RD AUDITORS

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Investin
Prepared on: 26 April 2021
Platform: Binance Smart Chain
Language: Solidity

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 Investin
Platform	BSC / Solidity
File 1	CloneFactory.sol
MD5 hash	EF71D28FFE6FD3CAB144033953ED402E
SHA256 hash	5EDEC7204F3EA4D2FE5FE80CDB14037F5669853805312FFE87E78619C66488C4
File 2	FundDec.sol
MD5 hash	D41D8CD98F00B204E9800998ECF8427E
SHA256 hash	E3B0C44298FC1C149AFBF4C8996FB92427AE41E4649B934CA495991B7852B855
File 3	FundFactory.sol
MD5 hash	A9C0D5978E57E2374066D404E0537659
SHA256 hash	2D8A1E31189762533EB11934E526DB20B30311E996A75C97BC49D52CAC4D5B3A
File 4	InvestInMultiCall.sol
MD5 hash	82CF02A00A30B23C0DE77D6653582210
SHA256 hash	F32B21462953ACDF48D27C7D5EEAA5FC1BFF923AA1033D69117AE248A063E501
File 5	IVNYToken.sol
MD5 hash	EDC851A8C5B860F9D6958B5796E3F55F
SHA256 hash	53454B566B93CF8CDBE9E17EE68318C2091823AB2B0D12959D743F64EDF6CB82
File 6	RouterDecBEP20.sol
MD5 hash	5B98635E031F359E5074880DBBAD1F51
SHA256 hash	107D3D1F10DBDA876792CEFAC3FF3FA58B904C9F386128BBA52F6810B224CACD
Date	26/04/2021

Introduction

RD Auditors (Consultant) was contracted by Investin (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 16 - 26 April 2021.

This contract consists of six files.

Project Scope

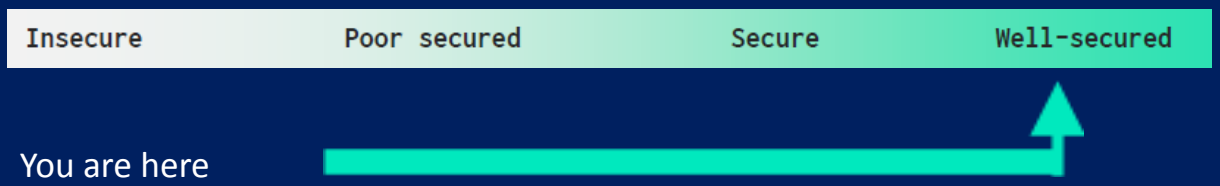
The scope of the project is a smart contract.

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

Investin consists of many smart contract files. This smart contract also contains many open source libraries from openZeppelin.

The libraries within the files are 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.

Investin has also conducted unit tests using scripts provided through the same github link which fortify functionality and security of the contract, which also helped us to determine the integrity of the code in an automated way.

Overall, the code is not commented. Commenting provides rich documentation for functions, return variables and more and also helps auditors to quick cover the flow behind code logic. Use of Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given Investin and its supporting files in the form of the github link: https://github.com/Investin-pro/SmartContracts/tree/Decentralized-BSC-V3/contracts

The hash of that file is mentioned in the table. As mentioned, it's recommended to write comments in the 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

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects. And even core code blocks are written well and systematically.

AS-IS Overview

Investin Overview: Is a swap functionality provider with the architecture of a decentralized fund management protocol.

File And Function Level Report

File: Clonefactory.sol

Contract: Clonefactory
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	CreateClone	write	Passed	All Passed	No Issue	Passed
2	isClone	read	Passed	All Passed	No Issue	Passed

