HAECHI AUDIT

LOKA

Smart Contract Security Analysis

Published on : Dec 13, 2021

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

LOKA

Security Report Published by HAECHI AUDIT
v1.0 Dec 06, 2021
v2.0 Dec 13, 2021

Auditor : Hoon Won

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	-	-	-	-	-
Major	1	1	-	-	All issues resolved
Minor	1	-	-	-	-
Tips	2	2	-	-	All issues resolved

TABLE OF CONTENTS

2 Issues (0 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 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 Claimant smart contract created by LOKA team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by LOKA 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 LOKA 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/LOKA-audit/blob/master/contracts/LOKA.sol). The last commit of the code used for this Audit is "0ccafdca674b94f2b9638fbcacf1efbb35fd5011".

Issues HAECHI AUDIT found 0 critical issues, 1 major issue, and 1 minor issue. There are 2 tips explained that would improve the code's usability or efficiency upon modification.

update [v.2.0] In the new commit, 697100a2dd463cfa9bbee5c41b06bb55ebf839fc, 1 major issue and 2 tips have been modified.

Severity	Issue	Status
⚠ **MAJOR**	Membership#transferOwner() function works abnormally.	(Found - v1.0) (Resolved - v2.0)
MINOR	When LOKA#take() function is called, the user may not take the assigned token.	(Found - v1.0)
💡 **TIPS**	There are missing Events.	(Found - v1.0) (Resolved - v2.0)
💡 **TIPS**	There are functions not in use.	(Found - v1.0) (Resolved - v2.0)

OVERVIEW

Contracts subject to audit

- ERC20
- ERC20Capped
- Ownable
- Membership
- LOKA
- Claimant

FINDINGS

⚠ MAJOR

Membership#transferOwner() function works abnormally.

(Found - v.1.0) (Resolved - v.2.0)

```
598   function transferOwner(address newOwner) public onlyOwner {
599       address preOwner = owner;
600       owner = newOwner;
601       emit OwnerTransferred(preOwner, newOwner);
602       setMembership(preOwner, 0);
603       setMembership(newOwner, 1);
604   }
605
606  function setMembership(address key, uint256 level) public onlyOwner {
607       membership[key] = level;
608       emit MembershipChanged(key, level);
609   }
610
611   modifier onlyOwner() {
612       require(isOwner(), "Membership : caller is not the owner");
613       _;
614   }
615
616   function isOwner() public view returns (bool) {
617       return _msgSender() == owner;
618   }
```

[https://github.com/HAECHI-LABS/LOKA-audit/blob/master/contracts/LOKA.sol#L598-L618]

Issue

Membership#transferOwner() the owner by adjusting the memberships of preOwner and newOwner after changing the internal *Membership#owner* variable with input newOwner.

However, at the time when *Membership#setMembership()* function is called in *LOKA.sol:602-603*, *Membership#owner* has already been replaced. Thus, the preOwner corresponding to msg.sender at that point cannot pass the onlyOwner modifier of *Membership#setMembership()* function. Accordingly, a call of *Membership#transferOwner()* is always reverted.

Recommendation

It is recommended to modify the logic appropriately so that the *Membership#owner* variable update occurs after the adjustment of the memberships of preOwner and newOwner.

Update

[v2.0] - The problem has been resolved by fixing the logic so that the *Membership#owner* variable update occurs after the adjustment of the membership of preOwner and newOwner.

MINOR

When LOKA#take() function is called, the user may not take the assigned token.

(Found - v.1.0)

```
663   mapping(address ⇒ uint256) private _allocates;
664   function _allocate(address addr, uint256 amount) internal virtual {
665       _allocates[addr] += amount;
666   }
667   function allocate(address addr, uint256 amount) public onlyOwner {
668       _allocate(addr, amount);
669   }
670   function allocateOf(address account) public view virtual returns (uint256) {
671       return _allocates[account];
672   }
673   function take(address addr, uint256 amount) public {
674       address sender = _msgSender();
675       require(_allocates[sender]⩾amount, "TokenVesting: No takable amount");
676       _allocates[sender] -= amount;
677       _mint(addr, amount);
678   }
```

[https://github.com/HAECHI-LABS/LOKA-audit/blob/master/contracts/LOKA.sol#L663-L678]

Issue

LOKA#take() function mints the token allocated from *LOKA#allocate()* function in advance by the amount to the addr input as a parameter.
However, when calling *LOKA#allocate()* function, it does not check whether totalSupply + amount is less than*ERC20Capped#_cap*. Thus, at the time when *LOKA#take()* function is called, if totalSupply + amount exceeds *ERC20Capped#_cap* , the call becomes reverted. Accordingly, a user who is assigned a certain amount of via *LOKA#allocate()* function may be unable to take the token via *LOKA#take()*.

Acknowledgement

If the implementation is as intended, no modification is necessary.

💡 **TIPS**

There are missing Events.

(Found - v.1.0) (Resolved - v.2.0)

The following list shows the function missing Event.

Function	Expected Event	Emitted Event	Omitted Event
mint	Transfer, Mint	Transfer	Mint

Without Event, it is difficult to identify in real-time whether correct values are recorded on the blockchain. In this case, it becomes problematic to determine whether the corresponding value has been changed in the application and whether the corresponding function has been called.

