EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: SingularityNET Token
Website: singularitynet.io
Platform: Ethereum
Language: Solidity
Date: April 20th, 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.

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the SingularityNET smart contract from singularitynet.io was audited extensively. 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 20th, 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

- This contract inherits from several other contracts and libraries to implement functionalities like access control, ERC20 token standards, pausing, burning, and minting.
- Here's a breakdown of what this contract does:
    - **Libraries:** The contract imports and uses various libraries like EnumerableSet, SafeMath, and Address to implement set operations, safe arithmetic operations, and address-related functions.
    - **AccessControl:** The contract defines roles (MINTER_ROLE and PAUSER_ROLE) and assigns them to specific accounts. These roles control who can mint new tokens and pause/unpause the contract.
    - **ERC20 Token:** The contract implements the ERC20 token standard with functionalities like transferring tokens, approving spending, allowance management, total supply, balance inquiries, etc.
    - **Burnable:** This contract allows tokens to be burned (destroyed) by the token owner or by another authorized account.
    - **Pausable:** The contract can be paused and unpaused, preventing token transfers while paused to avoid potential issues or attacks.
    - **Constructor:** The constructor initializes the contract by setting up the default admin role and assigning the minter and pauser roles to the contract deployer.

- Overall, this contract provides a standard ERC20 token with additional features such as access control, burning, and pausing.

Audit scope

Name	**Code Review and Security Analysis Report for SingularityNET Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	SingularityNetToken.sol
Smart Contract Code	0x5b7533812759b45c2b44c19e320ba2cd2681b542
Audit Date	April 20th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: SingularityNET Token • Symbol: AGIX • Decimals: 8	**YES, This is valid.**
Other Specifications: • Minting new tokens by the minter role owner. • Pauses/Unpauses all token transfers by the pauser role owner.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based 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 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, 1 medium, 1 low, and 2 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	0%
Sell Tax	0%
Cannot Buy	No
Cannot Sell	No
Max Tax	0%
Modify Tax	No
Fee Check	Not Detected
Is Honeypot	Not Detected
Trading Cooldown	Not Detected
Can Pause Trade?	Not Detected
Pause Transfer?	No
Max Tax?	No
Is it Anti-whale?	Not Detected
Is Anti-bot?	Not Detected
Is it a Blacklist?	No
Blacklist Check	No
Can Mint?	Yes
Is it a Proxy?	No
Can Take Ownership?	No
Hidden Owner?	Not Detected
Self Destruction?	Not Detected
Auditor Confidence	High

Overall Audit Result: PASSED

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 SingularityNET 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 SingularityNET Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a SingularityNET Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are well commented on. And 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.

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

SingularityNetToken.sol

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	mint	write	Centralization Risk, Minter can mint unlimited token	Refer Audit Findings
3	pause	write	Centralization Risk	Refer Audit Findings
4	unpause	write	Centralization Risk	Refer Audit Findings
5	_beforeTokenTransfer	internal	Passed	No Issue
6	_beforeTokenTransfer	internal	Passed	No Issue
7	paused	read	Passed	No Issue
8	whenNotPaused	modifier	Passed	No Issue
9	whenPaused	modifier	Passed	No Issue
10	_pause	internal	whenNotPaused	No Issue
11	_unpause	internal	whenPaused	No Issue
12	burn	write	Centralization Risk	Refer Audit Findings
13	burnFrom	write	Passed	No Issue
14	name	read	Passed	No Issue
15	symbol	read	Passed	No Issue
16	decimals	read	Passed	No Issue
17	totalSupply	read	Passed	No Issue
18	balanceOf	read	Passed	No Issue
19	transfer	write	Passed	No Issue
20	allowance	write	Passed	No Issue
21	approve	write	Passed	No Issue
22	transferFrom	write	Passed	No Issue
23	increaseAllowance	write	Passed	No Issue
24	decreaseAllowance	write	Passed	No Issue
25	_transfer	internal	Passed	No Issue
26	_mint	internal	Passed	No Issue
27	_burn	internal	Passed	No Issue
28	_approve	internal	Passed	No Issue
29	_setupDecimals	internal	Passed	No Issue
30	_beforeTokenTransfer	internal	Passed	No Issue
31	hasRole	read	Passed	No Issue
32	getRoleMemberCount	read	Passed	No Issue
33	getRoleMember	read	Passed	No Issue
34	getRoleAdmin	read	Passed	No Issue
35	grantRole	write	Passed	No Issue
36	revokeRole	write	Passed	No Issue

