Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Vectous Token
Platform: Binance Smart Chain
Language: Solidity
Date: January 23rd, 2023

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Vectous team to perform the Security audit of the Vectous Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on January 23rd, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Vectous is a standard ERC20 token smart contract. This audit only considers Vectous token smart contracts, and does not cover any other smart contracts on the platform.
- Vectous Contracts have functions like NewEntry, EntryAction, CollectEntryProfit, collectLendReturn, lendOnDeposit, withdrawLoanedFunds, updateFinishedLoan, etc.

Audit scope

Name	**Code Review and Security Analysis Report for Vectous Token Smart Contract**
Platform	**BSC / Solidity**
File	Vectous.sol
File MD5 Hash	755bc1b75a92b29395550520d77e55ad
Audit Date	January 23rd, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Avarice • Symbol: BAVC • Decimals: 18 • Minimum Return Percent: 5% • Maximum Return Percent: 25% • Profit Global Virtual Amount: 15% • G Rate:10%	**YES, This is valid.**
Owner Specifications: • Owner can set a global extra virtual amount percentage that adds up to all the new deposits. • Owner can set a global extra virtual amount % that adds up to new deposits created from the user deposit's profit. • Owner can switch the loaning feature status. • Owner can set a new return percentage.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Poor Secured"**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 2 critical, 1 high, 0 medium and 1 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **Failed**

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Vectous Token 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 in the Vectous Token.

The Vectous Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

Documentation

We were given a Vectous Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are well commented on. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Require statements doest return error messages	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	_transferOwnership	internal	Passed	No Issue
7	name	read	Passed	No Issue
8	symbol	read	Passed	No Issue
9	decimals	read	Passed	No Issue
10	totalSupply	read	Passed	No Issue
11	balanceOf	read	Passed	No Issue
12	transfer	write	Passed	No Issue
13	allowance	read	Passed	No Issue
14	approve	write	Passed	No Issue
15	transferFrom	write	Passed	No Issue
16	increaseAllowance	write	Passed	No Issue
17	decreaseAllowance	write	Passed	No Issue
18	_transfer	internal	Passed	No Issue
19	_mint	internal	Passed	No Issue
20	_burn	internal	Passed	No Issue
21	_approve	internal	Passed	No Issue
22	_spendAllowance	internal	Passed	No Issue
23	_beforeTokenTransfer	internal	Passed	No Issue
24	_afterTokenTransfer	internal	Passed	No Issue
25	Update_globalVirtualAmountPercent	external	Wrong Error Message	Refer Audit Findings
26	Update_fromProfit_globalVirtualAmountPercent	external	Wrong Error Message	Refer Audit Findings
27	switchLoaningStatus	external	access only Owner	No Issue
28	_globalDataCheck	write	Passed	No Issue
29	setReturnPercent	external	Wrong Error Message	Refer Audit Findings
30	NewEntry	external	Passed	No Issue
31	NewEntry_fromProfits	external	The Require statement is assigning the value instead of comparing and always return due to wrong require condition, Missing Error Message	Refer Audit Findings
32	_newEntry	internal	An _amount calculation is wrong	Refer Audit Findings

33	EntryAction	external	The Require statement is assigning the value instead of comparing and always return due to wrong require condition, Duplicate Code,Missing Error Message	Refer Audit Findings
34	CollectEntryProfit	write	The Require statement is assigning the value instead of comparing and always return due to wrong require condition, Missing Error Message, Function getting reverted	Refer Audit Findings
35	calcDepositReturn	read	Passed	No Issue
36	calc_section_timestamps_same_day	read	Passed	No Issue
37	calc_starting_section	internal	Passed	No Issue
38	calc_middle_section	internal	Passed	No Issue
39	calc_ending_section	internal	Passed	No Issue
40	calcMDP	read	Passed	No Issue
41	_generateNextDepositID	write	Passed	No Issue
42	getOwnedDeposits	read	Passed	No Issue
43	getAverageReturnPerc	read	Missing Error Message	Refer Audit Findings
44	checkForROI	read	Passed	No Issue
45	getLoanOnDeposit	external	Wrong Error Message	Refer Audit Findings
46	cancelDepositLoanRequest	write	Missing Error Message	Refer Audit Findings
47	lendOnDeposit	external	Missing Error Message	Refer Audit Findings
48	withdrawLoanedFunds	external	Missing Error Message	Refer Audit Findings
49	clcLenderLendId	read	Passed	No Issue
50	collectLendReturn	external	Passed	No Issue
51	updateFinishedLoan	write	Missing Error Message, Function getting reverted	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token 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
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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

