RD AUDITORS

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Safe Token
Prepared on: 30/03/2021
Platform: Binance
Language: Solidity

info@rdauditors.com

Table of contents

THIS DOCUMENT MAY CONTAIN CONFIDENTIAL INFORMATION ABOUT ITS SYSTEMS AND INTELLECTUAL PROPERTY OF THE CUSTOMER AS WELL AS INFORMATION ABOUT POTENTIAL VULNERABILITIES AND METHODS OF THEIR EXPLOITATION.

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

Document

Name	Smart Contract Code Review and Security Analysis Report for Safe Token
Platform	Binance/Solidity
File	SAFETOKEN.sol
MD5 hash	D41D8CD98F00B204E9800998ECF8427E
SHA256 hash	E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934 CA495991B7852B855
Date	30/03/2021

info@rdauditors.com

Introduction

RD Auditors (Consultant) was contracted by Safe Token (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 March 28, 2021 – March 30, 2021.

This contract consists of 1 file.

Project Scope

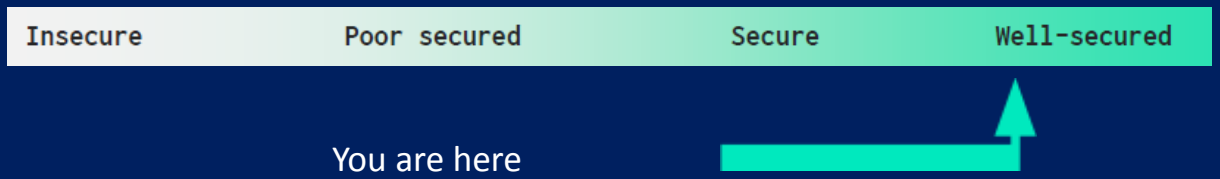
The scope of the project is a smart contract.

We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but not limited to):

- Reentrancy
- 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
- Implicit visibility level

Executive Summary

According to the assessment, the customer`s solidity smart contract is well secured.



Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low and 0 very low level issues.

Code Quality

Safe Token consists of a single smart contract file. This also contains safeMath and address, this is a compact and well written contract.

The library in SAFETOKEN is part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in SAFETOKEN.

Safe Token has **not** provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is well commented. Commenting can provide rich documentation for functions, return variables and more. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given Safe Token contract in the form of a github link: https://bscscan.com/address/0x31045e7023e6c388f9447e632a3f9eaff90393fa#code

The hash of those files are mentioned in the table. As mentioned, It's well commented smart contract code, so anyone can quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

Core code blocks are written well and systematically. No other dependencies except safeMath and Address.

AS-IS overview

Safe Token overview

Safe Token provides a safe way to invest and protect one's earnings.

File And Function Level Report

Contract: Ownable
inherit: Context
Observation: All passed including security check
Test Report: passed
Score: passed
Conclusion: passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	write	Passed	All Passed	No Issue	Passed
3	transferOwnership	write	passed	All Passed	No Issue	Passed
4	getunlockTime	read	passed	All Passed	No Issue	Passed
5	lock	write	passed	All Passed	No Issue	Passed
6	unlock	write	passed	All Passed	No Issue	Passed

Contract: SafeToken
Inherit: context, IERC20, Ownable
Observation: All passed including security check
Test Report: passed
Score: passed
Conclusion: passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	passed	All Passed	No Issue	Passed
4	totalSupply	read	passed	All Passed	No Issue	Passed
5	balanceOf	read	passed	All Passed	No Issue	Passed
6	transfer	write	passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	passed	All Passed	No Issue	Passed
10	IncreaseAllowance	write	passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	passed	All Passed	No Issue	Passed
12	IsExcludedFromReward	read	passed	All Passed	No Issue	Passed
13	totalFees	read	Passed	All Passed	No Issue	Passed
14	deliver	read	Passed	All Passed	No Issue	Passed
15	reflectionFromToken	read	passed	All Passed	No Issue	Passed
16	tokenFromReflection	read	passed	All Passed	No Issue	Passed
17	excludeFromFee	write	passed	All Passed	No Issue	Passed
18	SetLiquidityFeePercent	write	passed	All Passed	No Issue	Passed
19	SetMaxTxPercent	write	Passed	All Passed	No Issue	Passed
20	SetSwapAndLiquifyEnabled	write	Passed	All Passed	No Issue	Passed
21	_reflectFee	read	passed	All Passed	No Issue	Passed
22	_getValues	read	passed	All Passed	No Issue	Passed
23	_getTValues	read	passed	All Passed	No Issue	Passed
24	_getRValues	write	passed	All Passed	No Issue	Passed
25	_getRate	read	Passed	All Passed	NoIssue	Passed
26	_getCurrentSupply	read	Passed	All Passed	No Issue	Passed
27	_takeLiquidity	write	Passed	All Passed	No Issue	Passed
28	CalculateTaxFee	read	passed	All Passed	No Issue	Passed
29	CalculateLiquidityFee	read	passed	All Passed	No Issue	Passed
30	removeAllFee	read	passed	All Passed	No Issue	Passed
31	restoreAllFee	read	passed	All Passed	No Issue	Passed
32	isExcludedFromFee	read	Passed	All Passed	No Issue	Passed
33	_transferToExcluded	write	Passed	All Passed	No Issue	Passed
34	_approve	write	passed	All Passed	No Issue	Passed
35	_transferFromExcluded	write	passed	All Passed	No Issue	Passed
36	addLiquidity	write	passed	All Passed	No Issue	Passed
37	_tokenTransfer	write	passed	All Passed	No Issue	Passed
38	_transferStandard	write	Passed	All Passed	No Issue	Passed
39	_transfer	write	Passed	All Passed	No Issue	Passed
40	swapAndLiquidity	write	passed	All Passed	No Issue	Passed
41	includeInFee	write	passed	All Passed	No Issue	Passed
42	setTaxFeePercent	write	passed	All Passed	No Issue	Passed
43	_transferBothExcluded	write	passed	All Passed	No Issue	Passed
44	IncludeInReward	write	passed	All Passed	No Issue	Passed
45	excludeFromReward	write	passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
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
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

