HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Nucleon
Date: January 24th, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Nucleon
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	Voting; Yield Farming
Platform	CFX
Network	Conflux
Language	Solidity
Methodology	Link
Website	https://confluxnetwork.org/
Changelog	13.10.2022 - Initial Review 09.12.2022 - Second Review 19.01.2023 - Third Review 24.01.2023 - Fourth Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Nucleon (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/article-i/espace/tree/in-test
https://github.com/article-i/core-pos-pool/tree/test

Commit:
7a9aa71c9a7fe10d213b60ae9c75913dbba9d87b (espace)
8ce294e520127a973b2fbe6e626a545a8f904d31 (core-pos-pool)

Documentation:
Whitepaper (partial functional requirements provided)

Functional requirements

Integration and Unit Tests: No
Deployed Contracts Addresses: -
Contracts:

File: ./contracts/IExchange.sol
SHA3: 61eb39b071990344a69bc373650743c093b0ae1ac4aa34243d4fefac741cdd90

File: ./contracts/eSpace/CoreBridge_multipool.sol
SHA3: 6cdb4307022e9d0e13601b72f84ce35d006e99f8904e09c91d78f47e7f0e59f5

File: ./contracts/PoolContext.sol
SHA3: 94c801354857878c017680d3b15073e0881921ac0f28283b3cb199e81c94c625

File: ./contracts/PoSPoolmini.sol
SHA3: 8a03a2cb3f4f2c7a8ac9108ed86d9a056ee813eb38927d8efef58db4097445eb

File: ./contracts/VotePowerQueue.sol
SHA3: 681daac351c4b37e070998d30bfc9919a2a0c01e909ed4f89929942064504fa9

File: ./contracts/eSpace/Exchangeroom.sol
SHA3: b0f2248a7acb4eb698adc778e1720d3a1cd9c17bb027dba3d9007b2b6b9086be

File: ./contracts/eSpace/UnstakeQueueCFX.sol
SHA3: eced83c9430fa22112ce47a57e5335963be0839e807aa48cc4b8d5019a23295e

Second review scope

Repository:
https://github.com/article-i/espace/tree/in-test
https://github.com/article-i/core-pos-pool/tree/test

Commit:
c70be17a0bfebbe4a4456892b1f14a55b81356c2 (espace)
04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6 (core-pos-pool)

Documentation:
Whitepaper (partial functional requirements provided)

Functional requirements

Integration and Unit Tests: No
Deployed Contracts Addresses: -
Contracts:
File: ./contracts/IExchange.sol
SHA3: 61eb39b071990344a69bc373650743c093b0ae1ac4aa34243d4fefac741cdd90

File: ./contracts/eSpace/CoreBridge_multipool.sol
SHA3: 643096ae32d343e8a2975ef6d7ce038ae4b6f604a817f7516da7598bce110899

File: ./contracts/PoolContext.sol
SHA3: 59121afe857833d89710964de2bfeb7c95faa3b7d4af0cb262e58a1744bfe451

File: ./contracts/PoSPoolmini.sol
SHA3: 1d59d96c351f84271d65e275ea7bf5873dc353aa91850f3b338aa8b3afbc64ff

File: ./contracts/VotePowerQueue.sol
SHA3: 32a11531531207f71349703260d35edfd20931ba531a0b8492dc818519d0fb89

File: ./contracts/eSpace/Exchangeroom.sol
SHA3: c54856d0b25ce97b5ea3f4734cec9c61571959880c5567f3e84cb67482fd2129

File: ./contracts/XCFX.sol
SHA3: 20aac8420a1000e1cf3e0cafaf71887e71e9ed7c053c15bbfd0c56c5e259e146

Third review scope

Repository:
https://github.com/article-i/espace/tree/in-test
https://github.com/article-i/core-pos-pool/tree/test
Commit:
b793f4db6cf6644bd046ae25c1c5c19682c45fb0 (espace)
35299a5071cda0ee1b51659c5608127b7ca92e9f (core-pos-pool)

Documentation:
Whitepaper (partial functional requirements provided)

