Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: SUN COIN Token
Website: www.sunrewards.io
Platform: PulseChain Network
Language: Solidity
Date: June 24th, 2023

Table of contents

`

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.

Introduction

EtherAuthority was contracted by the SUN COIN team to perform the Security audit of the $UN 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 June 24th, 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 SUN COIN access token ($UN) is an ERC-20 token designed for the PulseChain (PLS) Network.
- The minimum value of $UN is limited to two decimal places (0,01).
- $UN COIN is limited to 5 billion tokens on the PulseChain network.
- Sun Coin is a smart contract that has functions like burn, unpause, pause, mint, etc.
- Sun Coin contract inherits the ERC20, ERC20Burnable, Pausable, and Ownable standard smart contracts from the OpenZeppelin library.
- These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for SUN COIN Token Smart Contract**
Platform	**PULSECHAIN (PLS) / Solidity**
File	UnCoin.sol
Initial Github Commit	2d02c4b37fbf2bd50d0819dfd8a3950e49a5e1fd
Final Github Commit	d72e41975c4b73e44bc4c1ea4bed0eefd7773154
Audit Date	June 24th, 2023
Revised Audit Date	June 29th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Token Name: $UN COIN ● Token Symbol: $UN ● Decimals: 2 ● Maximum Supply: 5 billion ● Minimum Supply: 1 million	**YES, This is valid.**
Owner has control over following functions: ● Set the pause / unpause state in the contract. ● Current owner can transfer ownership of the contract to a new account. ● Deleting ownership will leave the contract without an owner, removing any owner-only functionality.	**YES, This is valid.**
Other features: ● For every purchase made with $UN, a 20% burn of the product price will be made	**There is no coding in the smart contract for this. And hence, this should be done from the Dapp side.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based 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, 0 medium, 1 low and 2 very low level issues.

We confirm that these issues have been fixed / acknowledged in the revised code.

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

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 SUN COIN 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 SUN COIN Token.

The SUN COIN Token team has provided scenario and unit test scripts, which 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 used, which is a good thing.

Documentation

We were given a SUN COIN Token smart contract code in the form of a github repository. The hash of that code is mentioned above in the table.

As mentioned above, the 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 website https://www.sunrewards.io which provided rich information about the project architecture.

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	name	read	Passed	No Issue
3	symbol	read	Passed	No Issue
4	decimals	read	Passed	No Issue
5	totalSupply	read	Passed	No Issue
6	balanceOf	read	Passed	No Issue
7	transfer	write	Passed	No Issue
8	allowance	read	Passed	No Issue
9	approve	write	Passed	No Issue
10	transferFrom	write	Passed	No Issue
11	increaseAllowance	write	Passed	No Issue
12	decreaseAllowance	write	Passed	No Issue
13	_transfer	internal	Passed	No Issue
14	_update	internal	Passed	No Issue
15	_mint	internal	Passed	No Issue
16	_burn	internal	Passed	No Issue
17	_approve	internal	Passed	No Issue
18	_approve	internal	Passed	No Issue
19	_spendAllowance	internal	Passed	No Issue
20	burn	write	Passed	No Issue
21	burnFrom	write	Passed	No Issue
22	whenNotPaused	modifier	Removed	No Issue
23	whenPaused	modifier	Removed	No Issue
24	paused	read	Removed	No Issue
25	_requireNotPaused	internal	Removed	No Issue
26	_requirePaused	internal	Removed	No Issue
27	_pause	internal	Removed	No Issue
28	_unpause	internal	Removed	No Issue
29	onlyOwner	modifier	Passed	No Issue
30	owner	write	Passed	No Issue
31	_checkOwner	internal	Passed	No Issue
32	renounceOwnership	write	access only Owner	No Issue
33	transferOwnership	write	access only Owner	No Issue
34	_transferOwnership	internal	Passed	No Issue
35	decimals	read	Passed	No Issue
36	pause	write	Removed	No Issue
37	unpause	write	Removed	No Issue
38	burn	write	Passed	No Issue
39	burnFrom	write	Passed	No Issue
40	_beforeTokenTransfer	internal	Passed	No Issue
41	claim	write	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.

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

(1) Higher level of ownership control.

We understand this could be the requirement of the business. But from the users point of view, if the token transfer can be paused by anyone, then it creates trust issues.

Resolution: We advise renouncing the ownership once the ownership functions are not needed anymore.

Status: We got confirmation from $UN team that this feature is a required business function.

Very Low / Informational / Best practices:

(1) Make variables constants:

```
uint public MAX_SUPPLY = 5000000000 * 10 ** decimals();
uint public MIN_SUPPLY = 1000000 * 10 ** decimals();
```

These variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constants. Just put a constant keyword.

Status: Fixed

(2) Anti-bot or Anti-whale code would be advantageous.

This is not a requirement. But we have seen sniper bots attacking while adding liquidity in the DEX and eating up the liquidity. So adding its conditions would be beneficial.

Status: We got confirmation from $UN team that the anti-bot feature is not a required business function.

