Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Avarice Token
Platform: Binance Smart Chain
Language: Solidity
Date: May 4th, 2022

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 Avarice team to perform the Security audit of the Avarice 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 April 29th, 2022.

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

The Avarice contract is a BE20 standard smart contract which has functionalities like deposit for staking, sell, cancel or get loan on stake, enter lobby, lend on stake, buy stake.

Audit scope

Name	**Code Review and Security Analysis Report for Avarice Token Smart Contract**
Platform	**BSC / Solidity**
File	Avarice.sol
File MD5 Hash	FBD6662EA975C7115054B86ED30A1232
Updated File MD5 Hash	ED45E4DD6800C55A93539C750217FB22
Online Code Link	0x7f57f5bf9dc3409733c913176f3034f03be4aca5
Updated Online Code Link	0xaef3f73a20d19b998e85e6fcdb89cf4279be2853
Audit Date	April 29th, 2022
Revise Audit Date	June 4th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Avarice ● Symbol: AVC ● Decimals: 18 ● Lobby Pool: 3 million ● Max Stake Days: 300 ● Referred Bonus NR: 5% ● Referred Bonus NRR: 1% ● Dividends Pool Caps at 60 days. ● Bonus Calculation Ratio: 128 ● Lobby Pool Decrease Percentage: 5% ● Day Share Percentage ○ Avaric Team: 4% ○ Marketing: 1% ○ BuyBack: 1%	**YES, This is valid.** **Owner authorized wallet can set some percentage value and we suggest handling the private key of that wallet securely.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“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 0 critical, 0 high, 1 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	Moderated
	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: **PASSED**

Code Quality

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

The libraries in Avarice 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 Avarice Token.

The Avarice 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 smart contracts.

Documentation

We were given a Avarice Token smart contract code in the form of a BSCScan Web Link.The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. 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 used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access by isOwner	No Issue
5	transferOwnership	write	access by isOwner	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	switchLoaningStatus	external	access by isOwner	No Issue
26	switchStakeSellingStatus	external	access by isOwner	No Issue
27	flushFirstDayLobbyEntry	external	access by isOwner	No Issue
28	flushFirstDayLobbyEntrySwitch	external	access by isOwner	No Issue
29	do_changeMarketingAddress	external	access by isOwner	No Issue
30	flushdevShareOfStakeSells	external	access by isOwner	No Issue
31	_clcDay	read	Passed	No Issue
32	_updateDaily	write	Passed	No Issue
33	_updateLobbyPool	internal	Passed	No Issue
34	_sendDevShare	internal	Passed	No Issue
35	_sendMarketingShare	internal	Passed	No Issue
36	_buyLobbyBuybackShare	internal	Passed	No Issue
37	EnterLobby	external	Passed	No Issue
38	ExitLobby	external	Passed	No Issue
39	_clcTokenValue	read	Passed	No Issue
40	EnterStake	external	Passed	No Issue
41	calcStakeCount	read	Passed	No Issue
42	EndStake	external	Passed	No Issue

43	calcStakeCollecting	read	Passed	No Issue
44	calcBonusToken	write	Passed	No Issue
45	clcDayDivs	read	Passed	No Issue
46	updateActiveStakesTokens	read	Passed	No Issue
47	sellStakeRequest	external	Passed	No Issue
48	buyStakeRequest	external	Critical operation lacks event log	Refer audit findings
49	withdrawSoldStakeFunds	external	Critical operation lacks event log	Refer audit findings
50	getLoanOnStake	external	Passed	No Issue
51	cancelStakeLoanRequest	write	Critical operation lacks event log	Refer audit findings
52	cancelSellStakeRequest	internal	Passed	No Issue
53	lendOnStake	external	Passed	No Issue
54	clcLenderStakeId	read	Passed	No Issue
55	collectLendReturn	external	Critical operation lacks event log	Refer audit findings
56	updateFinishedLoan	internal	Passed	No Issue

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

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) “Out of Gas” Issue:
clcDayDivs, calcStakeCollecting functions are using a loop for days which can be reverted because of out of gas error. The calcStakeCollecting is used to execute inside EndStake and getLoanOnStake functions.