Functional requirements

Integration and Unit Tests: No
Deployed Contracts Addresses: -
Contracts:
File: ./contracts/IExchange.sol
SHA3: a8ecb49754298f2fbc87f719c9bbc8831a64845eaa761b7d0d3360e53667cdfb

File: ./contracts/eSpace/CoreBridge_multipool.sol
SHA3: 9f448adf501fd5a1234c65ec148f6f23f90ce3427e631313513c1cbd90911a1e

File: ./contracts/PoolContext.sol
SHA3: 59121afe857833d89710964de2bfeb7c95faa3b7d4af0cb262e58a1744bfe451

File: ./contracts/PoSPoolmini.sol
SHA3: f1840f41f2956077a9f3f8909f136484435ffbb17e4cdfaacb63189f72160e88

```
File: ./contracts/VotePowerQueue.sol
SHA3: 9631c1eed19018face1d6f26d630e5b621a3468947c9e06b9c3e0abc114efe06

File: ./contracts/eSpace/Exchangeroom.sol
SHA3: 03cec211403fcb140692215a1e7fb3561760d647f02a830b2474cbac6d0248e2

File: ./contracts/XCFX.sol
SHA3: 20aac8420a1000e1cf3e0cafaf71887e71e9ed7c053c15bbfd0c56c5e259e146
```

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are mostly provided.
- Technical description is provided.

Code quality

The total Code Quality score is **3** out of **10**.

- The code contains duplicate patterns and contracts.
- The development environment is configured.
- Style guide violations found in code, including public functions starting with underscores, deviations from mixed casing and variable naming conventions, and missing camel-case usage.
    - Slither can be used to detect and fix these style guide violations.
- There are two different repositories for code and one for the tests.

Test coverage

Test coverage of the project is **78.86%**.

- Unit tests are provided, but several contracts are not covered at all
- There are scenarios implemented for basic behavior and user interactions.

Security score

As a result of the audit, the code contains **5** low severity issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **7.6**.

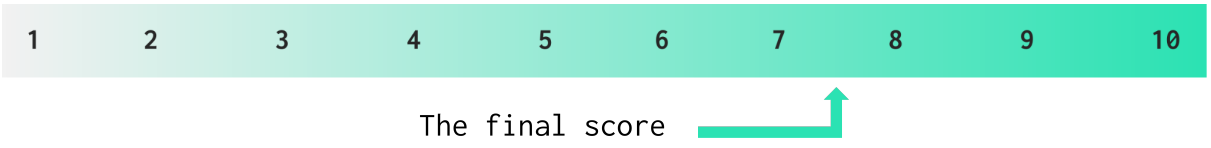

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
13 October 2022	23	9	6	5

05 December 2022	7	6	1	0
19 January 2023	5	0	1	0
24 January 2023	5	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Not Relevant
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed

Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Not Relevant
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Failed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed

Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Nucleon is a project that rewards its users based on the amount of CFX token they deposit:

- *Exchangeroom.sol* - A staking contract for CFX token.
- *CoreBridge_multipool.sol* - A bridge contract to connect Conflux POS pools and Exchange rooms.
- *PoSPoolmini.sol* - A small Conflux POS pool contract with basic usages.
- *PoolAPY.sol* - A library contract operates users' stake information.
- *PoolContext.sol* - An abstract contract to call *Staking* and *Register* contracts functions.
- *VotePowerQueue.sol* - A library contract queues users' voting rights and processes them sequentially.
- *UnstakeQueueCFX.sol* - A library contract adds users who unstake the CFX token to the queue.
- *IExchange.sol* - An interface contract used for the *PoSPoolmini* contract.

