17th MAY 2019

SMART CONTRACT AUDIT REPORT

version 2.0.0

ERC20 Smart Contract Security Audit and General Analysis

HAECHI LABS

Table of Contents

4 Issues (2 Critical, 0 Major, 2 Minor) Found

01. Introduction

This report was written to provide a security audit for the TEMCO smart contract, designed by team TEMCO. HAECHI LABS conducted the audit focusing on whether the TEMCO team's smart contract is designed and implemented in accordance with publicly released information and whether it has any security vulnerabilities.

The code used for the audit can be found at "HAECHI/audit-temco" Github storage(https://github.com/HAECHI-LABS/audit-temco/tree/master/contracts). The last commit used for the audit was "06a3e4ce660c83e8438f1b4ff8b2700183e961a8".

02. Summary

The TEMCO team implemented an ERC20 smart contract with the following features:

- Lock
- Mint
- Burn

HAECHI LABS found 2 Critical Issues, 0 Major Issues, and 2 Minor Issues; we also included 2 tips that could help improve the code's usability and efficiency.

Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage. Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed. Minor issues are some potential risks that require some degree of modification. HAECHI LABS advises addressing all the issues found in this report.

Updated

19.05.23 [v.2.0] - The Ethereum transaction "0xb53eb30bc5f1f85d02e79c6955c61aaa3235a86b7bcb2288103d8557b8ae24e1" no longer causes the 2 Critical Issues and 1 Minor Issue.

03. Contracts subject to audit

- ERC20
- Lockable
- Ownable
- SafeMath
- TemcoToken

04. About HAECHI LABS

HAECHI LABS is a leading tech company within the blockchain industry based on its self-developed blockchain technology solutions and R&D capacity. HAECHI LABS provides solutions essential to developing blockchain based services.

Their most prominent services and solutions include a smart contract security auditing service, an open-source smart contract development/deployment/testing CLI tool called VVISP, and a middleware that helps with instant reading and analyzing blockchain data 'Query Layer'(Tentative).

HAECHI LABS' current client lists range from major companies and startups - both domestic and abroad - such as SK Telecom and Kakao's blockchain subsidiary(Ground X) to global cryptocurrency exchange institutes such as Bit-Z, Coinall(OKEx), KuCoin, Liquid, CPDAX, and Huobi Korea.

It is HAECHI LABS' mission to provide top-of-the-shelf- solutions and services so that eventually anyone building a blockchain-based system will come to use our solutions.

Contact : hello@haechi.io
Website : https://haechi.io

05. Issues Found

The issues found are classified as CRITICAL, MAJOR, or MINOR, according to their severity.

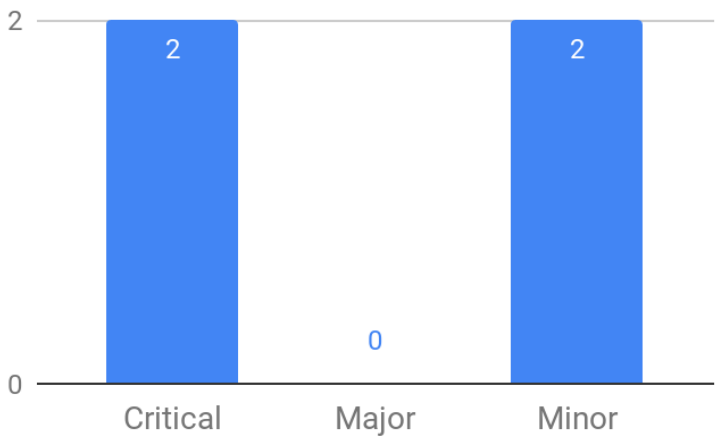
CRITICAL Critical issues are security vulnerabilities that MUST be addressed in order to prevent widespread and massive damage.

MAJOR Major issues contain security vulnerabilities or have faulty implementation issues and need to be fixed.

MINOR Minor issues are some potential risks that require some degree of modification.

HAECHI LABS strongly advises that TEMCO addresses all the issues found in this report.

