Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

# SMART CONTRACT

## Security Audit Report

| | |
|---|---|
| Project: | ThorSwap Token |
| Website: | https://thorswap.co |
| Platform: | CORE Chain Network |
| Language: | Solidity |
| Date: | April 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 ThorSwap Token team to perform the Security audit of the ThorSwap Token smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on April 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

- ThorSwap Token is a bep20 token.
- ThorSwap is a unique platform that combines exciting features of NFT trades, earn, gaming and launchpad built on CoreDao.

# Audit scope

| **Name** | **Code Review and Security Analysis Report for ThorSwap Token Smart Contract** |
|---|---|
| **Platform** | **CORE Chain Network / Solidity** |
| **File** | ThorSwapToken.sol |
| **File MD5 Hash** | FA11DC1CB80A17DA6981BDD46CD7BA56 |
| **Online code link** | 0x06F0A74EBC394328A08aBE3ef0D57008A7c2e38b |
| **Audit Date** | April 14th, 2023 |


**Email: audit@EtherAuthority.io**

# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
|---|---|
| **Tokenomics:**<br>• Name: ThorSwap<br>• Symbol: THOR<br>• Decimals: 18<br>• Maximum Supply: 100 Million | **YES, This is valid.** |
| **Owner Specifications:**<br>• Current owner can transfer ownership of the contract to a new account.<br>• Deleting ownership will leave the contract without an owner, removing any owner-only functionality. | **YES, This is valid.** |

# 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 and 0 low and some very low level issues.**

**Investors Advice:** Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

# Technical Quick Stats

| Main Category | Subcategory | Result |
|---|---|---|
| Contract Programming | Solidity version not specified | Passed |
| | Solidity version too old | Passed |
| | Integer overflow/underflow | Passed |
| | Function input parameters lack of check | Passed |
| | Function input parameters check bypass | Passed |
| | Function access control lacks management | Passed |
| | Critical operation lacks event log | 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**


**Email: audit@EtherAuthority.io**

## 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 ThorSwap 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 ThorSwap Token.

The ThorSwap Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are well commented on in the smart contracts. Ethereum's NatSpec commenting style is used, which is a good thing.

## Documentation

We were given a ThorSwap Token smart contract code in the form of a scan.coredao.org web link The hash of that code is mentioned above in the table.

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

Another source of information was its official website: https://thorswap.co 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 | owner | read | Passed | No Issue |
| 3 | onlyOwner | modifier | Passed | No Issue |
| 4 | renounceOwnership | write | access only Owner | No Issue |
| 5 | transferOwnership | write | access only Owner | No Issue |
| 6 | _transferOwnership | internal | Multiple SPDX license identifier used | Refer to audit findings |
| 7 | _msgSender | internal | Passed | No Issue |
| 8 | _msgData | internal | Multiple SPDX license identifier used | Refer to audit findings |
| 9 | getOwner | external | Passed | No Issue |
| 10 | decimals | external | Passed | No Issue |
| 11 | symbol | external | Passed | No Issue |
| 12 | name | external | Passed | No Issue |
| 13 | totalSupply | external | Passed | No Issue |
| 14 | balanceOf | external | Passed | No Issue |
| 15 | transfer | external | Passed | No Issue |
| 16 | allowance | external | Passed | No Issue |
| 17 | approve | external | Passed | No Issue |
| 18 | transferFrom | external | Passed | No Issue |
| 19 | increaseAllowance | write | Passed | No Issue |
| 20 | decreaseAllowance | write | Passed | No Issue |
| 21 | burn | write | Passed | No Issue |
| 22 | burnFrom | write | Passed | No Issue |
| 23 | _transfer | internal | Passed | No Issue |
| 24 | _burn | internal | Passed | No Issue |
| 25 | _approve | 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) Please use the latest compiler version when deploying contract:

This is not a severe issue, but we suggest using the latest compiler version at the time of contract deployment, which is 0.8.18 at the time of this audit. Using the latest compiler version is always recommended which prevents any compiler level issues.

(2) SafeMath Library:

```
359  contract ThorSwapToken is Context, IBEP20, Ownable {
360    using SafeMath for uint256;
361
362    mapping(address => uint256) private _balances;
363
364    mapping(address => mapping(address => uint256)) private _allowances;
365
```

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

**Resolution:** Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

(3) Multiple // SPDX-License-Identifier: UNLICENSED used:

```
352     function _transferOwnership(address newOwner) internal {
353       require(newOwner != address(0), 'Ownable: new owner is the zero address');
354       emit OwnershipTransferred(_owner, newOwner);
355       _owner = newOwner;
356     // SPDX-License-Identifier: UNLICENSED
357     }
358   }
```

```
99    // SPDX-License-Identifier: UNLICENSED
100     event Approval(address indexed owner, address indexed spender, uint256 value);
101   }
102   /**
```

```
253     function mod(
254       uint256 a,
255       uint256 b,
256       string memory errorMessage
257     ) internal pure returns (uint256) {
258       require(b != 0, errorMessage);
259       return a % b;
260     // SPDX-License-Identifier: UNLICENSED
261     }
262   }
```

```
277
278     function _msgData() internal view virtual returns (bytes calldata) {
279       this;
280
281       return msg.data;
282     // SPDX-License-Identifier: UNLICENSED
283     }
284   }
```

MIT License is mentioned still in 4 places found // SPDX-License-Identifier: UNLICENSED string.