Privileged roles

- The owner of the *CoreBridge_multipool* contract can add, delete and modify the pool addresses stored. The owner can set xCFX, bridge, treasury, and trusted triggers addresses.
- The trusted triggers addresses of the *CoreBridge_multipool* contract can synchronize all pools.
- The owner of the *Exchangeroom* contract can set lock periods, exchange limits, pool name and bridge, exchange, storage, xCFX addresses.
- The bridge and exchange addresses of the *Exchangeroom* contract can exchange CFX for xCFX, dequeue pending queue. These addresses can set locked votes, unstaked votes. The bridge can set directly *xcfxvalues, xCFXincrease* variables.
- The owner of the *PoSPoolmini* contract can register the pool contract to the PoS internal contract. The owner can set bridge addresses, lock periods, pool name, CFX count of one vote.
- The bridge of the *PoSPoolmini* contract can increase, decrease, and withdraw CFX tokens. The bridge can claim all the interest from the pool.

Risks

- The repository contains a code that is **out of the audit scope**. The secureness of contracts that are not mentioned in the Scope sedition of the report cannot be guaranteed.

Findings

■■■■ Critical

1. Token Supply Manipulation

XCFX tokens are backed by CFX tokens. The function transfers the CFX token amount to the *_bridgeAddress* and afterward mints XCFX tokens to the *Storage_addr* address. The function has the *onlyBridge* modifier, so this function can be executed from the *_bridgeAddress* address. If *_bridgeAddress* executes this function, since the receiver address is *_bridgeAddress* itself, *_bridgeAddress* will be sent the CFX token to itself. The balance of the *_bridgeAddress* will not change because it sends the CFX token to itself. Thus, this contract will be able to mint XCFX without sending any CFX tokens.

This may lead to XCFX token inflation.

Path: ./contracts/eSpace/Exchangeroom.sol : handleCFXexchangeXCFX()

Recommendation: Change the CFX receiver address.

Status: Mitigated (The Customer states, the transaction mechanism running between Conflux Espace and Core blockchains. _bridgeAddress is the mirror address of CoreBridge_multipool which is deployed in Conflux Core.)

2. Invalid Calculations

It is possible to perform mathematical operations between the different units of the same type of variables, such as time units. The function claims the CFX tokens of pools within the for loop. The *interest* variable is iterated inside the for loop. When the loop is finished, the *interest* variable will be equal to the last item of the *poolAddress* list, and the *systemCFXInterestsTemp* variable calculation will be made with this last element. However, this element does not reflect the total value of all pools.

This may lead to calculation errors which may lead to imbalances.

Path: ./contracts/eSpace/CoreBridge_multipool.sol : claimInterests()

Recommendation: Implement all pools to the *systemCFXInterestsTemp* calculation.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

3. Unverifiable Logic

PoolContext abstract contract uses mock *staking* and *register* contracts. Since these contracts are for testing, everyone can access the critical functions directly through *staking* and *register* contracts.

This may lead to fund losses.

Path: ./contracts/PoolContext.sol

Recommendation: Remove the mock contract imports. Implement production contracts.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

High

1. Data Consistency

The *handleUnstake()* function executes *posPool.poolSummary()*, it performs operation according to the last updated pool attributes. The pool attributes updates with the *_updatePoolShot()* function. *_updatePoolShot()* function should have be called before use the *poolSummary()* function. In *handleUnstake()*, executing *_updatePoolShot()* before the *poolSummary()* is based on the condition *votePower > 0*, if votePower is not greater than zero, then *handleUnstake()* function will use unupdated pool attributes in the calculations.

This may lead to calculation errors which may lead to imbalances.

Path: ./contracts/eSpace/CoreBridge_multipool.sol : handleUnstake()

Recommendation: Execute *_updatePoolShot()* before calling *poolSummary()*.

Status: Mitigated (The Customer states, the handleUnstake operation of the pos pool will not be triggered until the current total withdrawals reach 1000 cfx.)

2. Requirements Violation

The function copies the last item to the address index, then removes the last item. The *break* was never used inside the for loop, so even if the function removed the address from the array list, the for loop would continue to work.

This may lead to the transactions being reverted.

Path: ./contracts/eSpace/CoreBridge_multipool.sol : _delePoolAddress(), _changePoolAddress()

Recommendation: Use *break* keyword after using *pop()* function.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

3. Requirements Violation

The `transfer` function is used to send CFX to addresses. The execution will fail if a sender is a contract with a fallback function.

It would be impossible to use a system with another contract.