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 usually create trouble. Following are Admin functions:

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.
- _checkOwner: Throws if the sender is not the owner.

This ownership contract is not used anywhere. So, we suggest removing this from the code to make the code clean. However, the presence of this contract does not raise any security issues.

Conclusion

We were given a contract code in the form of a github repository, and we have used all possible tests based on given objects. We have observed 1 low severity issue and 2 Informational severity issues. These issues are fixed / acknowledged in the revised code. **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 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 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).

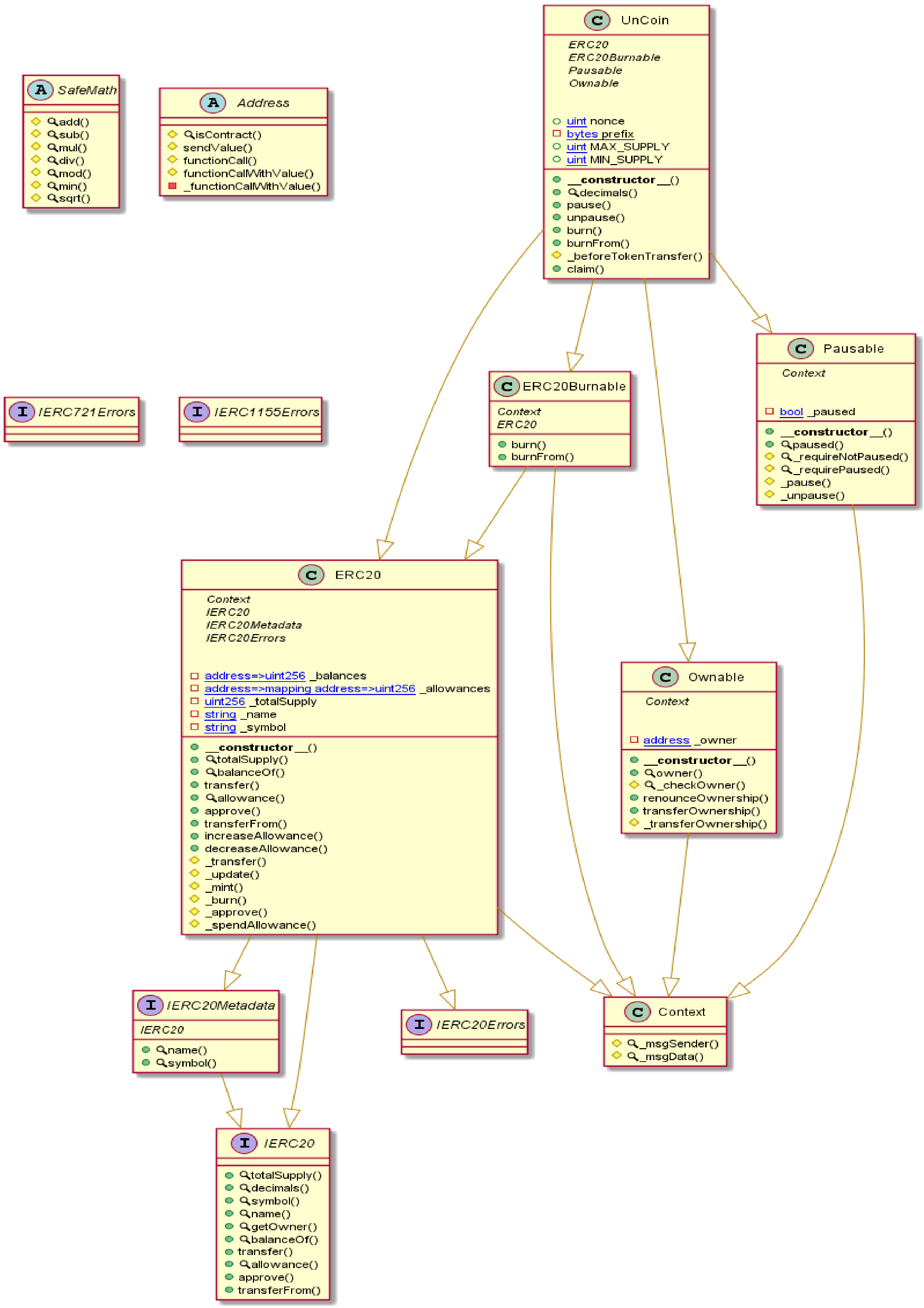
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither Log >> UnCoin.sol

```
UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32) (UnCoin.sol#1019-1040) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(block.timestamp <= _deadline,ERC20Permit: expired deadline) (UnCoin.sol#1027)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (UnCoin.sol#197-208) uses assembly
	- INLINE ASM (UnCoin.sol#204-206)
Address._functionCallWithValue(address,bytes,uint256,string) (UnCoin.sol#305-331) uses assembly
	- INLINE ASM (UnCoin.sol#323-326)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Address._functionCallWithValue(address,bytes,uint256,string) (UnCoin.sol#305-331) is never used and should be removed
Address.functionCall(address,bytes) (UnCoin.sol#252-254) is never used and should be removed
Address.functionCall(address,bytes,string) (UnCoin.sol#262-268) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (UnCoin.sol#281-287) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (UnCoin.sol#295-303) is never used and should be removed
Address.isContract(address) (UnCoin.sol#197-208) is never used and should be removed
Address.sendValue(address,uint256) (UnCoin.sol#226-232) is never used and should be removed
Context._msgData() (UnCoin.sol#463-466) is never used and should be removed
SafeMath.add(uint256,uint256) (UnCoin.sol#18-23) is never used and should be removed
SafeMath.div(uint256,uint256) (UnCoin.sol#96-98) is never used and should be removed
SafeMath.div(uint256,uint256,string) (UnCoin.sol#112-122) is never used and should be removed
SafeMath.min(uint256,uint256) (UnCoin.sol#161-163) is never used and should be removed
SafeMath.mod(uint256,uint256) (UnCoin.sol#136-138) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (UnCoin.sol#152-159) is never used and should be removed
SafeMath.mul(uint256,uint256) (UnCoin.sol#70-82) is never used and should be removed
SafeMath.sqrt(uint256) (UnCoin.sol#166-177) is never used and should be removed
SafeMath.sub(uint256,uint256) (UnCoin.sol#35-37) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (UnCoin.sol#49-58) is never used and should be removed
UnCoin._beforeTokenTransfer(address,address,uint256) (UnCoin.sol#1004-1017) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
UnCoin.MAX_SUPPLY (UnCoin.sol#968) is set pre-construction with a non-constant function or state variable:
	- 5000000000 * 10 ** decimals()
UnCoin.MIN_SUPPLY (UnCoin.sol#969) is set pre-construction with a non-constant function or state variable:
