Cybertech Ltd
ITN 7751139129/ 775101001

№ 14 date 14.08.2019

Audit of SWZL token Smart Contract

Initial Data:

Smart Contract Address in Ethereum: **0x946eA588417fFa565976EFdA354d82c01719a2EA**

Bytecode available at etherscan.io available at:
https://etherscan.io/address/0x946ea588417ffa565976efda354d82c01719a2ea#code

Source code of the contract available at: **https://github.com/SwapZilla/contract**

Token meets the specification of ERC20.

Token page on etherscan.io:
https://etherscan.io/token/0x946ea588417ffa565976efda354d82c01719a2ea?a=0x253442363cead3f9810b1033b3e6807318a9d4aa

Token is verified by the etherscan.io team .

Project website: **https://www.swapzilla.co**

Reddit: **https://www.reddit.com/user/swapzilla**

Facebook: **https://www.facebook.com/swapzilla/**

Twitter: **https://twitter.com/swapzillaco**

Telegram: **https://t.me/SwapZilla**

Medium: **https://medium.com/@swapzilla**

White paper: **https://www.swapzilla.co/WhitePaper-eng2.pdf**

Current state of the contract: The volume of token emission is fixed, tokens issued in full at the time of publication of the contract. Additional emission under this contract is not permitted.

Owner of the Ethereum contract has an address:
0x253442363CEaD3f9810b1033B3E6807318A9D4aa

30,000,000 SWZL tokens are issued.

D. A. Akimov

1. Source code analysis.

1.1. Checking for the use of unsafe mathematics.

6 cases of use of unsafe maths are found in the smart-contract's code. Each case must be analyzed to prevent an overflow of variables.

2 cases of the use of unsafe maths are found in the function «transfer»

```
function transfer(address _to, uint256 _value) public ret
    require(balances[msg.sender] >= _value);
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value); //solhint-dis
    return true;
}
```

Line 41. Value «balances[msg.sender] -= _value» is safe, because of the fact that it is being checkein line 40 that value «balances[msg.sender]» is greater than or equal to value «_value».

Line 42. Checking for overflow of a variable «balances[_to]» is not implemented. However, taking into consideration the smart-contract's current state and logic of its operating, it has become possible to detect that any user's balance cannot exceed the total amount of tokens emitted. That is the number «3000000000000000». So an overflow of variable «balances[_to]» is impossible in line 42.

3 cases of the use of unsafe maths are found in the function «transferFrom»:

```
function transferFrom(address _from, address _to, uint256 _value) public returns (bool
    uint256 allowance = allowed[_from][msg.sender];
    require(balances[_from] >= _value && allowance >= _value);
    balances[_to] += _value;
    balances[_from] -= _value;
    if (allowance < MAX_UINT256) {
        allowed[_from][msg.sender] -= _value;
    }
    emit Transfer(_from, _to, _value); //solhint-disable-line indent, no-unused-vars
    return true;
}
```

Line 50. Checks for overflow of the variable "balances [_to]" are not performed. However, from line 49 it follows that the value of "_value" cannot exceed the total volume of issued tokens. Thus, the overflow of the variable "balances [_to]" is impossible.

D. A. Akimov

Line 51 It follows from the condition on line 49 that the amount spent cannot exceed the available balance. Thus, the value of the variable "_value" cannot be greater than the current value of "balances [_from]". As a result, the logic of the contract is not violated.

Line 53. Possible negative consequences of the operation are explicitly bypassed by the total use of lines 49 and 52, which avoids the appearance of undesirable values in the variable "allowed".

In the "approve" function, 1 use case was detected.

```
function approve(address _spender, uint256 _value) public returns (bool succe
    allowed[msg.sender][_spender] = _value;
    emit Approval(msg.sender, _spender, _value); //solhint-disable-line inden
    return true;
}
```

Line 64. Overflow checks are not performed. However, it makes no sense for a legal user to transmit "_value" values greater than the sum of tokens available, there is a low probability of problems arising. Intentional overflow of this variable will not give any unauthorized opportunities due to the correspondence of the declared types of the argument _value and the variable allowed. In addition, a repeated call of the approve procedure will also not lead to negative consequences, since in the term 64 it is not a summation but an assignment of a value.

1.2. Checking access rights to key functions.

The contract does not provide any restrictions on access to key functions, all functions can be accessed by all users. Almost all of these functions provide an access to the msg.sender service. This ensures that the user of any of these functions can work - only within the limits of the own balance available. The entire initial emission volume is assigned to the owner of the contract, which has the address

0x253442363CEaD3f9810b1033B3E6807318A9D4aa.

The owner of the contract in accordance with the approved agreements makes further distribution of tokens.

2. Logic analysis of the Smart Contract.

Logic of operating of the function «approve» is different from accepted for the smart-contracts corresponding to ERC20 specification. As it increases the amount of tokens allowed to be governed by another user by the transmitted value but it does not make it equal to the transmitted value. However, this method can be used because of the presence of potential vulnerability in the generally accepted method.

D. A. Akimov

Conclusion

After conducting an audit and examining the source code of the smart contract by our experts, the following conclusions were drawn:

The source code does not contain any critical vulnerabilities.

The audit was conducted by the «CyberTech» specialists.

«Cybertech»

CTO of CyberTech Ltd, Ph.D.

D. A. Akimov