File: FundDec.sol

Contract: FundDec
Inherit: Fund
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Initialize	write	Passed	All Passed	No Issue	Passed
2	getTokenIds	read	Passed	All Passed	No Issue	Passed
3	getFundDetails	read	Passed	All Passed	No Issue	Passed
4	getAmountInRouter	read	Passed	All Passed	No Issue	Passed
5	getNumberofActiveInvestment	read	Passed	All Passed	No Issue	Passed
6	getTotalAmount	read	Passed	All Passed	No Issue	Passed
7	getPrePerf	read	Passed	All Passed	No Issue	Passed
8	geetTokenList	read	Passed	All Passed	No Issue	Passed
9	getTokenBalance	read	Passed	All Passed	No Issue	Passed
10	UpdateTotalAmountAndPrevPerf	write	Passed	All Passed	No Issue	Passed
11	UpdateMinimumAmount	write	Passed	All Passed	No Issue	Passed
12	UpdateMinimumReturn	write	Passed	All Passed	No Issue	Passed
13	freezefund	write	Passed	All Passed	No Issue	Passed
14	updateManagerfundStatus	write	Passed	All Passed	No Issue	Passed
15	MakeInvestment	write	Passed	All Passed	No Issue	Passed
16	transferToVault	write	Passed	All Passed	No Issue	Passed
17	ClaimPerformancefee	write	Passed	All Passed	No Issue	Passed
18	getInvestmentByID	read	Passed	All Passed	No Issue	Passed
19	withdraw	write	Passed	All Passed	No Issue	Passed
20	transferTokens	write	Passed	All Passed	No Issue	Passed
21	swapAllTokens	write	Passed	All Passed	No Issue	Passed
22	addToken	write	Passed	All Passed	No Issue	Passed
23	deleteToken	write	Passed	All Passed	No Issue	Passed
24	swapTo	write	Passed	All Passed	No Issue	Passed
25	swap	write	Passed	All Passed	No Issue	Passed

File: FundFactory.sol

Contract: FundFactoryDec
Inherit: CloneFactory
Observation: All Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	setRouterAddress	write	Passed	All Passed	No Issue	Passed
2	createFund	write	Passed	All Passed	No Issue	Passed
3	getManagerToFund	read	Passed	All Passed	No Issue	Passed
4	getFundMapping	read	Passed	All Passed	No Issue	Passed
5	UpdateFundMapping	write	Passed	All Passed	No Issue	Passed
6	getFundList	read	Passed	All Passed	No Issue	Passed

File: InvestmentInMulticall.sol

Contract: InvestInMultiCall
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	addFactory	write	Passed	All Passed	No Issue	Passed
2	getFactories	read	Passed	All Passed	No Issue	Passed
3	getInvestmentDetailsByTokenID	read	Passed	All Passed	No Issue	Passed
4	getInvestmentByInvestorAddress	read	Passed	All Passed	No Issue	Passed
5	getManagerDetails	read	Passed	All Passed	No Issue	Passed
6	getTokenWhiteLists	read	Passed	All Passed	No Issue	Passed
7	FromAllRouters	read	Passed	All Passed	No Issue	Passed
8	getAllFundDetails	read	Passed	All Passed	No Issue	Passed

File: IVNYToken.sol

Contract: IVNYToken
Import: ERC721.sol, AccessControl.sol, Counters.sol
Inherit: ERC721, AccessControl
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	UpdateInvestInVaultAddress	write	Passed	All Passed	No Issue	Passed
2	UpdateStatus	write	Passed	All Passed	No Issue	Passed
3	UpdateToken	write	Passed	All Passed	No Issue	Passed
4	UpdateInvestInAddress	write	Passed	All Passed	No Issue	Passed
5	mint	write	Passed	All Passed	No Issue	Passed
6	getTokenInfo	read	Passed	All Passed	No Issue	Passed
7	roleAssignment	write	Passed	All Passed	No Issue	Passed
8	transferFrom	write	Passed	All Passed	No Issue	Passed
9	safeTransferFrom	write	Passed	All Passed	No Issue	Passed

