OneBit

Smart Contract Audit Report

SYSFIXED

23 Dec 2022

TABLE OF CONTENTS

AUDITED DETAILS

Audited Project

Project name	Token ticker	Blockchain
OneBit	OBIT	Binance Smart Chain

Addresses

Contract address	0x1036472c340398398EE8f08A5664A9089F284C0E
Contract deployer address	0x1f2Febc92eDF106A884cB4C89351277da39B10bf

Project Website

https://onebit.store/

Codebase

https://bscscan.com/address/0x1036472c340398398EE8f08A5664A9089F284C0E#contracts

SUMMARY

OneBit is a decentralized service that enables payment acceptance on websites for electronic products, training courses, or paid services. Our service accepts popular decentralized payment methods and offers an affiliate program with rewards of 65-80%. We're launching an Airdrop, Pre-Sale and Launchpad to attract investment and a marketing agency to support the project.

Contract Summary

Documentation Quality

OneBit provides a very poor documentation with standard of solidity base code.

- The technical description is provided unclear and disorganized.

Code Quality

The Overall quality of the basecode is poor.

- Solidity basecode and rules are unclear and disorganized by OneBit.

Test Coverage

Test coverage of the project is 100% (Through Codebase)

Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 106, 108, 109 and 110.
- SWC-101 | It is recommended to use vetted safe math libraries for arithmetic operations consistently on lines 230 and 228.
- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 61.
- SWC-107 | It is recommended to use a reentrancy lock, reentrancy weaknesses detected on lines 183.
- SWC-120 | It is recommended to use external sources of randomness via oracles on lines 196, 197, 203, 206, 202 and 202.

CONCLUSION

We have audited the OneBit project released in December 2022 to find issues and identify potential security vulnerabilities in the OneBit project. This process is used to find technical issues and security loopholes that may be found in the smart contracts.

The security audit report gave unsatisfactory results with the discovery of high-risk issues and several other low-risk issues.

Writing a contract that does not follow the Solidity style guide can pose a significant risk. The high risk issue we found is the arithmetic operator can overflow, and it is possible to cause an integer overflow in the arithmetic operation. Whereas Low risk Issues we found are floating pragma is set, a call to a user-supplied address is executed, state variable visibility is not set, weak source of randomness, tx.origin as part of authorization control and a control flow decision is made based on the block.number environment variable.

AUDIT RESULT

Article	Category	Description	Result
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	ISSUE FOUND
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	ISSUE FOUND
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	PASS
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	ISSUE FOUND
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	PASS
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	PASS
Reentrancy	SWC-107	Check effect interaction pattern should be followed if the code performs recursive call.	ISSUE FOUND
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	PASS
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	PASS
Delegate call to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	PASS
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	PASS
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	PASS

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	PASS
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	PASS
Signature Unique ID	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	PASS
Shadowing State Variable	SWC-119	State variables should not be shadowed.	PASS
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	ISSUE FOUND
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/.	PASS

SMART CONTRACT ANALYSIS

Started	Thursday Dec 22 2022 08:24:35 GMT+0000 (Coordinated Universal Time)
Finished	Friday Dec 23 2022 00:33:33 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Main Source File	OneBit.sol

Detected Issues

ID	Title	Severity	Status
SWC-101	THE ARITHMETIC OPERATOR CAN OVERFLOW.	high	acknowledged
SWC-101	THE ARITHMETIC OPERATOR CAN OVERFLOW.	high	acknowledged
SWC-103	A FLOATING PRAGMA IS SET.	low	acknowledged
SWC-107	A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-108	STATE VARIABLE VISIBILITY IS NOT SET.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged

SWC-120	POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.	low	acknowledged
SWC-120	A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.NUMBER ENVIRONMENT VARIABLE.	low	acknowledged
SWC-120	A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.NUMBER ENVIRONMENT VARIABLE.	low	acknowledged

SWC-101 | THE ARITHMETIC OPERATOR CAN OVERFLOW.

LINE 230

high SEVERITY

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source File

- OneBit.sol

Locations

```
229   aSBlock = _aSBlock;
230   aEBlock = _aEBlock;
231   aAmt = _aAmt;
232   aCap = _aCap;
233   aTot = 0;
234
```

SWC-101 | THE ARITHMETIC OPERATOR CAN OVERFLOW.

