Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	SecureChain AI Token
Website:	https://securechain.ai
Platform:	Ethereum / BSC
Language:	Solidity
Date:	September 14th, 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 SecureChain AI team to perform the Security audit of the SCAI 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 September 14th, 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

- The SecureChain AI (SCAI) token is a standard token smart contract, having functions like updating buy and selling fees, updating LP pair addresses, updating enabled trading status, etc.
- The SCAI tokens are part of the SecureChain AI ecosystem and they are launched in Ethereum and BSC networks.

Audit scope

Name	**Code Review and Security Analysis Report for SecureChain AI (SCAI) Token Smart Contract**
Platform	**Ethereum / BSC**
File	SCAI.sol
Ethereum Code	0xe35009059cb55ded065027e9832a2c564aff7512
BSC Code	0x051A66a7750098fB1EC6548D36E275bb23749A78
Audit Date	September 14th, 2023

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: SecureChain AI • Symbol: SCAI • Decimals: 18 • ERC20 Supply: 100 Million • BEP20 Supply: 100 Million • No more minting possible	**YES, This is valid.**
• Buy fee: 2% • Selling fee: 2% • fee denominator: 1000 • Swap Threshold: 20,000	**YES, This is valid.** **The smart contract owner can not raise the taxes more than 2% which is a good thing.**
Ownership Control: • Set a no-fee wallet account enabled status. • Update a new pair address. • Toggle swap fee status. • Update the marketing wallet address. • Update the presale address. • EnableTrading status. • Current owner can transfer the ownership. • Owner can renounce ownership.	**YES, This is valid. We advise to renounce ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **“Secured”**. Also, these contracts contain owner control, which does not make them 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, 0 medium and 0 low and 2 very low level issues.

We confirm that these issues are acknowledged by the SecureChain AI team.

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: **PASSED**

Business Risk Analysis

Category	Result
Buy Tax	2%
Sell Tax	2%
Cannot Buy	Passed
Cannot Sell	Passed
Max Tax	2%
Modify Tax	Not Detected
Fee Check	Passed
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	No
Pause Transfer?	Detected
Max Tax?	Passed
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	Not Detected
Blacklist Check	Passed
Can Mint?	No
Is Proxy?	Not Detected
Can Take Ownership?	Not Detected
Hidden Owner?	Not Detected
Self Destruct?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 2 smart contracts. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in SCAI 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 SCAI Token.

The SCAI 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 not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a SCAI Token smart contract code in the form of Etherscan and BSCscan web links.

As mentioned above, code parts are not well commented on. but the logic is straightforward. 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.

Another source of information was its official project URL: https://securechain.ai which provided rich information about the project architecture and tokenomics.

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	Passed	No Issue
2	totalSupply	external	Passed	No Issue
3	decimals	external	Passed	No Issue
4	symbol	external	Passed	No Issue
5	name	external	Passed	No Issue
6	getOwner	external	Passed	No Issue
7	allowance	external	Passed	No Issue
8	balanceOf	read	Passed	No Issue
9	receive	external	Passed	No Issue
10	transfer	write	Passed	No Issue
11	approve	external	Passed	No Issue
12	_approve	internal	Passed	No Issue
13	transferFrom	external	Passed	No Issue
14	isNoFeeWallet	external	Passed	No Issue
15	setNoFeeWallet	write	access only Owner	No Issue
16	isLimitedAddress	internal	Passed	No Issue
17	is_buy	internal	Passed	No Issue
18	is_sell	internal	Passed	No Issue
19	is_transfer	internal	Passed	No Issue
20	canSwap	internal	Passed	No Issue
21	changeLpPair	external	access only Owner	No Issue
22	toggleCanSwapFees	external	access only Owner	No Issue
23	_transfer	internal	Passed	No Issue
24	changeWallets	external	access only Owner	No Issue
25	takeTaxes	internal	Passed	No Issue
26	internalSwap	internal	In the swap flag	No Issue
27	setPresaleAddress	external	access only Owner	No Issue
28	enableTrading	external	access only Owner	No Issue
29	inSwapFlag	modifier	Passed	No Issue
30	owner	read	Passed	No Issue
31	onlyOwner	modifier	Passed	No Issue
32	renounceOwnership	write	access only Owner	No Issue
33	transferOwnership	write	access only Owner	No Issue
34	_setOwner	write	Passed	No Issue
35	_msgSender	internal	Passed	No Issue
36	_msgData	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

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Consider renouncing ownership:

Once all the administrative functions are over, then we advise to renounce the ownership of the contract. This will make it fully decentralized. Fully decentralized contracts increase the trust in the users.