File: RouterDecBEP20.sol

Contract: RouterDec
Import: SafeERC20
Inherit: Router
Observation: Passed
Test Report: Passed
Score: Passed
Conclusion: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	WhitelistToken	write	Passed	All Passed	No Issue	Passed
2	blacklistToken	write	Passed	All Passed	No Issue	Passed
3	getTokenWhiteList	read	Passed	All Passed	No Issue	Passed
4	getTokenMapping	read	Passed	All Passed	No Issue	Passed
5	getBaseTokenAddress	read	Passed	All Passed	No Issue	Passed
6	makeInvestmentHelper	write	Passed	All Passed	No Issue	Passed
7	transferToVaultHelper	write	Passed	All Passed	No Issue	Passed
8	getAmountAndPerformance	read	Passed	All Passed	No Issue	Passed
9	getInvestmentValue	read	Passed	All Passed	No Issue	Passed
10	returnsCalculation	read	Passed	All Passed	No Issue	Passed
11	withdrawHelper	write	Passed	All Passed	No Issue	Passed
12	transferBaseToken	write	Passed	All Passed	No Issue	Passed
13	claimProtocolFee	write	Passed	All Passed	No Issue	Passed
14	getBaseTokenValuation	read	Passed	All Passed	No Issue	Passed
15	getLatestPrice	read	Passed	All Passed	No Issue	Passed

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored

Audit Findings

Critical

No high 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

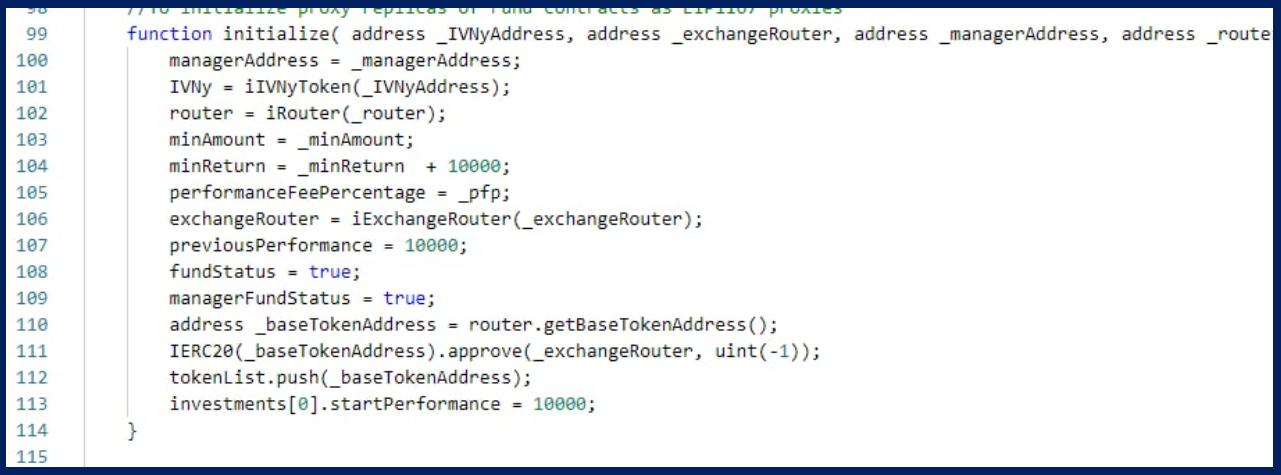
All hard coded addresses/bytes should be double checked before deployment.

File: CloneFactory.sol

For security reasons we do not recommend the use of assembly code, method/math would be more appropriate. This particular file is identical to the one available in the public domain, none of which reported any vulnerabilities. Note, the team should remember to check its hard coded bytes before deploying for production.

File: FundDec.sol

1. Line 99, no required/modifier defined for function "initialize" and the function is external. The number of times and who will call this function needs to be considered.



```
99    function initialize( address _IVNyAddress, address _exchangeRouter, address _managerAddress, address _route
100       managerAddress = _managerAddress;
101       IVNy = iIVNyToken(_IVNyAddress);
102       router = iRouter(_router);
103       minAmount = _minAmount;
104       minReturn = _minReturn  + 10000;
105       performanceFeePercentage = _pfp;
106       exchangeRouter = iExchangeRouter(_exchangeRouter);
107       previousPerformance = 10000;
108       fundStatus = true;
109       managerFundStatus = true;
110       address _baseTokenAddress = router.getBaseTokenAddress();
111       IERC20(_baseTokenAddress).approve(_exchangeRouter, uint(-1));
112       tokenList.push(_baseTokenAddress);
113       investments[0].startPerformance = 10000;
114   }
115
```

Update: After a discussion with the dev team, their response was as follows: “This was due to the inherent design of EIP 1167: Minimal Proxy Contracts, however we shall add a check to make it callable only once per clone contract deployment.” Hence it will be one step more logically solid.