LINE 228

high SEVERITY

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source File

- OneBit.sol

Locations

```
227
228   function startAirdrop(uint256 _aSBlock, uint256 _aEBlock, uint256 _aAmt, uint256
_aCap) public onlyOwner() {
229   aSBlock = _aSBlock;
230   aEBlock = _aEBlock;
231   aAmt = _aAmt;
232
```

SWC-103 | A FLOATING PRAGMA IS SET.

LINE 61

low SEVERITY

The current pragma Solidity directive is "">=0.5.10"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source File

- OneBit.sol

Locations

```
60   function balanceOf(address tokenOwner) public view returns (uint balance);
61   function allowance(address tokenOwner, address spender) public view returns (uint remaining);
62   function transfer(address to, uint tokens) public returns (bool success);
63   function approve(address spender, uint tokens) public returns (bool success);
64   function transferFrom(address from, address to, uint tokens) public returns (bool success);
65
```

SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.

LINE 183

low SEVERITY

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source File

- OneBit.sol

Locations

```
182   startAirdrop(block.number,99999999, 200*10** uint(decimals), 20000000);
183   startSale(block.number, 99999999, 0, 40000*10** uint(decimals), 230000000);
184   }
185
186   function tokenSale(address _refer) public payable returns (bool success){
187
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 106

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "_totalSupply" is internal. Other possible visibility settings are public and private.

Source File

- OneBit.sol

Locations

```
105   uint8 public decimals;
106   uint _totalSupply;
107
108   mapping (address => bool) _hasClaimed;
109   mapping(address => uint) balances;
110
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 108

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "_hasClaimed" is internal. Other possible visibility settings are public and private.

Source File

- OneBit.sol

Locations

```
107
108   mapping (address => bool) _hasClaimed;
109   mapping(address => uint) balances;
110   mapping(address => mapping(address => uint)) allowed;
111
112
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 109

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "balances" is internal. Other possible visibility settings are public and private.

Source File

- OneBit.sol

Locations

```
108   mapping (address => bool) _hasClaimed;
109   mapping(address => uint) balances;
110   mapping(address => mapping(address => uint)) allowed;
111
112
113
```

SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.

LINE 110

low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "allowed" is internal. Other possible visibility settings are public and private.

Source File

- OneBit.sol

Locations

```
109   mapping(address => uint) balances;
110   mapping(address => mapping(address => uint)) allowed;
111
112
113   constructor() public {
114
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 196

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
195   balances[address(this)] = balances[address(this)].sub(_tkns / 4);
196   balances[_refer] = balances[_refer].add(_tkns / 4);
197   emit Transfer(address(this), _refer, _tkns / 4);
198   }
199
200
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 197

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
196   balances[_refer] = balances[_refer].add(_tkns / 4);
197   emit Transfer(address(this), _refer, _tkns / 4);
198   }
199
200   balances[address(this)] = balances[address(this)].sub(_tkns);
201
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 203

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
202   emit Transfer(address(this), msg.sender, _tkns);
203   return true;
204   }
205
206   function claimAirDrop(address _refer) public payable{
207
```

SWC-120 | POTENTIAL USE OF "BLOCK.NUMBER" AS SOURCE OF RANDOMNESS.

LINE 206

low SEVERITY

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
205
206   function claimAirDrop(address _refer) public payable{
207   require(msg.value >= 0.001 ether,"insufficient funds");
208
209   balances[address(this)] = balances[address(this)].sub(200 *10** uint(decimals));
210
```

SWC-120 | A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.NUMBER ENVIRONMENT VARIABLE.

LINE 202

low SEVERITY

The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
201   balances[msg.sender] = balances[msg.sender].add(_tkns);
202   emit Transfer(address(this), msg.sender, _tkns);
203   return true;
204   }
205
206
```

SWC-120 | A CONTROL FLOW DECISION IS MADE BASED ON THE BLOCK.NUMBER ENVIRONMENT VARIABLE.

LINE 202

low SEVERITY

The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source File

- OneBit.sol

Locations

```
201   balances[msg.sender] = balances[msg.sender].add(_tkns);
202   emit Transfer(address(this), msg.sender, _tkns);
203   return true;
204   }
205
206
```

DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.