(1) The Require statement is assigning the value instead of comparing and always return due to wrong require condition:

EntryAction():

```
function EntryAction(bool noIncrease, uint256 Deposit_ID, uint256 amount) external {
    _globalDataCheck();

    require(checkForROI(msg.sender, Deposit_ID) == true);

    if (depositData_2[msg.sender][Deposit_ID].deposit_hasLoan == true) {
        updateFinishedLoan(mapRequestingLoans[msg.sender][Deposit_ID].lenderAddress, msg.sender, mapRequesting
    }

    require(depositData_2[msg.sender][Deposit_ID].deposit_hasLoan = false);
```

NewEntry_fromProfits():

```
function NewEntry_fromProfits(address referrerAddr, uint256 Deposit_ID) external {
    uint256 entryProfits = calcDepositReturn(msg.sender, Deposit_ID);
    require(entryProfits > 0);
    require(depositData_2[msg.sender][Deposit_ID].deposit_hasLoan = false);

    uint256 bonusAmount = entryProfits * fromProfit_globalVirtualAmountPercent / 100;
```

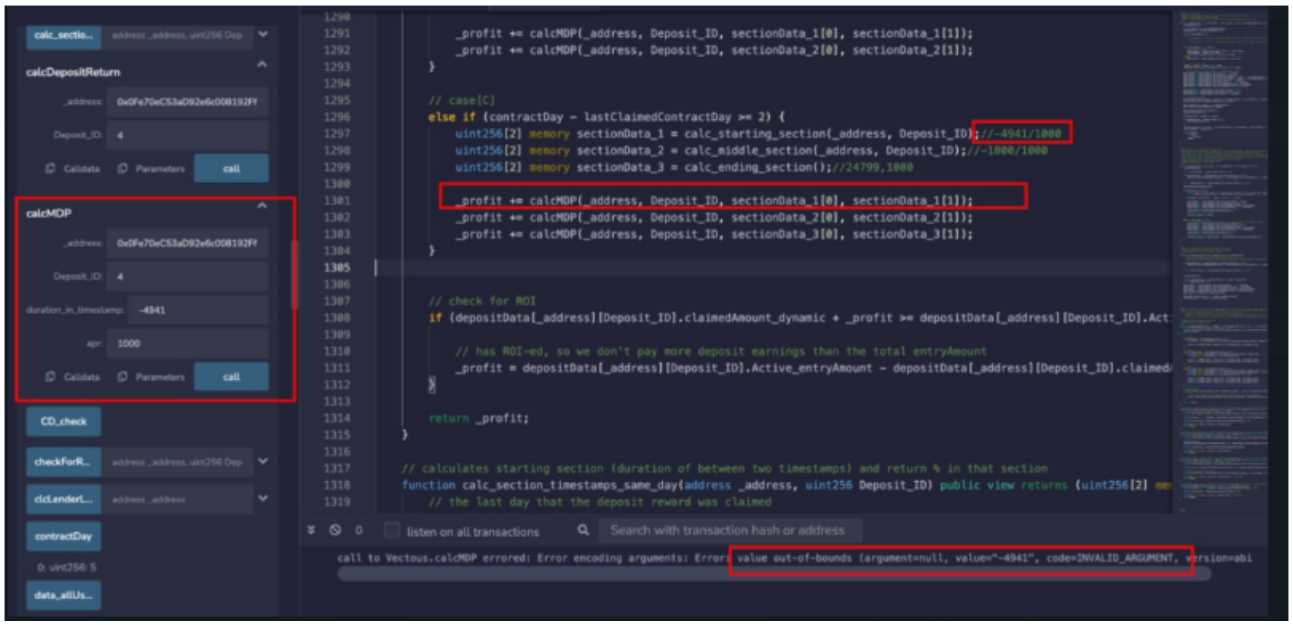
CollectEntryProfit():

```
function CollectEntryProfit(uint256 Deposit_ID) public nonReentrant {
    // require(depositData[msg.sender][Deposit_ID].ISActive == true,
    // 'the entry has been ROI-ed and now its been inactivated, the user needs to take action ("EntryAct

    if (depositData_2[msg.sender][Deposit_ID].deposit_hasLoan == true) {
        updateFinishedLoan(mapRequestingLoans[msg.sender][Deposit_ID].lenderAddress, msg.sender, mapRequ
    }

    require(depositData_2[msg.sender][Deposit_ID].deposit_hasLoan = false);
```

