Smart Contract Audit Report

Personal ICO Tokens

October 2nd, 2020

TABLE OF CONTENTS

SUMMARY

Audit Report prepared by Ali for Personal ICO Tokens.

Note: This is for demonstration purpose only.

PROCESS AND DELIVERY

An independent research was performed using the below mentioned tools. The debrief took place on October 2nd, 2020 and the final results are presented here.

Tool: https://tool.smartdec.net

AUDITED FILES

- ICO.sol
- Token.sol
- Migrations.sol

NOTES

Audit was performed on https://github.com/alijnmerchant21/Personal-ICO-Token.git and pragma version 0.4.x.

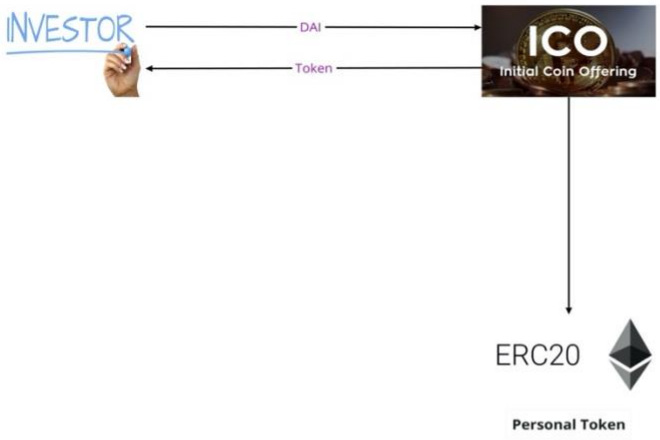
INTENDED BEHAVIOR

1. Product Context

Personal ICO Token is an ERC-20 based token designed for the purpose of raising money for start of business. *This ICO is for demonstration purpose only.*

2. Product Context

2.1 Algorithm overview



ISSUES FOUND

Hardcoded address

File: ICO.sol
Lines: 23-23
Severity: 1

The contract contains unknown address. This address might cause some malicious activity. Please check hardcoded address and it usage.

Recommendation:

It is required to check the address. Also, it is required to check the code of the called contract for vulnerabilities.

Example:

In the following contract, the address is specified in the source code:

```
pragma solidity 0.4.24;
  contract C {
    function f(uint a, uint b) pure returns (address) {
      address public multisig = 0xf64B584972FE6055a770477670208d737Fff282f;
      return multisig;
    }
  }
```

Do not forget to check the contract at the address

0xf64B584972FE6055a770477670208d737Fff282f

for vulnerabilities.

Compiler version not fixed

File: ICO.sol
Lines: 1-1
Severity: 1

Solidity source files indicate the versions of the compiler they can be compiled with.
pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.24; // good : compiles w 0.4.24 only
It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Recommendation:

Specify the exact compiler version (pragma solidity x.y.z;)

Example:

In the following example, an unsafe way to specify versions of the compiler is used:
pragma solidity ^0.4.17;
Safe alternative:
pragma solidity 0.4.24;

File: Token.sol
Lines: 1-1
Severity: 1

Solidity source files indicate the versions of the compiler they can be compiled with.
pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.24; // good : compiles w 0.4.24 only
It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Recommendation:

Specify the exact compiler version (pragma solidity x.y.z;)

Example:

In the following example, an unsafe way to specify versions of the compiler is used:
pragma solidity ^0.4.17;
Safe alternative:
pragma solidity 0.4.24

File: Migrations.sol
Lines: 2-2
Severity: 1

Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.24; // good : compiles w 0.4.24 only

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Recommendation:

Specify the exact compiler version (pragma solidity x.y.z;)

Example:

In the following example, an unsafe way to specify versions of the compiler is used:
pragma solidity ^0.4.17;

Safe alternative:

pragma solidity 0.4.24;

Use of SafeMath

File: ICO.sol
Lines: 8-8
Severity: 1

Compiler version not fixed
Solidity source files indicate the versions of the compiler they can be compiled with.

```
pragma solidity ^0.4.17; // bad: compiles w 0.4.17 and above
pragma solidity 0.4.24; // good : compiles w 0.4.24 only
```

It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee.

