



Smart Contract Security Audit Report



The SlowMist Security Team received the BaaS team's application for smart contract security audit of the BAAS Token on November 18, 2019. The following are the details and results of this smart contract security audit:

Token name :

BAAS

The File Name and HASH(SHA256) :

BaaS_v2.0.sol: 797889bbd5a717b39e15472f49e7fb458e1b22f0ee3fa010ca4ddf50e3d6dd6a

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Race Conditions Audit	-	Passed
3	Authority Control Audit	Permission vulnerability audit	Passed
		Excessive auditing authority	Some Risks
4	Safety Design Audit	Zeppelin module safe use	Passed
		Compiler version security	Passed
		Hard-coded address security	Passed
		Fallback function safe use	Passed
		Show coding security	Passed
		Function return value security	Passed
		Call function security	Passed
5	Denial of Service Audit	-	Passed
6	Gas Optimization Audit	-	Passed
7	Design Logic Audit	-	Passed
8	"False Deposit" vulnerability Audit	-	Passed
9	Malicious Event Log Audit	-	Passed
10	Scoping and Declarations Audit	-	Passed

11	Replay Attack Audit	ECDSA's Signature Replay Audit	Passed
12	Uninitialized Storage Pointers Audit	-	Passed
13	Arithmetic Accuracy Deviation Audit	-	Passed

Audit Result : Passed

Audit Number : 0X001911200003

Audit Date : November 20, 2019

Audit Team : SlowMist Security Team

(**Statement** : SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary: This is a token contract that contains the tokenVault section. SafeMath security Module is used, which is a recommend approach.

Some risks were found during the audit:

- 1、 The total amount of token can be changed, owner can arbitrarily mint tokens through mint function and burn tokens through burn function.**
- 2、 Owner can freeze and unfreeze arbitrary account through freezeAccount function and unfreezeAccount function.**
- 3、 Owner can arbitrarily lock any amount of token of any account through lock function.**
- 4、 The approveAndCall function does not restrict by the whenNotPaused modifier.**
- 5、 Lock function not set a limit for _releaseRate parameter , and will cause infinite loop during token transferring when it is set to 0 and result in failure in token transfer.**
- 6、 Lock function not set a limit for _termOfRound parameter , and will consume huge amount of gas during token transferring when it is set too low.**

The source code:

```
pragma solidity ^0.4.22;
```

```
//SlowMist// SafeMath security Module is used, which is a recommend approach
```

```
library SafeMath {

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;

        assert(c / a == b); //SlowMist// This kind of check is very good, can optimize Gas

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // assert(b > 0); // Solidity automatically throws when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {

        assert(b <= a); //SlowMist// This kind of check is very good, can optimize Gas

        return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;

        assert(c >= a); //SlowMist// This kind of check is very good, can optimize Gas

        return c;
    }
}

contract Ownable {
    address public owner;
    address public newOwner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    constructor() public {
        owner = msg.sender;
    }
}
```

```
    newOwner = address(0);
}

modifier onlyOwner() {
    require(msg.sender == owner);
    _;
}

modifier onlyNewOwner() {
    require(msg.sender != address(0));
    require(msg.sender == newOwner);
    _;
}
```

```
function transferOwnership(address _newOwner) public onlyOwner {
```

```
    require(_newOwner != address(0)); //SlowMist// This check is quite good in avoiding losing
```

control of the contract caused by user mistakes

```
    newOwner = _newOwner;
}

function acceptOwnership() public onlyNewOwner returns(bool) {
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
}
```

```
contract Pausable is Ownable {
    event Pause();
    event Unpause();
```

```
    bool public paused = false;
```

```
    modifier whenNotPaused() {
        require(!paused);
        _;
    }
```

```
    modifier whenPaused() {
        require(paused);
        _;
    }
```

