



**RociFi**

**Protocol**

**SMART CONTRACT AUDIT**

**31.12.2021**

**Made in Germany by Chainsulting.de**



## Table of contents

1. Disclaimer.....	4
2. About the Project and Company .....	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level .....	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology .....	8
4.2 Used Code from other Frameworks/Smart Contracts .....	9
4.3 Tested Contract Files .....	11
4.4 Metrics / CallGraph.....	13
4.5 Metrics / Source Lines & Risk.....	14
4.6 Metrics / Capabilities .....	15
4.7 Metrics / Source Unites in Scope .....	16
5. Scope of Work.....	19
5.1 Manual and Automated Vulnerability Test.....	20
5.1.1 Unlimited borrowing .....	20
5.1.2 Missing lending logic.....	21
5.1.3 Missing test cases .....	21
5.1.4 Divide before multiplying .....	22
5.1.5 Mathematical units .....	22
5.1.6 Redundant if statement .....	23
5.1.7 Spelling and Grammatical Errors.....	23
5.1.8 Naming conventions .....	25



5.1.9 Unrestricted visibility .....	26
5.1.10 Public functions should be declared as external.....	26
5.1.11 Floating compiler version.....	28
5.1.12 Unexplicit uint types.....	28
5.1.13 Unused imports .....	29
5.1.14 Check for Boolean equality.....	29
5.1.15 Missing inheritance .....	30
5.1.16 Wrong interface definition .....	30
5.1.17 Unused variables .....	31
5.1.18 Unclear variable names .....	31
5.2 Verify claims .....	32
5.3 Unit Tests.....	34
6. Executive Summary.....	35
7. Deployed Smart Contract .....	35

## 1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of ROCIFI LABS PTE. LTD. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (14.12.2021)	Layout
0.2 (16.12.2021)	Test Deployment
0.5 (19.12.2021)	Automated Security Testing Manual Security Testing
0.6 (20.12.2021)	Testing SWC Checks
0.7 (27.12.2021)	Verify Claims
0.9 (28.12.2021)	Summary and Recommendation
1.0 (31.12.2021)	Final document
1.1 (TBA)	Added deployed contract addresses

## 2. About the Project and Company

### Company address:

ROCIFI LABS PTE. LTD.  
1 GEORGE STREET #10-01  
ONE GEORGE STREET  
SINGAPORE (049145)

**Website:** <https://roci.fi>

**Twitter:** <https://twitter.com/rocifi>



## 2.1 Project Overview

RociFi is building a decentralized credit economy with transferable, blockchain-native credit scores designed to facilitate zero and under-collateralized lending. At its core, the protocol leverages on-chain data, Machine Learning, and loan risk-management to effectively and profitably facilitate under-collateralized loans via the blockchain.

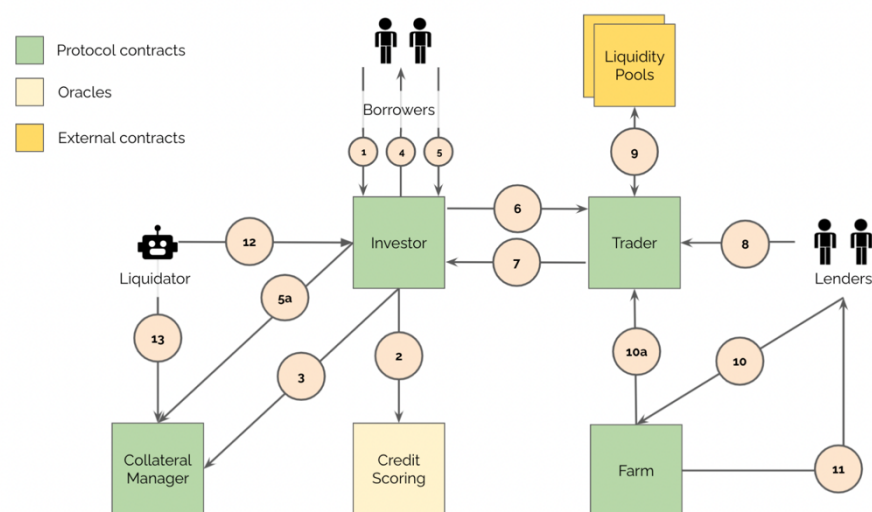
This innovative approach seeks to usher in a new wave of capital efficiency and revenue growth in DeFi lending. RociFi consists of on-chain lending protocol and credit score (off-chain, accessible via Chainlink oracle).

The main smart contracts that comprise the RociFi lending architecture are Bond, Investor, Trader, Payment and Collateral Manager. The smart contracts that comprise the liquidity management architecture are Liquidity Pools (LP) and Farms.

### Lending protocol actors

**Borrowers** - participants who want to raise money from the protocol against a certain amount of collateral.

**Lenders** - participants who want to lend money to the protocol to receive fixed returns by mining liquidity in the liquidity pools (staking the ERC-20 LP token).



