Audit report of Synergy.SURF

Prepared By: - Kishan Patel

Prepared On: - 06/01/2023

Prepared for: Synergy.SURF

Table of contents

THIS AUDIT REPORT WILL CONTAIN CONFIDENTIAL INFORMATION ABOUT THE SMART CONTRACT AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES OF THEIR EXPLOITATION.

THE INFORMATION FROM THIS AUDIT REPORT CAN BE USED INTERNALLY BY THE CUSTOMER OR IT CAN BE DISCLOSED PUBLICLY AFTER ALL VULNERABILITIES ARE FIXED - UPON THE DECISION OF THE CUSTOMER.

1. Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions). Because the total numbers of test cases are unlimited, the audit makes no statements or warranties on the security of the code.

It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

2. Introduction

Kishan Patel (Consultant) was contacted by Nifty Splits Ltd (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contracts and its code review conducted between 06/01/2023 – 08/01/2022.

The project has 8 files. It contains approx 1200 lines of Solidity code. All the functions and state variables are well commented on using the natspec documentation, but that does not create any vulnerability.

3. Project information

Token Name	Synergy.SURF
Token Symbol	Synergy.SURF
Platform	Binance Smart Chain
Order Started Date	06/01/2023
Order Completed Date	08/01/2023

4. List of attacks checked

- Over and under flows
- Short address attack
- Visibility & Delegate call
- Reentrancy / TheDAO hack
- Forcing BUSD to a contract
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference

5. Severity Definitions

Risk	Level Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution

6. Good things in code

- ## Good required condition in functions:-

File Name: - Syrfico.sol

- Here you are checking that ICOState is BEFORE then only allow starting ico.

```
105     //Start, Halt and End ICO
106     function startICO() external onlyAdmin {
107         require(ICOState == State.BEFORE, "ICO isn't in before stat
108         ICOState = State.RUNNING;
109     }
110
```

- Here you are checking that ICOState is RUNNING then only allow halt ico.

```
111     function haltICO() external onlyAdmin {
112         require(ICOState == State.RUNNING, "ICO isn't running yet")
113         ICOState = State.HALTED;
114     }
115
```

- Here you are checking that ICOState is HALTED then only allow resuming ico.

```
115
116     function resumeICO() external onlyAdmin {
117         require(ICOState == State.HALTED, "ICO State isn't halted y
118         ICOState = State.RUNNING;
```

- Here you are checking that ICOState is RUNNING then only allow end ico.

```
121     function endICO() external onlyAdmin {
122         require(ICOState == State.RUNNING, "ICO State isn't running
123         ICOState = State.END;
```

- Here you are checking that ICOState is not end transfer method of token is successfully called or not.
- Question: - If Ico is on halted or before then also you want to burn token?

```
137     function burn() external onlyAdmin returns (bool)  {
138         require(ICOState == State.END, "ICO isn't over yet");
139 
140         uint remainingTokens = token.balanceOf(address(this));
141         bool success = token.transfer(address(0), remainingTokens);
142         require(success, "Failed to burn remaining tokens");
```

- Here you are checking that amount which is user investing is smaller than hardCap value and transfer to msg.sender is successfully done or not.

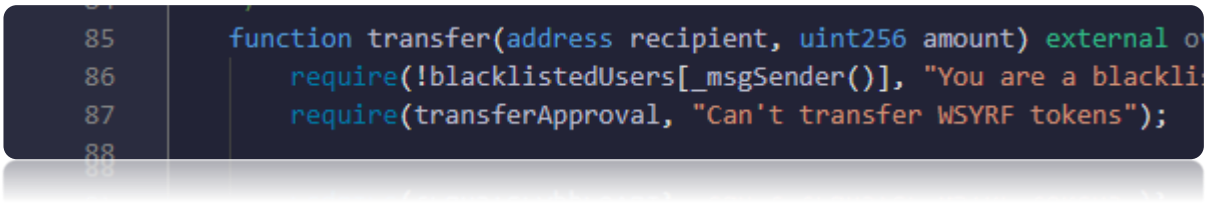
```
162 
163     //Invest
164     function invest() public payable returns (bool) {
165         require(ICOState == State.RUNNING, "ICO isn't running");
166         require(raisedAmount.add((msg.value.mul(getLatestPrice()))).
```

- Here you are checking that ICO state is running or not, token_amount is bigger than 0 and token_amount which user wants to buy is smaller than hardcap value.