The Required statement in EntryAction(), NewEntry_fromProfits(), CollectEntryProfit() functions are assigning the value instead of comparing.

Resolution: We suggest using double = to for check condition.

(2) The updateFinishedLoan and CollectEntryProfit getting reverted:



Users are not able to withdraw deposits and also not able to return the loan to lenders. updateFinishedLoan and CollectEntryProfit getting reverted due to negative value from internal functions calc_starting_section and with this negative value calcMDP is reverting saying 'out-of-bounds' error.

Resolution: We suggest checking the code logic and correcting it.

High Severity

(1) An _amount calculation is wrong:

```
function _newEntry(address _referrerAddr, uint256 _amount, uint256 _virtualBonusAmount) internal {
    _globalDataCheck();

    uint256 Deposit_ID = _generateNextDepositID();
    ownedDeposits[msg.sender].push(Deposit_ID);

    uint256 referredBonus = 0;
    /*
        "referredBonus" ==> If user is referred, because of using a refferal link they will get 1% extra virtual amount on

        "referralsBonus" ==> Users by reffering others will receive 1.5% of their deposit amount saved up in a map for thei
    */

    if (_referrerAddr != msg.sender) {
        /* No Self-referred */
        depositData[msg.sender][Deposit_ID].referrer = _referrerAddr;
        referredBonus = _amount * 10 /1000; // 1%
        referralsBonus[_referrerAddr] += _amount * 15 /1000; // 1.5%
    } else {
        depositData[msg.sender][Deposit_ID].referrer = address(0);
    }

    // pay the 2.7% dev fee
    _amount = _amount - (_amount * 27 / 1000);
    token_BUSD.transfer(address(devWallet), (_amount * 27 / 1000));
```

In _newEntry, _amount calculation is wrong as devWallet will receive an incorrect amount. Because the dev fee is deducted twice from _amount.

Resolution: We suggest to re-check and correcting the code logic for _amount calculation.

Medium

No medium severity vulnerabilities were found.

Low

(1) Duplicate Code:

EntryAction function:

```
if (noIncrease == false) {
    // user chosed to add to their entry (amount must be equal to half of their Active_entryAmount)
    require(amount == (depositData[msg.sender][Deposit_ID].Active_entryAmount / 2) , "ERR: Amount Not correct");

    token_BUSD.transferFrom(msg.sender, address(this), amount);

    depositData[msg.sender][Deposit_ID].Active_entryAmount += amount;
    depositData[msg.sender][Deposit_ID].claimedAmount_dynamic = 0;
    depositData[msg.sender][Deposit_ID].lastClaimedContractDay = contractDay;
    depositData[msg.sender][Deposit_ID].lastClaimedTime = block.timestamp;
    //depositData[msg.sender][Deposit_ID].ISActive == true ;
    depositData[msg.sender][Deposit_ID].ROICounter ++;

    totalEntry_dynamic += amount;

}
else if (noIncrease == true) {
    // user chosed not to add to their entry, so we reduce their entry amount to half
    depositData[msg.sender][Deposit_ID].Active_entryAmount /= 2;
    depositData[msg.sender][Deposit_ID].claimedAmount_dynamic = 0;
    depositData[msg.sender][Deposit_ID].lastClaimedContractDay = contractDay;
    depositData[msg.sender][Deposit_ID].lastClaimedTime = block.timestamp;
    //depositData[msg.sender][Deposit_ID].ISActive == true ;
    depositData[msg.sender][Deposit_ID].ROICounter ++;

    totalEntry_dynamic -= depositData[msg.sender][Deposit_ID].Active_entryAmount /2;
}
```

In EntryAction , noIncrease true and false both conditions have duplicate code.