The following explanations of each issue will use a {File name}#{Line number},{Contract name}#{Function/Variable name} format to refer to specific codes. For example, *Sample.sol:20* refers to the 20th line of the Sample.sol file, and *Sample#fallback()* refers to the fallback() function of the Sample contract.



[Chart 1] Issue Stats

Issues by severity level

Severity	Issue	Status
CRITICAL	*TemcoToken#mintTo()* can *burn* tokens. [Unintended Behavior]	**Resolved**
CRITICAL	Owner can mint limitlessly. [Unintended Behavior]	**Resolved**
MINOR	*TemcoToken#mintTo()* triggers an incorrect Event. [Unintended Behavior]	**Resolved**
MINOR	In *TemcoToken#transferFrom()*, when `_from` is locked, sending fails but the transaction seems successful. [Unintended Behavior]	Open

CRITICAL : *TemcoToken#mintTo()* can *burn* tokens. [Unintended Behavior] (Found - v.1.0) (Resolved - v.2.0)

CRITICAL

```
209     function mintTo(address _from, address _to, uint256 _amount) onlyOwner canMint e
210         require(_from != address(0)  && _to != address(0) && _amount > 0);
211         balances[_from] = balances[_from].sub(_amount);
212         balances[_to] = balances[_to].add(_amount);
213         emit Mint(_to, _amount);
214         emit Transfer(address(0), _to, _amount);
215         return true;
216     }
```

(TemcoToken.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/TemcoToken.sol#L209-L216)

Problem Statement

TemcoToken#mintTo() receives *_from*, *_to*, *_amount* as parameters. *TemcoToken.sol:211* subtracts *_amount* from *balances[_from]*. With this logic, *owner* can remove tokens without user approval.

Recommendation

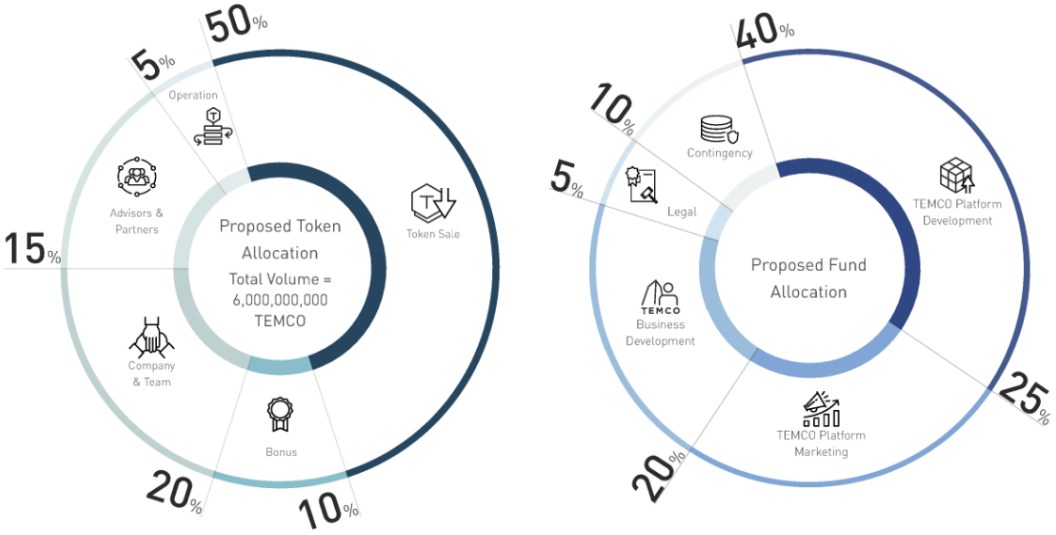
Delete the *_from* parameter and delete the token removing line.

Updated

19.05.23 [v.2.0] - *TemcoToken#mintTo()* can no longer be called upon as Ethereum transaction "0xb53eb30bc5f1f85d02e79c6955c61aaa3235a86b7bcb2288103d8557b8ae24e1" calls *TemcoToken#finishMinting()* instead.

