Smart Contract Security Audit Report

Contents

1. Executive Summary 1

2. Audit Methodology 2

3. Project Background 3

3.1 Project Introduction 3

3.2 Project Structure 4

4. Code Overview 8

4.1 Main Contract address 8

4.2 Contracts Description 8

4.3 Code Audit 16

4.3.1 Critical vulnerabilities 16

4.3.2 High-risk vulnerabilities 21

4.3.3 Medium-risk vulnerabilities 24

4.3.4 Low-risk vulnerabilities 25

4.3.5 Enhancement Suggestions 29

5. Audit Result 33

5.1 Conclusion 33

6. Statement 33

1. Executive Summary

On Jan. 22, 2021, the SlowMist security team received the Balanced.Network team's security audit application for balanced, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DeFi project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

SlowMist Smart Contract DeFi project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk vulnerabilities	Medium vulnerability will affect the operation of DeFi project. It is recommended to fix medium-risk vulnerabilities.

Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack
- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack

- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background

3.1 Project Introduction

Balanced is a DAO (Decentralized Autonomous Organization) consisting of a decentralized balance sheet of ICX collateral, with the vision of creating a vibrant ecosystem of tokens pegged to the value of real-world assets. It allows users to mint, borrow, and retire pegged assets, while also providing market makers with clearly defined arbitrage opportunities. The initial iteration of Balanced will support the minting and retiring of USD-pegged assets referred to as ICON Dollars (ICD).

Audit version file information

Initial audit files:

https://github.com/balancednetwork/contracts-private

commit: 1b6b02d81df032a89f6a7a2b7b0bfb92fc4c7fbe

Final audit files:

https://github.com/balancednetwork/contracts-private

commit: 7c796d5f3d0ae22ceff98d6c53d9286ec599e675

3.2 Project Structure

core_contracts

```
├── Rewards.ipynb
├── dex
│   ├── __init__.py
│   ├── dex.py
│   ├── package.json
│   ├── scorelib
│   │   ├── __init__.py
│   │   ├── bag.py
│   │   ├── consts.py
│   │   ├── id_factory.py
│   │   ├── iterable_dict.py
│   │   ├── linked_list.py
│   │   ├── set.py
│   │   └── utils.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── dividends
│   ├── __init__.py
│   ├── dividends.py
│   ├── package.json
│   ├── scorelib
│   │   ├── __init__.py
│   │   ├── binary_tree.py
│   │   ├── consts.py
│   │   ├── id_factory.py
│   │   ├── iterable_dict.py
│   │   └── utils.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── governance
│   ├── __init__.py
│   ├── data_objects.py
│   ├── governance.py
│   ├── package.json
```

```
│   ├── scorelib
│   │   ├── __init__.py
│   │   ├── consts.py
│   │   ├── id_factory.py
│   │   └── utils.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── loans
│   ├── __init__.py
│   ├── loans
│   │   ├── __init__.py
│   │   ├── assets.py
│   │   ├── loans.py
│   │   ├── positions.py
│   │   ├── replay_log.py
│   │   └── snapshots.py
│   ├── package.json
│   ├── scorelib
│   │   ├── __init__.py
│   │   └── id_factory.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── reserve_fund
│   ├── __init__.py
│   ├── package.json
│   ├── reserve_fund.py
│   ├── scorelib
│   │   ├── __init__.py
│   │   ├── consts.py
│   │   ├── id_factory.py
│   │   └── utils.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── rewards
│   ├── RewardData.py
│   ├── __init__.py
```

```
│   ├── package.json
│   ├── rewards.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── staking
│   ├── __init__.py
│   ├── package.json
│   ├── scorelib
│   │   ├── consts.py
│   │   ├── id_factory.py
│   │   └── linked_list.py
│   ├── staking.py
│   └── utils
│       ├── __init__.py
│       └── checks.py
└── tests
    ├── __init__.py
    ├── repeater.py
    └── test_integration_loan.py
token_contracts
├── bal
│   ├── __init__.py
│   ├── balance.py
│   ├── package.json
│   ├── tokens
│   │   ├── IIRC2.py
│   │   ├── IRC2.py
│   │   ├── IRC2burnable.py
│   │   ├── IRC2mintable.py
│   │   └── __init__.py
│   └── utils
│       ├── __init__.py
│       ├── checks.py
│       └── consts.py
├── bwt
│   ├── __init__.py
│   ├── package.json
│   ├── tokens
│   │   ├── IIRC2.py
│   │   ├── IRC2.py
```

```
|   |   └── __init__.py
|   ├── utils
|   |   ├── __init__.py
|   |   ├── checks.py
|   |   └── consts.py
|   └── worker_token.py
├── icd
|   ├── __init__.py
|   ├── icd.py
|   ├── package.json
|   ├── tokens
|   |   ├── IIRC2.py
|   |   ├── IRC2.py
|   |   ├── IRC2burnable.py
|   |   ├── IRC2mintable.py
|   |   └── __init__.py
|   └── utils
|       ├── __init__.py
|       ├── checks.py
|       └── consts.py
└── sicx
    ├── __init__.py
    ├── package.json
    ├── sicx.py
    ├── tokens
    |   ├── IIRC2.py
    |   ├── IRC2.py
    |   ├── IRC2burnable.py
    |   ├── IRC2mintable.py
    |   └── __init__.py
    └── utils
        ├── __init__.py
        ├── checks.py
        └── consts.py
```

4. Code Overview

4.1 Main Contract address

The contract has not yet been deployed on the mainnet.