2. Line 216, 313, 323 - the loop will fail unless the tokenIds.length is limited by default.

```
208     //Manager: To trasfer invested amount to fund for active utilization
209     function transferToVault() external {
210         require(msg.sender == managerAddress || msg.sender == IVNy.investinAddress(), "Manager/investin can
211         require(AmountInRouter > 0, "Nothing to transfer");
212         uint performance;
213         uint16 count;
214         performance = router.transferToVaultHelper(managerAddress, AmountInRouter);
215
216         for(uint i = tokenIdsLength; i < tokenIds.length; i++){
217             if(investments[tokenIds[i]].amount != 0){//To check for investments that didn't send back IVNy
218                 investments[tokenIds[i]].startPerformance = uint128(performance);
219                 count++;
220             }
221         }
222
```

Update: The dev team confirmed that it is limited by project plans, hence this is now ok.

3. The use of safeMath is recommended specially for '-' operation, for example in line 244, 246, 265, 270 and many others where '-' s are used. This will not be an issue if by default the overflow will never occur.

```
229     function claimPerformanceFee() external {
230         require(msg.sender == managerAddress, "Only Manager can call this");
231             uint128 amount = investments[0].amount;
232             uint128 startPerformance = investments[0].startPerformance;
233             uint _performanceFee;
234             uint baseTokenValue;
235             (_performanceFee, amount, startPerformance, baseTokenValue) = router.getInvestmentValue(addre
236             amount+=uint128(_performanceFee);
237
238             IERC20 token = IERC20(tokenList[0]);
239
240             require(token.balanceOf(address(this))>amount, "Not enough Balance in Base Token");
241
242             token.transfer(managerAddress, (amount*9)/10);
243             token.transfer(IVNy.investinVault(), amount/10);
244             totalAmount  = uint128(baseTokenValue-amount);
245             delete investments[0].amount;
246             --numberOfActiveInvestments;
247             previousPerformance = startPerformance;
248
249     }
```

```
257     function withdraw(uint _tokenId, uint128 _amount, address tokenOut) external {
258         address investorAddress  = IVNy.ownerOf(_tokenId);
259         require(msg.sender == investorAddress, "Only Investor can end");
260         require(router.getTokenMapping(tokenOut) == true || tokenOut == address(0), "tokenOut not valid");
261
262         if(investments[_tokenId].startPerformance == 0){
263             if(_amount >= investments[_tokenId].amount){
264                 router.transferBaseToken(msg.sender, investments[_tokenId].amount);
265                 AmountInRouter -= investments[_tokenId].amount;
266                 IVNy.updateToken(_tokenId, address(0));
267             } else {
268                 router.transferBaseToken(msg.sender, _amount);
269                 investments[_tokenId].amount -= _amount;
270                 AmountInRouter -= _amount;
271             }
272         }
```

Update: the dev team´s response "Subtraction overflows are neither intended nor possible in places where '–' is used, for instance in line 244, amount can never be greater than baseTokenValue. In addition to it, we have used uint (-1) as an indicator for maximum value of a uint256 to get infinite approvals on tokens being utilized by the contracts. We would also re-evaluate code size limits to include safeMath checks if required."

This is fine if unitX (where x is the size of a unit) for overflow is double checked. This would provide more confidence from a fail-safe perspective.

File:FundFactoryDecBeP20.sol

No issue found

File:InvestinMulticall.sol

No issue found (if loop fails in view function there is no harm, dependent logic can be checked if used in other places).

Update: It has now been clarified that the intended purpose of the subject function is to provide an easier integration to the UI.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is ready to go for production, according to the above updates.

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. However we discussed as much as possible to assure the intended goal to frame/cover max use cases under given time.

The security state of the reviewed contract is now “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 the blockchain. 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

Email: info@rdauditors.com

Website: www.rdauditors.com