Resolution: We suggest correcting the logic.
Status: Fixed.

Low

(1) Critical operation lacks event log:
There is a missing event log for some functions.
Functions are listed below:

- buyStakeRequest
- withdrawSoldStakeFunds
- cancelStakeLoanRequest
- collectLendReturn

Resolution: We suggest adding logs for these functions.

Very Low / Informational / Best practices:

(1) Unused variable:

```
/* Starting amount of tokens that are minted daily to be distributed
uint256 internal constant startingLobbyPool = 3 * 1e8;
/* last amount of lobby pool which starts from startingLobbyPool */
uint256 internal lastLobbyPool = startingLobbyPool;
```

```
/* Min penalty for canceling stake */
uint256 internal constant penalty_min = 2;

/* Max penalty for canceling stake */
uint256 internal constant penalty_max = 50;
```

A startingLobbyPool variable is used to initiate the lastLobbyPool variable and not used any other place. penalty_min , penalty_max have not been used in code.

Resolution: We suggest initiating the lastLobbyPool variable by the same value of startingLobbyPool and removing unused variables.
Status: Fixed.

(2) Hardcoded value:

```
/* Address of flush accs */
address internal constant avariceTeam_addr_1 = 0xba38ec6D3b67562c94BF5e08A2e80F432C7f32F5;
address internal constant avariceTeam_addr_2 = 0x8FB1c0840e86306b4036c228Bf875D1DaC3C6Dd7;
address internal constant avariceTeam_addr_3 = 0xde08C84C9005dcE3BD82FaD861Aa191f85CF14e0;
```

```
/* Time of contract launch (0000-00-00|00:00:00) */
uint256 internal constant LAUNCH_TIME = 1650881563;
```

These variables are set with hardcoded values and cannot be changed after contract deployment..

Resolution: Deployer needs to confirm before deploying the contract.

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:

- switchLoaningStatus: Owner can switch the loaning feature status.
- switchStakeSellingStatus: Owner can switch the stake selling feature status.
- flushFirstDayLobbyEntry: Owner can flush the 1st day's lobby entry to dev address.
- flushFirstDayLobbyEntrySwitch: Owner can turn off the first day flush functionality.
- do_changeMarketingAddress: Owner can change marketing wallet address.
- flushdevShareOfStakeSells: Owner can flushes the dev share from stake sells.

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. And we have used all possible tests based on given objects as files. We have observed some issues. And those are fixed in revised code. So, **it's good to go to production**.

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.

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

Security state of the reviewed contract, based on standard audit procedure scope, is **"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).