4.2 Contracts Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

DEX			
Function Name	Decorated	Mutability	Modifiers
Swap	eventlog	-	-
MarketAdded	eventlog	-	-
Add	eventlog	-	-
Remove	eventlog	-	-
Deposit	eventlog	-	-
Withdraw	eventlog	-	-
TransferSingle	eventlog	-	-
ApprovalForAll	eventlog	-	-
URI	eventlog	-	-
Snapshot	eventlog	-	-
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
getAdmin	external	-	-
setAdmin	external	Can modify state	only_governance
getSicx	external	-	-
setSicx	external	Can modify state	only_admin
setDividends	external	Can modify state	only_admin
getDividends	external	-	-

setStaking	external	Can modify state	only_admin
getStaking	external	-	-
setGovernance	external	Can modify state	only_owner
getGovernance	external	-	-
setRewards	external	Can modify state	only_admin
getRewards	external	-	-
setIcd	external	Can modify state	only_admin
getIcd	external	-	-
setMarketName	external	Can modify state	only_governance
turnDexOn	external	Can modify state	only_governance
getDexOn	external	-	-
getDay	external	-	-
setTimeOffset	external	Can modify state	only_governance
getTimeOffset	external	-	-
fallback	Payable	Can modify state	-
cancelSicxicxOrder	external	Can modify state	-
tokenFallback	external	Can modify state	-
precompute	external	-	-
transfer	external	Can modify state	-
_transfer	-	Can modify state	-
getDeposit	external	-	-
getPoolId	external	-	-
getNonce	external	-	-
getNamedPools	external	-	-
lookupPid	external	-	-
getPoolTotal	external	-	-
totalSupply	external	-	-
balanceOf	external	-	-
getFees	external	-	-
getPoolBase	external	-	-
getPoolQuote	external	-	-
getPrice	external	-	-
getPriceByName	external	-	-
getInversePrice	external	-	-
getICXBalance	external	-	-
getICXWithdrawLock	external	-	-
setApprovalForAll	external	Can modify state	-

isApprovedForAll	external	Can modify state	-
transferFrom	external	Can modify state	-
_transfer_from	-	Can modify state	-
hasUsedDex	external	-	-
totalDexAddresses	external	-	-
_get_exchange_rate	-	Can modify state	-
exchange	-	Can modify state	-
_get_sicx_rate	-	Can modify state	-
_swap_icx	-	Can modify state	-
_take_new_day_snapshot	-	Can modify state	-
_check_distributions	-	Can modify state	-
_update_account_snapshot	-	Can modify state	-
_update_total_supply_snapshot	-	Can modify state	-
balanceOfAt	external	-	-
totalSupplyAt	external	-	-
getTotalValue	external	-	-
loadBalancesAtSnapshot	external	-	-
getDataBatch	external	-	-
permit	external	Can modify state	only_governance
withdraw	external	Can modify state	-
remove	external	Can modify state	-
add	external	Can modify state	-

Dividends			
Function Name	Decorated	Mutability	Modifiers
Transfer	eventlog	-	-
FundTransfer	eventlog	-	-
TokenTransfer	eventlog	-	-
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
setGovernance	external	Can modify state	-
getGovernance	external	-	-
setAdmin	external	Can modify state	only_governance
getAdmin	external	-	-

setLoans	external	Can modify state	only_admin
getLoans	external	-	-
getBalances	external	-	-
distribute	external	Can modify state	-
claim	external	Can modify state	-
tokenFallback	external	Can modify state	-
_send_ICX	-	Can modify state	-
fallback	Payable	Can modify state	-

Governance			
Function Name	Decorated	Mutability	Modifiers
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
launchBalanced	external	Can modify state	only_owner
setAddresses	external	Can modify state	only_owner
getAddresses	external	-	-
setContractAddresses	external	Can modify state	only_owner
toggleBalancedOn	external	Can modify state	only_owner
_set_launch_day	external	Can modify state	-
getLaunchDay	external	-	-
_set_launch_time	external	Can modify state	-
getLaunchTime	external	-	-
getDay	external	-	-
addAsset	external	Can modify state	only_owner
toggleAssetActive	external	Can modify state	only_owner
addRewardsDataSource	external	Can modify state	only_owner
updateBalTokenDistPercentage	external	Can modify state	only_owner
dexPermit	external	Can modify state	only_owner
setMarketName	external	Can modify state	only_owner
tokenFallback	external	Can modify state	-
fallback	Payable	-	-

