HAECHI AUDIT

Boomco

Smart Contract Security Analysis

Published on :Aug 01, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

Boomco

Security Report Published by HAECHI AUDIT
v1.0 July 22, 2022
v2.0 Aug 01, 2022

Auditor : Felix Kim

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	1	1	-	-	-
Major	1	1	-	-	-
Minor	1	-	-	1	-
Tips	-	-	-	-	-

TABLE OF CONTENTS

3 Issues (1 Critical, 1 Major, 1 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 in 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 Boomco team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by Boomco 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 Boomco 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/audit-boomco/blob/800fe446e421cfe389da22ccbf4c8876967d43be/contracts). The last commit of the code used for this Audit is “800fe446e421cfe389da22ccbf4c8876967d43be”.

Issues	HAECHI AUDIT found 1 critical issue, 1 major issue, and 1 minor issue. There are 0 Tips issues explained to improve the code’s usability or efficiency upon modification.
Update	[v2.0] - From the new commit, “097bd2968cc7e0ea39ac1a1b47240755da08e974”, 1 critical issue and 1 major issue have been modified.

Severity	Issue	Status
CRITICAL	The owner can send any user’s tokens.	(Found - v1.0) (Resolved - v2.0)
MAJOR	chainId is variable arbitrarily.	(Found - v1.0) (Resolved - v2.0)
MINOR	ERC165’s interface id shows an incorrect value.	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- ❖ Gov
- ❖ Commitable
- ❖ Interface

FINDINGS

CRITICAL

The owner can send any user's tokens. (Found - v.1.0) (Resolved - v.2.0)

```
   function approveForSpending(address owner, uint rawAmount) onlyCommitter
isNotDestroyed external{
        uint96 amount;
        if (rawAmount == uint(-1)) {
            amount = uint96(-1);
        } else {
            amount = safe96(rawAmount, "Comp::approve: amount exceeds 96 bits");
        }

        allowances[owner][msg.sender] = amount;

        emit Approval(owner, msg.sender, amount);
   }
```

[https://github.com/HAECHI-LABS/audit-boomco/blob/800fe446e421cfe389da22ccbf4c8876967d43be/contracts/DAO/Gov.sol#L106-L117]

Issue

The *Gov#approveForSpending()* function allows the committer to change any user's allowance when the Commitable#isDestroyed variable is not true. Because the owner can arbitrarily add or remove the committer, this means that the owner can eventually transfer any user's tokens.

Recommendation

It is advised to delete the *Gov#approveForSpending()* function.

Update

[v2.0] - The issue has been resolved as the function was deleted from the new commit, “097bd2968cc7e0ea39ac1a1b47240755da08e974”.

⚠ **MAJOR**

chainId is variable arbitrarily. (Found - v.1.0) (Resolved - v.2.0)

```
function setChainId(uint _chainId) external {
    chainId = _chainId;
}
```

[https://github.com/HAECHI-LABS/audit-boomco/blob/800fe446e421cfe389da22ccbf4c8876967d43be/contracts/DAO/Gov.sol#L320-L322]

Issue

The *Gov#setChainId()* is callable by anyone and changes the chainId variable of contracts. Because anyone can call the function, the consistency of the contract's chainId variable cannot be guaranteed.

If so, even if the contract concerned is actually running on the same chain, it is possible to conduct DoS attacks by changing the chainId through the *Gov#setChainId()* function to prevent the *Gov#delegateBySig()* from running normally. In extreme cases, replay attacks are also possible, using signatures issued on other chains.

Recommendation

It is recommended to delete the *Gov#setChainId()* function because the chainId values can be received by EVM assembly.

Update

[v2.0] - The issue has been resolved as the function was deleted from the new commit, “097bd2968cc7e0ea39ac1a1b47240755da08e974”.

