Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Sustainix Renewable
Website:	sustainix.org
Platform:	Ethereum
Language:	Solidity
Date:	August 5th, 2024

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 Sustainix Renewable Provider Limited to perform the Security audit of the Sustainix Renewable smart contract code. The audit was performed using manual analysis and automated software tools. This report presents all the findings regarding the audit performed on August 5th, 2024.

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 Solidity smart contract for a token called "Sustainix Renewable" (SXR). This contract extends the ERC20 standard with additional features such as burnable tokens, fee mechanisms, swap thresholds, and automated market maker (AMM) management.

Here's a brief overview of the contract features and structure:

- **ERC20 Standard and Burnable Tokens:** Inherits from `ERC20` and `ERC20Burnable`.
- **Ownership:**
    - Uses `Ownable2Step` to manage ownership.
    - The `Initializable` ensures that certain functions run only once after deployment.
- **Fees and Liquidity:**
    - Supports three types of fees: buy, sell, and transfer.
    - Fees can be allocated for research and liquidity.
    - Uses UniswapV2 for liquidity operations.
- **Exclusions:** Addresses can be excluded from fees and trade limits.
- **Trading Limits:**
    - Enforces maximum buy and sell amounts.
    - Implements trade cooldown periods to prevent rapid trades.
- **Swap and Liquify:**
    - Automatically swaps tokens for ETH and adds liquidity.
    - Supports adding liquidity from leftover tokens.

- **Error Handling:** Custom error messages for various validation checks.

This contract is designed to handle various aspects of an ERC20 token, including fees, liquidity management, and trading restrictions, making it suitable for a decentralized finance (DeFi) application.

Audit scope

Name	**Code Review and Security Analysis Report for Sustainix Renewable Smart Contract**
Platform	**Ethereum / Solidity**
File	Sustainix_Renewable.sol
Etherscan Smart Contract Address	0x135acfcc634f8b28c37518bc4070d484d2a7d524
Audit Date	August 5th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Sustainix Renewable • Symbol: SXR • Decimals: 18 • Maximum buy amount: 50 million • Maximum sell amount: 30 million • Swap Threshold Ratio: 50	**YES, This is valid.**
Liquidity Fees: • Buy Fee:0.5% • Sell Fee:0.5% **Research Fees:** • Buy Fee:1% • Sell Fee:1%	**YES, This is valid.**
The owner has control over the following functions: • Allows the owner to recover tokens. • Allows the owner to recover foreign tokens. • Updates the swap threshold ratio. • Set up the research address. • Set up research fees. • Set up liquidity fees. • Excludes an address from fees. • Sets an address as an AMM. • Excludes an address from limits. • Updates the maximum buy/sell amount. • Updates the trade cooldown time. • The current owner can transfer the ownership. • The owner can renounce ownership.	**YES, This is valid.** **We suggest renouncing 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-based smart contracts are **“secured”**. This token contract contains 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 a 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. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 3 very low-level issues.

Investor Advice: A 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	The solidity version is not specified	Passed
	The solidity version is too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack 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	Moderated
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 is not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Business Risk Analysis

Category	Result
Buy Tax	1.5%
Sell Tax	1.5%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	Yes
Fee Check	Yes
Is Honeypot	Not Detected
Trading Cooldown	30 mins
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	2.5%
Is it Anti-whale?	Not Detected
Is Anti-bot?	Yes
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	No
Is it a Proxy Contract?	No
Can Take Ownership?	Yes
Creator Percentage?	0.00%
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

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

The libraries in the Sustainix Renewable 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 Sustainix Renewable.

The Sustainix Renewable team has not provided scenario and unit test scripts, which would help 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 Sustainix Renewable smart contract code in the form of an etherscan weblink.