Resolution: We suggest keeping that code out of the if conditions.

Very Low / Informational / Best practices:

(1) Missing Error Message:

```
    */
    function Update_fromProfit_globalVirtualAmountPercent(uint256 num) external onlyOwner() {
        require(num >= 0);
        require(num <= 100);

        fromProfit_globalVirtualAmountPercent = num;
    }
```

```
function cancelDepositLoanRequest(uint256 Deposit_ID) public {
    require(depositData_2[msg.sender][Deposit_ID].deposit_hasLoan == false);

    depositData_2[msg.sender][Deposit_ID].deposit_forLoan = false;
}
```

```
function lendOnDeposit(address loanerAddress , uint256 Deposit_ID, uint256 amount) external nonReentrant {
    _globalDataCheck();

    require(loaningIsPaused == false, 'functionality is paused');
    require(depositData[loanerAddress][Deposit_ID].investor != msg.sender, 'no self lend');
    require(depositData_2[loanerAddress][Deposit_ID].deposit_hasLoan == false, 'Target deposit has an active
    require(depositData_2[loanerAddress][Deposit_ID].deposit_forLoan == true, 'Target deposit is not requesti

    uint256 loanAmount = mapRequestingLoans[loanerAddress][Deposit_ID].loanAmount;
    uint256 returnAmount = mapRequestingLoans[loanerAddress][Deposit_ID].returnAmount;
    uint256 rawAmount = amount;

    require(rawAmount == mapRequestingLoans[loanerAddress][Deposit_ID].loanAmount);

    token_BUSD.transferFrom(msg.sender, address(this), amount);
```

Some functions doest return error messages from require statements.

Functions are:

- Update_globalVirtualAmountPercent
- Update_fromProfit_globalVirtualAmountPercent
- NewEntry_fromProfits
- lendOnDeposit
- updateFinishedLoan
- EntryAction
- CollectEntryProfit
- getAverageReturnPerc
- withdrawLoanedFunds
- cancelDepositLoanRequest

- setReturnPercent

Resolution: We suggest adding a revert message for required statements.

(2) SafeMath Library is included in contract but not used:

- SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.
- SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

(3) Wrong Error Message:

```
function getLoanOnDeposit(uint256 Deposit_ID ,uint256 loanAmount, uint256 returnAmount) external {
    _globalDataCheck();

    require(loaningIsPaused == false, 'functionality is paused');
    require(returnAmount > (loanAmount * 2 /100 + loanAmount), 'loan return must at least 3% higher than loam amount');
    require(depositData[msg.sender][Deposit_ID].investor == msg.sender, 'auth failed');
    require(depositData_2[msg.sender][Deposit_ID].deposit_hasLoan == false, 'Target deposit has an active loan on it');
    require(returnAmount >= depositData[msg.sender][Deposit_ID].Active_entryAmount - depositData[msg.sender][Deposit_ID].claimedAm

    depositData_2[msg.sender][Deposit_ID].deposit_forLoan = true;

    /* data of the requesting loan */
    mapRequestingLoans[msg.sender][Deposit_ID].loanerAddress = msg.sender;
    mapRequestingLoans[msg.sender][Deposit_ID].Deposit_ID = Deposit_ID;
    mapRequestingLoans[msg.sender][Deposit_ID].loanAmount = loanAmount;
    mapRequestingLoans[msg.sender][Deposit_ID].returnAmount = returnAmount;
    mapRequestingLoans[msg.sender][Deposit_ID].loanIsPaid = false;


    emit E_getLoanOnDeposit(
        msg.sender,
        block.timestamp,
        loanAmount,
        returnAmount,
        Deposit_ID
    );
}
```

In the getLoanOnDeposit function, Loan amount is calculated with 2% and the error message is "loan return must be at least 3% higher than loan amount". So either the error message Or calculation is incorrect.

Resolution: We advise to check the calculation and error message and correct it.