(2) Trading is paused by default:

We understand this is a design of SAFU protocol of launchpad, and only SAFU dev can enable the trading. And we understand that trading can not be disabled after being enabled, which is a good thing.
However, we advise to renounce the ownership to make the smart contract fully decentralized.

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:

SCAI.sol

- setNoFeeWallet: No fee wallet account enabled status can be set by the owner.
- changeLpPair: A new pair address can be updated by the owner.
- toggleCanSwapFees: Swap fee status can be toggled by the owner.
- changeWallets: The marketing wallet address can be updated by the owner.
- setPresaleAddress: The presale address can be updated by the owner.
- enableTrading: Trading status can be enabled by the owner.

Ownable.sol

- renounce Ownership: Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transfer ownership: The current owner can transfer ownership of the contract to a new account.

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

Conclusion

We were given a contract code in the form of Etherscan and BSCscan web links. And we have used all possible tests based on given objects as files. We had observed 2 informational issues in the smart contracts. And those issues are acknowledged. So, **it's good to go for the 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 smart 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).

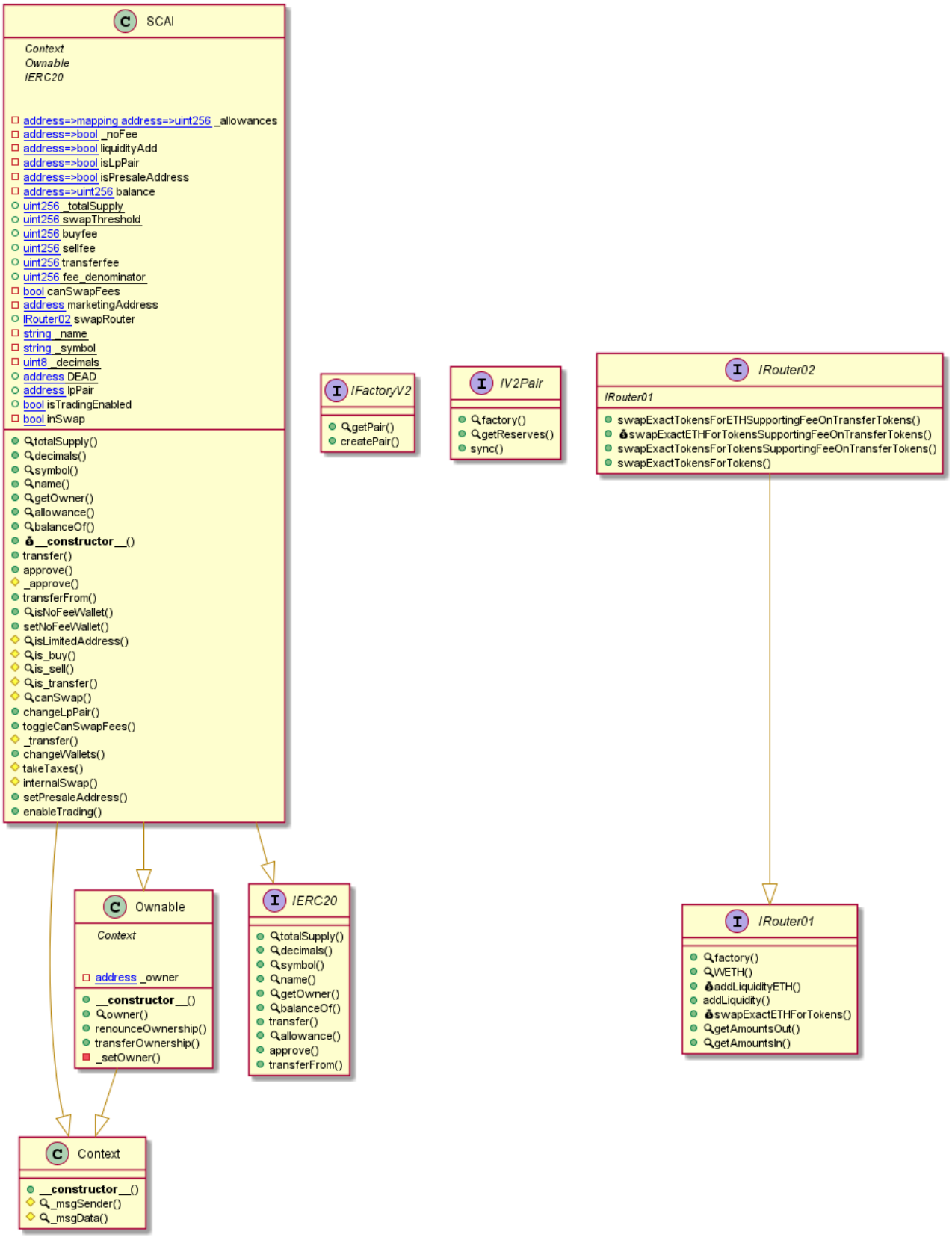
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