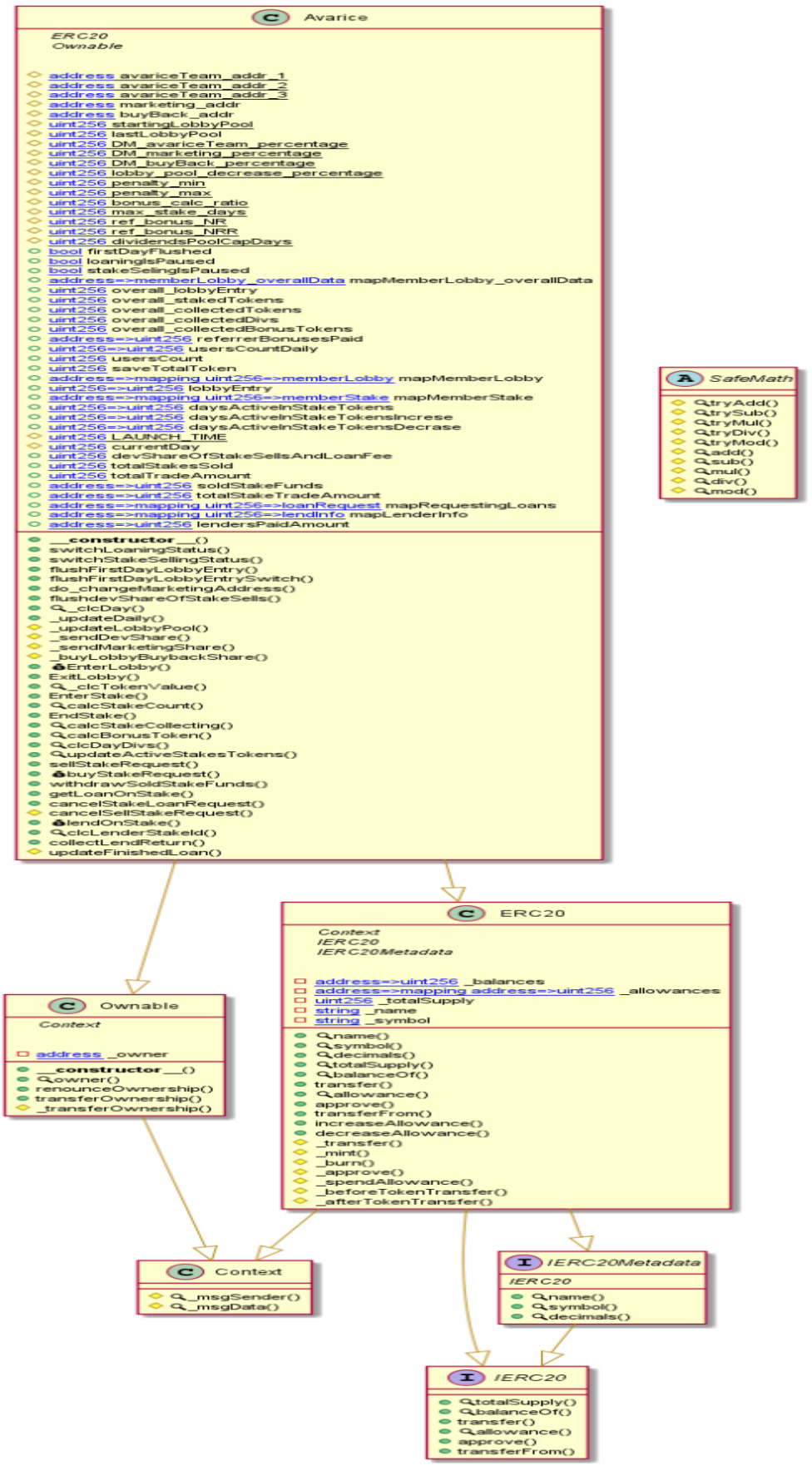
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 - Avarice Token