Path: ./contracts/eSpace/Exchangeroom.sol : CFX_exchange_XCFX()

Recommendation: Replace transfer and send functions with call or provide special mechanism for interacting with a smart contract.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

4. Unverifiable Logic

collectEndedVotes() and sumEndedVotes() functions contains a check if `q.items[i].endBlock > block.number`. If this check applies the issued function a continue statement which will allow an execution to continue with the next item. However, since this is a queue, the items later then [i] will match the if check stated above, and the execution will move on for no reason and will increase the Gas cost.

Since these functions are used in critical operations such as withdrawals, the increased Gas consumption will not be optimal. In addition, in extreme cases, this will cause out-of-Gas exceptions.

Path: /espace/contracts/VotePowerQueue.sol : collectEndedVotes(), sumEndedVotes()

Recommendation: Use the `break` statement instead of `continue`.

Status: Mitigated (Since the system always uses sorted data, this issue cannot be created.)

5. Requirements Violation

The contract uses mock contracts functions. Mock contract functions are not working as expected.

register(), *increaseStake()*, *decreaseStake()* functions should perform CFX token transfer, but in fact, these functions do not perform CFX token transfer.

The CFX token was not sent to the contract, *withdrawStake()* function will not work as it will try to send the CFX token to the executor, but the balance of the contract is insufficient.

Path: ./core-pos-pool/contracts/PoSPoolmini.sol : register(), increaseStake(), decreaseStake(), withdrawStake()

Recommendation: Remove the mock contract imports. Implement production contracts.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

6. Highly Permissive Role Access

The owner can burn tokens at any time without notifying the users.

Path: ./espace/contracts/XFXCX.sol: burnTokens()

Recommendation: Either notify the users about this functionality or remove this functionality.

Status: Fixed (https://docs.nucleon.network/about-nucleon/risks)

Medium

1. Unoptimized Loop Usage

In the *Exchangeroom* contract, the functions take *unstakeLen()* as a for loop repeat value and perform external calls while iterating over them.

This can lead to out-of-Gas exceptions.

Paths: ./contracts/VotePowerQueue.sol : sumEndedVotes(), collectEndedVotes(), queueItems(), queueItems,

./contracts/PoolAPY.sol : clearOutdatedNode(),

./contracts/VotePowerQueue.sol : queueItems(), collectEndedVotes(), sumEndedVotes(), clear()

Recommendation: Implement array length limitations.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

2. Unchecked Return Value

The function dequeue returns *Node*. The call made to the *handleAllUnstakeTask()* does not check its return value. This means that the contract will continue its execution even if there is an erroneous situation.

This can lead to unexpected behavior in the contract.

Path: ./espace/contracts/eSpace/Exchangeroom.sol : handleAllUnstakeTask()

Recommendation: Implement a check of the returning value.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

3. Missing Balance Check before Transfer

The current balance of the contract should be checked before Ether and token transfers.

This can lead to reverts.

Path: ./espace/contracts/PoSPoolmini.sol : withdrawStake()

Recommendation: Check that the contract balance is sufficient for transfer before calling transfer.

Status: Mitigated (The Customer specified this function will be used inside the CoreBridge_multipool and the contract has the necessary checks.)

4. **Checks-Effects-Interactions Violation**

   The contract state is being updated after external calls (addTokens(), transfer(), burnTokens()) in the Exchangeroom.sol contract.

   This can lead to reentrancies, race conditions, or Denial-of-Service vulnerabilities.

   Path: ./contracts/eSpace/CoreBridge_multipool.sol : handleUnstake()

   Recommendation: Implement function according to the Checks-Effects-Interactions pattern or use ReentrancyGuards.

   Status: Fixed (Revised commit: 35299a5071cda0ee1b51659c5608127b7ca92e9f)

5. **Unverifiable Logic**

   The *Exchangeroom* contract uses *IXCFX* contract, which is not in the audit scope.

   This can lead to unexpected behaviors.

   Path: ./espace/contracts/eSpace/Exchangeroom.sol

   Recommendation: Use audited contracts.

   Status: Mitigated (The implementation of the issued contract has been provided.)