Loans			
Function Name	Decorated	Mutability	Modifiers

__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
snapIndexes	external	-	-
turnLoansOn	external	Can modify state	only_governance
toggleLoansOn	external	Can modify state	only_governance
getLoansOn	external	-	-
getDay	external	-	-
getNonzeroPositionCount	external	-	-
getPositionStanding	external	-	-
getAssetTokens	external	-	-
getCollateralTokens	external	-	-
getTotalCollateral	external	-	-
getAccountPositions	external	-	-
getPositionByIndex	external	-	-
getAvailableAssets	external	-	-
assetCount	external	-	-
borrowerCount	external	-	-
hasDebt	external	-	-
getEvent	external	-	-
getSnapshot	external	-	-
addAsset	external	Can modify state	only_admin
toggleAssetActive	external	Can modify state	only_admin
precompute	external	Can modify state	-
getTotalValue	external	-	-
getDataCount	external	-	-
getDataBatch	external	-	-
checkForNewDay	external	Can modify state	-
checkDistributions	external	Can modify state	-
addCollateral	payable	Can modify state	-
tokenFallback	external	Can modify state	-
_deposit_and_borrow	-	Can modify state	-
_repay_loan	-	Can modify state	-
_retire_asset	-	Can modify state	-
bd_redeem	-	Can modify state	-
originateLoan	external	Can modify state	-

withdrawCollateral	external	Can modify state	-
updateStanding	external	Can modify state	-
liquidate	external	Can modify state	-
replayEvents	external	Can modify state	-
_send_token	-	Can modify state	-
fallback	Payable	-	-
setGovernance	external	Can modify state	only_owner
setAdmin	external	Can modify state	only_governance
setDividends	external	Can modify state	only_admin
setReserve	external	Can modify state	only_admin
setRewards	external	Can modify state	only_admin
setStaking	external	Can modify state	only_admin
setReplayBatchSize	external	Can modify state	only_admin
setMiningRatio	external	Can modify state	only_admin
setLockingRatio	external	Can modify state	only_admin
setLiquidationRatio	external	Can modify state	only_admin
setOriginationFee	external	Can modify state	only_admin
setRedemptionFee	external	Can modify state	only_admin
setRedeemMinimum	external	Can modify state	only_admin
setTimeOffset	external	Can modify state	only_governance
getParameters	external	-	-
ContractActive	eventlog	-	-
AssetActive	eventlog	-	-
Transfer	eventlog	-	-
FundTransfer	eventlog	-	-
TokenTransfer	eventlog	-	-
AssetAdded	eventlog	-	-
CollateralReceived	eventlog	-	-
OriginateLoan	eventlog	-	-
LoanRepaid	eventlog	-	-
AssetRedeemed	eventlog	-	-
Liquidate	eventlog	-	-
BadDebt	eventlog	-	-
TotalDebt	eventlog	-	-
FeePaid	eventlog	-	-
OraclePriceUpdate	eventlog	-	-
PositionStanding	eventlog	-	-

Diagnostic	eventlog	-	-
Snapshot	eventlog	-	-

ReserveFund			
Function Name	Decorated	Mutability	Modifiers
Transfer	eventlog	-	-
FundTransfer	eventlog	-	-
TokenTransfer	eventlog	-	-
RedeemFail	eventlog	-	-
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
setGovernance	external	Can modify state	only_owner
getGovernance	external	-	-
setAdmin	external	Can modify state	only_governance
getAdmin	external	-	-
setLoans	external	Can modify state	only_admin
getLoans	external	-	-
setBaln	external	Can modify state	only_admin
getBaln	external	-	-
setSicx	external	Can modify state	only_admin
getSicx	external	-	-
getBalances	external	-	-
redeem	external	Can modify state	-
tokenFallback	external	Can modify state	-
_send_token	-	Can modify state	-
fallback	Payable	-	-

Rewards			
Function Name	Decorated	Mutability	Modifiers
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-
name	external	-	-
getBalnHoldings	external	-	-

distStatus	external	-	-
updateBalTokenDistPercentage	external	Can modify state	only_governance
getDataSourceNames	external	-	-
getRecipients	external	-	-
getRecipientsSplit	external	-	-
addNewDataSource	external	Can modify state	only_governance
getDataSources	external	-	-
_reward_distribution	-	Can modify state	-
distribute	external	Can modify state	-
claimRewards	external	Can modify state	-
_get_day	-	Can modify state	-
_bal_token_dist_per_day	-	Can modify state	-
tokenFallback	external	Can modify state	-
setGovernance	external	Can modify state	only_owner
getGovernance	external	-	-
setAdmin	external	Can modify state	only_governance
getAdmin	external	-	-
setBaln	external	Can modify state	only_admin
getBaln	external	-	-
setBwt	external	Can modify state	only_admin
getBwt	external	-	-
setReserve	external	Can modify state	only_admin
getReserve	external	-	-
setBatchSize	external	Can modify state	only_admin
getBatchSize	external	-	-
setTimeOffset	external	Can modify state	only_governance
getTimeOffset	external	-	-