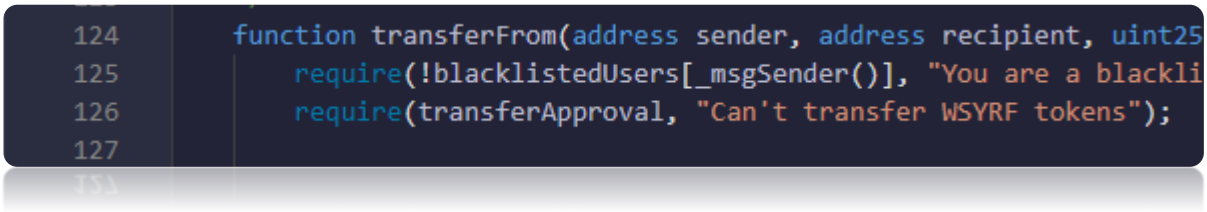
```
180 
181     //Buy SYRF with other tokens
182     function buyWithTokens(uint token_amount, address _tokenAddr) p
183         require(ICOState == State.RUNNING, "ICO isn't running");
184         require(token_amount > 0, "Amount can't be zero numbers");
185         require(raisedAmount.add(token_amount.div(tokenPrice.mul(1e
186 
```

File Name: -WSYRF.sol

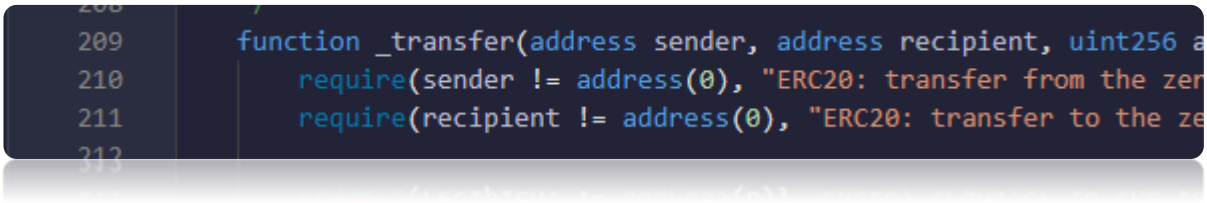
- Here you are checking that msg.sender is not balcklisted user and transfer process has been started by owner.



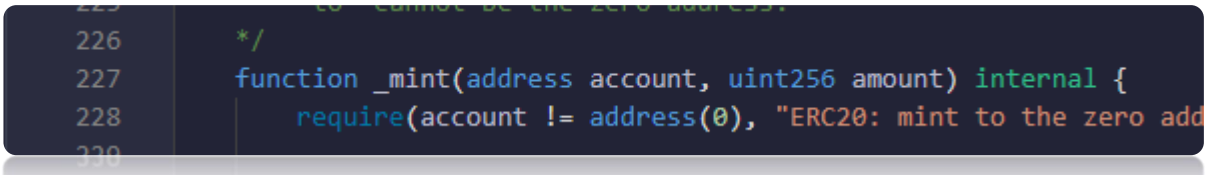
- Here you are checking that msg.sender is not balcklisted user and transfer process has been started by owner.
- Suggestions: - I think you have to check that sender is also not blacklisted otherwise some blacklisted account (sender) can give permission to not blacklisted account (msg.sender) and they can withdraw money.

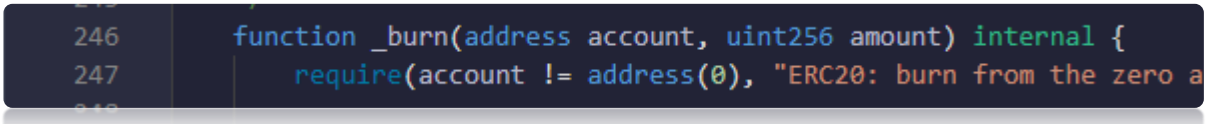


- Here you are checking that sender and recipient addresses are valid and proper.



- Here you are checking that account address is valid and proper.