6. **Tautology**

   Since the contract uses uint values _poolSummary.unlocked can never be smaller than zero.

   Path: ./core-pos-pool/contracts/PoSPoolmini.sol : withdrawStake()

   Recommendation: Remove related require statement.

   Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

7. **Redundant Functions**

   The fallback and receive functions are redundant and increase the code size.

   In addition to that, since there is no way to withdraw ETH from the contract, these functions will cause more ETH to be locked on the contract.

Path: ./core-pos-pool/contracts/PoSPoolmini.sol : fallback(), receive()

Recommendation: Delete these functions.

Status: Fixed (Revised commit: 35299a5071cda0ee1b51659c5608127b7ca92e9f)

8. Unoptimized Loop Usage

In the *CoreBridge_multipool* contract, the *claimInterests()*, *withdrawVotes()* functions perform external calls while iterating over arrays.

This can lead to out-of-Gas exceptions.

Path: ./core-pos-pool/contracts/CoreBridge_multipool.sol : claimInterests(), withdrawVotes()

Recommendation: Implement array length limitations.

Status: Fixed (Revised commit: 35299a5071cda0ee1b51659c5608127b7ca92e9f)

9. Contradiction

According to *_setLockPeriod()* function, the _poolLockPeriod_out value should be greater than *_poolLockPeriod_in* value. However, in the *_setLockPeriod()* the validation is missed.

Path: ./contracts/Exchangeroom.sol: _setLockPeriod()

Recommendation: Implement checks to enforce that _poolLockPeriod_out is greater than _poolLockPeriod_in.

Status: Fixed (Revised commit: b793f4db6cf6644bd046ae25c1c5c19682c45fb0)

10. Contradiction

The implementation contains commented code which looks like it should be uncommented to finalize the code.

The contract contains commented function.

Paths: ./contracts/VotePowerQueue.sol: clear(),

./eSpace/contracts/VotePowerQueue.sol:

Recommendation: Remove the commented code or finalize its implementation.

Status: Fixed (Revised commit: b793f4db6cf6644bd046ae25c1c5c19682c45fb0)

Low

1. **Duplicate Array Items**

   It is possible to add the same addresses to the list again.

   This may lead to unnecessary Gas consumption.

   Path: ./contracts/eSpace/CoreBridge_multipool.sol: _addPoolAddress()

   Recommendation: Implement duplicate checks.

   Status: Reported

2. **Floating Pragma**

   The project uses floating pragma ^0.8.0

   Paths: ./espace/contracts/VotePowerQueue.sol

   ./espace/contracts/eSpace/UnstakeQueueCFX.sol

   ./espace/contracts/eSpace/Exchangeroom.sol

   ./core-pos-pool/contracts/VotePowerQueue.sol

   ./core-pos-pool/contracts/PoolAPY.sol

   ./core-pos-pool/contracts/PoolContext.sol

   ./core-pos-pool/contracts/PoSPoolmini.sol

   ./contracts/eSpace/CoreBridge_multipool.sol

   Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

   Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

3. **State Variable Default Visibility**

   Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

   Paths: ./contracts/eSpace/Exchangeroom.sol: XCFX_address, Storage_addr

   ./contracts/eSpace/CoreBridge_multipool.sol : CFX_COUNT_OF_ONE_VOTE, CFX_VALUE_OF_ONE_VOTE, Unstakebalanceinbridge, identifier, trusted_node_trigers

   Recommendation: Variables can be specified as being public, internal, or private. Explicitly define visibility for all state variables.

   Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

4. **Redundant Variable**

   The usage of *zero_addr* is unnecessary for the contract.

Paths: ./contracts/eSpace/Exchangeroom.sol : initialize()

./contracts/PoSPoolmini.sol : initialize()

Recommendation: Remove assignments inside the *initialize* function.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

5. **Redundant Assignment**

ONE_DAY_BLOCK_COUNT, *CFX_COUNT_OF_ONE_VOTE*, *CFX_VALUE_OF_ONE_VOTE* variables already assigned when variable defined, then *initialize* function assigned the same value again.