Staking			
Function Name	Decorated	Mutability	Modifiers
Transfer	eventlog	-	-
FundTransfer	eventlog	-	-
TokenTransfer	eventlog	-	-
__init__	-	Can modify state	-
on_install	-	Can modify state	-
on_update	-	Can modify state	-

name	external	-	-
getTodayRate	external	-	-
getRate	-	Can modify state	-
getSicxAddress	external	-	-
getTotalStake	external	-	-
getLifetimeReward	external	-	-
getTopPreps	external	-	-
getUnstakeInfo	external	-	-
getUserUnstakeInfo	external	-	-
getUnstakeAmount	external	-	-
setSicxAddress	external	Can modify state	-
_set_top_preps	-	Can modify state	-
_reset_top_preps	-	Can modify state	-
_check_for_iscore	-	Can modify state	-
_check_unstake_result	-	Can modify state	-
_perform_checks	-	Can modify state	-
_evenly_distrubuted_amount	-	Can modify state	-
addCollateral	Payable	Can modify state	-
_claim_iscore	-	Can modify state	-
_stake	-	Can modify state	-
_delegations	-	Can modify state	-
tokenFallback	external	Can modify state	-
_unstake	-	Can modify state	-
_send_ICX	-	Can modify state	-
fallback	Payable	-	-

4.3 Code Audit

4.3.1 Critical vulnerabilities

4.3.1.1 Transfer record error

Duplicate recording of the number of AVAILABLE for the _from address at the time of transfer will lead to the problem that the transfer is successful but the number of AVAILABLE for the _from

address remains the same.

Fix suggestion: When transferring money, the correct logic should be: the number of AVAILABLE in the _from address decreases, and the number of AVAILABLE in the _to address increases.

Code location: token_contracts/bal/balance.py

```
    @external
    def transfer(self, _to: Address, _value: int, _data: bytes = None):

        _from = self.msg.sender
        self._check_first_time(_from)
        self._check_first_time(_to)
        self._make_available(_from)
        self._make_available(_to)

        if self._staked_balances[_from][Status.AVAILABLE] < _value:
            revert(f"{TAG}: Out of available balance. Please check staked and total balance")

        self._staked_balances[_from][Status.AVAILABLE] = self._staked_balances[_from][Status.AVAILABLE] -
_value
        self._staked_balances[_from][Status.AVAILABLE] = self._staked_balances[_from][Status.AVAILABLE] +
_value

        IRC2Mintable.transfer(self, _to, _value, _data)
```

Fix status: Fixed.

4.3.1.2 Add liquidity ratio calculation error

When users add liquidity, if _pid has been created, _quoteValue will be calculated according to a certain ratio through _baseValue and checked.

The ratio is calculated by the getPrice function in the contract, specifically: the total number of baseTokens in the pool divided by the total number of quoteTokens. Finally, it is multiplied with the _baseValue passed in when the user adds liquidity to obtain _quoteValue.

But the ratio calculated in this way does not match the actual.

The correct calculation method should be:

*_baseValue * (self._pool_total[_pid][self._pool_quote[_pid]] /*

self._pool_total[_pid][self._pool_base[_pid]])

In this way, the quoteValue required to add liquidity can be correctly calculated.

Fix suggestion: Can refer to the practice of uniswapV2

https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/libraries/UniswapV2Libr

ary.sol#L39

Code location: core_contracts/dex/dex.py

```
@external
def add(self, _baseToken: Address, _quoteToken: Address, _baseValue: int, _quoteValue: int):
    """
    Adds liquidity to a pool for trading, or creates a new pool. Rules:
    - The quote coin of the pool must be one of the allowed quote currencies.
    - Tokens must be deposited in the pool's ratio.
    - If ratio is incorrect, it is advisable to call `swap` first.
    """
    _owner = self.msg.sender
    _pid = self._pool_id[_baseToken][_quoteToken]
    if _baseToken == _quoteToken:
        revert("Pool must contain two token contracts")
    if not _baseValue:
        revert("Please send initial value for first currency")
    if not _quoteValue:
        revert("Please send initial value for second currency")
    if _pid == 0:
        if not (_quoteToken == self._icd.get()) and not (_quoteToken == self._sicx.get()):
            revert("Second currency must be ICD or sICX")
        self._pool_id[_baseToken][_quoteToken] = self._nonce.get()
        self._pool_id[_quoteToken][_baseToken] = self._nonce.get()
        _pid = self._nonce.get()
        liquidity = DEFAULT_INITAL_LP
        self._nonce.set(self._nonce.get() + 1)
```

```
                self.active[_pid] = True
                self._pool_base[_pid] = _baseToken
                self._pool_quote[_pid] = _quoteToken
                self.MarketAdded(_pid, _baseToken, _quoteToken,
                                 _baseValue, _quoteValue)
            else:
                token1price = self.getPrice(_pid)
                if not int(_baseValue * (token1price / 10**10)) == _quoteValue:
                    revert('disproportionate amount')
                liquidity = int(self._total[_pid] * (self._pool_total[_pid]
                                                     [_baseToken] + _baseValue) /
self._pool_total[_pid][_baseToken])
            self._pool_total[_pid][_baseToken] += _baseValue
            self._pool_total[_pid][_quoteToken] += _quoteValue
            self._deposit[_baseToken][self.msg.sender] -= _baseValue
            self._deposit[_quoteToken][self.msg.sender] -= _quoteValue
            self._balance[_pid][_owner] += liquidity
            self._total[_pid] += liquidity
            self.Add(_pid, _owner, liquidity)
            self._withdraw_lock[self.msg.sender][_pid] = self.now()
            if self.msg.sender not in self._funded_addresses:
                self._funded_addresses.add(self.msg.sender)
            self._update_account_snapshot(_owner, _pid)
            self._update_total_supply_snapshot(_pid)
```

Fix status: Fixed.