//SlowMist// Suspending all transactions upon major abnormalities is a recommended approach.

```
function pause() onlyOwner whenNotPaused public {
    paused = true;
    emit Pause();
}

function unpause() onlyOwner whenPaused public {
    paused = false;
    emit Unpause();
}

contract ERC20 {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function allowance(address owner, address spender) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    function transferFrom(address from, address to, uint256 value) public returns (bool);
    function approve(address spender, uint256 value) public returns (bool);

    event Approval(address indexed owner, address indexed spender, uint256 value);
    event Transfer(address indexed from, address indexed to, uint256 value);
}

interface TokenRecipient {
    function receiveApproval(address _from, uint256 _value, address _token, bytes _extraData) external;
}

contract BaaSidToken is ERC20, Ownable, Pausable {

    uint128 internal MONTH = 30 * 24 * 3600; // 1 month

    using SafeMath for uint256;

    struct LockupInfo {
        uint256 releaseTime;
        uint256 termOfRound;
```

```
uint256 unlockAmountPerRound;
uint256 lockupBalance;
}

string public name;
string public symbol;
uint8 public decimals;
uint256 internal initialSupply;
uint256 internal totalSupply_;

mapping(address => uint256) internal balances;
mapping(address => bool) internal locks;
mapping(address => bool) public frozen;
mapping(address => mapping(address => uint256)) internal allowed;
mapping(address => LockupInfo) internal lockupInfo;

event Unlock(address indexed holder, uint256 value);
event Lock(address indexed holder, uint256 value);
event Burn(address indexed owner, uint256 value);
event Mint(uint256 value);
event Freeze(address indexed holder);
event Unfreeze(address indexed holder);

modifier notFrozen(address _holder) {
    require(!frozen[_holder]);
    _;
}

constructor() public {
    name = "BaaSId";
    symbol = "BAAS";
    decimals = 18;
    initialSupply = 10000000000;
    totalSupply_ = initialSupply * 10 ** uint(decimals);
    balances[owner] = totalSupply_;
    emit Transfer(address(0), owner, totalSupply_);
}

function () public payable {
    revert();
}
```

```
function totalSupply() public view returns (uint256) {  
    return totalSupply_;  
}
```

```
function transfer(address _to, uint256 _value) public whenNotPaused notFrozen(msg.sender) returns (bool) {  
    if (locks[msg.sender]) {  
        autoUnlock(msg.sender);  
    }
```

```
    require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake
```

leading to the loss of token during transfer

```
    require(_value <= balances[msg.sender]);
```

```
    // SafeMath.sub will throw if there is not enough balance.
```

```
    balances[msg.sender] = balances[msg.sender].sub(_value);
```

```
    balances[_to] = balances[_to].add(_value);
```

```
    emit Transfer(msg.sender, _to, _value);
```

```
    return true; //SlowMist// The return value conforms to the EIP20 specification
```

```
}
```

```
function balanceOf(address _holder) public view returns (uint256 balance) {  
    return balances[_holder] + lockupInfo[_holder].lockupBalance;  
}
```

```
function transferFrom(address _from, address _to, uint256 _value) public whenNotPaused notFrozen(_from) returns  
(bool) {
```

```
    if (locks[_from]) {  
        autoUnlock(_from);  
    }
```

```
    require(_to != address(0)); //SlowMist// This kind of check is very good, avoiding user mistake
```

leading to the loss of token during transfer

```
    require(_value <= balances[_from]);
```

```
    require(_value <= allowed[_from][msg.sender]);
```

```
    balances[_from] = balances[_from].sub(_value);
```

```
    balances[_to] = balances[_to].add(_value);
```

```
allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);  
emit Transfer(_from, _to, _value);
```

```
return true; //SlowMist// The return value conforms to the EIP20 specification
```

```
}
```

```
function approve(address _spender, uint256 _value) public whenNotPaused returns (bool) {  
    allowed[msg.sender][_spender] = _value;  
    emit Approval(msg.sender, _spender, _value);
```

```
return true; //SlowMist// The return value conforms to the EIP20 specification
```

```
}
```

```
//SlowMist// The approveAndCall function does not restrict by the whenNotPaused
```

modifier

```
function approveAndCall(address _spender, uint256 _value, bytes _extraData) public returns (bool success) {  
    require(isContract(_spender));  
    TokenRecipient spender = TokenRecipient(_spender);  
    if (approve(_spender, _value)) {  
        spender.receiveApproval(msg.sender, _value, this, _extraData);  
        return true;  
    }  
}
```

```
}
```

```
function allowance(address _holder, address _spender) public view returns (uint256) {  
    return allowed[_holder][_spender];  
}
```

```
//SlowMist// Not set a limit for _releaseRate parameter, and will cause infinite loop during
```

token transferring when it is set to 0 and and result in failure in token transfer

```
//SlowMist// Not set a limit for _termOfRound parameter, and will consume huge amount
```