### 3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

## 4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

### 4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
  - i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
  - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
  - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
  - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
  - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.



## 4.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol	<a href="https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol">https://github.com/smartcontractkit/chainlink/blob/develop/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol</a>
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/access/OwnableUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/access/OwnableUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/proxy/utils/Initializable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/proxy/utils/Initializable.sol</a>
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/proxy/utils/UUPSUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/proxy/utils/UUPSUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/security/PausableUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/security/PausableUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/ERC721Upgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/ERC721Upgradeable.sol</a>
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721BurnableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/extensions/ERC721BurnableUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/extensions/ERC721BurnableUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol</a>
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/utils/CountersUpgradeable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/tree/v4.4.0/contracts/utils/CountersUpgradeable.sol</a>

Dependency / Import Path	Source
@openzeppelin/contracts/access/Ownable.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/access/Ownable.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/access/Ownable.sol</a>
@openzeppelin/contracts/token/ERC1155/ERC1155.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/ERC1155.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/ERC1155.sol</a>
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/IERC1155.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/IERC1155.sol</a>
@openzeppelin/contracts/token/ERC1155/utils/ERC1155Holder.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/utils/ERC1155Holder.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC1155/utils/ERC1155Holder.sol</a>
@openzeppelin/contracts/token/ERC20/IERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/IERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/IERC20.sol</a>
@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/utils/SafeERC20.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/token/ERC20/utils/SafeERC20.sol</a>
@openzeppelin/contracts/utils/cryptography/ECDSA.sol	<a href="https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/utils/cryptography/ECDSA.sol">https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.2.0/contracts/utils/cryptography/ECDSA.sol</a>

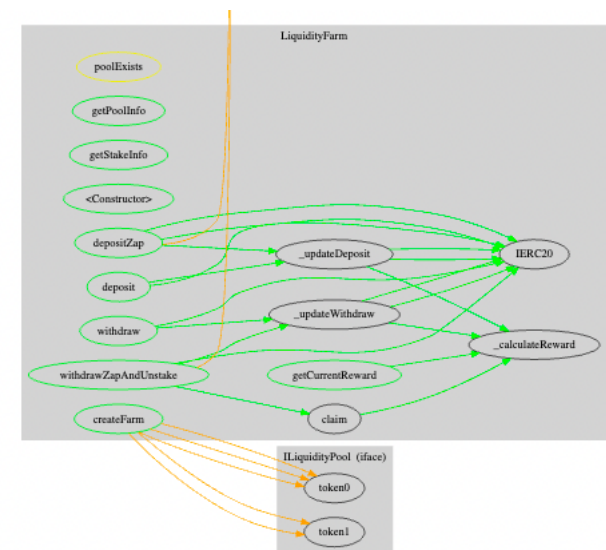
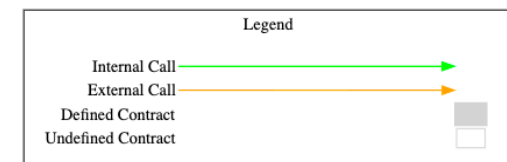
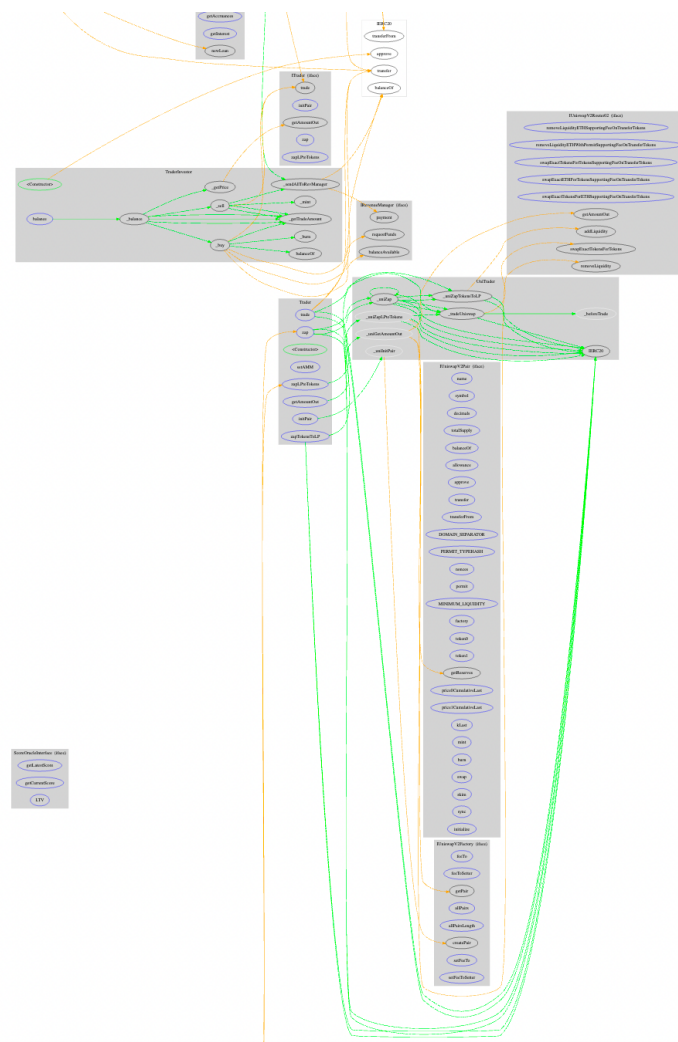
## 4.3 Tested Contract Files