Paths: ./core-pos-pool/contracts/PoSPoolmini.sol: initialize()

Recommendation: Remove assignments inside the *initialize* function.

Status: Fixed (Revised commit: 35299a5071cda0ee1b51659c5608127b7ca92e9f)

6. **Redundant Calculation**

The *register* function checks if the votePower with a require statement, and then checks multiplies CFX_VALUE_OF_ONE_VOTE by votePower, which is 1.

While calculating the Unstakebalanceinbridge, the code divides Unstakebalanceinbridge to CFX_VALUE_OF_ONE_VOTE and then multiplies it with CFX_VALUE_OF_ONE_VOTE.

Paths: ./core-pos-pool/contracts/PoSPoolmini.sol: : *register*()

./contracts/eSpace/CoreBridge_multipool.sol: handleUnstake()

Recommendation: Remove redundant multiplication.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

7. **Natspec Mismatch**

The ExchangeSummary definition states that different fields than the actual struct defined below.

Path: ./contracts/eSpace/Exchangeroom.sol

Recommendation: Re-define the Natspec.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

8. **Redundant Function**

The *_blockNumber()* function is redundant since the *block.number* is a globally defined variable in Solidity.

The *initialize()* function is defined as *public* and has no parameter. The function is called inside the *constructor*. Since the function has no parameter and is called inside the *constructor*, there is no need for *initialize()* function.

Paths: ./contracts/eSpace/Exchangeroom.sol: *_blockNumber()*

./core-pos-pool/contracts/PoolContext.sol : *_blockNumber()*

./contracts/eSpace/CoreBridge_multipool.sol : initialize()

Recommendation: Remove redundant function.

Remove *initialize()* function and imported *initializer* contract. Implement all of the *initialize()* function logic to the *constructor*.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

9. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x00.

This can lead to unwanted external calls to 0x0.

Path: ./contracts/PoSPoolmini.sol : _setbridges()

Recommendation: Implement zero address checks.

Status: Reported

10. Hardcoded Maths

Hard-coded values are used in computations.

Path: ./contracts/eSpace/Exchangeroom.sol: initialize(), XCFX_burn(),

Recommendation: Convert these variables into constants.

Status: Reported. (Putting the exact same values into constants would not lead to any changes in the event of a hard fork. As the official documentation states, the values of the constant variables cannot be changed after the contract construction.)

11. Redundant Use of SafeMath

Since Solidity v0.8.0, the overflow/underflow check is implemented via ABIEncoderV2 on the language level - it adds validation to the bytecode during compilation.

There is no need to use the SafeMath library.

Path: ./contracts/eSpace/Exchangeroom.sol

Recommendation: Remove the SafeMath library.

Status: Reported

12. Setting Uint64 Values to Uint256 Variables

The *_setLockPeriod()* takes two uint64 parameters and sets them directly to uint256 variables. This set operation is syntactically correct but can lead to miscalculations in mathematical operations.

Paths: ./contracts/eSpace/Exchangeroom.sol: _setLockPeriod

./contracts/eSpace/CoreBridge_multipool.sol: _setPoolUserShareRatio

Recommendation: Re-adjust this setting.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

13. Unused Function

The *isContract()* function is never used.

Paths: ./contracts/eSpace/Exchangeroom.sol: isContract

./contracts/VotePowerQueue.sol : queueItems(), clear(),

./contracts/eSpace/CoreBridge_multipool.sol : queryespacexCFXincrease(), queryInterest()

Recommendation: Remove unused function.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

14. Missing Events

Events for critical state changes should be emitted for tracking things off-chain.