Slither Log >> SCAI.sol

```
SCAI.internalSwap(uint256) (SCAI.sol#352-377) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations

Reentrancy in SCAI._transfer(address,address,uint256) (SCAI.sol#304-329):
        External calls:
        - internalSwap(contractTokenBalance) (SCAI.sol#317)
                - swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(contractTokenBalance,0,path,address(this),bloc
k.timestamp) (SCAI.sol#362-370)
                - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
        External calls sending eth:
        - internalSwap(contractTokenBalance) (SCAI.sol#317)
                - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
        State variables written after the call(s):
        - balance[from] -= amount (SCAI.sol#324)
        SCAI.balance (SCAI.sol#168) can be used in cross function reentrancies:
        - SCAI._transfer(address,address,uint256) (SCAI.sol#304-329)
        - SCAI.balanceOf(address) (SCAI.sol#158-160)
        - SCAI.constructor() (SCAI.sol#204-228)
        - SCAI.takeTaxes(address,bool,bool,uint256) (SCAI.sol#338-350)
        - balance[to] += amountAfterFee (SCAI.sol#325)
        SCAI.balance (SCAI.sol#168) can be used in cross function reentrancies:
        - SCAI._transfer(address,address,uint256) (SCAI.sol#304-329)
        - SCAI.balanceOf(address) (SCAI.sol#158-160)
        - SCAI.constructor() (SCAI.sol#204-228)
        - SCAI.takeTaxes(address,bool,bool,uint256) (SCAI.sol#338-350)
        - amountAfterFee = takeTaxes(from,is_buy(from,to),is_sell(from,to),amount) (SCAI.sol#324)
                - balance[address(this)] += feeAmount (SCAI.sol#345)
        SCAI.balance (SCAI.sol#168) can be used in cross function reentrancies:
        - SCAI._transfer(address,address,uint256) (SCAI.sol#304-329)
        - SCAI.balanceOf(address) (SCAI.sol#158-160)
        - SCAI.constructor() (SCAI.sol#204-228)
        - SCAI.takeTaxes(address,bool,bool,uint256) (SCAI.sol#338-350)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
```

```
Reentrancy in SCAI._transfer(address,address,uint256) (SCAI.sol#304-329):
        External calls:
        - internalSwap(contractTokenBalance) (SCAI.sol#317)
                - swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(contractTokenBalance,0,path,address(this),bloc
k.timestamp) (SCAI.sol#362-370)
                - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
        External calls sending eth:
        - internalSwap(contractTokenBalance) (SCAI.sol#317)
                - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
        Event emitted after the call(s):
        - Transfer(from,address(this),feeAmount) (SCAI.sol#346)
                - amountAfterFee = takeTaxes(from,is_buy(from,to),is_sell(from,to),amount) (SCAI.sol#324)
        - Transfer(from,to,amountAfterFee) (SCAI.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Context._msgData() (SCAI.sol#18-21) is never used and should be removed
SCAI.is_transfer(address,address) (SCAI.sol#282-285) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version=0.8.19 (SCAI.sol#7) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8.1
6
solc-0.8.19 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in SCAI.internalSwap(uint256) (SCAI.sol#352-377):
        - (success,None) = marketingAddress.call{gas: 35000,value: address(this).balance}() (SCAI.sol#375)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Function IRouter01.WETH() (SCAI.sol#73) is not in mixedCase
Event SCAI_enableTrading() (SCAI.sol#197) is not in CapWords
Event SCAI_setPresaleAddress(address,bool) (SCAI.sol#198) is not in CapWords
Event SCAI_toggleCanSwapFees(bool) (SCAI.sol#199) is not in CapWords
Event SCAI_changePair(address) (SCAI.sol#200) is not in CapWords
Event SCAI_changeWallets(address) (SCAI.sol#201) is not in CapWords
Function SCAI.is_buy(address,address) (SCAI.sol#272-275) is not in mixedCase
Function SCAI.is_sell(address,address) (SCAI.sol#277-280) is not in mixedCase
Function SCAI.is_transfer(address,address) (SCAI.sol#282-285) is not in mixedCase
Constant SCAI._totalSupply (SCAI.sol#171) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SCAI.swapThreshold (SCAI.sol#172) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SCAI.fee_denominator (SCAI.sol#176) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SCAI._name (SCAI.sol#182) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SCAI._symbol (SCAI.sol#183) is not in UPPER_CASE_WITH_UNDERSCORES
Constant SCAI._decimals (SCAI.sol#184) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (SCAI.sol#19)" inContext (SCAI.sol#10-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

Variable IRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (SCAI.sol#85)
is too similar to IRouter01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (SCAI
.sol#86)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

SCAI.buyfee (SCAI.sol#173) should be constant
SCAI.sellfee (SCAI.sol#174) should be constant
SCAI.transferfee (SCAI.sol#175) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

SCAI.lpPair (SCAI.sol#186) should be immutable
SCAI.swapRouter (SCAI.sol#181) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
SCAI.sol analyzed (8 contracts with 84 detectors), 30 result(s) found
```