The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./contracts/interfaces/ScoreOracleInterface.sol	92396d4100dcfde1d78fb59ca558e714
./contracts/interfaces/IBonds.sol	6cb858efe58b4dc46d7549a9b19e6138
./contracts/interfaces/ICollateralManager.sol	074f7e5a8e2a1cb1426f439ce6d05f92
./contracts/interfaces/IERC20PaymentStandard.sol	ea2cbc9f03341b38fd003ca9f4112093
./contracts/interfaces/IPriceFeed.sol	12b1fee0188933e29b24d9b9b9bce6dc
./contracts/interfaces/ITrader.sol	b5ef4d710a0d1ad3b73bb55f920ad3d2
./contracts/interfaces/revManager/IRevenueManager.sol	a07ff63e4945eb020155dc9284786ca0
./contracts/interfaces/revManager/IPaymentSplitter.sol	38e3b19bc36b80ccbeace5eaa1e417dc
./contracts/interfaces/uniswap/IUniswapV2Pair.sol	3fa31c3860f2b9585c3a98c64850829e
./contracts/interfaces/uniswap/IUniswapV2Factory.sol	ec39b5372f4f4d52101b4da561e2aee6
./contracts/interfaces/uniswap/IUniswapV2Router02.sol	3fd36426274a5b35f7e702f87ba08a8e
./contracts/interfaces/uniswap/IUniswapV2Router01.sol	95d8729b2c698dc663322167b3c085fe
./contracts/farming/LiquidityFarm.sol	3a8c83cf28a2211af210d396d6c3669f
./contracts/Globals.sol	52dedc94503f57c0faa111973ab7a98c
./contracts/RevenueManager/RevenueManager.sol	46b6ecfeed678c785572ca0e7d29285d
./contracts/RevenueManager/StakingStub.sol	fde9438497174a6db1d6a5d126709502
./contracts/RevenueManager/PaymentSplitter.sol	f290a85c513f8fd586c113a18a32deb2
./contracts/NFCS/RociCreditTokenInterface.sol	561741297ca996cfe92c31af297497f9
./contracts/NFCS/RociCreditTokenV2.sol	d317d4b2bf930025edc1d2ecad2ec196
./contracts/PriceFeed/PriceFeed.sol	e7ce43119a8b5dc498adba49aacd7194
./contracts/collateralManager/CollateralManager.sol	543e402d472a3d1c50caec512b0eb1f1
./contracts/trader/UniTrader.sol	ecd724aa1ccac9944a5adeba09fe8ee8
./contracts/trader/Trader.sol	dc64df5827213c6dd9de6e8ffc3d73f7

./contracts/Bonds/Bonds.sol	16f3e621d9faa78743a04518b4849da0
./contracts/investorContract/ERC20CollateralPayment.sol	43a8e8366c525c1e1504cefa3bd8def8
./contracts/investorContract/ERC20PaymentStandard.sol	95a584d783b1b7d967a60e04bf5f4ad0
./contracts/investorContract/TraderInvestor.sol	01f7eedf77d8c967c1482d187f0b241b
./contracts/investorContract/Investor.sol	9fce2482cfbce4c039fa04c66cec3771

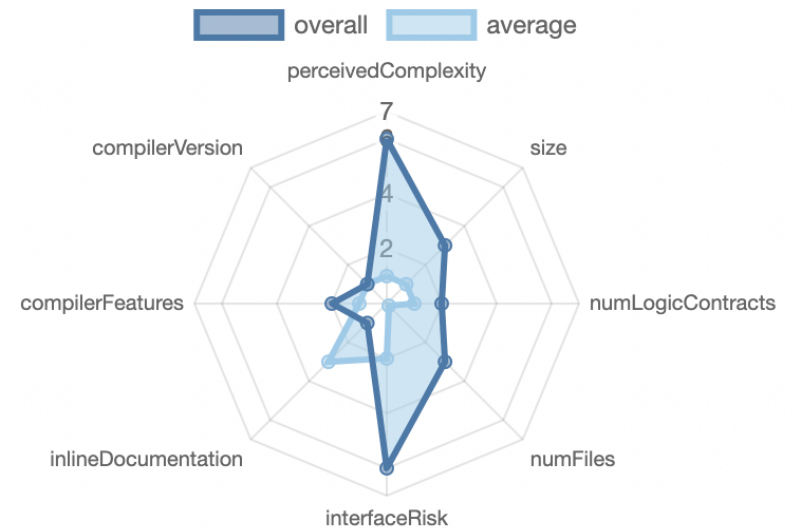
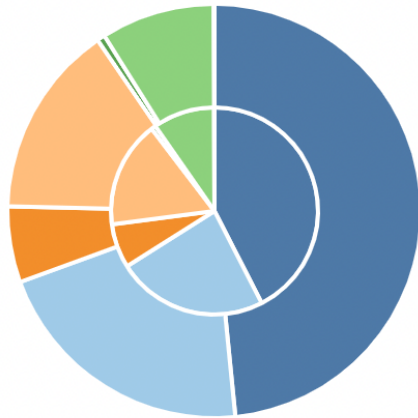
Source: <https://github.com/loan-wolf/RociFi-audit> (commit 785d16827a236c508d876623f5b9e4bf845ccd11)