(4) Revert message is not correct:

Revert message saying the return amount should be at least 3% from the loan amount but the formula says the return amount should be at least 2% of the loan amount.

Resolution: We suggest to give the correct revert message to avoid confusion for the users.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- Update_globalVirtualAmountPercent: Owner can set a global extra virtual amount percentage that adds up to all the new deposits.
- Update_fromProfit_globalVirtualAmountPercent: Owner can set a global extra virtual amount percentage that adds up to new deposits created from user deposit's profit.
- switchLoaningStatus: Owner can switch the loaning feature status.
- setReturnPercent: Owner can set a new return percentage.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of a file and we have used all possible tests based on given objects as files. We have observed 2 critical issues,1 high issue, 1 low issue and some Informational issues in the smart contracts. **So smart contracts are not good to go for the mainnet deployment.**

Since possible test cases can be unlimited for such smart contracts protocol, 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.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Poor Secured".**

Email: audit@EtherAuthority.io

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

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract 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).

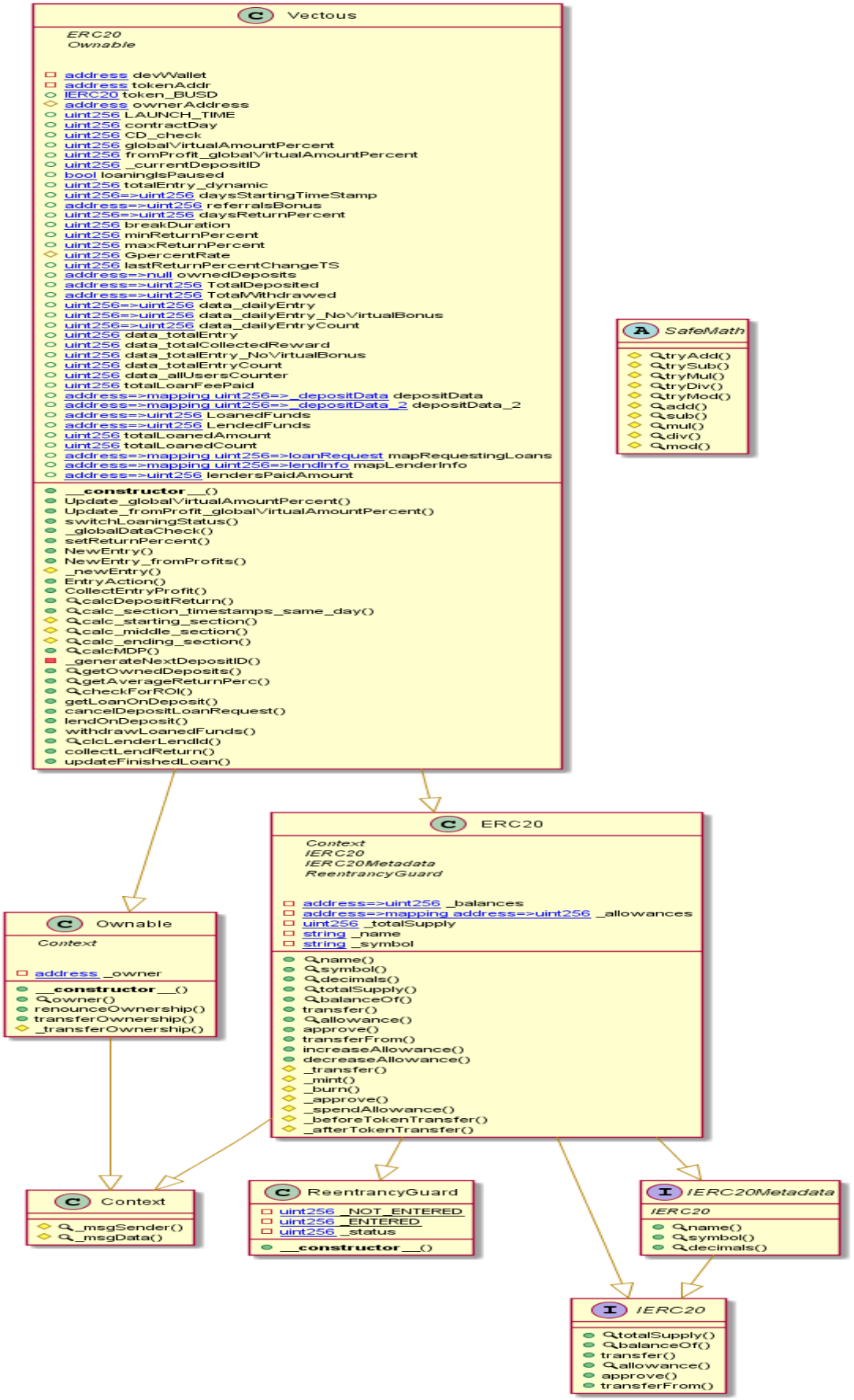
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the 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.