4.3.1.3 Liquidity calculation error when adding liquidity

When calculating the liquidity obtained by the user, the total number of baseTokens in the current pool is added to the _baseValue added by the user, then divided by the total number of baseTokens in the current pool, and finally multiplied by _total[_pid] in the current pool to get Liquidity acquired by the user.

Since the original number of _baseToken in the pool is added to the calculation, the liquidity obtained by the end user is more than the current total liquidity.

Fix suggestion: The original baseToken quantity should not be added when calculating:

*liquidity = int(self._total[_pid] * (_baseValue) / self._pool_total[_pid][_baseToken])*

You can refer to the practice of uniswapV2:

https://github.com/Uniswap/uniswap-v2-core/blob/master/contracts/UniswapV2Pair.sol#L123

Code location: core_contracts/dex/dex.py

```
    @external
    def add(self, _baseToken: Address, _quoteToken: Address, _baseValue: int, _quoteValue: int):
        """
        Adds liquidity to a pool for trading, or creates a new pool. Rules:
        - The quote coin of the pool must be one of the allowed quote currencies.
        - Tokens must be deposited in the pool's ratio.
        - If ratio is incorrect, it is advisable to call `swap` first.
        """
        _owner = self.msg.sender
        _pid = self._pool_id[_baseToken][_quoteToken]
        if _baseToken == _quoteToken:
            revert("Pool must contain two token contracts")
        if not _baseValue:
            revert("Please send initial value for first currency")
        if not _quoteValue:
            revert("Please send initial value for second currency")
        if _pid == 0:
            if not (_quoteToken == self._icd.get()) and not (_quoteToken == self._sicx.get()):
                revert("Second currency must be ICD or sICX")
            self._pool_id[_baseToken][_quoteToken] = self._nonce.get()
            self._pool_id[_quoteToken][_baseToken] = self._nonce.get()
            _pid = self._nonce.get()
            liquidity = DEFAULT_INITAL_LP
            self._nonce.set(self._nonce.get() + 1)
            self.active[_pid] = True
            self._pool_base[_pid] = _baseToken
            self._pool_quote[_pid] = _quoteToken
            self.MarketAdded(_pid, _baseToken, _quoteToken,
                             _baseValue, _quoteValue)
        else:
            token1price = self.getPrice(_pid)
            if not int(_baseValue * (token1price / 10**10)) == _quoteValue:
```

```
                    revert('disproportionate amount')
                liquidity = int(self._total[_pid] * (self._pool_total[_pid]
                                                     [_baseToken] + _baseValue) /
self._pool_total[_pid][_baseToken])
            self._pool_total[_pid][_baseToken] += _baseValue
            self._pool_total[_pid][_quoteToken] += _quoteValue
            self._deposit[_baseToken][self.msg.sender] -= _baseValue
            self._deposit[_quoteToken][self.msg.sender] -= _quoteValue
            self._balance[_pid][_owner] += liquidity
            self._total[_pid] += liquidity
            self.Add(_pid, _owner, liquidity)
            self._withdraw_lock[self.msg.sender][_pid] = self.now()
            if self.msg.sender not in self._funded_addresses:
                self._funded_addresses.add(self.msg.sender)
            self._update_account_snapshot(_owner, _pid)
            self._update_total_supply_snapshot(_pid)
```

Fix status: Fixed.

4.3.2 High-risk vulnerabilities

4.3.2.1 Risk of overflow

When the user adds liquidity, the amount of added tokens needs to be deducted from the user's _deposit, but it is not checked whether the deducted amount is less than or equal to the amount in _deposit.

Fix suggestion: It is recommended to check whether _baseValue and _quoteValue are less than or equal to the amount in _deposit when adding liquidity.