**Resolution:** We suggest removing unwanted or unrelated commented lines.

(4) State variables declared as public which is redundant:

```
367     uint8 public _decimals;
368     string public _symbol;
369     string public _name;
```

Below State variables declared as public which is redundant.

- uint8 public _decimals;
- string public _symbol;
- string public _name;

**Resolution:** Declare mentioned State variables as private which can be accessed with the help of already declared functions.

- decimals()
- symbol()
- name()

# 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:

### 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 a scan.coredao.org link and we have used all possible tests based on given objects as files. We have observed some informational severity issues in the token smart contract. But those issues are not critical ones. So, **it's good to go for the 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).

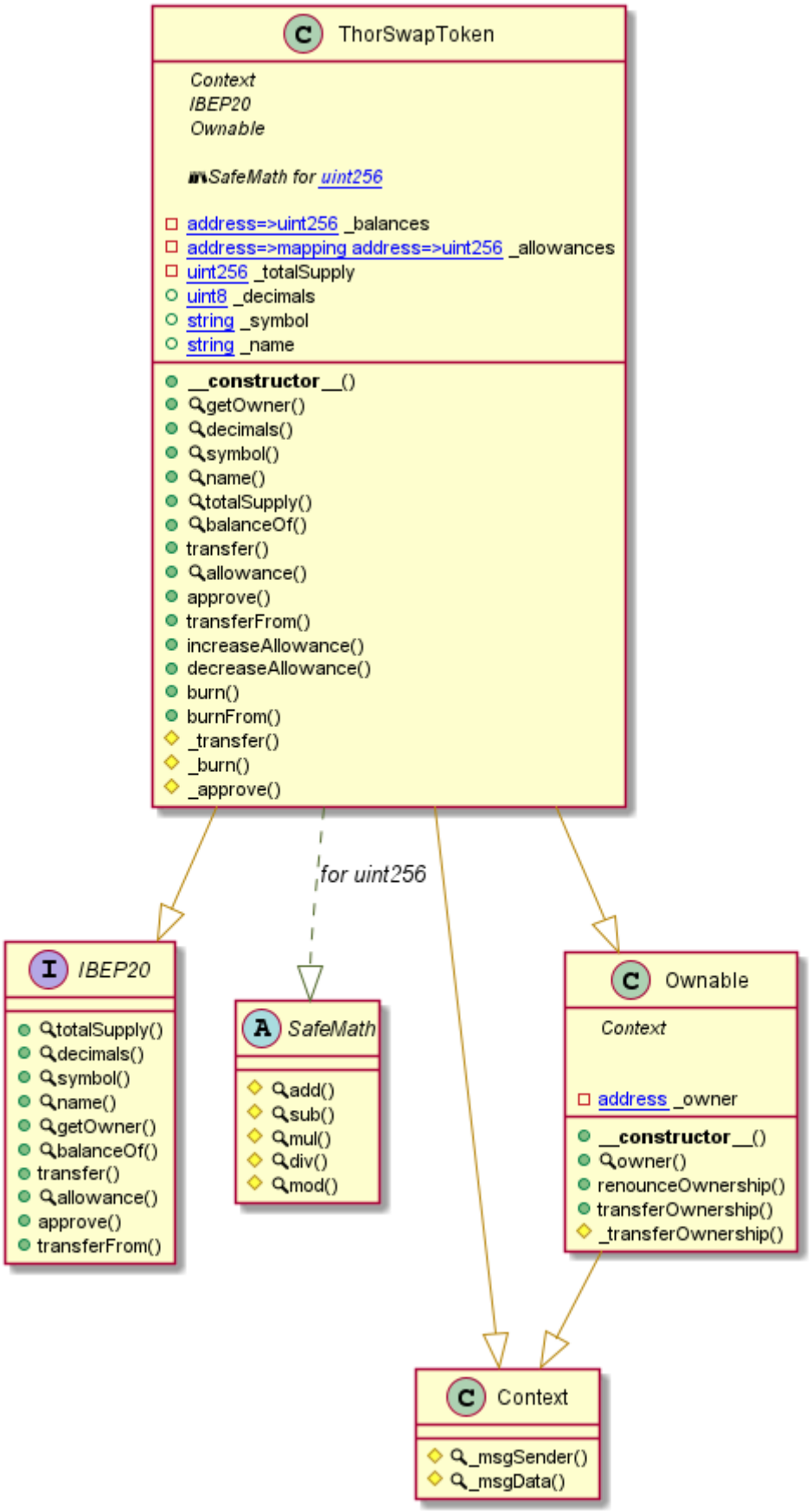
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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