	- 1000000 * 10 ** decimals()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Pragma version^0.8.17 (UnCoin.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6/0.8
.16
solc-0.8.17 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (UnCoin.sol#226-232):
	- (success) = recipient.call{value: amount}() (UnCoin.sol#230)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (UnCoin.sol#305-331):
	- (success,returndata) = target.call{value: weiValue}(data) (UnCoin.sol#314)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._hashedMessage (UnCoin.sol#1020) is not in mixedCase
Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._value (UnCoin.sol#1021) is not in mixedCase
Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._deadline (UnCoin.sol#1022) is not in mixedCase
Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._v (UnCoin.sol#1023) is not in mixedCase
Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._r (UnCoin.sol#1024) is not in mixedCase
Parameter UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32)._s (UnCoin.sol#1025) is not in mixedCase
Constant UnCoin.prefix (UnCoin.sol#967) is not in UPPER_CASE_WITH_UNDERSCORES
Variable UnCoin.MAX_SUPPLY (UnCoin.sol#968) is not in mixedCase
Variable UnCoin.MIN_SUPPLY (UnCoin.sol#969) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (UnCoin.sol#464)" inContext (UnCoin.sol#458-467)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

UnCoin.slitherConstructorVariables() (UnCoin.sol#965-1041) uses literals with too many digits:
	- MAX_SUPPLY = 5000000000 * 10 ** decimals() (UnCoin.sol#968)
UnCoin.slitherConstructorVariables() (UnCoin.sol#965-1041) uses literals with too many digits:
	- MIN_SUPPLY = 1000000 * 10 ** decimals() (UnCoin.sol#969)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
```

```
UnCoin (UnCoin.sol#965-1041) does not implement functions:
	- IERC20.getOwner() (UnCoin.sol#357)
	- IERC20Metadata.name() (UnCoin.sol#431)
	- IERC20Metadata.symbol() (UnCoin.sol#436)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions

UnCoin.MAX_SUPPLY (UnCoin.sol#968) should be immutable
UnCoin.MIN_SUPPLY (UnCoin.sol#969) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
UnCoin.sol analyzed (13 contracts with 84 detectors), 43 result(s) found
```

Solidity Static Analysis

UnCoin.sol

Security

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: 70:16:

Gas & Economy

Gas costs:

Gas requirement of function UnCoin.claim 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: 62:4:

Miscellaneous

Similar variable names:

UnCoin.claim(bytes32,uint256,uint256,uint8,bytes32,bytes32) : Variables have very similar names "_r" and "_s". Note: Modifiers are currently not considered by this static analysis.
Pos: 78:64:

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: 80:8:

Solhint Linter

UnCoin.sol

```
UnCoin.sol:2:1: Error: Compiler version ^0.8.17 does not satisfy the
r semver requirement
UnCoin.sol:10:28: Error: Constant name must be in capitalized
SNAKE_CASE
UnCoin.sol:11:17: Error: Variable name must be in mixedCase
UnCoin.sol:12:17: Error: Variable name must be in mixedCase
UnCoin.sol:14:5: Error: Explicitly mark visibility in function (Set
ignoreConstructors to true if using solidity >=0.7.0)
UnCoin.sol:14:44: Error: Code contains empty blocks
UnCoin.sol:70:17: Error: Avoid to make time-based decisions in your
business logic
```

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