Code location: core_contracts/dex/dex.py

```
    @external
    def add(self, _baseToken: Address, _quoteToken: Address, _baseValue: int, _quoteValue: int):
        """
        Adds liquidity to a pool for trading, or creates a new pool. Rules:
```

```
        - The quote coin of the pool must be one of the allowed quote currencies.
        - Tokens must be deposited in the pool's ratio.
        - If ratio is incorrect, it is advisable to call `swap` first.
        """
        _owner = self.msg.sender
        _pid = self._pool_id[_baseToken][_quoteToken]
        if _baseToken == _quoteToken:
            revert("Pool must contain two token contracts")
        if not _baseValue:
            revert("Please send initial value for first currency")
        if not _quoteValue:
            revert("Please send initial value for second currency")
        if _pid == 0:
            if not (_quoteToken == self._icd.get()) and not (_quoteToken == self._sicx.get()):
                revert("Second currency must be ICD or sICX")
            self._pool_id[_baseToken][_quoteToken] = self._nonce.get()
            self._pool_id[_quoteToken][_baseToken] = self._nonce.get()
            _pid = self._nonce.get()
            liquidity = DEFAULT_INITAL_LP
            self._nonce.set(self._nonce.get() + 1)
            self.active[_pid] = True
            self._pool_base[_pid] = _baseToken
            self._pool_quote[_pid] = _quoteToken
            self.MarketAdded(_pid, _baseToken, _quoteToken,
                             _baseValue, _quoteValue)
        else:
            token1price = self.getPrice(_pid)
            if not int(_baseValue * (token1price / 10**10)) == _quoteValue:
                revert('disproportionate amount')
            liquidity = int(self._total[_pid] * (self._pool_total[_pid]
                                                 [_baseToken] + _baseValue) /
self._pool_total[_pid][_baseToken])
        self._pool_total[_pid][_baseToken] += _baseValue
        self._pool_total[_pid][_quoteToken] += _quoteValue
        self._deposit[_baseToken][self.msg.sender] -= _baseValue
        self._deposit[_quoteToken][self.msg.sender] -= _quoteValue
        self._balance[_pid][_owner] += liquidity
        self._total[_pid] += liquidity
        self.Add(_pid, _owner, liquidity)
        self._withdraw_lock[self.msg.sender][_pid] = self.now()
        if self.msg.sender not in self._funded_addresses:
            self._funded_addresses.add(self.msg.sender)
```

```
        self._update_account_snapshot(_owner, _pid)
        self._update_total_supply_snapshot(_pid)
```

Fix status: Fixed

4.3.2.2 Error sending fee

During the token exchange, the amount of toToken the user needs to redeem was incorrectly sent to the dividends score.

Fix suggestion: Baln_fees should be transferred

Code location: core_contracts/dex/dex.py

```
        # Pay each of the user and the dividends score their share of the tokens
        to_token_score = self.create_interface_score(_toToken, TokenInterface)
        to_token_score.transfer(_receiver, send_amt)
        from_token_score = self.create_interface_score(_fromToken, TokenInterface)
        from_token_score.transfer(self._dividends.get(), send_amt)
        self.Swap(_fromToken, _toToken, _sender, _receiver, _value, send_amt)
```

Fix status: Fixed

4.3.2.3 Risk of lack of access control

There are updateBalTokenDistPercentage function and addNewDataSource function in the Rewards contract. These two functions can adjust the distribution ratio of reward tokens and add data sources. But there is no permission control on these two functions, so any user can call them.

Fix suggestion: It is recommended to control the calling permissions of this function

Code location: core_contracts/rewards/rewards.py

```
    @external
    def updateBalTokenDistPercentage(self, _recipient_list : List[DistPercentDict]) -> None:
        ......
```

```
@external
def addNewDataSource(self, _data_source_name: str, _contract_address: Address) -> None:
    ......
```

Fix status: Fixed

4.3.3 Medium-risk vulnerabilities

4.3.3.1 Distribute the call failure issue

In the _distribute function of the RewardData contract, it will call the precompute function of the loans contract and check its return value, but the precompute function will directly perform the revert, which will cause the entire transaction to fail. The same is true for getTotalValue function.

Fix suggestion: It is recommended to remove the revert logic.

Code location: core_contracts/loans/loans/loans.py

```
@external
def precompute(self, _snapshot_id: int, batch_size: int) -> bool:
    """
    prepares the position data snapshot to send to the rewards SCORE.
    """
    revert(f'_snapshot_id: {_snapshot_id}')
    if self.msg.sender != self._rewards.get():
        revert(f'The precompute method may only be invoked by the rewards SCORE.')
    self.checkForNewDay() # Only does something if it is internal on a DEX tx.
    # Iterate through all positions in the snapshot to bring them up to date.
    if self._positions._calculate_snapshot(_snapshot_id, batch_size):
        return Complete.DONE
    return Complete.NOT_DONE
```

Fix status: Fixed.

4.3.4 Low-risk vulnerabilities

4.3.4.1 Risk of excessive authority

The admin role exists in the token contract. The admin role can mint unlimited tokens via the mintTo function and burn tokens of any user without being approved via the burnFrom function. Since the owner can set any address as the admin role through the setAdmin function, and the owner authority cannot be transferred, this will lead to the risk of excessive authority.

Fix suggestion: In business design, the admin authority will be transferred to other contracts, so the setAdmin function can be designed to be called only once to alleviate the risk of excessive authority.