Slither Results Log

Slither log >> Avarice.sol

```
INFO:Detectors:
Avarice.do_changeMarketingAddress(address).adr (Avarice.sol#1013) lacks a zero-check on :
                - marketing_addr = adr (Avarice.sol#1014)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Avarice._updateDaily() (Avarice.sol#1058-1077) uses timestamp for comparisons
        Dangerous comparisons:
        - currentDay != _clcDay() (Avarice.sol#1059)
Avarice._sendDevShare() (Avarice.sol#1085-1094) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(currentDay > 1) (Avarice.sol#1086)
Avarice._sendMarketingShare() (Avarice.sol#1097-1102) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(currentDay > 1) (Avarice.sol#1098)
Avarice._buyLobbyBuybackShare() (Avarice.sol#1105-1110) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(currentDay > 1) (Avarice.sol#1106)
Avarice.ExitLobby(uint256) (Avarice.sol#1161-1191) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(targetDay < currentDay) (Avarice.sol#1164)
Avarice._clcTokenValue(address,uint256) (Avarice.sol#1198-1211) uses timestamp for comparisons
        Dangerous comparisons:
        - entryDay != 0 && entryDay < currentDay (Avarice.sol#1204)
Avarice.EndStake(uint256) (Avarice.sol#1276-1314) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(mapMemberStake[msg.sender][stakeId].endDay < currentDay,Stakes end day not reached yet) (Avarice
.sol#1277)
Avarice.sellStakeRequest(uint256,uint256) (Avarice.sol#1451-1475) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(mapMemberStake[msg.sender][stakeId].endDay > currentDay,Target stake is ended) (Avarice.sol#1458
)
Avarice.buyStakeRequest(address,uint256) (Avarice.sol#1483-1526) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool)(mapMemberStake[sellerAddress][stakeId].endDay > currentDay) (Avarice.sol#1492)
Avarice.getLoanOnStake(uint256,uint256,uint256,uint256) (Avarice.sol#1582-1620) uses timestamp for comparisons
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Avarice.switchLoaningStatus() (Avarice.sol#980-987) compares to a boolean constant:
        -loaningIsPaused == false (Avarice.sol#984)
Avarice.switchLoaningStatus() (Avarice.sol#980-987) compares to a boolean constant:
        -loaningIsPaused == true (Avarice.sol#981)
Avarice.switchStakeSellingStatus() (Avarice.sol#990-997) compares to a boolean constant:
        -stakeSelingIsPaused == false (Avarice.sol#994)
Avarice.switchStakeSellingStatus() (Avarice.sol#990-997) compares to a boolean constant:
        -stakeSelingIsPaused == true (Avarice.sol#991)
Avarice.flushFirstDayLobbyEntry() (Avarice.sol#1000-1005) compares to a boolean constant:
        -require(bool)(firstDayFlushed == false) (Avarice.sol#1001)
Avarice.ExitLobby(uint256) (Avarice.sol#1161-1191) compares to a boolean constant:
        -require(bool,string)(mapMemberLobby[msg.sender][targetDay].hasCollected == false,ERR: Already collected) (Avarice.sol#
1162)
Avarice.EndStake(uint256) (Avarice.sol#1276-1314) compares to a boolean constant:
        -require(bool)(mapMemberStake[msg.sender][stakeId].stakeCollected == false) (Avarice.sol#1279)
Avarice.EndStake(uint256) (Avarice.sol#1276-1314) compares to a boolean constant:
        -require(bool)(mapMemberStake[msg.sender][stakeId].stake_hasLoan == false) (Avarice.sol#1280)
Avarice.EndStake(uint256) (Avarice.sol#1276-1314) compares to a boolean constant:
        -require(bool)(mapMemberStake[msg.sender][stakeId].stake_hasSold == false) (Avarice.sol#1281)
Avarice.clcDayDivs(address,uint256,uint256) (Avarice.sol#1368-1412) compares to a boolean constant:
        -_isOdd == true (Avarice.sol#1382)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Context._msgData() (Avarice.sol#26-29) is never used and should be removed
SafeMath.add(uint256,uint256) (Avarice.sol#668-670) is never used and should be removed
SafeMath.div(uint256,uint256) (Avarice.sol#710-712) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Avarice.sol#766-775) is never used and should be removed
SafeMath.mod(uint256,uint256) (Avarice.sol#726-728) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Avarice.sol#792-801) is never used and should be removed
SafeMath.mul(uint256,uint256) (Avarice.sol#696-698) is never used and should be removed
SafeMath.sub(uint256,uint256) (Avarice.sol#682-684) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Avarice.sol#743-752) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (Avarice.sol#597-603) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (Avarice.sol#639-644) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (Avarice.sol#651-656) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (Avarice.sol#622-632) is never used and should be removed
SafeMath.trySub(uint256,uint256) (Avarice.sol#610-615) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.0 (Avarice.sol#8) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Struct Avarice.memberLobby_overallData (Avarice.sol#908-913) is not in CapWords
Struct Avarice.memberLobby (Avarice.sol#938-943) is not in CapWords
Struct Avarice.memberStake (Avarice.sol#953-966) is not in CapWords