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

Another source of information was its website sustainix.org which provided rich information about the project architecture.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure 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	Hardcoded address	Refer Audit Findings
2	afterConstructor	external	initializer	No Issue
3	decimals	write	Passed	No Issue
4	recoverToken	external	The owner can drain tokens	Refer Audit Findings
5	recoverForeignERC20	external	access only Owner	No Issue
6	receive	external	Passed	No Issue
7	_swapTokensForCoin	write	Passed	No Issue
8	updateSwapThreshold	write	access only Owner	No Issue
9	getSwapThresholdAmount	read	Passed	No Issue
10	getAllPending	read	Passed	No Issue
11	researchAddressSetup	write	access only Owner	No Issue
12	researchFeesSetup	write	access only Owner	No Issue
13	_swapAndLiquify	write	Liquidity transferred to dead address	Refer Audit Findings
14	_addLiquidity	write	Liquidity transferred to dead address	Refer Audit Findings
15	addLiquidityFromLeftoverTokens	external	Passed	No Issue
16	liquidityFeesSetup	write	access only Owner	No Issue
17	excludeFromFees	write	access only Owner	No Issue
18	_updateRouterV2	write	Passed	No Issue
19	setAMM	external	access only Owner	No Issue
20	_setAMM	write	Passed	No Issue
21	excludeFromLimits	external	access only Owner	No Issue
22	_excludeFromLimits	internal	Passed	No Issue
23	_maxTxSafeLimit	read	Passed	No Issue
24	updateMaxBuyAmount	write	access only Owner	No Issue
25	updateMaxSellAmount	write	access only Owner	No Issue

26	updateTradeCooldownTime	write	access only Owner	No Issue
27	_update	internal	Passed	No Issue
28	_beforeTokenUpdate	internal	Passed	No Issue
29	_afterTokenUpdate	internal	Passed	No Issue
30	name	read	Passed	No Issue
31	symbol	read	Passed	No Issue
32	decimals	read	Passed	No Issue
33	totalSupply	read	Passed	No Issue
34	balanceOf	read	Passed	No Issue
35	transfer	write	Passed	No Issue
36	allowance	read	Passed	No Issue
37	approve	write	Passed	No Issue
38	transferFrom	write	Passed	No Issue
39	_transfer	internal	Passed	No Issue
40	_update	internal	Passed	No Issue
41	_mint	internal	Passed	No Issue
42	_burn	internal	Passed	No Issue
43	_approve	internal	Passed	No Issue
44	_approve	internal	Passed	No Issue
45	_spendAllowance	internal	Passed	No Issue
46	burn	write	Passed	No Issue
47	burnFrom	write	Passed	No Issue
48	pendingOwner	read	Passed	No Issue
49	transferOwnership	write	access only Owner	No Issue
50	_transferOwnership	internal	Passed	No Issue
51	acceptOwnership	write	Passed	No Issue
52	initializer	modifier	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 a 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 a 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) Liquidity transferred to dead address:

```
function _swapAndLiquify(uint256 tokenAmount) private returns (uint256 leftover) {    infinite gas
    // Sub-optimal method for supplying liquidity
    uint256 halfAmount = tokenAmount / 2;
    uint256 otherHalf = tokenAmount - halfAmount;

    _swapTokensForCoin(halfAmount);

    uint256 coinBalance = address(this).balance;

    if (coinBalance > 0) {
        (uint amountToken, uint amountCoin, uint liquidity) = _addLiquidity(otherHalf, coinBalance);

        emit LiquidityAdded(amountToken, amountCoin, liquidity);

        return otherHalf - amountToken;
    } else {
        return otherHalf;
    }
}

function _addLiquidity(uint256 tokenAmount, uint256 coinAmount) private returns (uint, uint, uint) {    infinite gas
    _approve(address(this), address(routerV2), tokenAmount);

    return routerV2.addLiquidityETH{value: coinAmount}(address(this), tokenAmount, 0, 0, address(0), block.timestamp);
}
```

While swap and liquify, liquidity has been added to address(0).

Resolution: We suggest confirming this if this is a required feature.

