RD AUDITORS

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: eMarket
Prepared on: 26/03/2021
Platform: Ethereum
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 eMarket
Platform	ethereum / Solidity
File1	eMarketDex.sol
MD5 hash	44D8DE8123EB9ECCAB80711FAD2DBCDC
SHA256 hash	B1B8294E9E6D4D0FC48C1925F09EF5E7725578DD2C992F489F6FF0AD8B913394
File2	eMarketToken.sol
MD5 hash	BDDF9D54374402BD9F18A9285246422F
SHA256 hash	0A6DABC76AA8DCA76A21851158CC10232A08545EED5F037161EE9732D7249656
Date	26/03/2021

Introduction

RD Auditors (Consultant) was contracted by eMarketDex Team (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 24, 2021 – March 26, 2021.

This contract consists of 2 files.

Project Scope

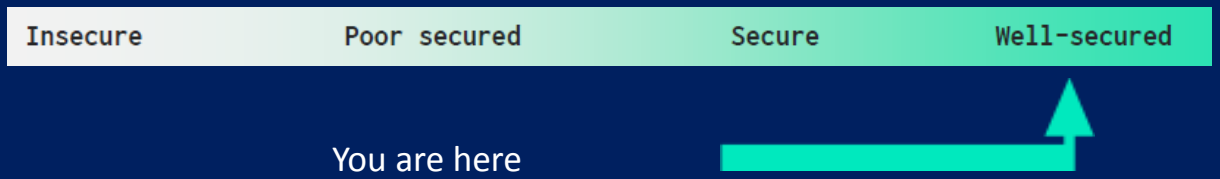
The scope of the project is a smart contract.

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

eMarket consists of two smart contract files.This also contains safeMath, both are compact and well written contracts.

The library in the eMarket 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 the eMarket

The eMarket team 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 eMarketDex contract and eMarketToken in the form of a github link:
https://rinkeby.etherscan.io/address/0x12f5e20f5e34b426513c7df0195a8105e1262e6f#code

https://rinkeby.etherscan.io/address/0x8a87a77fd75ca40a82233861280aced23c18eb3b#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.

AS-IS overview

eMarket contract overview

eMarket consists of two contracts, one is for the purpose of decentralized exchange and another is ERC20 standard token.

File And Function Level Report

File: eMarketDex.sol

Contract: eMarketDex
Inherit: owned
Observation: All passed including security check
Test Report: passed
Score: passed
Conclusion: passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	CalculatePercentage	read	Passed	All Passed	No Issue	Passed
2	changeFeeAccount	write	Passed	All Passed	No Issue	Passed
3	changeTradingFee	write	passed	All Passed	No Issue	Passed
4	availableTradingFeeOwner	read	passed	All passed	No Issue	Passed
5	withdrawTradingFeeOwner	write	passed	All Passed	No Issue	passed
6	deposit	write	Passed	All Passed	No Issue	Passed
7	withdraw	write	Passed	All Passed	No Issue	Passed
8	depositToken	write	Passed	All Passed	No Issue	Passed
9	withdrawToken	write	Passed	All Passed	No Issue	Passed
10	balanceOf	read	Passed	All Passed	No Issue	Passed
11	order	read	passed	All Passed	No Issue	Passed
12	trade	write	passed	All passed	No Issue	Passed
13	splittrade	write	passed	All Passed	No Issue	passed

14	TradeBalances	write	Passed	All Passed	No Issue	Passed
15	testTrade	read	Passed	All Passed	No Issue	Passed
16	availableVolume	read	Passed	All Passed	No Issue	Passed
17	amountFilled	read	Passed	All Passed	No Issue	Passed
18	CancelOrder	read	Passed	All Passed	No Issue	Passed
19	TransferFromPreSigned	write	passed	All Passed	No Issue	Passed
20	transferPreSigned	write	passed	All passed	No Issue	Passed
21	testSender	read	passed	All Passed	No Issue	passed
22	manualWithdrawTokens	write				removed
23	manualWithdrawEther	write				removed
24	destructContract	write				removed
25	ChangeSafeguardStatus	write	passed	All Passed	No Issue	Passed
26	PresignDeposit	write	passed	All passed	No Issue	Passed
27	PresignWithdraw	write	passed	All Passed	No Issue	passed
28	PresignTrade	write	Passed	All Passed	No Issue	Passed
29	splitPreSignTrade	write	Passed	All Passed	No Issue	Passed
30	PresignTradeBalances	write	Passed	All Passed	No Issue	Passed