of gas during token transferring when it is set too low

```
//SlowMist// Owner can arbitrarily lock tokens of any account
```

```
function lock(address _holder, uint256 _amount, uint256 _releaseStart, uint256 _termOfRound, uint256 _releaseRate)  
public onlyOwner returns (bool) {  
    require(locks[_holder] == false);  
    require(balances[_holder] >= _amount);
```

```
balances[_holder] = balances[_holder].sub(_amount);
lockupInfo[_holder] = LockupInfo(_releaseStart, _termOfRound, _amount.div(100).mul(_releaseRate), _amount);

locks[_holder] = true;

emit Lock(_holder, _amount);

return true;
}

function unlock(address _holder) public onlyOwner returns (bool) {
    require(locks[_holder] == true);
    uint256 releaseAmount = lockupInfo[_holder].lockupBalance;

    delete lockupInfo[_holder];
    locks[_holder] = false;

    emit Unlock(_holder, releaseAmount);
    balances[_holder] = balances[_holder].add(releaseAmount);

    return true;
}

//SlowMist// Owner can arbitrarily freeze account

function freezeAccount(address _holder) public onlyOwner returns (bool) {
    require(!frozen[_holder]);
    frozen[_holder] = true;
    emit Freeze(_holder);
    return true;
}

//SlowMist// Owner can arbitrarily unfreeze account

function unfreezeAccount(address _holder) public onlyOwner returns (bool) {
    require(frozen[_holder]);
    frozen[_holder] = false;
    emit Unfreeze(_holder);
    return true;
}

function getNowTime() public view returns(uint256) {
    return now;
}
```

```
function showLockState(address _holder) public view returns (bool, uint256, uint256, uint256, uint256) {
    return (locks[_holder], lockupInfo[_holder].lockupBalance, lockupInfo[_holder].releaseTime,
lockupInfo[_holder].termOfRound, lockupInfo[_holder].unlockAmountPerRound);
}

function distribute(address _to, uint256 _value) public onlyOwner returns (bool) {
    require(_to != address(0));
    require(_value <= balances[owner]);

    balances[owner] = balances[owner].sub(_value);
    balances[_to] = balances[_to].add(_value);
    emit Transfer(owner, _to, _value);
    return true;
}

function distributeWithLockup(address _to, uint256 _value, uint256 _releaseStart, uint256 _termOfRound, uint256
_releaseRate) public onlyOwner returns (bool) {
    distribute(_to, _value);
    lock(_to, _value, _releaseStart, _termOfRound, _releaseRate);
    return true;
}

function claimToken(ERC20 token, address _to, uint256 _value) public onlyOwner returns (bool) {
    token.transfer(_to, _value);
    return true;
}

//SlowMist// Owner can arbitrarily burn tokens

function burn(uint256 _value) public onlyOwner returns (bool success) {
    require(_value <= balances[msg.sender]);
    address burner = msg.sender;
    balances[burner] = balances[burner].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    emit Burn(burner, _value);
    return true;
}

//SlowMist// Owner can arbitrarily mint tokens

function mint( uint256 _amount) onlyOwner public returns (bool) {
    totalSupply_ = totalSupply_.add(_amount);
    balances[owner] = balances[owner].add(_amount);
}
```

```
emit Transfer(address(0), owner, _amount);
return true;
}

function isContract(address addr) internal view returns (bool) {
    uint size;
    assembly{size := extcodesize(addr)}
    return size > 0;
}

function autoUnlock(address _holder) internal returns (bool) {
    if (lockupInfo[_holder].releaseTime <= now) {
        return releaseTimeLock(_holder);
    }
    return false;
}

function releaseTimeLock(address _holder) internal returns(bool) {
    require(locks[_holder]);
    uint256 releaseAmount = 0;
    // If lock status of holder is finished, delete lockup info.

    for( ; lockupInfo[_holder].releaseTime <= now ; )
    {
        if (lockupInfo[_holder].lockupBalance <= lockupInfo[_holder].unlockAmountPerRound) {
            releaseAmount = releaseAmount.add(lockupInfo[_holder].lockupBalance);
            delete lockupInfo[_holder];
            locks[_holder] = false;
            break;
        } else {
            releaseAmount = releaseAmount.add(lockupInfo[_holder].unlockAmountPerRound);
            lockupInfo[_holder].lockupBalance =
lockupInfo[_holder].lockupBalance.sub(lockupInfo[_holder].unlockAmountPerRound);

            lockupInfo[_holder].releaseTime = lockupInfo[_holder].releaseTime.add(lockupInfo[_holder].termOfRound);
        }
    }

    emit Unlock(_holder, releaseAmount);
    balances[_holder] = balances[_holder].add(releaseAmount);
}
```

```
return true;  
}  
// TODO:  
  
}
```



Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
[@SlowMist_Team](https://twitter.com/SlowMist_Team)

WeChat Official Account

