that exceed available balances.

```
// canisters/swap/src/business/token/transfer.rs
// check balance
let (balance, fee_to) = with_state(|s|
s.business_token_balance_of_with_fee_to(token.canister_id, self.from));
fee_to = caller.fee_to(fee_to, self.to);
let amount = self.transfer_amount_without_fee.clone() + fee_to.map(|_|
token.fee.clone()).unwrap_or_default();
```

```
if balance < amount {
    return Err(BusinessError::insufficient_balance(token.canister_id, balance));
}
```

Impact:

- Potential for insufficient balance transfers
- Inconsistent account states
- Failed internal operations

Recommended Mitigation:

Add comprehensive balance validation before processing internal transfers to ensure sufficient funds are available.

Solution:

```
// canisters/swap/src/business/token/transfer.rs
// check balance
let (balance, fee_to) = with_state(|s|
s.business_token_balance_of_with_fee_to(token.canister_id, self.from));
fee_to = caller.fee_to(fee_to, self.to);
let amount = self.transfer_amount_without_fee.clone() + fee_to.map(|_|
token.fee.clone()).unwrap_or_default();

// Added: Pre-check balance
if balance < amount {
    return Err(BusinessError::insufficient_balance(token.canister_id, balance));
}

// Added: Verify destination account exists
with_state(|s| s.business_token_balance_of(token.canister_id, self.to))?;
```

8.6 VLS_SCN_006: Liquidity Removal Calculation Error

Risk Level: High

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/59e0717

Description:

Incorrect calculation logic in liquidity removal operations could lead to improper distribution of liquidity tokens and potential fund loss.

```
// canisters/swap/src/stable/v001/types/common/amm/cpmm.rs
let amount0 = liquidity_without_fee.clone() * balance0 / _total_supply.clone();
let amount1 = liquidity_without_fee.clone() * balance1 / _total_supply.clone();
```

```
// ! check amount before change data
let arg = &guard.arg.arg;
if amount0 == *ZERO || amount1 == *ZERO {
    return Err(BusinessError::Liquidity("INSUFFICIENT_LIQUIDITY_BURNED".into()));
}
```

Impact:

- Incorrect liquidity token distribution
- Potential fund loss for liquidity providers
- Inaccurate pool state representation

Recommended Mitigation:

Fix mathematical precision in liquidity calculations to ensure accurate token distribution and proper pool state management.

Solution:

```
// canisters/swap/src/stable/v001/types/common/amm/cpmm.rs
// Fixed mathematical precision issues
let amount0 = (liquidity_without_fee.clone() * balance0) / _total_supply.clone();
let amount1 = (liquidity_without_fee.clone() * balance1) / _total_supply.clone();

// ! check amount before change data
let arg = &guard.arg.arg;
if amount0 == *ZERO || amount1 == *ZERO {
    return Err(BusinessError::Liquidity("INSUFFICIENT_LIQUIDITY_BURNED".into()));
}

// Added: Validate calculation results
if amount0 > balance0 || amount1 > balance1 {
    return Err(BusinessError::Liquidity("INVALID_CALCULATION".into()));
}
```

8.7 VLS_SCN_007: Fee Account Operation Exemption

Risk Level: Medium

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/ff060b2

Description:

Fee account operations were not exempted from fee charges, leading to unnecessary fee deductions from fee collection accounts.

```
// canisters/swap/src/stable/v001/types/common/principal.rs
/// ! check token fee to required or not
```

```
pub fn fee_to(&self, fee_to: Option<Account>, to: Account) -> Option<Account> {
    if fee_to.is_some_and(|fee_to| fee_to.owner == self.0 || fee_to == to) {
        return None;
    }
    fee_to
}
```

Impact:

- Unnecessary fee deductions from fee accounts
- Reduced fee collection efficiency
- Inconsistent fee handling logic

Recommended Mitigation:

Implement fee exemption for fee account operations to prevent circular fee deductions and improve fee collection efficiency.

