Smart Contract Security Audit Report

The SlowMist Security Team received the Asia Influencer Platform team's application for smart contract security audit of the AIP on Nov. 18, 2020. The following are the details and results of this smart contract security audit:

Token name :

AIP

The Contract address :

0x97836C0CB03b03843A8d77152F5f15096522f98d

Link address :

https://etherscan.io/address/0x97836C0CB03b03843A8d77152F5f15096522f98d

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	Passed
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 : 0X002011240003

Audit Date : Nov. 24, 2020

Audit Team : SlowMist Security Team

Summary: This is a token contract that contains the tokenVault section. The total amount of contract tokens can be changed, the owner can burn his own tokens through the burn function. SafeMath security module is used, which is a recommend approach. The contract does not have the Overflow and the Race Conditions issue. During the audit, we found:

1. The owner can lock any user's tokens through the setLock function.

2. The owner can lock any user's tokens in batches through the setLockList function.

3. The owner can freeze his own tokens through the freeze function.

4. The owner can unfreeze his own tokens through the unfreeze function.

The source code:

```
//SlowMist// The contract does not have the Overflow and the Race Conditions issue
```

```
pragma solidity 0.4.25;

contract ERC20Basic {
    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function transfer(address   to, uint256 value) public returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
}

contract ERC20 is ERC20Basic {
    function allowance(address owner, address spender) public view returns (uint256);
    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);
}

contract DetailedERC20 is ERC20 {
    string public name;
    string public symbol;
    uint8 public decimals;

    constructor(string _name, string _symbol, uint8 _decimals) public {
        name = _name;
        symbol = _symbol;
        decimals = _decimals;
    }
}

contract BasicToken is ERC20Basic {
    using SafeMath for uint256;
    event Approval(address indexed owner, address indexed spender, uint256 value);
    mapping(address => uint256)   balances;
    uint256   _totalSupply;

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


    function transfer(address _to, uint256 _value) public returns (bool) {
```

```
        //SlowMist// This kind of check is very good, avoiding user mistake leading to the loss of token during transfer
        require(_to != address(0) && _value != 0 &&_value <= balances[msg.sender],"Please check the amount of transmission error and the amount you send.");
        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 _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }
}

contract ERC20Token is BasicToken, ERC20 {
    using SafeMath for uint256;
    event Approval(address indexed owner, address indexed spender, uint256 value);
    mapping (address => mapping (address => uint256))   allowed;
    mapping (address => uint256) public freezeOf;

    function approve(address _spender, uint256 _value) public returns (bool) {

        require(_value == 0 || allowed[msg.sender][_spender] == 0,"Please check the amount you want to approve.");
        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

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

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

    function increaseApproval(address _spender, uint256 _addedValue) public returns (bool success) {
        allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
```

```
    }

    function decreaseApproval(address _spender, uint256 _subtractedValue) public returns (bool success) {
        uint256 oldValue = allowed[msg.sender][_spender];
        if (_subtractedValue >= oldValue) {
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
}

contract Ownable {

    address public owner;
    mapping (address => bool) public admin;
    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

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

    modifier onlyOwner() {
        require(msg.sender == owner,"I am not the owner of the wallet.");
        _;
    }

    modifier onlyOwnerOrAdmin() {
        require(msg.sender == owner || admin[msg.sender] == true,"It is not the owner or manager wallet address.");
        _;
    }

    function transferOwnership(address newOwner) onlyOwner public {
        require(newOwner != address(0) && newOwner != owner && admin[newOwner] == true,"It must be the existing
manager wallet, not the existing owner's wallet.");
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
    }

    function setAdmin(address newAdmin) onlyOwner public {
```

```
        require(admin[newAdmin] != true && owner != newAdmin,"It is not an existing administrator wallet, and it must not be the owner wallet of the token.");
        admin[newAdmin] = true;
    }

    function unsetAdmin(address Admin) onlyOwner public {
        require(admin[Admin] != false && owner != Admin,"This is an existing admin wallet, it must not be a token holder wallet.");
        admin[Admin] = false;
    }

}

contract Pausable is Ownable {
    event Pause();
    event Unpause();
    bool public paused = false;

    modifier whenNotPaused() {
        require(!paused,"There is a pause.");
        _;
    }

    modifier whenPaused() {
        require(paused,"It is not 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();
    }

}
```

//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;
        require(c / a == b,"An error occurred in the calculation process");
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b !=0,"The number you want to divide must be non-zero.");
        uint256 c = a / b;
        require(c * b == a,"An error occurred in the calculation process");
        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a,"There are more to deduct.");
        return a - b;
    }