Appendix

Code Flow Diagram - Vectous Token



Slither Results Log

Slither Log >> Vectous.sol

```
Vectous.Update_globalVirtualAmountPercent(uint256) (Vectous.sol#871-876) should emit an event for:
        - globalVirtualAmountPercent = num (Vectous.sol#875)
Vectous.EntryAction(bool,uint256,uint256) (Vectous.sol#1031-1052) should emit an event for:
        - totalEntry_dynamic += amount (Vectous.sol#1046)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in Vectous.CollectEntryProfit(uint256) (Vectous.sol#1060-1074):
        External calls:
        - token_BUSD.transfer(address(msg.sender),depositEarnings) (Vectous.sol#1072)
        State variables written after the call(s):
        - TotalWithdrawed[msg.sender] += depositEarnings (Vectous.sol#1073)
Reentrancy in Vectous.EntryAction(bool,uint256,uint256) (Vectous.sol#1031-1052):
        External calls:
        - CollectEntryProfit(Deposit_ID) (Vectous.sol#1037)
                - token_BUSD.transfer(address(msg.sender),depositEarnings) (Vectous.sol#1072)
        - token_BUSD.transferFrom(msg.sender,address(this),amount) (Vectous.sol#1043)
        State variables written after the call(s):
        - totalEntry_dynamic += amount (Vectous.sol#1046)
Reentrancy in Vectous._newEntry(address,uint256,uint256) (Vectous.sol#977-1018):
        External calls:
        - token_BUSD.transfer(address(devWallet),(_amount * 27 / 1000)) (Vectous.sol#992)
        State variables written after the call(s):
        - TotalDeposited[msg.sender] += _amount (Vectous.sol#1004)
        - data_allUsersCounter ++ (Vectous.sol#1007)
        - data_dailyEntry[contractDay] += _amount + _virtualBonusAmount + referralsBonus[msg.sender] (Vectous.sol#995)
        - data_dailyEntryCount[contractDay] ++ (Vectous.sol#1001)
        - data_dailyEntry_NoVirtualBonus[contractDay] += _amount (Vectous.sol#996)
        - data_totalEntry += _amount + _virtualBonusAmount (Vectous.sol#998)
        - data_totalEntryCount ++ (Vectous.sol#1002)
        - data_totalEntry_NoVirtualBonus += _amount (Vectous.sol#999)
        - referralsBonus[msg.sender] = 0 (Vectous.sol#1011)
        - totalEntry_dynamic += (_amount + _virtualBonusAmount + referredBonus + referralsBonus[msg.sender]) (Vectous.sol#1010
)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
Reentrancy in Vectous.NewEntry(address,uint256) (Vectous.sol#946-953):
        External calls:
        - token_BUSD.transferFrom(msg.sender,address(this),amount) (Vectous.sol#949)
        - _newEntry(referrerAddr,amount,bonusAmount) (Vectous.sol#952)
                - token_BUSD.transfer(address(devWallet),(_amount * 27 / 1000)) (Vectous.sol#992)
        Event emitted after the call(s):
        - E_newEntry(msg.sender,block.timestamp,_amount) (Vectous.sol#1013-1017)
                - _newEntry(referrerAddr,amount,bonusAmount) (Vectous.sol#952)
Reentrancy in Vectous._newEntry(address,uint256,uint256) (Vectous.sol#977-1018):
        External calls:
        - token_BUSD.transfer(address(devWallet),(_amount * 27 / 1000)) (Vectous.sol#992)
        Event emitted after the call(s):
        - E_newEntry(msg.sender,block.timestamp,_amount) (Vectous.sol#1013-1017)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Vectous._globalDataCheck() (Vectous.sol#907-926) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp > CD_check,next day not reached yet) (Vectous.sol#908)
        - timeCheck >= 1 (Vectous.sol#913)
Vectous.setReturnPercent(uint256) (Vectous.sol#932-938) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(block.timestamp > lastReturnPercentChangeTS + breakDuration) (Vectous.sol#933)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
Vectous.switchLoaningStatus() (Vectous.sol#893-900) compares to a boolean constant:
        -loaningIsPaused == false (Vectous.sol#897)
Vectous.switchLoaningStatus() (Vectous.sol#893-900) compares to a boolean constant:
        -loaningIsPaused == true (Vectous.sol#894)
Vectous.EntryAction(bool,uint256,uint256) (Vectous.sol#1031-1052) compares to a boolean constant:
        -noIncrease == true (Vectous.sol#1049)
Vectous.EntryAction(bool,uint256,uint256) (Vectous.sol#1031-1052) compares to a boolean constant:
        -noIncrease == false (Vectous.sol#1039)
Vectous.EntryAction(bool,uint256,uint256) (Vectous.sol#1031-1052) compares to a boolean constant:
        -require(bool)(checkForROI(msg.sender,Deposit_ID) == true) (Vectous.sol#1034)