Code location: token_contracts/**/tokens/IRC2Mintable.sol & IRC2Burnable.sol

```
@external
def mint(self, _amount: int, _data: bytes = None) -> None:
        """
        Creates `_amount` number of tokens, and assigns to caller account.
        Increases the balance of that account and total supply.
        See {IRC2-_mint}

        :param _amount: Number of tokens to be created at the account.
        """
        # revert(f'Yes, got to here! Minting {_amount} {self.symbol()} to {self.msg.sender}. '
        #                 f'Forwarding data = {_data.decode("utf-8")}')
        if _data is None:
                _data = b'None'
        self._mint(self.msg.sender, _amount, _data)

@external
def mintTo(self, _account: Address, _amount: int, _data: bytes = None) -> None:
        """
        Creates `_amount` number of tokens, and assigns to `_account`.
        Increases the balance of that account and total supply.
        See {IRC2-_mint}
```

```
                :param _account: The account at which token is to be created.
                :param _amount: Number of tokens to be created at the account.
                """
                # revert(f'This should print...')
                if _data is None:
                        _data = b'None'
                self._mint(self.tx.origin, _amount, _data)
                self._transfer(self.tx.origin, _account, _amount, _data)

    @external
    def burnFrom(self, _account: Address, _amount: int) -> None:
                """
                Destroys `_amount` number of tokens from the specified `_account` account.
                Decreases the balance of that account and total supply.
                See {IRC2-_burn}

                :param _account: The account at which token is to be destroyed.
                :param _amount: Number of tokens to be destroyed at the `_account`.
                """
                self._burn(_account, _amount)
```

Fix status: After communicating with the project party, the project party's feedback: The installation script sets the admin for the sICX token to the staking contract, the admin for the BALN token to the rewards contract, the admin for the ICD token to the Loans contract. and the admin for the BWT token to the Governance contract. For additional security, it would be possible to make the admin address an installation parameter for these token contracts so the admin can't be changed after deployment; it would be fixed to the respective admin contract. And the owner has considerable control over ICON contracts that can't be relinquished, but this would at least make it impossible to change the admin without a contract update.

4.3.4.2 Event logging error

update_asset_value uses the try-except method to update the price. The normal logic should be to

trigger the OraclePrice event record after the price is successfully updated, instead of the OraclePrice event record after the price update fails.

Fix suggestion: It is suggested to record the event only after the price is successfully updated.

Code location: token_contracts/icd/icd.py

```
    def update_asset_value(self) -> None:
        """
        Calls the oracle method for the asset and updates the asset
        value in loop.
        """
        base = self._peg.get()
        quote = "ICX"
        oracle_address = self._oracle_address.get()
        try:
            oracle = self.create_interface_score(oracle_address, OracleInterface)
            priceData = oracle.get_reference_data(base, quote)
            self._last_price.set(priceData['rate'])
            self._price_update_time.set(self.now())
        except BaseException as e:
            self.OraclePriceUpdateFailed(base + quote, self._oracle_name.get(), oracle_address, f'Exception: {e}')
        self.OraclePrice(base + quote, self._oracle_name.get(), oracle_address, priceData['rate'])
```

Fix status: Fixed.

4.3.4.3 Date calculation issue

In the rewards function, the number of days from the beginning to the present can be obtained through the _get_day function. It is calculated by dividing the difference between the present and the microsecond timestamp at the beginning by DAY_IN_MICROSECONDS

But DAY_IN_MICROSECONDS is the number of microseconds in a day divided by 24.

Fix suggestion: Need to calculate the number of days correctly, DAY_IN_MICROSECONDS should not be divided by 24

The same goes for the U_SECONDS_DAY parameter in the loans contract

Code location: core_contracts/rewards/utils/constss.py

```
DAY_IN_MICROSECONDS = 86400 * 10**6 // 24


    def _get_day(self) -> int:
        today = (self.now() - self._start_timestamp.get()) // DAY_IN_MICROSECONDS
        return today
```

Fix status: Fixed.

4.3.4.4 The check is not rigorous when originateLoan

The originateLoan function exists in the loans contract for the user to carry out the loan operation, but when the user calls the originateLoan function to carry out the loan operation, this function does not check the active and is_collateral of the asset.

Fix suggestion: It is recommended to check the active and is_collateral of the asset when the user calls the originateLoan function for lending operations.

Code location: core_contarcts/loans/loans/loans.py

```
    @external
    def originateLoan(self, _asset: str, _amount: int, _from: Address = None) -> None:
        """
        Originate a loan of an asset if there is sufficient collateral.

        :param _asset: Symbol of the asset.
        :type _asset: str
        :param _value: Number of tokens sent.
        :type _value: int
        """
        if _from == None:
            _from = self.msg.sender
        if not self._positions._exists(_from):
            revert(f'This address does not have a position on Balanced. '
                   f'Collateral must be deposited before originating a loan.')
```

```
        if self._assets[_asset].dead():
            revert(f'No new loans of {_asset} can be originated since '
                   f'it is in a dead market state.')
        pos = self._positions.get_pos(_from)
        if pos.get_standing() == Standing.INDETERMINATE:
            revert(f'Position must be up to date with replay of all retirement '
                   f'events before executing this transaction.')

......
```

Fix status: Fixed.

4.3.5 Enhancement Suggestions

4.3.5.1 Partial code redundancy

The setSicxSupply function in the staking contract is a test function. Keeping this function in the contract will affect the security of the contract.