<https://chainsulting.de/wp-content/uploads/2021/12/solidity-metrics-rocifi.html>

## 4.5 Metrics / Source Lines & Risk

source comment single block mixed  
empty todo blockEmpty





## 4.6 Metrics / Capabilities


Solidity Versions observed		 Experimental Features	 Can Receive Funds	 Uses Assembly	 Has Destroyable Contracts
<div><div>^0.8.0</div><div>&gt;=0.5.0</div><div>&gt;=0.6.2</div></div>			<div>yes</div>	<div>yes</div> <div>(1 asm blocks)</div>	<div></div>
 Transfers ETH	 Low-Level Calls	 DelegateCall	 Uses Hash Functions	 ECRrecover	 New/Create/Create2
<div>yes</div>	<div></div>	<div></div>	<div>yes</div>	<div>yes</div>	

### Exposed Functions











This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable				
190	4				
External	Internal	Private	Pure	View	
140	162	12	18	67	

















### StateVariables















Total	 Public
36	23

## 4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complexity Score	Capabilities
	contracts/interfaces/ScoreOracleInterface.sol	_____	1	11	6	3	2	7	_____
	contracts/interfaces/IBonds.sol	_____	1	25	7	4	1	21	_____
	contracts/interfaces/ICollateralManager.sol	_____	1	13	10	3	6	7	_____
	contracts/interfaces/IERC20PaymentStandard.sol	_____	1	124	27	17	71	27	_____
	contracts/interfaces/IPriceFeed.sol	_____	1	9	8	5	1	3	_____
	contracts/interfaces/ITrader.sol	_____	1	10	5	3	1	11	_____
	contracts/interfaces/revManager/IRvenueManager.sol	_____	1	11	8	4	1	7	_____
	contracts/interfaces/revManager/IPaymentSplitter.sol	_____	1	8	6	3	1	3	_____
	contracts/interfaces/uniswap/IUniswapV2Pair.sol	_____	1	52	7	5	_____	55	_____
	contracts/interfaces/uniswap/IUniswapV2Factory.sol	_____	1	17	6	4	_____	17	_____



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex Score	Capabilities
	contracts/interfaces/uniswap/IUniswapV2Router02.sol	_____	1	44	6	4	_____	16	
	contracts/interfaces/uniswap/IUniswapV2Router01.sol	_____	1	95	4	3	_____	48	
	contracts/farming/LiquidityFarm.sol	1	2	252	198	141	40	95	_____
	contracts/Globals.sol	_____	_____	6	6	4	2	_____	_____
	contracts/RevenueManager/RevenueManager.sol	1	_____	52	52	26	20	32	
	contracts/RevenueManager/StakingStub.sol	1	_____	6	6	3	2	1	
	contracts/RevenueManager/PaymentSplitter.sol	1	_____	78	78	38	29	41	_____
	contracts/NFCS/RociCreditTokenInterface.sol	_____	1	35	6	3	7	13	_____
	contracts/NFCS/RociCreditTokenV2.sol	1	_____	377	289	188	68	162	
	contracts/PriceFeed/PriceFeed.sol	1	_____	41	41	25	10	14	_____
	contracts/collateralManager/CollateralManager.sol	1	_____	121	108	58	39	41	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complexity Score	Capabilities
	contracts/trader/UniTrader.sol	1	_____	151	151	88	40	79	_____
	contracts/trader/Trader.sol	1	_____	168	135	66	52	64	
	contracts/Bonds/Bonds.sol	1	_____	277	240	124	89	59	
	contracts/investorContract/ERC20CollateralPayment.sol	1	_____	207	187	98	66	68	_____
	contracts/investorContract/ERC20PaymentStandard.sol	1	_____	292	259	144	99	57	
	contracts/investorContract/TraderInvestor.sol	1	_____	141	141	78	48	69	
	contracts/investorContract/Investor.sol	1	1	179	155	77	64	51	
	<b>Totals</b>	<b>14</b>	<b>16</b>	<b>2802</b>	<b>2152</b>	<b>1219</b>	<b>759</b>	<b>1068</b>	