Vectous.getLoanOnDeposit(uint256,uint256,uint256) (Vectous.sol#1208-1222) compares to a boolean constant:
        -require(bool,string)(loaningIsPaused == false,functionality is paused) (Vectous.sol#1211)
Vectous.lendOnDeposit(address,uint256,uint256) (Vectous.sol#1239-1255) compares to a boolean constant:
        -require(bool,string)(loaningIsPaused == false,functionality is paused) (Vectous.sol#1242)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
SafeMath.trySub(uint256,uint256) (Vectous.sol#22-27) is never used and should be removed
Vectous.calc_ending_section() (Vectous.sol#1122-1129) is never used and should be removed
Vectous.calc_middle_section(address,uint256) (Vectous.sol#1118-1119) is never used and should be removed
Vectous.calc_starting_section(address,uint256) (Vectous.sol#1111-1115) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version0.8.4 (Vectous.sol#2) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Event VectousE_newEntry(address,uint256,uint256) (Vectous.sol#781-785) is not in CapWords
Event VectousE_getLoanOnDeposit(address,uint256,uint256,uint256,uint256) (Vectous.sol#787-793) is not in CapWords
Event VectousE_lendOnDeposit(address,address,uint256,uint256) (Vectous.sol#795-800) is not in CapWords
Function Vectous.Update_globalVirtualAmountPercent(uint256) (Vectous.sol#871-876) is not in mixedCase
Function Vectous.Update_fromProfit_globalVirtualAmountPercent(uint256) (Vectous.sol#882-887) is not in mixedCase
```

```
Parameter Vectous.cancelDepositLoanRequest(uint256).Deposit_ID (Vectous.sol#1229) is not in mixedCase
Parameter Vectous.lendOnDeposit(address,uint256,uint256).Deposit_ID (Vectous.sol#1239) is not in mixedCase
Parameter Vectous.collectLendReturn(uint256,uint256).Deposit_ID (Vectous.sol#1293) is not in mixedCase
Parameter Vectous.updateFinishedLoan(address,address,uint256,uint256).Deposit_ID (Vectous.sol#1304) is not in mixedCase
Variable Vectous.token_BUSD (Vectous.sol#771) is not in mixedCase
Variable Vectous.LAUNCH_TIME (Vectous.sol#807) is not in mixedCase
Variable Vectous.CD_check (Vectous.sol#813) is not in mixedCase
Variable Vectous.fromProfit_globalVirtualAmountPercent (Vectous.sol#819) is not in mixedCase
Variable Vectous._currentDepositID (Vectous.sol#822) is not in mixedCase
Variable Vectous.totalEntry_dynamic (Vectous.sol#828) is not in mixedCase
Variable Vectous.GpercentRate (Vectous.sol#841) is not in mixedCase
Variable Vectous.TotalDeposited (Vectous.sol#848) is not in mixedCase
Variable Vectous.TotalWithdrawed (Vectous.sol#849) is not in mixedCase
Variable Vectous.data_dailyEntry (Vectous.sol#850) is not in mixedCase
Variable Vectous.data_dailyEntry_NoVirtualBonus (Vectous.sol#851) is not in mixedCase
Variable Vectous.data_dailyEntryCount (Vectous.sol#852) is not in mixedCase
Variable Vectous.data_totalEntry (Vectous.sol#853) is not in mixedCase
Variable Vectous.data_totalCollectedReward (Vectous.sol#854) is not in mixedCase
Variable Vectous.data_totalEntry_NoVirtualBonus (Vectous.sol#855) is not in mixedCase
Variable Vectous.data_totalEntryCount (Vectous.sol#856) is not in mixedCase
Variable Vectous.data_allUsersCounter (Vectous.sol#857) is not in mixedCase
Variable Vectous.LoanedFunds (Vectous.sol#1192) is not in mixedCase
Variable Vectous.LendedFunds (Vectous.sol#1193) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (Vectous.sol#222)" inContext (Vectous.sol#216-225)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Vectous.GpercentRate (Vectous.sol#841) is never used in Vectous (Vectous.sol#767-1315)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
```

```
Vectous.GpercentRate (Vectous.sol#841) should be constant
Vectous.breakDuration (Vectous.sol#838) should be constant
Vectous.devWallet (Vectous.sol#768) should be constant
Vectous.maxReturnPercent (Vectous.sol#840) should be constant
Vectous.minReturnPercent (Vectous.sol#839) should be constant
Vectous.tokenAddr (Vectous.sol#769) should be constant
Vectous.totalLoanFeePaid (Vectous.sol#858) should be constant
Vectous.totalLoanedAmount (Vectous.sol#1195) should be constant
Vectous.totalLoanedCount (Vectous.sol#1196) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

Vectous.LAUNCH_TIME (Vectous.sol#807) should be immutable
Vectous.ownerAddress (Vectous.sol#804) should be immutable
Vectous.token_BUSD (Vectous.sol#771) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
Vectous.sol analyzed (8 contracts with 84 detectors), 109 result(s) found
```