(2) Hardcoded address:

```
1076    constructor()
1077        ERC20(unicode"Sustainix Renewable", unicode"SXR")
1078        Ownable(msg.sender)
1079    {
1080        address supplyRecipient = 0x31585060788dDf05100B87360F01601E8567754C;
1081
1082        updateSwapThreshold(50);
1083
1084        researchAddressSetup(0x690615f7fF1e0109bCC6b046C4617Ddbd8C04Ca8);
1085        researchFeesSetup(100, 100, 0);
1086
1087        liquidityFeesSetup(50, 50, 0);
1088
1089        excludeFromFees(supplyRecipient, true);
1090        excludeFromFees(address(this), true);
1091
1092        _excludeFromLimits(supplyRecipient, true);
1093        _excludeFromLimits(address(this), true);
1094        _excludeFromLimits(address(0), true);
1095
1096        updateMaxBuyAmount(500000000 * (10 ** decimals()) / 10);
1097        updateMaxSellAmount(300000000 * (10 ** decimals()) / 10);
1098
1099        updateTradeCooldownTime(1800);
1100
1101        _mint(supplyRecipient, 100000000000 * (10 ** decimals()) / 10);
1102        _transferOwnership(0x31585060788dDf05100B87360F01601E8567754C);
```

A hardcoded address is used to set. The same address is used for _transferOwnership and supplyRecipient

Resolution: We suggest confirming the address before deployment and also the same variable can be used to transfer ownership.

(3) The owner can drain tokens:

The owner can drain the SXR token from the contract.

Resolution: We suggest keeping the owner's private key safe to not allow anyone to misuse the tokens.

Centralization

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

Sustainix_Renewable.sol

- recoverToken: Allows the owner to recover tokens.
- recoverForeignERC20: Allows the owner to recover foreign tokens.
- updateSwapThreshold: The owner can update the swap threshold ratio.
- researchAddressSetup: The owner can set up a research address.
- researchFeesSetup: The owner can set up a research fee.
- liquidityFeesSetup: The owner can set up liquidity fees.
- excludeFromFees: The owner can exclude an address from fees.
- setAMM: The owner can set an address as an AMM.
- excludeFromLimits: The owner can exclude an address from the limit.
- updateMaxBuyAmount: The owner can update the maximum buy amount.
- updateMaxSellAmount: The owner can update the maximum sell amount.
- updateTradeCooldownTime: The owner can update the trade cooldown time.

Ownable2Step.sol

- transferOwnership: Current owner can transfer ownership of the contract to a new account.

Ownable.sol

- renounceOwnership Deleting ownership will leave the contract without an owner, removing any owner-only functionality.
- transferOwnership: Current owner can transfer ownership of the contract to a new account.

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 an etherscan weblink, and we used all possible tests based on the given objects. We have observed 3 Informational severity issues. but these issues are not critical. **So, the smart contract is ready for 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 the 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 **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the 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 white box penetration testing. We look at the project's website 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, and 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).

