BLOCKHAT
SECURITY

Crypto Index Pool

Smart Contract Security Audit

Prepared by BlockHat

February 26th, 2023 - February 27th, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	golive106
Version	0.1
Classification	Public

Scope

The Crypto Index Pool Contract in the Crypto Index Pool Repository

Repo	Owner
https://arbiscan.io/token/0x50B871fb5Bba2895425e5Fc6ebA219197f21D6D5	0x50B871fb5Bba2895425e5Fc6ebA219197f21D6D5

Files	MD5 Hash
CryptoIndexPool.sol	7effdf51f0442cc8f2eeca00657ab31d

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

Crypto Index Pool engaged BlockHat to conduct a security assessment on the Crypto Index Pool beginning on February 26th, 2023 and ending February 27th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Crypto Index Pool

-

Issuer	golive106
Website	–
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

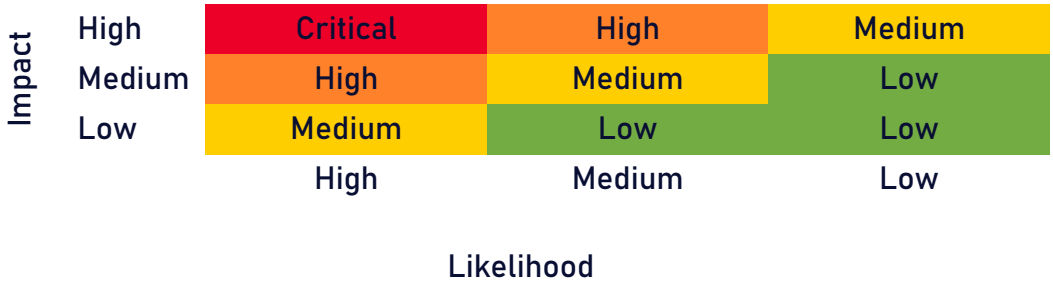
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Crypto Index Pool implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 1 critical-severity, 1 medium-severity, 1 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Centralization risk	CRITICAL	Not Fixed
Missing address verification	MEDIUM	Not Fixed
Owner Can Renounce Ownership	LOW	Not Fixed

3 Finding Details

A CryptoIndexPool.sol

A.1 Centralization risk [CRITICAL]

Description:

The owner can set any value in _maxWalletAmount variable. This represents a significant centralization risk on the user side. The owner can set 0 as a value and pause transfers.

Code:

Listing 1: CryptoIndexPool.sol

```
487     function changeMaxWalletAmount(
488         uint256 _maxWalletAmount
489     ) external onlyOwner returns (bool) {
490         maxWalletAmount = _maxWalletAmount;

492         return true;
493     }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

To prevent the owner from setting the max wallet amount to zero, which would pause transfers, we recommend adding a 'require' statement that limits the _maxWalletAmount to a minimum value.

Status - Not Fixed

A.2 Missing address verification [MEDIUM]

Description:

Certain functions lack a safety check in the address, the address-type argument _contract should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 2: CryptoIndexPool.sol

```
561     function addStakingContract(address _contract) public onlyOwner {
562         stakingContract = _contract;
563     }
```

Risk Level:

Likelihood – 1
Impact - 4

Recommendation:

We recommend that you make sure the address provided in the argument is different from the address(0).

Status - Not Fixed

A.3 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the

renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 3: CryptoIndexPool.sol

```
195     function renounceOwnership() external onlyOwner {
196         _transferOwnership(address(0));
197     }
```

Listing 4: CryptoIndexPool.sol

```
250 contract CryptoIndexPool is Context, IERC20, IERC20Metadata, Ownable {
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Not Fixed

4 Best Practices

BP.1 Public functions can be external

Description:

Functions with a public scope that are not called inside the contract should be declared external to reduce the gas fees

Code:

Listing 5: CryptoIndexPool.sol

```
561     function addStakingContract(address _contract) public onlyOwner {
562         stakingContract = _contract;
563     }
```

5 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
CryptoIndexPool.transfer(address,uint256).owner (CryptoIndexPool.sol
   ↪ #330) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool.allowance(address,address).owner (CryptoIndexPool.sol
   ↪ #339) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool.approve(address,uint256).owner (CryptoIndexPool.sol#356)
   ↪  shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool.increaseAllowance(address,uint256).owner (
   ↪ CryptoIndexPool.sol#404) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool.decreaseAllowance(address,uint256).owner (
   ↪ CryptoIndexPool.sol#427) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool._approve(address,address,uint256).owner (CryptoIndexPool
   ↪ .sol#528) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
CryptoIndexPool._spendAllowance(address,address,uint256).owner (
   ↪ CryptoIndexPool.sol#545) shadows:
       - Ownable.owner() (CryptoIndexPool.sol#177-179) (function)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #local-variable-shadowing

CryptoIndexPool.changeMaxWalletAmount(uint256) (CryptoIndexPool.sol
    ↪ #487-493) should emit an event for:
        - maxWalletAmount = _maxWalletAmount (CryptoIndexPool.sol#490)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic

CryptoIndexPool.addStakingContract(address)._contract (CryptoIndexPool.
    ↪ sol#561) lacks a zero-check on :
                - stakingContract = _contract (CryptoIndexPool.sol#562)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-zero-address-validation

Parameter CryptoIndexPool.changeMaxWalletAmount(uint256).
    ↪ _maxWalletAmount (CryptoIndexPool.sol#488) is not in mixedCase
Parameter CryptoIndexPool.addStakingContract(address)._contract (
    ↪ CryptoIndexPool.sol#561) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions

CryptoIndexPool.constructor() (CryptoIndexPool.sol#269-273) uses
    ↪ literals with too many digits:
        - _mint(msg.sender,10000000 * 10 ** decimals()) (CryptoIndexPool.
            ↪ sol#272)
CryptoIndexPool.slitherConstructorVariables() (CryptoIndexPool.sol
    ↪ #250-564) uses literals with too many digits:
        - maxWalletAmount = 1000000 * 10 ** decimals() (CryptoIndexPool.
            ↪ sol#256)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits

addStakingContract(address) should be declared external:
```

```
        - CryptoIndexPool.addStakingContract(address) (CryptoIndexPool.
            ↪ sol#561-563)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

We examined the design and implementation of Crypto Index Pool in this audit and found several issues of various severities. We advise golive106 team to implement the recommendations contained in all 3 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT
SECURITY

For a Contract Audit, contact us at contact@blockhat.io