Solution:

```
// canisters/swap/src/stable/v001/types/common/principal.rs
/// ! check token fee to required or not
pub fn fee_to(&self, fee_to: Option<Account>, to: Account) -> Option<Account> {
    // Fixed: Properly handle fee exemption logic
    if let Some(fee_to) = fee_to {
        if fee_to.owner == self.0 || fee_to == to {
            return None; // Exempt fee
        }
        return Some(fee_to);
    }
    None
}
```

8.8 VLS_SCN_008: Liquidity Removal Account Lock Omission

Risk Level: High

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/7d526b3

Description:

Missing account locking during liquidity removal operations could lead to concurrent modification vulnerabilities and inconsistent state updates.

```
// canisters/swap/src/business/pair/liquidity/remove.rs
let locks =
    match
super::super::super::lock_token_block_chain_and_swap_block_chain_and_token_balances
_and_token_pairs(
        fee_tokens,
        required,
        vec![arg.pa],
        retries.unwrap_or_default(),
    )? {
        LockResult::Locked(locks) => locks,
        LockResult::Retry(retries) => {
            return retry_pair_liquidity_remove(self_canister.id(), args,
retries).await;
        }
    };
```

Impact:

- Concurrent modification vulnerabilities
- Inconsistent state updates
- Potential for fund loss during high-frequency operations

Recommended Mitigation:

Add proper account locking mechanisms during liquidity removal operations to prevent concurrent modifications and ensure data consistency.

Solution:

```
// canisters/swap/src/business/pair/liquidity/remove.rs
// Added: Lock user account
let user_account = TokenAccount::new(arg.pa.dummy_canister_id(), arg.from);
required.push(user_account);

let locks =
    match
super::super::super::lock_token_block_chain_and_swap_block_chain_and_token_balances
_and_token_pairs(
        fee_tokens,
        required,
        vec![arg.pa],
        retries.unwrap_or_default(),
    )? {
        LockResult::Locked(locks) => locks,
        LockResult::Retry(retries) => {
            return retry_pair_liquidity_remove(self_canister.id(), args,
retries).await;
        }
    };
```

8.9 VLS_SCN_009: Pre-swap Pool Balance Verification

Risk Level: Low

Status: Fixed & Validated

Commit: https://github.com/ICPExLabs/swap-canister-next/commit/a56a62d

Description:

Lack of pool balance verification before swap execution could lead to failed swaps due to insufficient liquidity.

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// check balance in
let balance_in =
    balance_in.unwrap_or_else(|| with_state(|s|
s.business_token_balance_of(args.path[0].token.0, args.from)));
if balance_in < args.amount_in {
    return Err(BusinessError::insufficient_balance(args.path[0].token.0,
balance_in));
}
```

Impact:

- Failed swap operations
- Poor user experience
- Inefficient transaction processing

Recommended Mitigation:

Implement comprehensive pre-swap balance checks to ensure sufficient liquidity is available before executing swap operations.

Solution:

```
// canisters/swap/src/business/pair/swap/pay_exact.rs
// check balance in
let balance_in =
    balance_in.unwrap_or_else(|| with_state(|s|
s.business_token_balance_of(args.path[0].token.0, args.from)));
if balance_in < args.amount_in {
    return Err(BusinessError::insufficient_balance(args.path[0].token.0,
balance_in));
}

// Added: Verify pool balances
for pool in &args.path {
    let pool_balance = with_state(|s| s.business_token_balance_of(pool.token.0,
pool.pool_account()));
    if pool_balance < args.amount_in {
        return Err(BusinessError::Swap("INSUFFICIENT_POOL_LIQUIDITY".into()));
    }
}
```

9 Risk Distribution Summary

- **High Risk Issues:** 2 (22.2%)
- **Medium Risk Issues:** 3 (33.3%)
- **Low Risk Issues:** 4 (44.4%)

10 Conclusion

All identified security issues have been successfully resolved and independently verified by the audit team. The development team demonstrated excellent responsiveness and commitment to security throughout the audit process. The implemented fixes significantly improve the overall security posture of the ICPEx smart contracts. The audit team recommends continued monitoring and regular security assessments as the platform scales and new features are added. The current codebase demonstrates strong security practices and is ready for production deployment.