37	renounceRole	write	Passed	No Issue
38	_setupRole	internal	Passed	No Issue
39	_setRoleAdmin	internal	Passed	No Issue
40	_grantRole	write	Passed	No Issue
41	_revokeRole	write	Passed	No Issue
42	_msgSender	internal	Passed	No Issue
43	_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 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.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Centralization Risk:

```
function pause() public virtual {    infinite gas
    require(hasRole(PAUSER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must have pauser role to pause");
    _pause();
}
```

```
function unpause() public virtual {    infinite gas
    require(hasRole(PAUSER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must have pauser role to unpause");
    _unpause();
}
```

```
function mint(address to, uint256 amount) public virtual {    infinite gas
    require(hasRole(MINTER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must have minter role to mint");
    _mint(to, amount);
}
```

The PAUSER_ROLE can pause and unpause transfer,

The MINTER_ROLE can mint tokens in the contract.

Resolution: We suggest making smart contracts 100% decentralized.

Low

(1) Minter can mint unlimited tokens:

```
function mint(address to, uint256 amount) public virtual {    infinite gas
    require(hasRole(MINTER_ROLE, _msgSender()), "ERC20PresetMinterPauser: must have minter role to mint");
    _mint(to, amount);
}
```

There is no limit for minting tokens. Thus the owner can mint unlimited tokens to any account.

Resolution: There should be a limit for minting or need to confirm, if it is a part of the plan then disregard this issue.

Very Low / Informational / Best practices:

(1) Multiple pragma:

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

/** ...
library EnumerableSet { ...
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.6.2;

/** ...
library Address { ...
}
// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;

```

There are multiple pragmas with different compiler versions.

Resolution: We suggest using only one pragma and removing the other.

(2) Assembly:

```
function isContract(address account) internal view returns (bool) {
    // This method relies in extcodesize, which returns 0 for contracts in
    // construction, since the code is only stored at the end of the
    // constructor execution.

    uint256 size;
    // solhint-disable-next-line no-inline-assembly
    assembly { size := extcodesize(account) }
    return size > 0;
}
```

```
function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private
    require(isContract(target), "Address: call to non-contract");

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = target.call{ value: weiValue }(data);
    if (success) {
        return returndata;
    } else {
        // Look for revert reason and bubble it up if present
        if (returndata.length > 0) {
            // The easiest way to bubble the revert reason is using memory via assembly