Legend: [ ]

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

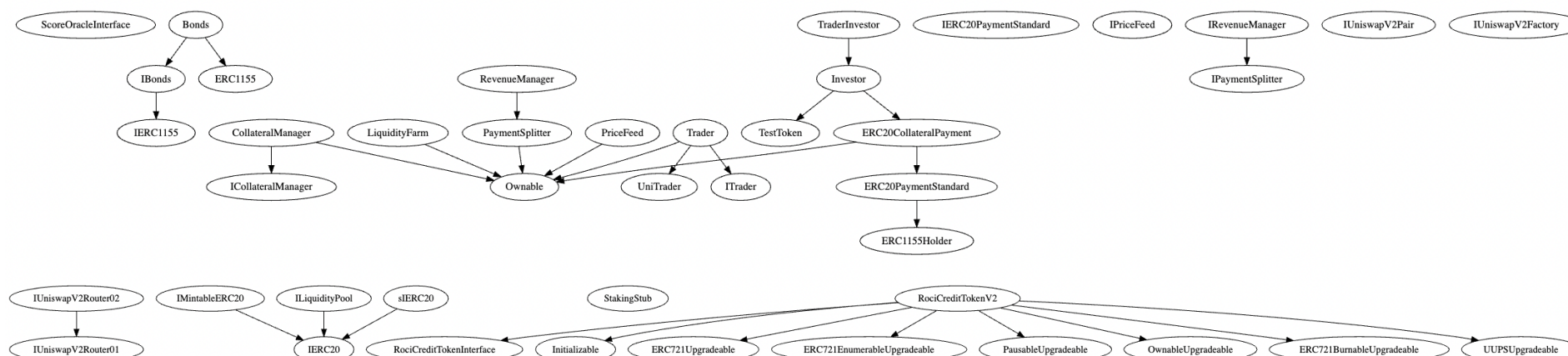
## 5. Scope of Work

The RociFi Team provided us with the files that needs to be tested. The scope of the audit are the RociFi protocol contracts.

The team put forward the following assumptions regarding the security, usage of the contracts:

- The smart contract is coded according to the newest standards and in a secure way
- ScoreDB and NFCS Credit score flow can't be manipulated or gamified by malicious actors
- Lenders are allowed to sell their ERC-20 debt tokens (rDAIx) and withdraw their capital anytime
- The collateral is correctly calculated and can't be manipulated or gamified by malicious actors
- Lenders receive profits from their investments via RociFi yield farms that distribute earned interest via rewards, that profits are correctly calculated.
- RociFi Protocol cannot be effected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzx

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.



## 5.1 Manual and Automated Vulnerability Test

### CRITICAL ISSUES

#### 5.1.1 Unlimited borrowing

Severity: CRITICAL

Status: ACKNOWLEDGED

File(s) affected: Investor.sol, HardcodedCreditScores.sol

Attack / Description	Code Snippet	Result/Recommendation
The current implementation is a borrow function to borrow a given token amount for a given collateral. In addition, there is no logic for the CreditScores. Any user can borrow any amount of tokens for any amount of collateral.	NA	It is recommended to restrict the amount of borrowed tokens related to the collateral and implement a good credit scoring to limit the amount of asset to be borrowed.

## HIGH ISSUES

### 5.1.2 Missing lending logic

Severity: HIGH

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current implementation has no logic for lending tokens to the protocol. The protocol does not work if there is no lending logic and nothing to lend.	NA	It is recommended to add lending logic to the contracts to check borrowing against lending funds.

## MEDIUM ISSUES

### 5.1.3 Missing test cases

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The overall test coverage is limited. The current implementation has only a few unit-tests which cover a fraction of the contract code.	NA	It is highly recommended to write testcases for every function and control the behaviour at any time of contract executions. It is necessary to check the desired functionality of executed code. Tests are

		also a good proof, that the written code is working in the intended way.
--	--	--

#### 5.1.4 Divide before multiplying

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: Bonds.sol

Attack / Description	Code Snippet	Result/Recommendation
Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.	line 180- 181: return(periodsStaked * (( _stakingAmount * _interest) / ONE_HUNDRED_PERCENT));	We highly recommend ordering multiplications before division.

#### 5.1.5 Mathematical units

Severity: MEDIUM

Status: ACKNOWLEDGED

File(s) affected: Investor.sol

Attack / Description	Code Snippet	Result/Recommendation
The current implementation has a calculation for time units.	line 87: uint256 accrualPeriod = 60 * 60 * 24 * 30;	It is highly recommended to use solidity integrated time units to increase code readability and avoid calculation error. For accrualPeriod the unit 30 days should be used.  <a href="https://docs.soliditylang.org/en/v0.8.10/units-and-global-variables.html#time-units">https://docs.soliditylang.org/en/v0.8.10/units-and-global-variables.html#time-units</a>

--	--	--

## LOW ISSUES

### 5.1.6 Redundant if statement

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: Bonds.sol

Attack / Description	Code Snippet	Result/Recommendation
The current implementation has a redundant if statement.	<pre> line 165 -169:     bool r;     if (pc.addInterest(toMint, id)) {         r = true;     } else {         r = false;     } </pre>	It is highly recommended to remove the redundant if statement and set the variable directly equal to the return parameter.