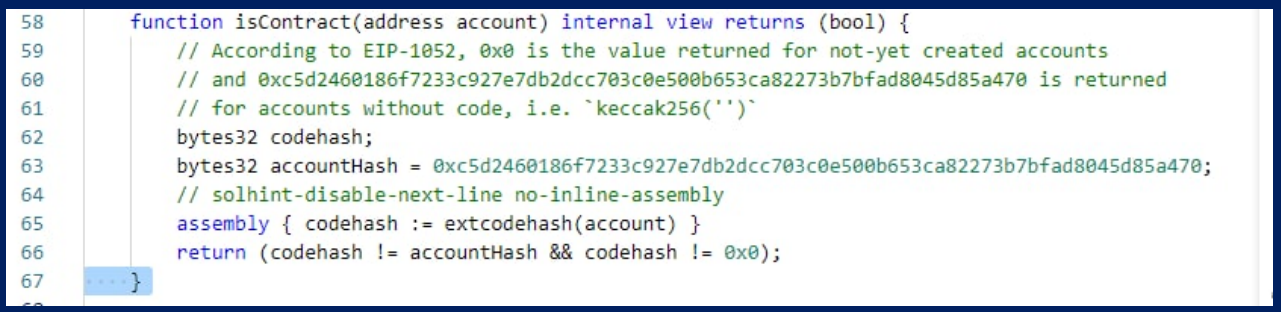
No Low severity vulnerabilities were found.

Very Low

No very Low severity vulnerabilities were found.

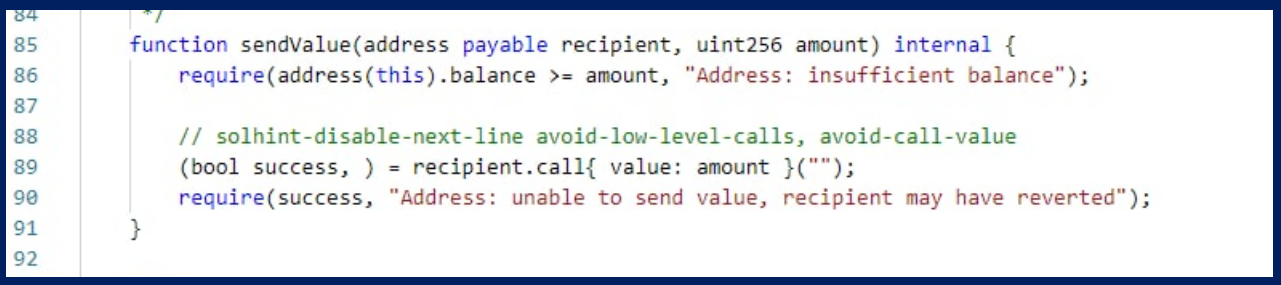
Discussion:

1) Hardcoded address should be checked before deployment



```
58      function isContract(address account) internal view returns (bool) {
59          // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
60          // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
61          // for accounts without code, i.e. `keccak256('')`
62          bytes32 codehash;
63          bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
64          // solhint-disable-next-line no-inline-assembly
65          assembly { codehash := extcodehash(account) }
66          return (codehash != accountHash && codehash != 0x0);
67      }
```

2) Unused functions inside library can be removed like “sendValue”



```
84      */
85      function sendValue(address payable recipient, uint256 amount) internal {
86          require(address(this).balance >= amount, "Address: insufficient balance");
87
88          // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
89          (bool success, ) = recipient.call{ value: amount }("");
90          require(success, "Address: unable to send value, recipient may have reverted");
91      }
92
```

Conclusion

We were given contract files. And we have used all possible tests based on the given object. The contracts are written systematically. We found no critical issues So it is good to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, so we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Security state of reviewed contract is “well secured”.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

RD Auditors 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 number of test cases are unlimited, so the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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 security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on 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.

RD
AUDITORS

info@rdauditors.com