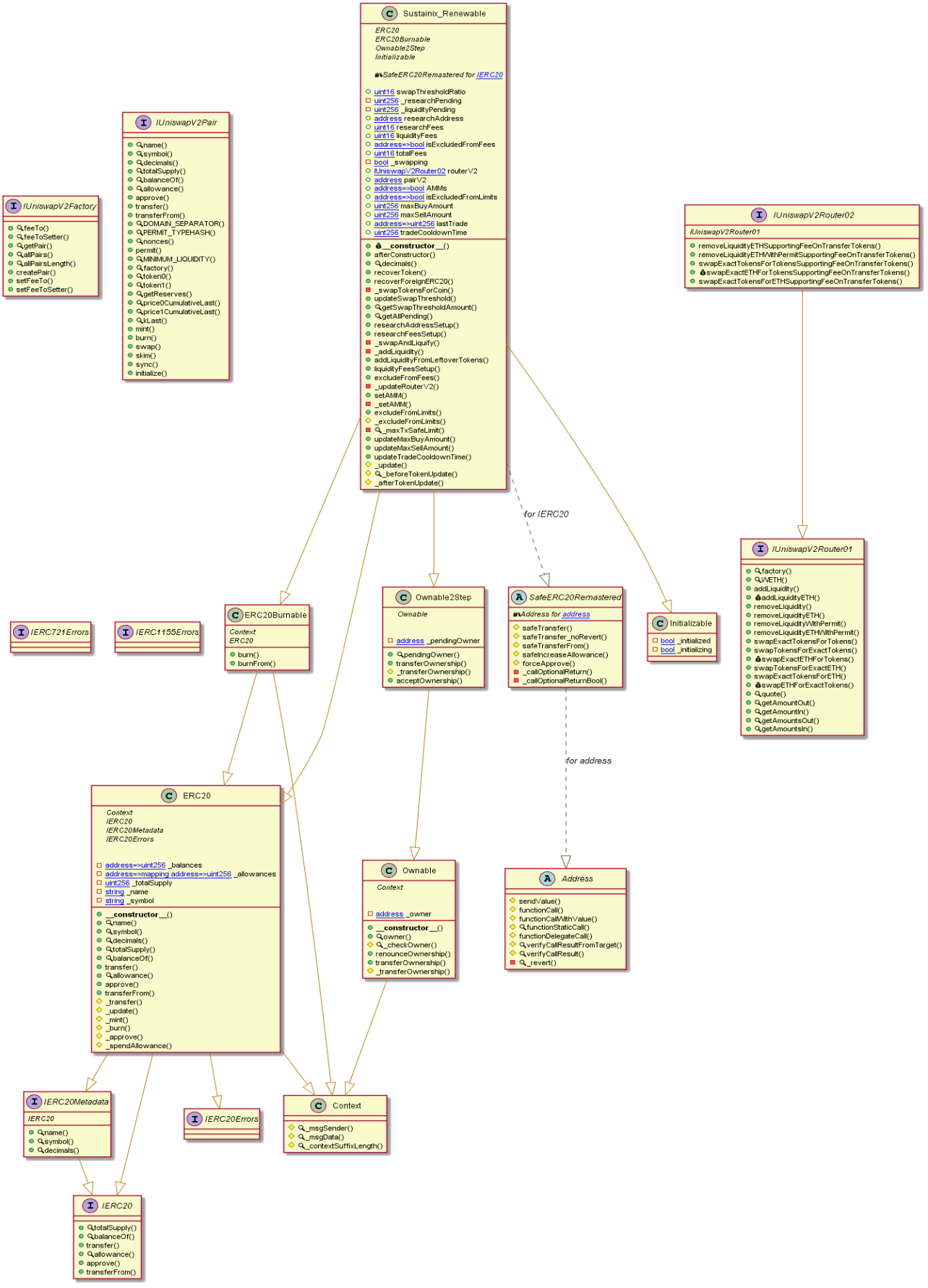
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While 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 the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Sustainix Renewable