Struct Avarice.loanRequest (Avarice.sol#1544-1555) is not in CapWords
Struct Avarice.lendInfo (Avarice.sol#1557-1565) is not in CapWords
```

```
Variable Avarice.overall_collectedDivs (Avarice.sol#923) is not in mixedCase
Variable Avarice.overall_collectedBonusTokens (Avarice.sol#925) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Avarice.sol#27)" inContext (Avarice.sol#21-30)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Reentrancy in Avarice.EndStake(uint256) (Avarice.sol#1276-1314):
        External calls:
        - _updateDaily() (Avarice.sol#1283)
                - address(buyBack_addr).transfer((lobbyEntry[currentDay - 1] * DM_buyBack_percentage) / 100) (Avarice.sol#1109)
                - address(marketing_addr).transfer((lobbyEntry[currentDay - 1] * DM_marketing_percentage) / 100) (Avarice.sol#1
101)
```

```
        - mapRequestingLoans[loanerAddress][stakeId].loanIsPaid = true (Avarice.sol#1742)
Reentrancy in Avarice.withdrawSoldStakeFunds() (Avarice.sol#1531-1536):
        External calls:
        - address(msg.sender).transfer(soldStakeFunds[msg.sender]) (Avarice.sol#1534)
        State variables written after the call(s):
        - soldStakeFunds[msg.sender] = 0 (Avarice.sol#1535)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4
INFO:Detectors:
Variable Avarice.avariceTeam_addr_1 (Avarice.sol#856) is too similar to Avarice.avariceTeam_addr_2 (Avarice.sol#857)
Variable Avarice.avariceTeam_addr_1 (Avarice.sol#856) is too similar to Avarice.avariceTeam_addr_3 (Avarice.sol#858)
Variable Avarice.avariceTeam_addr_2 (Avarice.sol#857) is too similar to Avarice.avariceTeam_addr_3 (Avarice.sol#858)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Avarice.penalty_min (Avarice.sol#876) is never used in Avarice (Avarice.sol#805-1748)
Avarice.penalty_max (Avarice.sol#879) is never used in Avarice (Avarice.sol#805-1748)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
Avarice.buyBack_addr (Avarice.sol#860) should be constant
ERC20._name (Avarice.sol#238) should be constant
ERC20._symbol (Avarice.sol#239) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Avarice.sol#79-81)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Avarice.sol#87-90)
name() should be declared external:
        - ERC20.name() (Avarice.sol#258-260)
symbol() should be declared external:
        - ERC20.symbol() (Avarice.sol#266-268)
decimals() should be declared external:
        - ERC20.decimals() (Avarice.sol#283-285)
totalSupply() should be declared external:
        - ERC20.totalSupply() (Avarice.sol#290-292)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (Avarice.sol#309-313)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Avarice.sol#332-336)
```

```
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (Avarice.sol#332-336)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (Avarice.sol#354-363)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (Avarice.sol#377-381)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (Avarice.sol#397-406)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Avarice.sol analyzed (7 contracts with 75 detectors), 149 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Avarice.sol

Security

Check-effects-interaction:

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

more

Pos: 1020:5:

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: 1251:13:

Gas & Economy

Gas costs:

Gas requirement of function Avarice.name 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: 258:5:

Gas costs:

Gas requirement of function Avarice.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: 1718:5:

Miscellaneous

Constant/View/Pure functions:

Avarice.clcDayDivs(address,uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1368:5:

Similar variable names:

Avarice.clcDayDivs(address,uint256,uint256) : Variables have very similar names "_dSC" and "_day". Note: Modifiers are currently not considered by this static analysis.

Pos: 1373:48:

Email: audit@EtherAuthority.io

No return:

IERC20Metadata.decimals(): Defines a return type but never explicitly returns a value.
Pos: 202:5:

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: 1661:9:

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: 1732:9:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 1081:27:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 1089:41:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 1091:46:

Solhint Linter

Avarice.sol

```
Avarice.sol:401:18: Error: Parse error: missing ';' at '{'
Avarice.sol:434:18: Error: Parse error: missing ';' at '{'
Avarice.sol:483:18: Error: Parse error: missing ';' at '{'
Avarice.sol:534:22: Error: Parse error: missing ';' at '{'
Avarice.sol:598:18: Error: Parse error: missing ';' at '{'
Avarice.sol:611:18: Error: Parse error: missing ';' at '{'
Avarice.sol:623:18: Error: Parse error: missing ';' at '{'
Avarice.sol:640:18: Error: Parse error: missing ';' at '{'
Avarice.sol:652:18: Error: Parse error: missing ';' at '{'
Avarice.sol:748:18: Error: Parse error: missing ';' at '{'
Avarice.sol:771:18: Error: Parse error: missing ';' at '{'
Avarice.sol:797:18: 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