Solidity Static Analysis

SCAI.sol

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: 367:12:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 375:52:

Gas costs:

Gas requirement of function SCAI.setPresaleAddress 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: 379:8:

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 153:4:

Similar variable names:

SCAI._approve(address,address,uint256) : Variables have very similar names "sender" and "spender". Note: Modifiers are currently not considered by this static analysis.
Pos: 246:28:

No return:

IERC20.getOwner(): Defines a return type but never explicitly returns a value.
Pos: 139:4:

No return:

IERC20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 144:4:

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: 39:8:

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: 332:8:

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: 342:28:

Solhint Linter

SCAI.sol

```
Compiler version =0.8.19 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:6
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:10
Code contains empty blocks
Pos: 19:10
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:29
Error message for require is too long
Pos: 9:47
Function name must be in mixedCase
Pos: 5:72
Provide an error message for revert
Pos: 96:151
Provide an error message for revert
Pos: 91:152
Constant name must be in capitalized SNAKE_CASE
Pos: 5:170
Constant name must be in capitalized SNAKE_CASE
Pos: 5:171
Constant name must be in capitalized SNAKE_CASE
Pos: 5:175
Constant name must be in capitalized SNAKE_CASE
Pos: 5:181
Constant name must be in capitalized SNAKE_CASE
Pos: 5:182
Constant name must be in capitalized SNAKE_CASE
Pos: 5:183
Event name must be in CamelCase
Pos: 5:196
Event name must be in CamelCase
Pos: 5:197
Event name must be in CamelCase
Pos: 5:198
Event name must be in CamelCase
Pos: 5:199
Event name must be in CamelCase
Pos: 5:200
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:203
Code contains empty blocks
Pos: 32:229
Function name must be in mixedCase
Pos: 5:271
Variable name must be in mixedCase
```

```
Pos: 9:272
Function name must be in mixedCase
Pos: 5:276
Variable name must be in mixedCase
Pos: 9:277
Function name must be in mixedCase
Pos: 5:281
Variable name must be in mixedCase
Pos: 9:282
Error message for require is too long
Pos: 9:305
Error message for require is too long
Pos: 9:306
Error message for require is too long
Pos: 9:307
Provide an error message for require
Pos: 9:331
Avoid making time-based decisions in your business logic
Pos: 13:366
Code contains empty blocks
Pos: 11:367
```

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