MINOR

ERC165's interface id shows an incorrect value. (Acknowledged - v.2.0)

```
contract Interface {
   /*
    * bytes4(keccak256('supportsInterface(bytes4)')) == 0x01ffc9a7
    */
   bytes4 private constant _INTERFACE_ID_ERC165 = 0x36372b07;


   /**
    * @dev 지원 여부에 대한 인터페이스 ID의 매핑
Mapping of interface ids to whether or not it's supported.
    */
   mapping(bytes4 => bool) private _supportedInterfaces;
```

[https://github.com/HAECHI-LABS/audit-boomco/blob/800fe446e421cfe389da22ccbf4c8876967d43be/contracts/Utils/Interface.sol#L5]

Issue

The _INTERFACE_ID_ERC165 variable, which is defined in the Interface contract, actually points to the interface id of ERC20. Thus, if an outsider intends to check whether the interface contract is supported, incorrect information may be provided.

Recommendation

We advise changing the _INTERFACE_ID_ERC165 variable to an appropriate value.

Update

The Boomco team answered that the team is aware of this issue and the contract is no longer in use and has been deleted, so the issue status has been changed to Acknowledged.

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 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.

```
Gov
  #constructor()
    ✔ should set name properly (69ms)
    ✔ should set symbol properly
    ✔ should set decimals properly
    ✔ should set initial supply properly
    ✔ supports ERC20 interfaces
    1) supports ERC165 interfaces
  ERC20 Spec
    #transfer()
      ✔ should fail if recipient is ZERO_ADDRESS (62ms)
      ✔ should fail if sender's amount is lower than balance (41ms)
      when succeeded
        ✔ sender's balance should decrease
        ✔ recipient's balance should increase
        ✔ should emit Transfer event
    #transferFrom()
      ✔ should fail if sender is ZERO_ADDRESS
      ✔ should fail if recipient is ZERO_ADDRESS
      ✔ should fail if sender's amount is lower than transfer amount
      ✔ should fail if allowance is lower than transfer amount
      ✔ should fail even if try to transfer sender's token without approval process
      when succeeded
        ✔ sender's balance should decrease
        ✔ recipient's balance should increase
        ✔ should emit Transfer event
        ✔ allowance should decrease
        ✔ should emit Approval event
    #approve()
      valid case
        ✔ allowance should set appropriately
        ✔ should emit Approval event
    #approveForSpending()
      ✔ should fail if msg.sender is not committer
      ✔ should fail if contract is destroyed (38ms)
    valid case
      ✔ allowance should set appropriately
      ✔ should emit Approval event
Gov spec
  #delegate()
    Delegation: address(0) -> address(0)
      ✔ Nothing happens
    Delegation: address(0) -> delegatee
      ✔ delegatee Votes changes
      ✔ should emit DelegateChanged Event
      ✔ should emit DelegateVotesChanged Event
    Delegation: nonzero delegatee -> new nonzero delegatee
      ✔ delegatee Votes change
      ✔ should emit DelegateChanged Event
      ✔ should emit DelegateVotesChanged Event - previous delegatee
      ✔ should emit DelegateVotesChanged Event - new delegatee
    Delegation: nonzero delegatee -> address(0)
      ✔ delegatee Votes change
      ✔ should emit DelegateChanged Event
      ✔ should emit DelegateVotesChanged Event
    Delegation: multiple delegator set same delegatee
      ✔ delegatee Votes change
      ✔ should emit DelegateChanged Event
      ✔ should emit DelegateVotesChanged Event
  #getPriorVotes()
    ✔ should fail if blockNumber is not finalized
    ✔ should return 0 if account does not have any checkpoint
    valid case - numCheckpoints == 1
      ✔ return the appropriate number of votes (141ms)
    valid case - numCheckpoints > 1
      ✔ return the appropriate number of votes (2984ms)
  #delegateBySig()
    ✔ should fail if nonce mismatch
    ✔ should fail if signature expired
    valid case
      ✔ delegate success
      ✔ should emit DelegateChanged event
      ✔ should emit DelegateVotesChanged event
```

End of Document