CRITICAL : owner can mint limitlessly. [Unintended Behavior] (Found - v.1.0) (Resolved - v.2.0)

CRITICAL



[Figure 1] Distribution of TemcoToken - https://docsend.com/view/49j5ngu

Problem Statement

Upon calling *TemcoToken#mint()*, *mint* can be executed regardless of *totalSupply*. TEMCO's whitepaper states that 6,000,000,000 is the total volume of tokens, but this maximum is not implemented in the smart contract.

Recommendation

Check to see if *totalSupply* after executing *mint* exceeds 6,000,000,000.

Updated

19.05.23 [v.2.0] - *TemcoToken#mintTo()* can no longer be called upon as Ethereum transaction "0xb53eb30bc5f1f85d02e79c6955c61aaa3235a86b7bcb2288103d8557b8ae24e1" calls *TemcoToken#finishMinting()* instead.

MINOR : *TemcoToken#mintTo()* triggers an incorrect Event. [Unintended Behavior] (Found - v.1.0) (Resolved - v.2.0)

MINOR

```
209     function mintTo(address _from, address _to, uint256 _amount) onlyOwner canMint e
210         require(_from != address(0)  && _to != address(0) && _amount > 0);
211         balances[_from] = balances[_from].sub(_amount);
212         balances[_to] = balances[_to].add(_amount);
213         emit Mint(_to, _amount);
214         emit Transfer(address(0), _to, _amount);
215         return true;
216     }
```

(TemcoToken.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/TemcoToken.sol#L209-L216)

Problem Statement

TemcoToken#mintTo() is a function that moves *_value* amount of tokens from *_from* to *_to* . However, among the events triggered, there is one event in *Transfer* that moves *_value* from address(0) to *_to*. Considering most ethereum block explorers determine token transfers based on the *Transfer* event, the ethereum block explorer might consider it an incorrect transfer of assets.

Recommendation

We recommend taking measures suggested for fixing the Critical Issue in *TemcoToken#mintTo()*. If not, we suggest changing to *Transfer(_from,_to,_amount)*.

Updated

19.05.23 [v.2.0] - *TemcoToken#mintTo()* can no longer be called upon as Ethereum transaction "0xb53eb30bc5f1f85d02e79c6955c61aaa3235a86b7bcb2288103d8557b8ae24e1" calls *TemcoToken#finishMinting()* instead.

MINOR : In *TemcoToken#transferFrom()*, when `_from` is locked, sending fails but the transaction seems successful. [Unintended Behavior] (Found - v.1.0)

MINOR

```
80      function transferFrom(address _from, address _to, uint256 _value) public returns
81          require(_to != address(0));
82          require(_to != address(this));
83          require(_value <= balances[_from]);
84          require(_value <= allowed[_from][msg.sender]);
85          if(nolockedUp(_from) == false){
86              return false;
87          }
88          balances[_from] = balances[_from].sub(_value);
89          balances[_to] = balances[_to].add(_value);
90          allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
91          emit Transfer(_from, _to, _value);
92          return true;
93      }
```

(TemcoToken.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/TemcoToken.sol#L80-L93)

Problem Statement

TemcoToken.sol:85 is designed using the *if* statement to return *false* if the given condition is *false*. In this case, ethereum protocols classify the transaction as successful, and therefore the user will be under the false impression of a successful transaction.

Recommendation

We suggest using the *require()* statement so that if the given condition is *false*, then running *revert*.

06. Tips

TIPS : Unused Variable

TIPS

```
12  contract Lockable is Ownable {
13
14      /**
15      * @dev hold lock up address and duration
16      */
17      mapping(address => uint256) public lockedUp;
18
19      uint public nowTime;
20
21      constructor () public {
22          nowTime = now;
23      }
24
25      /**
```

(Lockable.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/Lockable.sol#L22)

The nowTime variable in *Lockable* is not a variable used in the contract nor its inheriting contracts. It creates unnecessary gas costs upon deployment. We suggest deleting the variable.

TIPS : Event triggers are omitted.

