HAECHI AUDIT

Iskra (Token & Vesting)

Smart Contract Security Analysis

Published on : Nov 20, 2022

Version v1.0

HAECHI AUDIT

Smart Contract Audit Certificate

Iskra (Token & Vesting)

Security Report Published by HAECHI AUDIT
v1.0 Nov 20, 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, 0 tips) 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 400+ 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 the Iskra team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by the Iskra 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 that the Iskra 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 on the following repository and commit hash.

Repository : https://github.com/iskraworld/iskra-console-cmd

Commit hash : 24acebe0c41cae19dc38b3e0003cab0fa65e093a

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

- ❖ IKIP7
- ❖ IKIP7Metadata
- ❖ IKIP7Receiver
- ❖ IKIP13
- ❖ IKIP17
- ❖ IKIP17Enumberable
- ❖ IKIP17Metadata
- ❖ GameToken
- ❖ MultiToken
- ❖ Vesting

FINDINGS

No Issues are found.

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 main network. 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.

```
GameToken
  #constructor()
    ✔ name set properly
    ✔ symbol set properly
    ✔ decimals set properly
    ✔ initialSupply set properly
    ✔ totalSupply set properly
  #supportsInterface
    ✔ should check interface properly
  ERC20 Spec
    #approve()
      valid case
        ✔ allowance should set appropriately
        ✔ should emit Approval event
    #transfer()
      ✔ should fail if recipient is AddressZero
      ✔ should fail if sender's amount is more than balance
      valid case
        ✔ sender's balance should decrease
        ✔ recipient's balance should increase
        ✔ should emit Transfer event
    #transferFrom()
      ✔ should fail if recipient is AddressZero
      ✔ should fail if sender's amount is more than balance
      ✔ should fail if sender's amount is more than allowance
      ✔ should fail if try to transfer sender's token without approval process
      valid case
        ✔ sender's balance should decrease
        ✔ recipient's balance should increase
        ✔ allowance should decrease
        ✔ should emit Transfer event
        ✔ should emit Approval event
  #safeTransfer
    ✔ should fail if try to transfer to abnormal address

 MultiToken
  #constructor()
    ✔ register ERC1155 interface
  #balanceOf()
  ✔ should fail if query for ZERO_ADDRESS
  ✔ returns 0 if user does not have any tokens
  ✔ returns the amount of tokens owned by user
#balanceOfBatch()
  ✔ should fail if query for ZERO_ADDRESS
  ✔ should fail if length mismatch
  ✔ returns 0 if user does not have any tokens
  ✔ returns the amount of tokens owned by user
#setApprovedForAll()
  ✔ should fail if try to approve msg.sender
  valid case
    when approved flag is true
      ✔ approved for all
      ✔ approved user can transfer any tokens
      ✔ should emit ApprovalForAll event
    when approved flag is false
      ✔ approved for all clear
      ✔ should emit ApprovalForAll event
#safeTransferFrom()
  ✔ should fail if msg.sender is not owner nor approved
  ✔ should fail if recipient is ZERO_ADDRESS
  ✔ should fail if CA recipient has invalid return value (46ms)
  ✔ should fail if CA recipient reverts (42ms)
  when recipient: EOA
    normal transferFrom executed
      ✔ token balance change
      ✔ should emit TransferSingle event
    when recipient: CA with _ERC1155_RECEIVED return value
      normal transferFrom executed
        ✔ token balance should change
        ✔ should emit TransferSingle event
#safeBatchTransferFrom()
  ✔ should fail if msg.sender is not owner nor approved
  ✔ should fail if recipient is ZERO_ADDRESS
  ✔ should fail if CA recipient has invalid return value (48ms)
  ✔ should fail if CA recipient reverts (44ms)
  when recipient: EOA
    normal transferFrom executed
      ✔ token balance change
      ✔ should emit TransferBatch event
    when recipient: CA with _ERC1155_BATCH_RECEIVED return value
      normal transferFrom executed
        ✔ token balance should change
        ✔ should emit TransferBatch event
#mint()
  ✔ should fail if msg.sender is not owner
  valid case
    ✔ token mint
#mintBatch()
```

```
    ✔ should fail if msg.sender is not owner
    ✔ should fail if mint fail
    valid case
      ✔ token mint
  #burn()
    ✔ should fail if msg.sender does not have token with respect to tokenId
    ✔ should fail if msg.sender does not have enough amount token to burn
    valid case
      ✔ token burn
    #burnBatch()
      ✔ should fail if msg.sender does not have token with respect to tokenId
      ✔ should fail if msg.sender does not have enough amount token to burn
      valid case
        ✔ token burn

Vesting
  #prepare()
    ✔ should fail if msg.sender is not owner
    ✔ should fail if status is not CREATED
    ✔ should fail if beneficiary is addressZero
    ✔ should fail if duration is zero
    ✔ should fail if amount is zero
    ✔ should fail if amount is less than duration
    ✔ should fail if allowance is less than initialVestingAmount
    ✔ should fail if invalid initial unlocked amount is given
    valid case
      ✔ token should be set properly
      ✔ duration should be set properly
      ✔ beneficiary should be set properly
      ✔ initialVestingAmount should be set properly
      ✔ unlockUnit should be set properly
      ✔ remainder should be set properly
      ✔ VestingStatus should be set to PREPARED
      ✔ distributor's balance should decrease
      ✔ vesting contact's balance should increase
      ✔ should emit Prepared event
  #setStart()
    ✔ should fail if msg.sender is not owner
    ✔ should fail if start is zero
    ✔ should fail if VestingStatus is not PREPARED
    valid case
      ✔ start should be set properly
      ✔ end should be set properly
      ✔ status should be set to ACTIVE
      ✔ should emit SetStart event
  #revoke()
    ✔ should fail if msg.sender is not owner
    ✔ should fail if VestingStauts is CREATED
    ✔ should not transfer if vesting contract's balance is zero (50ms)
    valid case
      ✔ vesting contract's balance should be zero
      ✔ reclaimer's balance should increase
      ✔ status should be set to REVOKED
      ✔ should emit Revoked event
  #changeBeneficiary()
    ✔ should fail if msg.sender is not beneficiary
    ✔ should fail if VestingStatus is CREATED
    ✔ should fail if VestingStatus is REVOKED (42ms)
    valid case
      ✔ beneficiary should be set properly
      ✔ should emit SetBeneficiary event
  #getNextUnlock()
    ✔ should fail if VestingStatus is not ACTIVE
    ✔ should fail if current time is greater than end (38ms)
    valid case
      t <= start
        ✔ should return start + UNLOCK_PERIOD
      start < t < end
        ✔ t = start + 100 hours
        ✔ t = start + 730 hours
        ✔ t = start + 1000 hours
        ✔ t = start + 9000 hours
  #getClaimableAmount()
    ✔ should fail if VestingStatus is not ACTIVE
    valid case
      t < start
        ✔ t = start - 1 hours
      t >= end
        ✔ t = end + 1 hours
      start <= t < end
        ✔ t = start + 500 hours
        ✔ t = start + 1500 hours
        ✔ t = start + 3000 hours
        ✔ t = start + 10000 hours
        ✔ t = start + 1500 + 1500 hours
        ✔ t = start + 1500 + 1500 + 5000 hours (47ms)
  #claim()
    ✔ should fail if VestingStatus is not ACTIVE
    ✔ should fail if msg.sender is not beneficiary
    ✔ should fail if amount is zero (42ms)
    ✔ should fail if amount is greater than claimable amount
    valid case
      ✔ vesting contract's balance should decrease
      ✔ beneficiary's balance should increase
      ✔ should emit Claimed event
```

End of Document