Thus, we recommended adding Events corresponding to the change occurring in the function.

Update

[v2.0] - The missing Event has been added.

💡 TIPS

There are functions not in use.

(Found - v.1.0) (Resolved - v.2.0)

```
125    function _msgData() internal view virtual returns (bytes calldata) {
126        return msg.data;
127    }
```

[https://github.com/HAECHI-LABS/LOKA-audit/blob/master/contracts/LOKA.sol#L125-L127]

```
397   function _burn(address account, uint256 amount) internal virtual {
398       require(account != address(0), "ERC20: burn from the zero address");
399
400       _beforeTokenTransfer(account, address(0), amount);
401
402       uint256 accountBalance = _balances[account];
403       require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
404       unchecked {
405           _balances[account] = accountBalance - amount;
406       }
407       _totalSupply -= amount;
408
409       emit Transfer(account, address(0), amount);
410
411       _afterTokenTransfer(account, address(0), amount);
412   }
```

[https://github.com/HAECHI-LABS/LOKA-audit/blob/master/contracts/LOKA.sol#L397-L412]

Issue

Context#_msgData() function and *ERC20#_burn()* function are not in use.

Update

[v2.0] - The unused *Context#_msgData()* function and *ERC20#_burn()* function have been deleted.

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 net . 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.

```
Claimant
  #constructor()
    ✓ should set LOKA token properly
    ✓ LOKA token contract should be set to admin
  #initialize()
    ✓ should fail if msg.sender is not admin
    valid case
      ✓ Claimant contract should be initialized properly
  #register()
    ✓ should fail if msg.sender is not admin
    valid case
      ✓ account should be set to claimant
  #claim()
    ✓ should fail if msg.sender is not claimant
    ✓ should fail if current time is less than started
    ✓ should fail if msg.sender has no claimable amount (45ms)
    valid case
      ✓ case: t = 0
      ✓ case: t = 0 + 9
      ✓ case: t = 0 + 9 + 4 (41ms)
      ✓ case: t = 0 + 9 + 4 + 10 (sequence > times) (55ms)
      ✓ case: t = 0 + 9 + 4 + 10 + 10 (sequence > times) (64ms)
  #totalClaimed()
    ✓ should return claimed properly
  #getStarted()
    ✓ should return started properly
  #getTerm()
    ✓ should return term properly
  #getBlockTimestamp()
    ✓ should return timestamp properly

Membership
  #constructor()
    ✓ should set msg.sender to owner
    ✓ should set msg.sender to admin
```

```
  #transferOwner()
    ✓ should fail if msg.sender is not owner
    valid case
      1) owner should be nonOwner
      2) owner should be nonAdmin
      3) newOwner should be owner
      4) newOwner should be admin
      5) should emit OwnerTransferred event
  #setMembership()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ membership should be set properly
      ✓ should emit MembershipChanged event

Ownable
  #constructor()
    ✓ should set msg.sender to owner
  #transferOwnership()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if try to transfer ownership to AddressZero
    valid case
      ✓ should change owner to newOwner
      ✓ should emit OwnershipTransferred event
  #renounceOwnership()
    ✓ should fail if msg.sender is not owner
    valid case
      ✓ should change owner to AddressZero
      ✓ should emit OwnershipTransferred event

LOKA
  #constructor()
    ✓ should fail if cap is zero
    ✓ should set name properly
    ✓ should set symbol properly
    ✓ should set decimals properly
    ✓ should set initial supply properly
  ERC20 Spec
    #transfer()
      ✓ should fail if recipient is ZERO_ADDRESS
      ✓ should fail if sender's amount is lower than balance
      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 approve 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()
    ✓ should fail if spender is ZERO_ADDRESS
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #increaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if overflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
  #decreaseAllowance()
    ✓ should fail if spender is ZERO_ADDRESS
    ✓ should fail if overflows
    valid case
      ✓ allowance should set appropriately
      ✓ should emit Approval event
ERC20 Mintable spec
  #mint()
    ✓ should fail if msg.sender is not minter
    ✓ should fail if try to mint more than cap
    ✓ should fail if try to mint to ZERO_ADDRESS
    valid case
      ✓ receiver's amount should increase
      ✓ totalSupply should increase
      ✓ should emit Transfer event
      6) should emit Mint event
LOKA Spec
  #allocate()
    ✓ should fail if msg.sender is not owner
    7) should fail if totalSupply + amount is greater than cap
    valid case
```

```
      ✓ accounts' allocates should increase
  #take()
    ✓ should fail if msg.sender has not takable amount
    ✓ should fail if totalSupply + amount is greater than cap
    valid case
      ✓ accounts' balance should increase
  #retain()
    ✓ should fail if msg.sender is not owner
    ✓ should fail if totalSupply + amount is greater than cap
    ✓ should fail if address is not contract
    valid case
      ✓ Claimant contract's LOKA balance should increase
      ✓ Claimant contract should be initialized properly
Unused
  ✓ #_msgData()
  ✓ #_burn()
```

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/					
LOKA.sol	100	100	100	100	

[Table 1] Test Case Coverage

End of Document