File: eMarketToken.sol

Contract: eMarketToken

Inherit: owned

Observation: All passed including security check

Test Report: passed

Score: passed

Conclusion: passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_transfer	write	Passed	All Passed	No Issue	Passed
2	transfer	write	Passed	All Passed	No Issue	Passed
3	transferFrom	write	passed	All Passed	No Issue	Passed
4	approve	write	passed	All passed	No Issue	Passed
5	burn	write	passed	All Passed	No Issue	passed
6	burnFrom	write	Passed	All Passed	No Issue	Passed
7	freezeAccount	write	Passed	All Passed	No Issue	Passed
8	mintToken	write	Passed	All Passed	No Issue	Passed
9	manualWithdrawTokens	write				removed
10	manualWithdrawEther	write				removed
11	ChangeSafeguardStatus	write	passed	All Passed	No Issue	Passed
12	changeTokenSwapStatus	write	passed	All passed	No Issue	Passed

13	StartNewPassiveAirDrop	write	passed	All Passed	No Issue	passed
14	stopPassiveAirDropCompletely	write	Passed	All Passed	No Issue	Passed
15	claimpassiveAirdrop	write	Passed	All Passed	No Issue	Passed
16	changePassiveAirDropAmount	write	Passed	All Passed	No Issue	Passed
17	isContract	read	Passed	All Passed	No Issue	Passed
18	UpdateAirdropFee	write	Passed	All Passed	No Issue	Passed
19	airDropActive	write	passed	All Passed	No Issue	Passed
20	ChangeWhitelistingStatus	write	passed	All passed	No Issue	Passed
21	Whitelistuser	write	passed	All Passed	No Issue	passed
22	WhitelistManyusers	write	Passed	All Passed	No Issue	Passed
23	setPrices	write	Passed	All Passed	No Issue	Passed
24	buyTokens	write	Passed	All Passed	No Issue	Passed
25	sellTokens	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. There is a function “destructContract” in eMarketDex. The Owner can destroy the contract at any moment , if it is part of the plan than ok.

```
433
434   //selfdestruct function. just in case owner decided to destruct this contract.
435   function destructContract()onlyOwner public{
436       selfdestruct(owner);
437   }
438
```

2. There is a function “manualWithdrawther” in eMarketDex here owner can withdraw all ether from the contract if it is part of the plan than ok.

```
422
423   //Just in case, owner wants to transfer Ether from contract to owner address
424   function manualWithdrawEther()onlyOwner public{
425       address(owner).transfer(address(this).balance);
426   }
```

3. There is a function “manualWithdrawTokens” in eMarketDex here owner can withdraw all tokens from the contract, if it is part of the plan than ok.

```
428     //Just in case, owner wants to transfer Tokens from contract to owner address
429     //tokenAmount must be in WEI
430     function manualWithdrawTokens(address token, uint256 tokenAmount)onlyOwner public{
431         ERC20Essential(token).transfer(msg.sender, tokenAmount);
432     }
```

3. There is a function “manualWithdrawTokens” in eMarketToken here owner can withdraw all tokens from the contract, if it is part of the plan than ok.

```
293
294     function manualWithdrawTokens(uint256 tokenAmount) public onlyOwner{
295         // no need for overflow checking as that will be done in transfer function
296         _transfer(address(this), owner, tokenAmount);
297     }
```

3. There is a function “manualWithdrawEther” in eMarketToken here owner can withdraw all tokens from the contract, if it is part of the plan than ok.

```
299     //Just in rare case, owner wants to transfer Ether from contract to owner addr
300     function manualWithdrawEther()onlyOwner public{
301         address(owner).transfer(address(this).balance);
302     }
```

4 “eMarketDex” and “eMarketToken” contract codes are openly available in the public domain.

UPDATE(26/3/2021):

After our suggestion,“destructContract”,“manualWithdrawTokens” and “manualWithdrawEther” removed from the codeblock of eMarketDex and eMarketToken contract for public trust.

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