Solidity Static Analysis

Vectous.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Vectous.lendOnDeposit(address,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1239:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 1217:12:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 1252:12:

Gas & Economy

Gas costs:

Gas requirement of function Vectous.collectLendReturn is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 1293:4:

Gas costs:

Gas requirement of function Vectous.updateFinishedLoan is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Pos: 1304:4:

Miscellaneous

Constant/View/Pure functions:

Vectous.collectLendReturn(uint256,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1293:4:

Similar variable names:

Vectous.updateFinishedLoan(address,address,uint256,uint256) : Variables have very similar names "ownerAddress" and "loanerAddress". Note: Modifiers are currently not considered by this static analysis.
Pos: 1310:24:

No return:

Vectous.calcMDP(address,uint256,uint256,uint256): Defines a return type but never explicitly returns a value.
Pos: 1137:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 1262:8:

Solhint Linter

Vectous.sol

```
Vectous.sol:10:18: Error: Parse error: missing ';' at '{'
Vectous.sol:23:18: Error: Parse error: missing ';' at '{'
Vectous.sol:35:18: Error: Parse error: missing ';' at '{'
Vectous.sol:52:18: Error: Parse error: missing ';' at '{'
Vectous.sol:64:18: Error: Parse error: missing ';' at '{'
Vectous.sol:160:18: Error: Parse error: missing ';' at '{'
Vectous.sol:183:18: Error: Parse error: missing ';' at '{'
Vectous.sol:209:18: Error: Parse error: missing ';' at '{'
Vectous.sol:587:18: Error: Parse error: missing ';' at '{'
Vectous.sol:620:18: Error: Parse error: missing ';' at '{'
Vectous.sol:669:18: Error: Parse error: missing ';' at '{'
Vectous.sol:720:22: Error: Parse error: missing ';' at '{'
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io