### 5.1.7 Spelling and Grammatical Errors

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: NA

Attack / Description	Code Snippet	Result/Recommendation
Spelling and grammatical errors were identified in the	Spelling errors are formatted as follows:	<u>Documentation (gitbook):</u>



<p>codebase. Fixing these mistakes can help improve the end-user experience by providing clear information on errors encountered and improve the maintainability and auditability of the codebase.</p>	<p>Occurrence: wrong spelling (how it is) → correct spelling (how it should be)</p>	<p>Bond &gt; Overview &gt; 7<sup>th</sup> paragraph: Alice's loan is for 20,000 → 10,000  Bond &gt; Bond.sol &gt; stake(): ammount → amount  NFCS &gt; RociCreditToken.sol: non-trasnferable → non-transferrable  ScoreDB &gt; ScoreDB: is contract → ScoreDB is a contract</p> <p><u>Contracts:</u>  Bonds.sol:  line 23: ustake → unstake , itterate → iterate  line 24: itterate → iterate  line 28, 59, 80, 129, 131, 152: ammount → amount  line 145 varaibles → variables</p> <p>CollateralManager.sol:  line 107: based off → based of  line 110: withdarawl → withdraw  line 111: withdrawln collateral → withdrawn collateral  line 114: cannot withdrawal → cannot withdraw</p> <p>LiquidityFarm.sol:  line 21: in given in → is given in</p> <p>IERC20PaymentStandard:  line 28: based off → based of  line 45, 98, 106: ammount → amount  line 98: _erc20Ammount → _erc20Amount</p> <p>ERC20PaymentStandard:  line 13: overriden, flexible → overridden, flexible  line 107, 211, 237: ammount → amount</p>
--	---	---



		<p>line 237, 239: <code>_erc20Ammount</code> → <code>_erc20Amount</code></p> <p>ERC20CollateralPayment:  line 16: of a override of ERC20PaymentStandard → of an overridden ERC20PaymentStandard  line 16: to be be added → to be added  line 35: <code>collaterall</code> → <code>collateral</code>  line 36: <code>ammount</code> → <code>amount</code>  line 197: <code>overriden</code> → <code>overridden</code></p> <p>Investor:  line 71: <code>durration</code> → <code>duration</code>  line 117, 119: <code>_erc20Ammount</code> → <code>_erc20Amount</code></p>
--	--	---

### 5.1.8 Naming conventions

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: NA

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several solidity naming conventions are violated. Function parameters are not declared consistently (sometime with underscore sometimes without) and structs as well as constants are written in lower case.	<p>Constants:  Bonds.head (line 40)  ERC20PaymentStandard.poolScore (line 16)  Investor.interestRateAnnual (line 26)  UniTrader.uniswap (line 19)</p> <p>Structs:  Bonds.node (line 33)  CollateralManager.collateral (line 15)  IERC20PaymentStandard.loan (line 12)</p>	<p>It is recommended to use a consistent naming of function parameters (starting with underscores most common). It is also recommended to write constants in upper case and structs in caps words style. In general, it's recommended to follow the official solidity style guide for a better readable code.</p> <p><a href="https://docs.soliditylang.org/en/v0.8.11/style-guide.html#naming-conventions">https://docs.soliditylang.org/en/v0.8.11/style-guide.html#naming-conventions</a></p>

	Variables: Bonds.IOU.ID	
--	----------------------------	--

#### 5.1.9 Unrestricted visibility

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: RevenueManager.sol, LiquidityFarm.sol, RevenueManager.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions could be a restricted visibility.	RevenueManager.balanceAvailable should be restricted to view LiquidityFarm.calculateReward should be restricted to pure RevenueManager.balanceAvailable should be restricted to view	It is recommended to restrict the function visibility to the most restrict type as possible to enhance code readability.

#### 5.1.10 Public functions should be declared as external

Severity: LOW

Status: ACKNOWLEDGED

File(s) affected: LiquidityFarm.sol, ERC20PaymentStandard.sol, PriceFeed.sol, ERC20CollateralPayment.sol, Investor.sol ,  
TraderInvestor.sol, RociCreditTokenV2.sol, PaymentSplitter.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several functions are declared as public where they could be external. For public functions	LiquidityFarm.getpoolInfo (line 54) LiquidityFarm.getStakeinfo (line 64) LiquidityFarm.getCurrentReward (line 84) LiquidityFarm.createFarm (line 108)	We recommend declaring functions as external if they are not used internally. This leads to lower gas consumption and better code readability.