- Here you are checking that owner and spender addresses are valid and proper.

```
267     function _approve(address owner, address spender, uint256 amoun
268         require(owner != address(0), "ERC20: approve from the zero
269         require(spender != address(0), "ERC20: approve to the zero
```

- Here you are checking that _value is not same as currently status of address in blacklistedUsers array.

```
290     function setBlacklisted(address _addr, bool _value) external on
291         require(blacklistedUsers[_addr] != _value, "Not changed");
292         blacklistedUsers[_addr] = _value;
```

- Here you are checking that _value is not same as transferApproval .

```
299     function setTransferApproval(bool _value) external onlyOwner {
300         require(transferApproval != _value, "Not changed");
301         transferApproval = _value;
```

File Name: -Ownable.sol

- Here you are checking that newOwner address is valid or not.

```
62      function transferOwnership(address newOwner) public virtual only
63          require(newOwner != address(0), "Ownable: new owner is the z
64          _transferOwnership(newOwner);
65      }
```

7. Critical vulnerabilities in code

- **No Critical vulnerabilities found**

8. Medium vulnerabilities in code

- **No Medium vulnerabilities found**

9. Low vulnerabilities in code

9.1. Suggestions to add code validations:-

=> You have implemented required validation in contract.
=> There are some place where you can improve validation and security of your code.
=> These are all just suggestion it is not bug.

File Name: - Syrfico.sol

- **Function: - withdrawBNB**

```
147
148     //WithdrawBNB
149     function withdrawBNB() public onlyAdmin {
150         uint256 balance = address(this).balance;
151         payable(msg.sender).transfer(balance);
152     }
```

- Here in withdrawBNB function you can check that transfer to msg.sender address is successfully done or not.

- Please use admin address instead of msg.sender if possible because only admin is able to call this method.

- **Function: - withdrawERC20**

```
154     //WithdrawToken
155     function withdrawToken(address _tokenAddr) public onlyAdmin {
156         require(IBEP20(_tokenAddr).balanceOf(address(this)) > 0, "Su
157
158         IBEP20(_tokenAddr).transfer(msg.sender, IBEP20(_tokenAddr).b
159     }
```

- Here in withdrawBNB function you can check that transfer to msg.sender address is successfully done or not.

- Please use admin address instead of msg.sender if possible because only admin is able to call this method.

- **Function: - withdrawERC20**

```
188
189         if(_tokenAddr == BUSD_Addr) {
190             BUSD.transferFrom(msg.sender, address(this), token_amou
191         } else if(_tokenAddr == USDT_Addr) {
192             USDT.transferFrom(msg.sender, address(this), token_amou
193         } else if(_tokenAddr == USDC_Addr) {
194             USDC.transferFrom(msg.sender, address(this), token_amou
195         } else {
196             return false;
197         }
198
199         investedAmountOf[msg.sender] = investedAmountOf[msg.sender]
200         raisedAmount = raisedAmount.add(tokens.div(1e18));
201         token.transfer(msg.sender, tokens); // Send WSYRF tokens to
```

- Here in buyWithTokens function you can check that transferFrom methods are successfully called or not.

- It would be nice if you can check that transfer method of token is successfully called or not.

File Name: - WSYRF.sol

- **Function: - _approve**

```
267     function _approve(address owner, address spender, uint256 amount
268         require(owner != address(0), "ERC20: approve from the zero a
269         require(spender != address(0), "ERC20: approve to the zero a
270 
271         _allowances[owner][spender] = amount;
272         emit Approval(owner, spender, amount);
273     }
```

- Here in _approve function you can check that amount value is not bigger than balance of owner.
- It could be possible that owner has no sufficient balance and give more allowance to spender.

- **Function: - airdrop**

```
324     function airdrop(address tokenAddress) external onlyOwner {
325         for(uint256 i = 0; i < airdropAddress.length ; i ++) {
326             IERC20(tokenAddress).transfer(airdropAddress[i], airdro
327         }
328     }
```

- Here in airdrop function you can check that transfer method of tokenAddress is successfully called or not.

10. Summary

- **Number of problems in the smart contract as per severity level**

Critical	Medium	Low
0	0	5

According to the assessment, the smart contract code is well secured. The code is written with all validation and all security is implemented. Code is performing well and there is no way to steal funds from this contract.

- **Good Point:** Code performance and quality are good. All kind of necessary validation added into smart contract and all validations are working as excepted.

- **Suggestions:** Please try to implement suggested code validations.