TIPS

```
45      /**
46      * @dev add lock up investor to mapping
47      * @param _investor lock up address
48      * @param _duration lock up period. unit is days
49      */
50      function addLockUp(address _investor, uint _duration ) onlyOwner public {
51          require(_investor != address(0) && _duration > 0);
52          lockedUp[_investor] = now + _duration * 1 days;
53      }
54
```

(Lockable.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/Lockable.sol#L50)

```
55      /**
56      * @dev remove lock up address from mapping
57      * @param _investor lock up address to be removed from mapping
58      */
59      function removeLockUp(address _investor ) onlyOwner public {
60          require(_investor != address(0));
61          delete lockedUp[_investor];
62      }
63
64
```

(Lockable.sol - https://github.com/HAECHI-LABS/audit-temco/blob/06a3e4ce660c83e8438f1b4ff8b2700183e961a8/contracts/Lockable.sol#L59)

Lockable#addLockUp(), *Lockable#removeLockUp()* functions do not trigger any events. These do not harm the smart contract's security, but when no events are triggered, it becomes difficult to read whether the function has been called upon. . We recommend adding an appropriate event trigger to each function.

07. Test Results

The following are the results of a unit test that covers the major logics of the smart contract under audit. The parts in red contain issues and therefore have failed the test.

```
Contract: ERC20
  #totalSupply()
    ✓ should returns the total amount of tokens
  #balanceOf()
    ✓ should return the total amount of tokens
  #transfer()
    ✓ should revert when the sender does not have enough balance (37759 gas)
    ✓ should transfer the requested amount (65101 gas)
    ✓ should emit a transfer event (65101 gas)
    ✓ should revert when the recipient is the zero address (31362 gas)
    ✓ should revert when sender is locked (78626 gas)
    ✓ should revert when the recipient is the contract itself (35073 gas)
  #transferFrom()
    ✓ should transfer the requested amount (118201 gas)
    ✓ should decrease the spender allowance (118201 gas)
    ✓ should emit a transfer event (118201 gas)
    ✓ should revert when the initial holder does not have enough balance (86276 gas)
    ✓ should revert when the spender does not have enough approved balance (89109 gas)
    ✓ should revert when the initial holder does not have enough balance (32667 gas)
    ✓ should revert when the recipient is the zero address (79806 gas)
    1) should revert when sender is locked
    ✓ should revert when the recipient is the contract itself (29905 gas)
  #approve()
    ✓ should emit an approval event (53609 gas)
    ✓ should approve the requested amount (53609 gas)
    ✓ should approve the requested amount and replaces the previous one (91834 gas)
    ✓ should revert when msg.sender is locked (78252 gas)
  #decreaseApproval()
    ✓ should emit an approval event (85225 gas)
    ✓ should decrease the spender allowance subtracting the requested amount (100353 gas)
    ✓ should set the spender allowance to 0 when _substractedValue is bigger than allowed amount
(81242 gas)
    ✓ should set the allowance to zero when all allowance is removed (85225 gas)
  #increaseApproval()
    ✓ should emit an approval event (60013 gas)
    ✓ should approve the requested amount when the previous approved amount was zero (60013 gas)
```

```
    ✓ should increase the spender allowance adding the requested amount (98238 gas)
  #mint()
    ✓ should revert if msg.sender is not owner (25966 gas)
    ✓ should revert a null account (32031 gas)
    ✓ should increment totalSupply
    ✓ should increment recipient balance
    ✓ should emit Transfer event
    ✓ should revert if called after finishMinting() (69103 gas)
    2) should revert if amount is bigger than 6,000,000,000 * 10 ** decimal
  #mintTo()
    ✓ should revert if msg.sender is not owner (27653 gas)
    ✓ should revert a null account (33738 gas)
    3) should increment totalSupply
    ✓ should increment recipient balance
    ✓ should emit Transfer event
    ✓ should revert if called after finishMinting() (70790 gas)
  #finishMinting()
    ✓ should revert if msg.sender is not owner (24389 gas)
    ✓ should revert if called twice (67526 gas)
    ✓ should emit MintFinished event (39408 gas)
  #burn()
    ✓ should revert if msg.sender is not owner (24499 gas)
    ✓ should revert if amount is bigger than balance (28733 gas)
    ✓ should decrease totalSupply
    ✓ should increment recipient balance
    ✓ should emit Burn event
    ✓ should emit Transfer event

Contract: Lockable
  #addLockUp()
    ✓ should be reverted if msg.sender is not owner (25933 gas)
    ✓ should be reverted if _investor is zero address (28203 gas)
    ✓ should be reverted if _duration is zero (29366 gas)
    4) should emit Locked event for successful transaction
    ✓ should change lockedUp status (51386 gas)
  #removeLockUp()
    ✓ should revert if msg.sender is not owner (25627 gas)
    ✓ should revert if _investor is zero address (27871 gas)
    5) should emit RemoveLockUp event for successful transaction
    ✓ should change lockedUp to zero after valid case (20430 gas)

Contract: Ownable
```

```
  #addOwnership()
    ✓ should fail if msg.sender is not owner (25583 gas)
    ✓ should fail for attempt to add zero address as owner (27893 gas)
    ✓ should emit OwnershipAdded event and owner[newOwner] should be true (53601 gas)
  #removeOwnership()
    ✓ should fail if msg.sender is not owner (25539 gas)
    ✓ should fail for attempt to add zero address as owner (27849 gas)
    ✓ should fail for attempt to add zero address as owner (31557 gas)
    ✓ should emit OwnershipRemoved event and owner[newOwner] should be false (25397 gas)


Contract: SafeMath
  add
    ✓ adds correctly
    ✓ reverts on addition overflow
  sub
    ✓ subtracts correctly
    ✓ reverts if subtraction result would be negative
  mul
    ✓ multiplies correctly
    ✓ multiplies by zero correctly
    ✓ reverts on multiplication overflow
  div
    ✓ divides correctly
    ✓ divides zero correctly
    ✓ returns complete number result on non-even division
    ✓ reverts on divison by zero
```