<p>Solidity immediately copies array arguments to memory, while external functions can read directly from calldata. Because memory allocation is expensive, the gas consumption of public functions is higher.</p>	<p>LiquidityFarm.deposit (line 133)  LiquidityFarm.depositZap (line 142)  LiquidityFarm.withdraw (line 183)  LiquidityFarm.withdrawZapAndUnstake (line 192)  ERC20PaymentStandard.issueBonds (line 77)  ERC20PaymentStandard.getLoanInfo (line 135)  ERC20PaymentStandard.isDelinquent (line 156)  ERC20PaymentStandard.configureNew (line 173)  ERC20PaymentStandard.payment (line 239)  ERC20CollateralPayment.addCollateral (line 39)  ERC20CollateralPayment.issueBonds (line 69)  Investor.payment (line 119)  TraderInvestor.payment (line 60)  RociCreditTokenV2.initialize (line 40)  RociCreditTokenV2.pause (line 50)  RociCreditTokenV2.unpause (line 57)  RociCreditTokenV2.getToken (line 208)  RociCreditTokenV2.remintToken (line 269)  RociCreditTokenV2.version (line 284)  RociCreditTokenV2.tokenExistence (line 288)  PriceFeed.getLatestPriceUSD (line 29)  PaymentSplitter.payment (line 64)</p>	
--	--	--

## INFORMATIONAL ISSUES

### 5.1.11 Floating compiler version

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: All

Attack / Description	Code Snippet	Result/Recommendation
The current pragma solidity directive is floating. 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.	pragma solidity ^0.8.0	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. i.e. Pragma solidity 0.8.0  See SWC-103: <a href="https://swcregistry.io/docs/SWC-103">https://swcregistry.io/docs/SWC-103</a>

### 5.1.12 Unexplicit uint types

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: IBonds.sol, ICollateralManager.sol, IPriceFeed.sol, ITrader.sol, PaymentSplitter.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several unsigned integer variables have an unexplicit type (uint).	IBonds line 13, 17 ICollateralManager line 10, 11 IPriceFeed line 8 ITrader line 5, 7, 8, 9	It is recommended to use explicit variable declaration such as uint8 or uint256. It improves code readability and can help to make sure variables have an intended size.

	PaymentSplitter line 16, 21, 29, 38, 41, 52, 53, 64, 69	
--	---	--

#### 5.1.13 Unused imports

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: IPriceFeed.sol, ERC20PaymentStandard.sol, ERC20CollateralPayment.sol, Investor.sol, Trader.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several contracts have unused imported files.	IPriceFeed: Ownable (line 4) and AggregatorV3Interface(line 5) ERC20PaymentStandard: ScoreOracleInterface (line 8) ERC20CollateralPayment: hardhat/console (line 11) Investor: ONE_HUNDERED_PERCENT (line 8) Trader: hardhat/console (line 4)	Remove unused imports to reduce contract size.

#### 5.1.14 Check for Boolean equality

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: RociCreditTokenV2.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation several require checks are using a comparison to a Boolean constant. This leads	line 215: require(_mintedNonce[tokenOwner] == true)	It is recommended to remove the equality check to the Boolean constant and use the value itself.

to unnecessary gas consumption.		
---------------------------------	--	--

#### 5.1.15 Missing inheritance

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: ERC20PaymentStandard.sol, IERC20PaymentStandard.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation the implementing contract is not inheriting from its defining interface.	ERC20PaymentStandard.sol should inherit from IERC20PaymentStandard.sol	It is recommended to inherit from the interface to ensure functions are implemented correctly. In addition, the loan struct should not be redefined in the implementing contract.

#### 5.1.16 Wrong interface definition

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: RociCreditTokenInterface.sol

Attack / Description	Code Snippet	Result/Recommendation
The project structure contains an extra folder for interfaces. The current implementations has an interface outside of this folder violating naming conventions.	NA	Rename the RociCreditTokenInterface to IRociCreditTokenInterface and move it into the interfaces folder to clarify this file is an interface.

### 5.1.17 Unused variables

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: TraderInvestor.sol, ERC20CollateralPayment.sol, UniTrader.sol, ERC20PaymentStandard.sol, PriceFeed.sol

Attack / Description	Code Snippet	Result/Recommendation
The current implementation contains several unused variables.	ERC20PaymentStandard.poolScore (line 16) TraderInvestor._liquidate._receiver (line 69) ERC20CollateralPayment._ liquidate. _erc20Contract (line 200) ERC20CollateralPayment.claimCollateral (line 91 – 96) UniTrader._uniGetAmountOut.x (line 125) PriceFeed.getLatestPriceUSD (line 33 - 37)	We highly recommend removing unused variables to decrease storage usage and gas consumption.