            // solhint-disable-next-line no-inline-assembly
            assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }
        } else {
```

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Resolution: It is recommended to use assembly only when necessary.

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. The following are Admin functions:

SingularityNETToken.sol

- mint: Minting new tokens by the minter role owner.
- pause: Pauses all token transfers by the pauser role owner.
- unpause: Unpauses all token transfers by the pauser role owner.

AccessControl.sol

- grantRole: Grants `role` to `account` can be set by the admin.
- revokeRole: Revokes `role` from `account` by the admin.
- renounceRole: Renounce Role from `account` by the admin.

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 Etherscan web links. And we have used all possible tests based on given objects as files. We observed 1 medium,1 low, and 2 Informational issues in the smart contracts. but those are not critical. 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 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).

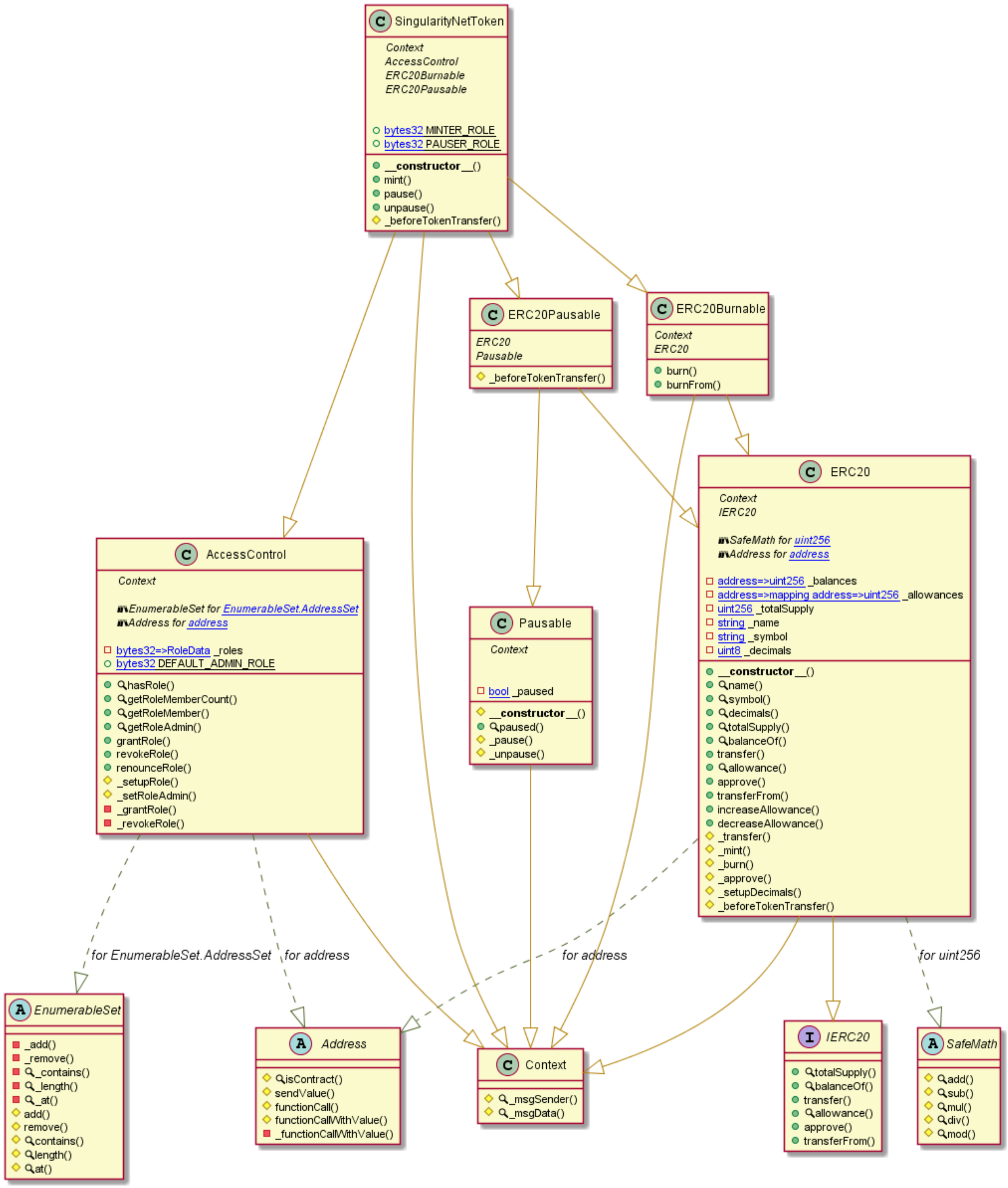
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While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

```
INFO:Detectors:
ERC20.constructor(string,string).name (SingularityNetToken.sol#913) shadows:
        - ERC20.name() (SingularityNetToken.sol#922-924) (function)
ERC20.constructor(string,string).symbol (SingularityNetToken.sol#913) shadows:
        - ERC20.symbol() (SingularityNetToken.sol#930-932) (function)
SingularityNetToken.constructor(string,string).name (SingularityNetToken.sol#1363) shadows:
        - ERC20.name() (SingularityNetToken.sol#922-924) (function)
SingularityNetToken.constructor(string,string).symbol (SingularityNetToken.sol#1363) shadows:

        - ERC20.symbol() (SingularityNetToken.sol#930-932) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shado
wing
INFO:Detectors:
Address.isContract(address) (SingularityNetToken.sol#273-282) uses assembly
        - INLINE ASM (SingularityNetToken.sol#280)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
AccessControl._setRoleAdmin(bytes32,bytes32) (SingularityNetToken.sol#596-599) is never used
and should be removed
Address._functionCallWithValue(address,bytes,uint256,string) (SingularityNetToken.sol#366-371
) is never used and should be removed
Address.functionCall(address,bytes) (SingularityNetToken.sol#326-328) is never used and shoul
d be removed
Address.functionCall(address,bytes,string) (SingularityNetToken.sol#336-338) is never used an
d should be removed
Address.functionCallWithValue(address,bytes,uint256) (SingularityNetToken.sol#351-353) is nev
er used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (SingularityNetToken.sol#361-364)
 is never used and should be removed
```