**Email: audit@EtherAuthority.io**

# Slither Results Log

## Slither Log >> ThorSwapToken.sol

```
ThorSwapToken.allowance(address,address).owner (ThorSwapToken.sol#449) shadows:
        - Ownable.owner() (ThorSwapToken.sol#317-319) (function)
ThorSwapToken._approve(address,address,uint256).owner (ThorSwapToken.sol#653) shadows:
        - Ownable.owner() (ThorSwapToken.sol#317-319) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Context._msgData() (ThorSwapToken.sol#278-283) is never used and should be removed
SafeMath.div(uint256,uint256) (ThorSwapToken.sol#199-201) is never used and should be removed
SafeMath.div(uint256,uint256,string) (ThorSwapToken.sol#214-225) is never used and should be removed
SafeMath.mod(uint256,uint256) (ThorSwapToken.sol#238-240) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (ThorSwapToken.sol#253-261) is never used and should be removed
SafeMath.mul(uint256,uint256) (ThorSwapToken.sol#174-186) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Pragma version^0.8.0 (ThorSwapToken.sol#2) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Variable ThorSwapToken._decimals (ThorSwapToken.sol#367) is not in mixedCase
Variable ThorSwapToken._symbol (ThorSwapToken.sol#368) is not in mixedCase
Variable ThorSwapToken._name (ThorSwapToken.sol#369) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Redundant expression "this (ThorSwapToken.sol#279)" inContext (ThorSwapToken.sol#273-284)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements

ThorSwapToken._decimals (ThorSwapToken.sol#367) should be immutable
ThorSwapToken._name (ThorSwapToken.sol#369) should be immutable
ThorSwapToken._symbol (ThorSwapToken.sol#368) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
ThorSwapToken.sol analyzed (5 contracts with 84 detectors), 17 result(s) found
```

# Solidity Static Analysis

**ThorSwapToken.sol**

## Gas & Economy

### Gas costs:

Gas requirement of function ThorSwapToken.burn 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: 561:2:

### Gas costs:

Gas requirement of function ThorSwapToken.burnFrom 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: 576:2:

## ERC

### ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type
more
Pos: 13:2:

## Miscellaneous

### Constant/View/Pure functions:

SafeMath.mod(uint256,uint256) : Is constant but potentially should not be.
Note: Modifiers are currently not considered by this static analysis.
more
Pos: 238:2:

### Similar variable names:

ThorSwapToken._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.
Pos: 636:39:

### No return:

IBEP20.transferFrom(address,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 81:2:

### 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: 658:4:

### 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: 221:16:

# Solhint Linter

**ThorSwapToken.sol**

```
ThorSwapToken.sol:2:1: Error: Compiler version ^0.8.3 does not
satisfy the r semver requirement
ThorSwapToken.sol:127:21: Error: Use double quotes for string
literals
ThorSwapToken.sol:142:22: Error: Use double quotes for string
literals
ThorSwapToken.sol:183:25: Error: Use double quotes for string
literals
ThorSwapToken.sol:200:22: Error: Use double quotes for string
literals
ThorSwapToken.sol:239:22: Error: Use double quotes for string
literals
ThorSwapToken.sol:308:3: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
ThorSwapToken.sol:325:37: Error: Use double quotes for string
literals
ThorSwapToken.sol:353:37: Error: Use double quotes for string
literals
ThorSwapToken.sol:371:3: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
ThorSwapToken.sol:372:13: Error: Use double quotes for string
literals
ThorSwapToken.sol:373:15: Error: Use double quotes for string
literals
ThorSwapToken.sol:497:9: Error: Use double quotes for string literals
ThorSwapToken.sol:550:9: Error: Use double quotes for string literals
ThorSwapToken.sol:580:9: Error: Use double quotes for string literals
ThorSwapToken.sol:606:35: Error: Use double quotes for string
literals
ThorSwapToken.sol:607:38: Error: Use double quotes for string
literals
ThorSwapToken.sol:611:7: Error: Use double quotes for string literals
ThorSwapToken.sol:629:36: Error: Use double quotes for string
literals
ThorSwapToken.sol:633:7: Error: Use double quotes for string literals
ThorSwapToken.sol:657:34: Error: Use double quotes for string
literals
ThorSwapToken.sol:658:36: Error: Use double quotes for string
literals
```

**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**