JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **SATT** (company/team) **Smart Advertising Transaction Token (SATT)** project smart contract code audit requirements on **2020/09/21.**

Project Name：Smart Advertising Transaction Token (SATT)

Smart Contract Address：
https://etherscan.io/address/0xdf49c9f599a0a9049d97cff34d0c30e468987389#code

Audit Number：20200912

Audit Date：20200921

Audit Category and Result:

Class	SubClass	Result(Pass/Not Pass)
Code programming	Integer overflow	Pass
	Race condition	Not Pass
	Logical flaw	Pass
	Denial of service	Pass
	Function parameter check	Not Pass
	Random number generation	Pass
	Compiler version	Pass
	Hardcoded address	Pass
	ERC20/ERC223 standard	Not Pass
	Code specification	Not Pass
Special service	Business risk	Pass
	Contract owner privileges	Pass
	"short address" attack	Pass
	"Fake recharge" attack	Pass
GAS optimization	-	Pass
Automated fuzzing	-	Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in this audit responsibility)

Audit Result：PASS

Auditor：John Wick Security Lab

(Disclaimer: The John Wick Security Lab issues this report based on the facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information

provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.)

Audit Details:

//JohnWick: 47L

The decimal point of this contract is 18, which is consistent with the decimal point of Ethereum's base currency `Ether(ETH)`, in line with the recommended practice.

//JohnWick: **[Low Risk]** 20L

This contract does not use the `SafeMath` library to avoid potential integer overflow issues, which is not in line with the recommended practice.

//JohnWick: **[Low Risk]** 20L

The function `transferOwnership(address payable newOwner)` does not check if `newOwner` is address(0). If the `owner` is set to address(0) by mistake, the contract will be out of control.

//JohnWick: **[Low Risk]** 48L

```
uint256 public constant totalSupply = 20000000000000000000000000000;
```

This way of writing is not conducive to improving the code readability of this smart contract, the code should be written as:

```
uint256 public constant totalSupply = 20000000000 * (10 ** uint256(decimals));
```

//JohnWick: **[Low Risk]** 103L

```
require(balanceOf[_to] + _value > balanceOf[_to]);
```

This will cause an exception to be thrown if the `_value` is 0, which need to change `>` to `>=`.

//JohnWick: **[Low Risk]** 118L

The function `approve(address _spender, uint256 _value)` has a race condition problem.

We recommend adding the following check code after 119L:

```
require(_value == 0 || allowance[msg.sender][_spender] == 0);
```

Or use the `increaseApproval` or `decreaseApproval` functions of the OpenZeppelin open source framework to achieve atomic increase or decrease `allowance[msg.sender][_spender]` to avoid this problem.

//JohnWick: **[Low Risk] 81L**

Function `transfer (address to, uint256 value, bytes memory data)` does not comply with the ERC223 standard. According to the standard, `_data` can be empty, but the implementation of this contract is that if `_data` is empty, the transfer transaction will not be triggered.

//JohnWick: 125L

`transferToken (address token,address to,uint256 val) public onlyOwner` function allows the contract `owner` to return the ERC20 token which was mistakenly transferred to this contract to the `to` address, which avoids the loss caused by the misoperation and conforms to the recommended practice.

Note: The line number of the code involved in the audit details is based on the verified contract source code uploaded by the project party at etherscan.io, which is also displayed as a backup in the `Smart Contract Source Code` section of this report.

Smart Contract Source Code:

```
/**
 *Submitted for verification at Etherscan.io on 2019-03-19
*/

pragma solidity ^0.5.6;

contract owned {
    address payable public owner;

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

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

    function transferOwnership(address payable newOwner) onlyOwner public {
        owner = newOwner;
    }
```

```
    function() external payable  {
    }

     function withdraw() onlyOwner public {
        owner.transfer(address(this).balance);
    }
}




interface ERC20 {
  function transfer(address receiver, uint256 value) external returns (bool ok);
}


interface ERC223Receiver {
    function tokenFallback(address _from, uint _value, bytes32 _data) external ;
}



contract SaTT is owned,ERC20 {

    uint8 public constant decimals = 18;
    uint256 public constant totalSupply = 20000000000000000000000000000; // 20
billions and 18 decimals
    string public constant symbol = "SATT";
    string public constant name = "Smart Advertising Transaction Token";



    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed tokenOwner, address indexed spender, uint256
tokens);


    constructor () public {
        balanceOf[msg.sender] = totalSupply;
    }
```

```
    function isContract(address _addr) internal view returns (bool is_contract)
{
     bytes32 hash;

     assembly {
          //retrieve the size of the code on target address, this needs assembly
          hash := extcodehash(_addr)
     }
     return                              (hash                              !=
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 && hash !=
bytes32(0));

   }

    function transfer(address to, uint256 value) public returns (bool success)
{
       _transfer(msg.sender, to, value);
       return true;
   }

    function transfer(address to, uint256 value,bytes memory  data) public
returns (bool success) {
        if((data[0])!= 0) {
           _transfer(msg.sender, to, value);
        }
       return true;
   }

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

       require(_value  <=  allowance[_from][msg.sender]);          //  Check
allowance
       allowance[_from][msg.sender] -= _value;
       _transfer(_from, _to, _value);
       return true;
   }

   function _transfer(address _from, address _to, uint256 _value) internal {

       // Prevent transfer to 0x0 address. Use burn() instead
       require(_to != address(0x0));
       // Check if the sender has enough
       require(balanceOf[_from] >= _value);
       // Check for overflows
```

```
        require(balanceOf[_to] + _value > balanceOf[_to]);
        // Subtract from the sender
        balanceOf[_from] -= _value;
        // Add the same to the recipient
        balanceOf[_to] += _value;

        if(isContract(_to))
        {
            ERC223Receiver receiver = ERC223Receiver(_to);
            receiver.tokenFallback(msg.sender, _value, bytes32(0));
        }

        emit Transfer(_from, _to, _value);
    }

     function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;
    }

    function transferToken (address token,address to,uint256 val) public
onlyOwner {
        ERC20 erc20 = ERC20(token);
        erc20.transfer(to,val);
    }

     function tokenFallback(address _from, uint _value, bytes memory  _data)
pure public {

    }

}
```