```
Address.functionCallWithValue(address,bytes,uint256,string) (SingularityNetToken.sol#361-364)
 is never used and should be removed
Address.isContract(address) (SingularityNetToken.sol#273-282) is never used and should be rem
oved
Address.sendValue(address,uint256) (SingularityNetToken.sol#300-306) is never used and should
 be removed
Context._msgData() (SingularityNetToken.sol#389-392) is never used and should be removed
EnumerableSet.add(EnumerableSet.UintSet,uint256) (SingularityNetToken.sol#208-210) is never u
sed and should be removed
EnumerableSet.at(EnumerableSet.UintSet,uint256) (SingularityNetToken.sol#246-248) is never us
ed and should be removed
EnumerableSet.contains(EnumerableSet.UintSet,uint256) (SingularityNetToken.sol#225-227) is ne
ver used and should be removed
EnumerableSet.length(EnumerableSet.UintSet) (SingularityNetToken.sol#232-234) is never used a
nd should be removed
EnumerableSet.remove(EnumerableSet.UintSet,uint256) (SingularityNetToken.sol#218-220) is neve
r used and should be removed
```

```
SafeMath.div(uint256,uint256,string) (SingularityNetToken.sol#814-820) is never used and shou
ld be removed
SafeMath.mod(uint256,uint256) (SingularityNetToken.sol#834-836) is never used and should be r
emoved
SafeMath.mod(uint256,uint256,string) (SingularityNetToken.sol#850-853) is never used and shou
ld be removed
SafeMath.mul(uint256,uint256) (SingularityNetToken.sol#772-784) is never used and should be r
emoved
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.6.0 (SingularityNetToken.sol#9) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#399) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#618) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#698) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#860) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#1170) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#1212) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#1304) allows old versions
Pragma version^0.6.0 (SingularityNetToken.sol#1332) allows old versions
solc-0.6.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-o
f-solidity
INFO:Detectors:
Redundant expression "this (SingularityNetToken.sol#390)" inContext (SingularityNetToken.sol#
384-393)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

INFO:Slither:SingularityNetToken.sol analyzed (11 contracts with 93 detectors), 35 result(s)
found
```


Email: audit@EtherAuthority.io

Solidity Static Analysis

Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

SingularityNetToken.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.
more
Pos: 280:8:

Gas costs:

Gas requirement of function SingularityNetToken.renounceRole 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: 565:4:

Gas costs:

Gas requirement of function SingularityNetToken.mint 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: 1382:4:

Gas costs:

Gas requirement of function SingularityNetToken.unpause 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: 1410:4:

Constant/View/Pure functions:

SingularityNetToken._beforeTokenTransfer(address,address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 1415:4:

No return:

Address._functionCallWithValue(address,bytes,uint256,string): Defines a return type but never explicitly returns a value.

Pos: 366:4:

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: 1411:8:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

more

Pos: 104:12:

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: 781:16:

Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

SingularityNetToken.sol

```
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:8
Error message for require is too long
Pos: 9:136
Variable "recipient" is unused
Pos: 24:299
Error message for require is too long
Pos: 9:361
Variable "data" is unused
Pos: 53:365
Variable "weiValue" is unused
Pos: 72:365
Variable "errorMessage" is unused
Pos: 90:365
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:398
Error message for require is too long
Pos: 9:530
Error message for require is too long
Pos: 9:545
Error message for require is too long
Pos: 9:565
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:617
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:697
Error message for require is too long
Pos: 9:780
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:859
Error message for require is too long
Pos: 9:1065
Error message for require is too long
Pos: 9:1066
Error message for require is too long
Pos: 9:1106
Error message for require is too long
```

```
Pos: 9:1129
Error message for require is too long
Pos: 9:1130
Code contains empty blocks
Pos: 94:1161
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1169
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1211
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1303
Error message for require is too long
Pos: 9:1325
Compiler version ^0.6.0 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:1331
Error message for require is too long
Pos: 9:1382
Error message for require is too long
Pos: 9:1396
Error message for require is too long
Pos: 9:1410
```

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

EtherAuthority

Email: audit@EtherAuthority.io