Slither Results Log

Slither Log >> Sustainix_Renewable.sol

```
INFO:Detectors:
Sustainix_Renewable.researchAddressSetup(address)._newAddress
(Sustainix_Renewable.sol#1160) lacks a zero-check on :
        - researchAddress = _newAddress (Sustainix_Renewable.sol#1163)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Sustainix_Renewable._swapAndLiquify(uint256)
(Sustainix_Renewable.sol#1181-1199):
    External calls:
    - _swapTokensForCoin(halfAmount) (Sustainix_Renewable.sol#1186)
        -
routerV2.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,addres
s(this),block.timestamp) (Sustainix_Renewable.sol#1141)
    - (amountToken,amountCoin,liquidity) = _addLiquidity(otherHalf,coinBalance)
(Sustainix_Renewable.sol#1191)
        - routerV2.addLiquidityETH{value:
coinAmount}(address(this),tokenAmount,0,0,address(0),block.timestamp)
(Sustainix_Renewable.sol#1204)
    External calls sending eth:
    - (amountToken,amountCoin,liquidity) = _addLiquidity(otherHalf,coinBalance)
(Sustainix_Renewable.sol#1191)
        - routerV2.addLiquidityETH{value:
coinAmount}(address(this),tokenAmount,0,0,address(0),block.timestamp)
(Sustainix_Renewable.sol#1204)
    State variables written after the call(s):
    - (amountToken,amountCoin,liquidity) = _addLiquidity(otherHalf,coinBalance)
(Sustainix_Renewable.sol#1191)
        - _allowances[owner][spender] = value (Sustainix_Renewable.sol#965)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Sustainix_Renewable._update(address,address,uint256) (Sustainix_Renewable.sol#1298-1373)
has a high cyclomatic complexity (16).
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#cyclomatic-complexity
INFO:Detectors:
Pragma version^0.8.20 (Sustainix_Renewable.sol#4) necessitates a version too recent to be
trusted. Consider deploying with 0.8.18.
solc-0.8.20 is not recommended for deployment
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
```

```
Parameter Sustainix_Renewable.liquidityFeesSetup(uint16,uint16,uint16)._sellFee
(Sustainix_Renewable.sol#1215) is not in mixedCase
Parameter Sustainix_Renewable.liquidityFeesSetup(uint16,uint16,uint16)._transferFee
(Sustainix_Renewable.sol#1215) is not in mixedCase
Parameter Sustainix_Renewable.setAMM(address,bool).AMM (Sustainix_Renewable.sol#1242)
is not in mixedCase
Parameter Sustainix_Renewable.updateMaxBuyAmount(uint256)._maxBuyAmount
(Sustainix_Renewable.sol#1273) is not in mixedCase
Parameter Sustainix_Renewable.updateMaxSellAmount(uint256)._maxSellAmount
(Sustainix_Renewable.sol#1281) is not in mixedCase
Parameter Sustainix_Renewable.updateTradeCooldownTime(uint256)._tradeCooldownTime
(Sustainix_Renewable.sol#1289) is not in mixedCase
Variable Sustainix_Renewable.AMMs (Sustainix_Renewable.sol#1042) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-c
onventions
INFO:Detectors:
Sustainix_Renewable.constructor() (Sustainix_Renewable.sol#1076-1103) uses literals with too
many digits:
        - updateMaxBuyAmount(500000000 * (10 ** decimals()) / 10)
(Sustainix_Renewable.sol#1096)
Sustainix_Renewable.constructor() (Sustainix_Renewable.sol#1076-1103) uses literals with too
many digits:
        - updateMaxSellAmount(300000000 * (10 ** decimals()) / 10)
(Sustainix_Renewable.sol#1097)
Sustainix_Renewable.constructor() (Sustainix_Renewable.sol#1076-1103) uses literals with too
many digits:
        - _mint(supplyRecipient,100000000000 * (10 ** decimals()) / 10)
(Sustainix_Renewable.sol#1101)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Slither:Sustainix_Renewable.sol analyzed (18 contracts with 93 detectors), 63 result(s)
found
```

Solidity Static Analysis

Sustainix_Renewable.sol

Inline assembly:
The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
Pos: 222:12:

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.
Pos: 1568:75:

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.
Pos: 1523:43:

Gas costs:
Gas requirement of function Sustainix_Renewable.renounceOwnership 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: 763:10:

Gas costs:
Gas requirement of function Sustainix_Renewable.updateSwapThreshold 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: 1317:10:

ERC20:
ERC20 contract's "decimals" function should have "uint8" as return type
Pos: 409:4:

Similar variable names:
Sustainix_Renewable.setAMM(address,bool) : Variables have very similar names "AMM" and "isAMM". Note: Modifiers are currently not considered by this static analysis.
Pos: 1416:35:

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.
Pos: 805:14:

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: 1269:33:

Solhint Linter

Sustainix_Renewable.sol

```
Compiler version ^0.8.20 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:3
Avoid to use low level calls.
Pos: 51:175
Avoid to use inline assembly. It is acceptable only in rare cases
Pos: 13:221
Function name must be in mixedCase
Pos: 5:474
Avoid to use low level calls.
Pos: 51:539
Error message for require is too long
Pos: 9:804
Contract has 17 states declarations but allowed no more than 15
Pos: 1:1175
Contract name must be in CamelCase
Pos: 1:1175
Variable name must be in mixedCase
Pos: 22:1239
Explicitly mark visibility in function (Set ignoreConstructors to
true if using solidity >=0.7.0)
Pos: 5:1248
Visibility modifier must be first in list of modifiers
Pos: 60:1280
Avoid making time-based decisions in your business logic
Pos: 106:1376
Variable name must be in mixedCase
Pos: 21:1414
Variable name must be in mixedCase
Pos: 22:1420
Avoid making time-based decisions in your business logic
Pos: 70:1567
Avoid making time-based decisions in your business logic
Pos: 77:1568
Variable "amount" is unused
Pos: 58:1564
```

Software analysis result:

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

Email: audit@EtherAuthority.io