Paths: ./contracts/Exchangeroom.sol : _setLockPeriod(), _setminexchangelimits(), _setPoolName(), _setBridge(), _setCoreExchange(), _setStorageaddr(), _setXCFXaddr(), setxCFXValue(), setlockedvotes(), handlexCFXadd(), handleUnstake()

./contracts/CoreBridge_multipool.sol : _clearTheStates(), _setPoolUserShareRatio(), _setCfxCountOfOneVote(), _settrustedtrigers(), _seteServicetreasuryAddress(), _seteSpacebridgeAddress(), _seteSpacexCFXAddress(), _seteSpaceExroomAddress(), _delePoolAddress(), _changePoolAddress(), _addPoolAddress()

./contracts/PoSPoolmini.sol : _set_bridges(), _setLockPeriod(), _setPoolName(), _setCfxCountOfOneVote(), _reStake(), claimAllInterest(),

Recommendation: Create and emit related events.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

15. Unused Variable

The variable *RATIO_BASE* is unused.

Path: ./contracts/eSpace/Exchangeroom.sol:

Recommendation: Remove unused variable.

Status: Fixed (Revised commit: 04cce4a32eb000ae279e1f0e6744f4f2d5fea4e6)

16. Redundant Modifier

getback_CFX() function calls the internal *withdraw()* function, called function has the same *onlyRegisted()* modifier.

There is no need to use the same modifier again, this will increase the Gas consumption.

Path: ./contracts/Exchangeroom.sol : getback_CFX()

Recommendation: Remove *onlyRegisted()* modifier from *withdraw()* function.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

17. Misleading Modifier Name

According to the modifier logic *msg.sender* can be a *bridge* or *exchange* contract address. *onlyBridge()* implements logic that contradicts its name.

This makes code harder to read.

Path: ./contracts/Exchangeroom.sol : onlyBridge()

Recommendation: Change the modifier name to fit the logic.

Status: Mitigated. (Implemented this way dues to future versions.)

18. Redundant Storage Usage

The function saves variables to storage multiple times instead of saving values to a local variable.

This will increase Gas consumption.

Path: ./contracts/Exchangeroom.sol : CFX_exchange_XCFX(), XCFX_burn(), withdraw()

Recommendation: Save the values returned from the *_exchangeSummary.totalxcfxs*, *userSummaries[msg.sender].unlocked* variables to the local memory. Use memory variables in the process.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

19. Code Duplication

uint256 temp_amount = userOutqueues[msg.sender].collectEndedVotes();

userSummaries[msg.sender].unlocked += temp_amount;

userSummaries[msg.sender].unlocking -= temp_amount;

These code blocks are used the same way in the two functions. There is no need to repeat the code.

This will increase the contract byte size.

Path: ./contracts/eSpace/Exchangeroom.sol: XCFX_burn(), withdraw()

Recommendation: Create function for these code blocks. Use the created function for XCFX_burn() and withdraw() functions.

Status: Fixed (Revised commit: adf3d1579d5a8f8386e6278f7d171f0eee6a0cbb)

20. Redundant Variable

_exchangeSummary.totalxcfxs is always equal to XCFX token total supply. There is no need to create *totalxcfxs* variable.

This will increase the Gas consumption.

Path: ./contracts/Exchangeroom.sol

Recommendation: Use XCFX token total supply in the calculations.

Status: Mitigated. (Kept for information purposes.)

21. Commented Code Parts

Commented parts of code in a contract. They will not cause security issues but will make code less clear.

Paths: ./contracts/eSpace/Exchangeroom.sol, ./contracts/eSpace/CoreBridge_multipool.sol,

./contracts/IExchange.sol

Recommendation: Remove commented parts of code.

Status: Fixed (Revised commit: b793f4db6cf6644bd046ae25c1c5c19682c45fb0)

22. Boolean Equality

Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/eSpace/CoreBridge_multipool.sol: Only_trusted_triggers

Recommendation: Remove boolean equality.

Status: Fixed (Revised commit: 35299a5071cda0ee1b51659c5608127b7ca92e9f)

23. Style Guide Violation

The contract names and functions do not follow the official guidelines.

The provided projects should follow the official guidelines.

Paths: ./contracts/PoSPoolmini.sol

./contracts/eSpace/storagesbridge.sol

./contracts/eSpace/systemstorages.sol

./contracts/eSpace/Exchangeroom.sol

./contracts/eSpace/CoreExchange.sol

./contracts/eSpace/CoreBridge_multipool.sol

Recommendation: Follow the official Solidity guidelines.

Status: Reported

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.