Gas					Block limit: 17592186044415 gas
Methods		5 gwei/gas		320476.04 krw/eth	
Contract	Method	Min	Max	Avg	# calls	krw (avg)
Lockable	addLockUp	-	-	51386	1	82.34
Lockable	removeLockUp	-	-	20430	1	32.74
Ownable	addOwnership	-	-	53601	1	85.89
Ownable	removeOwner	-	-	25397	1	40.70
TemcoToken	addLockUp	-	-	51795	2	83.00
TemcoToken	approve	38609	53737	52566	15	84.23
TemcoToken	decreaseApproval	27633	46744	34402	4	55.13
TemcoToken	finishMinting	-	-	39408	4	63.15
TemcoToken	increaseApproval	45013	60013	55013	3	88.15
TemcoToken	transfer	-	-	65101	2	104.32
TemcoToken	transferFrom	-	-	64592	3	103.50
Deployments				% of limit	
Lockable		-	-	2110826	0 %	3382.35
Ownable		-	-	1075665	0 %	1723.62
SafeMathMock		-	-	1345620	0 %	2156.19
TemcoToken		-	-	8692807	0 %	13929.18

[Figure 2] Ether Gas Report

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
ERC20.sol	100	100	100	100	
Lockable.sol	100	100	100	100	
Ownable.sol	100	100	100	100	
SafeMath.sol	100	100	100	100	
TemcoToken.sol	100	100	100	100	
contracts/mock/	100	100	100	100	
SafeMathMock.sol	100	100	100	100	
All files	100	100	100	100	

[Figure 3] Test Case Coverage

08. Disclaimer

This report is not an advice on investment, nor does it guarantee adequacy of a business model and/or a bug-free code. This report should be used only to discuss known technical problems. The code may include problems on Ethereum and/or Solidity that are not included in this report. It will be necessary to resolve addressed issues and conduct thorough tests to ensure the safety of the smart contract.