### 5.1.18 Unclear variable names

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

File(s) affected: Bonds.sol, CollateralManager.sol, ERC20PaymentStandard.sol, ERC20CollateralPayment.sol

Attack / Description	Code Snippet	Result/Recommendation
In the current implementation are several variables named with one or two letters. It is hard to understand what they stand for. A variable name like amm (for amount) is also misleading and could be understood as automated marked maker.	Bonds: IOU(26), lITail(43), n(75), pc(114, 156, 200), amm(117), _amm(133), amm (152), r(163), iou(197), _del(262) CollateralManager: c (115) ERC20PaymentStandard: ln (141, 244) ERC20CollateralPayment: ln (168)	It is recommended to use full names for variables to make clear what they stand for and increase code readability. It is better to write <i>linkedList</i> instead of <i>ll</i> .

## 5.2 Verify claims

**5.2.1** ScoreDB and NFCS Credit score flow can't be manipulated or gamified by malicious actors

**Status:** There is no ScoreDB implementation in the contracts.

**5.2.2** Lenders are allowed to sell their ERC-20 debt tokens (rDAIx) and withdraw their capital anytime

**Status:** Lending logic is missing in the contracts.

**5.2.3** The collateral is correctly calculated and can't be manipulated or gamified by malicious actors

**Status:** Collateral is not calculated. Borrowers can borrow any amount of tokens for any collateral.

**5.2.4** Lenders receive profits from their investments via RociFi yield farms that distribute earned interest via rewards, that profits are correctly calculated.

**Status:** Users can earn profits by staking tokens in the yield farm. The profits for staking tokens in the yield farm are calculated as follows:

```
250      return (amount * duration * apy) / ONE_YEAR / ONE_HUNDRED_PERCENT;
```

**5.2.5** The smart contract is coded according to the newest standards and in a secure way

**Status:** tested and verified **X**

**5.2.6** RociFi Protocol cannot be affected to hacks which happened to other lending platforms such as Celsius, Cream Finance or bzX

**Status:**

1. Celsius :

Celsius network was effected by the BadgerDAO hack in december 2021. This hack was a front-end attack, and the smart contracts were untouched. A hacker was able steal an active API key of cloudflare, a content delivery network. With the help of the API key, the attacker injected malicious front-end code where the user was asked to approve a malicious contract address to spend tokens. The approved contract drained all approved tokens out of the user's wallet.



This hack can potentially happen to other dApps like RociFi when lenders are asked for approval on lending. Users commonly do not double check the approved address and just allow the spending of tokens. To avoid this scenario, it is important to keep the front-end as safe as the contracts. It is recommended to update API keys and access keys like SSH keys regularly and keep them safe.

## 2. Cream Finance:

In the cream hack in October 2021 \$130m assets of the lending platform were stolen. The smart contracts have been outplayed with a flash loan attack. A borrower flash borrowed funds from Maker and borrowed loan tokens from cream vault multiple times. He manipulated the prices by burning the tokens.

This attack can happen to the RociFi protocol. RociFi can mitigate that risk with disabling multiple borrow transaction for one account (ex. within a block) .

## 3. bzX:

On the latest bzX hack, the private key of the deploying account was compromised by an attacker. An employer stored the mnemonic of the deployer wallet locally on his machine and received an email with a word document. The file ran a malicious macro and stole the mnemonic. With the privilege and funds of the deployer wallet, the attacker could drain out funds from the protocol by using the protocols tokens.

This attack could potentially happen to the RociFi platform as well. To prevent this scenario, it is recommended to have a security guideline that ensures the safe storage of private keys. Keys should never be stored on a machine that is used for network interactions such as customer requests. It is recommended to store the privileged private key securely offline. Other options can be multisig Wallets, such as Gnosis Safe.

## 5.3 Unit Tests

```
Borrowing and repayment flow
  ✓ Borrowing happy path (819ms)
  ✓ Repayment (621ms)

Deposit use case
  ✓ Farm created
  ✓ Deposit (84ms)

Liquidation
  1) "before each" hook for "Liquidation"

Liquidity farming
  ✓ Create farm
  ✓ Simple deposit (44ms)
  ✓ Test multiple deposits and getRewards (72ms)

List deposits use case
  ✓ Farms created
  ✓ Deposited
  - Can view deposits

Withdraw use case
  ✓ Farm created
  ✓ Deposited
  ✓ Withdraw (69ms)

12 passing (1m)
1 pending
1 failing

1) Liquidation
   "before each" hook for "Liquidation":
     Error: Timeout of 20000ms exceeded. For async tests and hooks, ensure "done()" is called; if returning a
     Promise, ensure it resolves. (. /RociFi-audit-main/test/Liquidation.js
)
    at listOnTimeout (internal/timers.js:557:17)
    at processTimers (internal/timers.js:500:7)
```

## 6. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase. The final debriefs took place on the December 28, 2021.

The main goal of the audit was to verify the claims regarding the security of the smart contract. During the audit, critical and high issues were found, after the manual and automated security testing and not all claims have been successfully verified. Please check all issues and get back to your auditor if issues have been fixed.

## 7. Deployed Smart Contract

PENDING