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a,"The number did not increase.");
        return c;
    }
}

contract BurnableToken is BasicToken, Ownable {

    event Burn(address indexed burner, uint256 amount);

    //SlowMist// The owner can burn his own tokens through the burn function

    function burn(uint256 _value) onlyOwner public {
        balances[msg.sender] = balances[msg.sender].sub(_value);
        _totalSupply = _totalSupply.sub(_value);
        emit Burn(msg.sender, _value);
        emit Transfer(msg.sender, address(0), _value);
    }


}
```

```
contract FreezeToken is BasicToken, Ownable {

    event Freezen(address indexed freezer, uint256 amount);
    event UnFreezen(address indexed freezer, uint256 amount);
    mapping (address => uint256) freezeOf;

    //SlowMist// The owner can freeze his own tokens through the freeze function

    function freeze(uint256 _value) onlyOwner public {
        balances[msg.sender] = balances[msg.sender].sub(_value);
        freezeOf[msg.sender] = freezeOf[msg.sender].add(_value);
        _totalSupply = _totalSupply.sub(_value);
        emit Freezen(msg.sender, _value);
    }

    //SlowMist// The owner can unfreeze his own tokens through the unfreeze function

    function unfreeze(uint256 _value) onlyOwner public {
        require(freezeOf[msg.sender] >= _value,"The number to be processed is more than the total amount and the number currently frozen.");
        balances[msg.sender] = balances[msg.sender].add(_value);
        freezeOf[msg.sender] = freezeOf[msg.sender].sub(_value);
        _totalSupply = _totalSupply.add(_value);
        emit Freezen(msg.sender, _value);
    }
}


contract AsiaInfluencerPlatform is BurnableToken,FreezeToken, DetailedERC20, ERC20Token,Pausable{
    using SafeMath for uint256;

    event Approval(address indexed owner, address indexed spender, uint256 value);
    event LockerChanged(address indexed owner, uint256 amount);
    mapping(address => uint) locker;

    string  private _symbol = "AIP";
    string  private _name = "Asia Influencer Platform";
    uint8   private _decimals = 18;
    uint256 private TOTAL_SUPPLY = 40*(10**8)*(10**uint256(_decimals));

    constructor() DetailedERC20(_name, _symbol, _decimals) public {
        _totalSupply = TOTAL_SUPPLY;
        balances[owner] = _totalSupply;
```

```
        emit Transfer(address(0x0), msg.sender, _totalSupply);
    }

    function transfer(address _to, uint256 _value)   public whenNotPaused returns (bool){
        require(balances[msg.sender].sub(_value) >= locker[msg.sender],"Attempting to send more than the locked number");
        return super.transfer(_to, _value);
    }

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

        require(_to > address(0) && _from > address(0),"Please check the address" );
        require(balances[_from] >= _value && allowed[_from][msg.sender] >= _value,"Please check the amount of transmission error and the amount you send.");
        require(balances[_from].sub(_value) >= locker[_from],"Attempting to send more than the locked number" );

        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 lockOf(address _address) public view returns (uint256 _locker) {
        return locker[_address];
    }
    //SlowMist// The owner can lock any user's tokens through the setLock function
    function setLock(address _address, uint256 _value) public onlyOwnerOrAdmin {
        require(_value <= _totalSupply &&_address != address(0),"It is the first wallet or attempted to lock an amount greater than the total holding.");
        locker[_address] = _value;
        emit LockerChanged(_address, _value);
    }
    //SlowMist// The owner can lock any user's tokens in batches through the setLockList function
    function setLockList(address[] _recipients, uint256[] _balances) public onlyOwnerOrAdmin{
```

```
        require(_recipients.length == _balances.length,"The number of wallet arrangements and the number of amounts are different.");

        for (uint i=0; i < _recipients.length; i++) {
            require(_recipients[i] != address(0),'Please check the address');

            locker[_recipients[i]] = _balances[i];
            emit LockerChanged(_recipients[i], _balances[i]);
        }
    }


    function transferList(address[] _recipients, uint256[] _balances) public onlyOwnerOrAdmin{
        require(_recipients.length == _balances.length,"The number of wallet arrangements and the number of amounts are different.");

        for (uint i=0; i < _recipients.length; i++) {
            balances[msg.sender] = balances[msg.sender].sub(_balances[i]);
            balances[_recipients[i]] = balances[_recipients[i]].add(_balances[i]);
            emit Transfer(msg.sender,_recipients[i],_balances[i]);
        }
    }


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

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

Github
https://github.com/slowmist