HAECHI AUDIT

Armor

Smart Contract Security Analysis

Published on : Mar 04, 2022

Version v1.0

HAECHI AUDIT

Smart Contract Audit Certificate

Armor

Security Report Published by HAECHI AUDIT
v1.0 Mar 04, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	-	-	-	-	-
Minor	-	-	-	-	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

0 Issues (0 Critical, 0 Major, 0 Minor) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring.

We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by Armor team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Armor team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends Armor team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found at GitHub (https://github.com/HAECHI-LABS/Armor-audit/tree/a0966bc7239dedae82d3bf98e5f6bb3741ed7133/contracts). The last commit of the code used for this Audit is “a0966bc7239dedae82d3bf98e5f6bb3741ed7133”.

Issues	HAECHI AUDIT found 0 critical issues, 0 major issues, and 0 minor issues. There are 0 Tips issues explained that would improve the code’s usability or efficiency upon modification.

OVERVIEW

Contracts subject to audit

- RcaController
- RcaRanking
- RcaShield
- RcaShieldAave
- RcaShieldBase
- RcaShieldCompound
- RcaShieldConvex
- RcaShieldOnsen
- RcaTresury
- Aave
- Compound
- Convex
- NexusMutual
- Sushiswap
- Governable
- RcaGovernable
- IRcaController
- IRcaShield
- IZapper
- MerkleProof
- MockERC20

FINDINGS

There were no issues found.

Test results attached in the appendix.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the mainnet. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
RCAController and RCAShield
  RCA controller spec
    controller constructor
      ✓ set apr properly
      ✓ set discount properly
      ✓ set withdrawalDelay properly
      ✓ set treasury properly
      ✓ set guardian properly
      ✓ set capOracle properly
      ✓ set priceOracle properly
    #initializeShield
      ✓ should fail if msg.sender is not governor
      ✓ should fail if length mismatch
      valid case
        ✓ initialize shield contract properly
        ✓ set shieldMapping properly
        ✓ set shieldMapping properly
        ✓ set lastShieldUpdate properly
        ✓ set shieldProtocolPercents properly
        ✓ emit ShieldCreated event
    #setLiqTotal
      ✓ should fail if msg.sender is not governor
      valid case
        ✓ set liqForClaimsRoot properly
        ✓ set reserveRoot properly
        ✓ updates system properly
      #setWithdrawalDelay
        ✓ should fail if msg.sender is not governor
        ✓ should fail if delay is too big
        valid case
          ✓ set withdrawalDelay properly
          ✓ updates system properly
      #setDiscount
        ✓ should fail if msg.sender is not governor
        ✓ should fail if discount is too big
```

```
    valid case
      ✓ set discount properly
      ✓ updates system properly
  #setApr
    ✓ should fail if msg.sender is not governor
    ✓ should fail if apr is too big
    valid case
      ✓ set apr properly
      ✓ updates system properly
  #setTreasury
    ✓ should fail if msg.sender is not governor
    valid case
      ✓ set treasury properly
      ✓ updates system properly
  #setPercentReserved
    ✓ should fail if msg.sender is not guardian
    valid case
      ✓ set reservedRoot properly
      ✓ updates system properly
  #setPrices
    ✓ should fail if msg.sender is not priceOracle
    valid case
      ✓ set priceRoot properly
      ✓ updates system properly
Shield spec
 #mintTo
   ✓ should fail if signature permission has expired
   ✓ should fail if capacity oracle signature is not valid
   valid case
     ✓ renew lastUpdate of shield
     ✓ user get RCA tokens properly
     ✓ underlyingToken moves to shield contract
 #redeemRequest and #redeemFinalize
   valid case
     redeem requests
       ✓ burn RCA tokens
       ✓ renew lastUpdate of shield
       ✓ increase pendingWithdrawal
       ✓ renew withdrawRequests of msg.sender
     redeem finalize(when zapper is false)
       ✓ should fail if withdrawal not allowed
       ✓ transfer utokens properly
       ✓ renew lastUpdate of shield
       ✓ decrease pendingWithdrawal
```

```
      ✓ delete withdrawRequests of msg.sender
    redeem finalize(when zapper is true)
      ✓ should fail if withdrawal not allowed
      ✓ get Eth instead of UToken
      ✓ renew lastUpdate of shield
      ✓ decrease pendingWithdrawal
      ✓ delete withdrawRequests of msg.sender
  #purchaseU
    ✓ should fail if price is wrong
    ✓ should fail if msg.value is wrong
    valid case
      ✓ set amtForSale properly
      ✓ user should get underlying token properly
      ✓ user's balance should decrease
      ✓ treasury should gets msg.value
  #purchaseRca
    ✓ should fail if price is wrong
    ✓ should fail if msg.value is wrong
    valid case
      ✓ set amtForSale properly
      ✓ user should get Rca token properly
      ✓ user's balance should decrease
      ✓ treasury should get msg.value
    #setTreasury
      ✓ shoild fail if msg.sender is not controller
      valid case
        ✓ set treasury properly
    #setPercentReserved
      ✓ shoild fail if msg.sender is not controller
      valid case
        ✓ set percentReserved properly
    #setWithdrawalDelay
      ✓ shoild fail if msg.sender is not controller
      valid case
        ✓ set withdrawalDelay properly
    #setDiscount
      ✓ shoild fail if msg.sender is not controller
      valid case
        ✓ set discount properly
    #setApr
      ✓ shoild fail if msg.sender is not controller
      valid case
        ✓ set apr properly
    #setController
```

```
      ✓ shoild fail if msg.sender is not governor
      valid case
        ✓ set controller properly

RcaRanking spec
  #stake
    ✓ protocol balances of msg.sender should increase
    ✓ ranks of protocol should increase
    ✓ balance of msg.sender should decrease
  #unstake
    ✓ protocol balances of msg.sender should decrease
    ✓ ranks of protocol should decrease
    ✓ balance of msg.sender should increase
sample shield
    RcaShieldConvex spec
      constructor
        ✓ set rewardPool properly
      #purchase
        ✓ should fail if _token is UToken
        ✓ should fail if price is wrong
        valid case
          ✓ msg.sender get _tokens
          ✓ stake UToken in rewardPool
    RcaShieldCompound spec
      constructor
        ✓ set comptroller properly
      #purchase
        ✓ should fail if _token is UToken
        ✓ should fail if price is wrong
        valid case
          ✓ msg.sender get _tokens
          ✓ uToken moves to shield
    RcaShieldOnsen spec
      constructor
        ✓ set masterchef properly
        ✓ set sushi properly
        ✓ set _pid properly
      #purchase
        ✓ should fail if _token is UToken
        ✓ should fail if price is wrong
        valid case
          ✓ msg.sender get _tokens
          ✓ deposits uTokens
    RcaShieldAave spec
```

```
    constructor
      ✓ set incentivesController properly
    #purchase
      ✓ should fail if _token is UToken
      ✓ should fail if price is wrong
      valid case
        ✓ msg.sender get _tokens
        ✓ uToken moves to shield
RcaTreasury spec
  #claimFor
    ✓ should fail if already claimed loss
    valid case
      ✓ set claimed true
      ✓ transfer to user for loss
  #setClaimsRoot
    ✓ should fail if msg.sender is not governor
    valid case
      ✓ set claimsRoots properly
  #withdraw
    ✓ should fail if msg.sender is not governor
    valid case
      ✓ transfer to _to address
```

End of Document