Recommendation:

Specify the exact compiler version (pragma solidity x.y.z;).

Example:

In the following example, an unsafe way to specify versions of the compiler is used:

```
pragma solidity ^0.4.17;
```

Safe alternative:

```
pragma solidity 0.4.24;
```

File: Token.sol
Lines: 7-7
Severity: 1

SafeMath library is used.

Recommendation:

We do not recommend using SafeMath library for all arithmetic operations. It is good practice to use explicit checks where it is really needed, and to avoid extra checks where overflow/underflow is impossible.

Example:

In the following example, the SafeMathlibrary is used:

```
pragma solidity 0.4.24;
import "../libraries/SafeMath.sol";
contract SafeSubAndDiv {
  using SafeMath for uint256;
function sub(uint a, uint b) public returns(uint) {
    return(a.sub(b));
  }

  function div(uint a, uint b) public returns(uint) {
    return(a.div(b));
  }
}
```

Best practice:

```
pragma solidity 0.4.24;
contract SafeSubAndDiv {
  function sub(uint a, uint b) public returns(uint) {
    // explicit check
    require(a >= b, "the result of the subtraction is negative");
    return(a - b);
  }

  function div(uint a, uint b) public returns(uint) {
    // EVM does not allow division by zero
    return(a / b);
  }
}
```

Prefer external to public visibility level

File: Migrations.sol
Lines: 16-18
Severity: 1

A function with public visibility modifier that is not called internally. Changing visibility level to external increases code readability. Moreover, in many cases functions with external visibility modifier spend less gas comparing to functions with public visibility modifier.

Recommendation:

Use the external visibility modifier for functions never called from the contract via internal call.

Example:

In the following example, functions with both public and external visibility modifiers are used:

```
contract Token {
  mapping (address => uint256) internal _balances;
   function transfer_public(address to, uint256 value) public {
      require(value <= _balances[msg.sender]);
      _balances[msg.sender] -= value;
      _balances[to] += value;
   }

   function transfer_external(address to, uint256 value) external {
      require(value <= _balances[msg.sender]);

      _balances[msg.sender] -= value;
      _balances[to] += value;
   }
}
```

The second function requires less gas.

Implicit visibility level

File: ICO.sol
Lines: 25-51
Severity: 1

The default function visibility level in contracts is public, in interfaces - external, state variable default visibility level is internal. In contracts, the fallback function can be external or public. In interfaces, all the functions should be declared as external. Explicitly define function visibility to prevent confusion.

Recommendation:

Avoid ambiguity: explicitly declare visibility levels.

Example:

In the following example, a specific modifier is not used when declaring a function:

```
function foo();
```

Preferred alternatives:

```
function foo() public;
```

```
function foo() internal;
```

File: Token.sol
Lines: 11-18
Severity: 1

The default function visibility level in contracts is public, in interfaces - external, state variable default visibility level is internal. In contracts, the fallback function can be external or public. In interfaces, all the functions should be declared as external. Explicitly define function visibility to prevent confusion.

Recommendation:

Avoid ambiguity: explicitly declare visibility levels.

Example:

In the following example, a specific modifier is not used when declaring a function:

```
function foo();
```

Preferred alternatives:

```
function foo() public;
function foo() internal;
```

CLOSING SUMMARY

Upon the audit of Personal ICO token's smart contract, it was observed that the contract contain issues at various criticality level.

A recommendation and ideal format was also attached. Resolving as per the recommendation given is as up to Personal ICO Token's discretion. The notes refer to improving the operations of the smart contract.

DISCLAIMER

The audit performed is for demonstration purpose only; not a security warranty, investment advice, or an endorsement of Personal ICO Tokens. The audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contract is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during the debrief process, in order to provide an unbiased delivery and protect the auditors from any legal and financial liability.