Fix suggestion: It is suggested to remove this test function

Code location: staking.py

```
    @external
    def setSicxSupply(self) -> None:
        """
        Only necessary for the dummy contract.
        """
        self._sICX_supply.set(self.sICX_score.totalSupply())
```

Fix status: Fixed.

4.3.5.2 Function visibility issue

In the sicx contract, the priceInLoop function and the lastPriceInLoop function both return the price of sICX in loop. The lastPriceInLoop function obtains the latest price by calling the priceInLoop

function, but the visibility of the priceInLoop function is external.

Fix suggestion: It is suggested to set the priceInLoop function to only allow internal calls.

Code location: token_contracts/sicx/sicx.py

```
    @external
    def priceInLoop(self) -> int:
        """
        Returns the price of sICX in loop.
        """
        staking_score = self.create_interface_score(self._staking_address.get(), stakingInterface)
        return staking_score.getTodayRate()

    @external(readonly=True)
    def lastPriceInLoop(self) -> int:
        """
        Returns the price of sICX in loop.
        """
        return self.priceInLoop()
```

Fix status: After communicating with the project party, the project party's feedback: Offering both methods allows the user to choose whether they want to get the latest price data, and pay for it, or get a readonly version of the price that may be outdated by a few minutes.

4.3.5.3 Missing event

The toggleAssetActive and toggleLoansOn function are missing event.

Fix suggestion: there should be an event.

Code location: core_contracts/loans/loans/loans.py

```
    @external
    @only_governance
    def toggleLoansOn(self) -> None:
        self._loans_on.set(not self._loans_on.get())
```

```
    @external
    @only_owner
    def toggleAssetActive(self, _symbol) -> None:
        self._assets[_symbol].active.set(not self._assets[_symbol].active.get())
```

Fix status: Fixed.

4.3.5.4 Risk of False top-up

Since the tokenFallback function can be called directly by the user, if there is a function called by any user in a token contract in the pool, the user can construct a transaction through this token contract to maliciously increase their balance.

Fix suggestion: The balance of this contract should be checked before and after the user transfers the token.

Code location: core_contracts/dex/dex.py

```
@external
    def tokenFallback(self, _from: Address, _value: int, _data: bytes):
        # if user sends some irc token to score
        Logger.info("called token fallback", self._TAG)

        # Update snapshots if necessary and notify the rewards score
        self._take_new_day_snapshot()
        self._check_distributions()

        unpacked_data = json_loads(_data.decode('utf-8'))
        _fromToken = self.msg.sender
        if unpacked_data["method"] == "_deposit":
            self._deposit[_fromToken][_from] += _value
            self.Deposit(_fromToken, _from, _value)
        elif unpacked_data["method"] == "_swap_icx":
            if _fromToken == self._sicx.get():
                self._swap_icx(_from, _value)
        elif unpacked_data["method"] == "_swap":
            max_slippage = 250
```

```
            if "maxSlippage" in unpacked_data["params"]:
                max_slippage = int(unpacked_data["params"]["maxSlippage"])
                if max_slippage > MAX_SLIPPAGE:
                    revert("Slippage cannot exceed 10% (1000 basis points)")
                if max_slippage <= 0:
                    revert("Max slippage must be a positive number")
            self.exchange(_fromToken, Address.from_string(
                unpacked_data["params"]["toToken"]), _from, _from, _value, max_slippage)
        else:
            revert("Fallback directly not allowed")
```

Fix status: Not fixed

4.3.5.5 Test code not removed

Part of the test code in the loans contract has not been removed, which will affect the normal business logic.

Fix suggestion: It is recommended to remove irrelevant code.

Code location: core_contracts/loans/loans/loans.py

```
    def on_update(self) -> None:
        super().on_update()
        self._redeem_minimum.set(REDEEM_MINIMUM)
        self._mining_ratio.set(DEFAULT_MINING_RATIO)
        self._locking_ratio.set(DEFAULT_LOCKING_RATIO)
        self._liquidation_ratio.set(DEFAULT_LIQUIDATION_RATIO)
        # # Create bad position for testing liquidation. Take out a loan that is too large.
        pos = self._positions.get_pos(TEST_ADDRESS)
        # Independently, 782769 * 10**15 =~$299 worth of collateral will be
        # deposited for this position.
        icd: int = 2 * 10**20 # $200 ICD debt
        self._assets['ICD'].mint(TEST_ADDRESS, icd)
        pos['ICD'] += icd
        pos.update_standing()
```

Fix status: Fixed

5. Audit Result

5.1 Conclusion

Audit Result : Low Risks

Audit Number : 0X002103020001

Audit Date : Mar. 02, 2021

Audit Team : SlowMist Security Team

Summary conclusion: The SlowMist security team use a manual and SlowMist Team analysis tool audit of the codes for security issues. There are sixteen security issues found during the audit. There are three critical-risk vulnerabilities, three high-risk vulnerabilities, one medium-risk vulnerabilities and four low-risk vulnerabilities. We also provide five enhancement suggestions. Due to the characteristics of ICON, the owner authority of the contract cannot be transferred to community governance, but the project party has adopted some measures to restrict the owner's authority. Therefore, the final audit conclusion is low risk.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis

and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist