Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	TallowFinance
Website:	https://tallow.finance
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 12th, 2021

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 PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

We were contracted by the TallowFinance team to perform the Security audit of the TallowFinance 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 12th, 2021.

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

TallowFinance is a community driven, fair launched DeFi Token. Three simple functions occur during each trade: Reflection, LP Acquisition, and Burn.

Audit scope

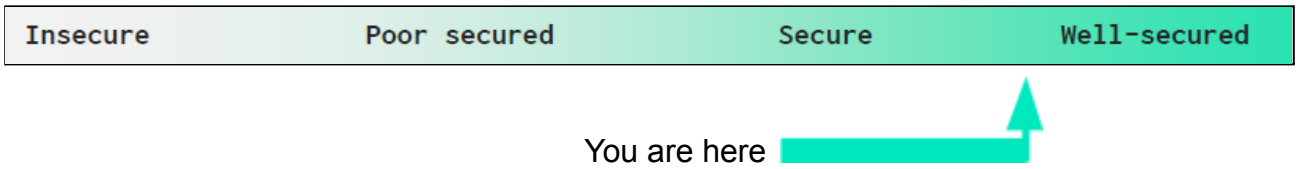
Name	**Code Review and Security Analysis Report for TallowFinance Token (TLF) Smart Contract**
Platform	**BSC / Solidity**
File	TallowFinance.sol
Smart Contract Online Code	https://bscscan.com/address/0x78a8258e09ce20ca79d4b13493a1fe59eb5fb763#code
File MD5 Hash	48E6DE7727A3E58E110F0BA90A2F0045
Audit Date	June 12th, 2021
Revised Smart Contract Code	https://bscscan.com/address/0x0818581b66677ef9680094988009044a373ffbc2#code
Revision Date	June 13th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: TallowFinance Token Symbol: TLF Decimals: 9	**YES, This is valid.**
No minting of tokens	**YES, This is valid.**
Chain: Binance Smart Chain (BEP-20)	**YES, This is valid.**
Tax fee: 2% Liquidity fee: 2% Burn fee: 2%	**YES, This is valid. Owner can change these fees.**
Max Transaction Amount: 700,000	**YES, This is valid. Owner can change this value.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **secured**. This contract also has owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.



We used various tools like MythX, Slither 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.

We found 0 critical, 0 high, 0 medium and 5 low and some very low level issues.

These issues are fixed / acknowledged in the revised smart contract code.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Resolved
	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	Resolved
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	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
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Resolved
	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. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. This is a compact and well written contract.

The libraries in the TallowFinance 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 TallowFinance Token.

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

Some code parts are **not well** commented on smart contracts.

Documentation

We were given TallowFinance Token smart contract code in the form of a BscScan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult 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://www.tallow.finance/ 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. And their core code blocks are written well.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

TallowFinance token is a smart contract, having functionality like Reflection, Burn, LP Acquisition, etc.

TallowFinance.sol

(1) Interface

(a) IERC20

(b) IUniswapV2Factory

(c) IUniswapV2Pair

(d) IUniswapV2Router01

(e) IUniswapV2Router02

(2) Inherited contracts

(a) Context

(b) Ownable

(c) IERC20

(3) Usages

(a) using SafeMath for uint256;

(b) using Address for address;

(4) Events

(a) event Transfer(address indexed from, address indexed to, uint256 value);

(b) event Approval(address indexed owner, address indexed spender, uint256 value);

(c) event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

(d) event PairCreated(address indexed token0, address indexed token1, address pair, uint);

(e) event Approval(address indexed owner, address indexed spender, uint value);

(f) event Transfer(address indexed from, address indexed to, uint value);

(g) event Mint(address indexed sender, uint amount0, uint amount1);

(h) event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);

(i) event Swap(address indexed sender,uint amount0In,uint amount1In,uint amount0Out,uint amount1Out,address indexed to);

(j) event Sync(uint112 reserve0, uint112 reserve1);

(k) event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);

(l) event SwapAndLiquifyEnabledUpdated(bool enabled);

(m) event SwapAndLiquify(uint256 tokensSwapped,uint256 ethReceived,uint256 tokensIntoLiqudity);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	lockTheSwap	modifier	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	isExcludedFromReward	read	Passed	No Issue
14	totalFees	read	Passed	No Issue
15	deliver	write	external instead of public	Resolved
16	_burnToken	write	Passed	No Issue
17	reflectionFromToken	read	external instead of public	Resolved
18	tokenFromReflection	read	Passed	No Issue
19	excludeFromReward	write	access only Owner	No Issue
20	includeInReward	external	Infinite loop possibility	Resolved
21	_transferBothExcluded	write	Passed	No Issue
22	excludeFromFee	write	Missing Events	Resolved
23	includeInFee	write	Missing Events	Resolved
24	setTaxFeePercent	external	Missing Events	Resolved
25	setLiquidityFeePercent	external	Missing Events	Resolved
26	setBurnFeePercent	external	Missing Events	Resolved

27	setMaxTxPercent	external	Missing Events	Resolved
28	setSwapAndLiquifyEnabled	write	Missing Events	Resolved
29	_reflectFee	write	Passed	No Issue
30	_getValues	read	Passed	No Issue
31	_getTValues	read	Passed	No Issue
32	_getRValues	write	Passed	No Issue
33	_getRate	read	Passed	No Issue
34	_getCurrentSupply	read	Passed	No Issue
35	_takeLiquidity	write	Passed	No Issue
36	calculateTaxFee	read	Passed	No Issue
37	calculateLiquidityFee	read	Passed	No Issue
38	calculateBurnFee	read	Passed	No Issue
39	removeAllFee	write	Passed	No Issue
40	restoreAllFee	write	Passed	No Issue
41	isExcludedFromFee	read	Passed	No Issue
42	_approve	write	Passed	No Issue
43	_transfer	write	Passed	No Issue
44	swapAndLiquify	write	Passed	No Issue
45	swapTokensForEth	write	Passed	No Issue
46	addLiquidity	write	Centralized risk in addLiquidity	Resolved
47	_tokenTransfer	write	Passed	No Issue
48	_transferStandard	write	Passed	No Issue
49	_transferToExcluded	write	Passed	No Issue
50	_transferFromExcluded	write	Passed	No Issue
51	owner	read	Passed	No Issue
52	onlyOwner	modifier	Passed	No Issue
53	renounceOwnership	write	access only Owner	No Issue
54	transferOwnership	write	access only Owner	No Issue
55	geUnlockTime	read	Passed	No Issue
56	lock	write	Removed	No Issue
57	unlock	write	Removed	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 functions
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

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Possible to gain ownership after renouncing the contract ownership. Owner can renounce ownership and make contract without owner but here is a catch, owner can misuse it by performing the following operations:

- Owner calls the lock function in contract to set the current owner as _previousOwner.
- Owner calls unlock to unlock contract and set _owner = _previousOwner.
- Owner called renounceOwnership to leave the contract without the owner.
- Owner calls unlock to regain ownership.

Resolution: This issue is resolved in revised smart contract code.

(2) Infinite loop possibility:

```
function includeInReward(address account) external onlyOwner() {
    require(_isExcluded[account], "Account is already excluded");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

If there are so many excluded wallets, then this logic will fail, as it might hit the block's gas limit. If there are very limited exceptions, then this will work, but will cost more gas.

Resolution: This issue is resolved in revised smart contract code.

(3) Make variables constant:

```
string private _name = "TallowFinance";
string private _symbol = "TLF";
uint8 private _decimals = 9;
```

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

- name
- symbol
- decimals

Resolution: This issue is resolved in revised smart contract code.

(4) Centralized risk in addLiquidity:

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router), tokenAmount);
     require( block.timestamp >= releaseTime);
    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        address(this),
        block.timestamp
    );
}
```

In the addLiquidity function, owner() gets TLF Tokens from the Pool. At some time, The owner will accumulate significant TLF tokens. If the _owner is an EOA (Externally Owned Account), mishandling of its private key can have devastating consequences to the project.

Resolution: This issue is acknowledged.

.

(5) Missing Events in following functions:

- excludeFromFee
- excludeFromReward
- includeInFee
- includeInReward
- setLiquidityFeePercent
- setMaxTxPercent
- setTaxFeePercent
- setSwapAndLiquifyEnabled

Resolution: This issue is resolved in revised smart contract code.

Very Low / Discussion / Best practices:

(1) external instead of public:
If any function is not called from inside the smart contract, then it is better to declare it as external instead of public. As it saves some gas as well.
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Resolution: This issue is acknowledged.

Centralization

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

- setSwapAndLiquifyEnabled: Owner can set swap and Liquify enable.
- setMaxTxPercent: Owner can set maximum TxAmount.
- setBurnFeePercent: Owner can set burn fee Percentage.
- setLiquidityFeePercent: Owner can set Liquidity Fee Percentage.
- setTaxFeePercent: Owner can set tax Fee Percentage.
- includeInFee: Owner can check include Fee.
- excludeFromFee: Owner can check exclude Fee.
- includeInReward: Owner can include rewards.
- excludeFromReward: Owner can check if the account is excluded.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those are fixed/acknowledged in the smart contracts. **So it is good to go for the production.**

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high level description of functionality was presented in As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed 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).

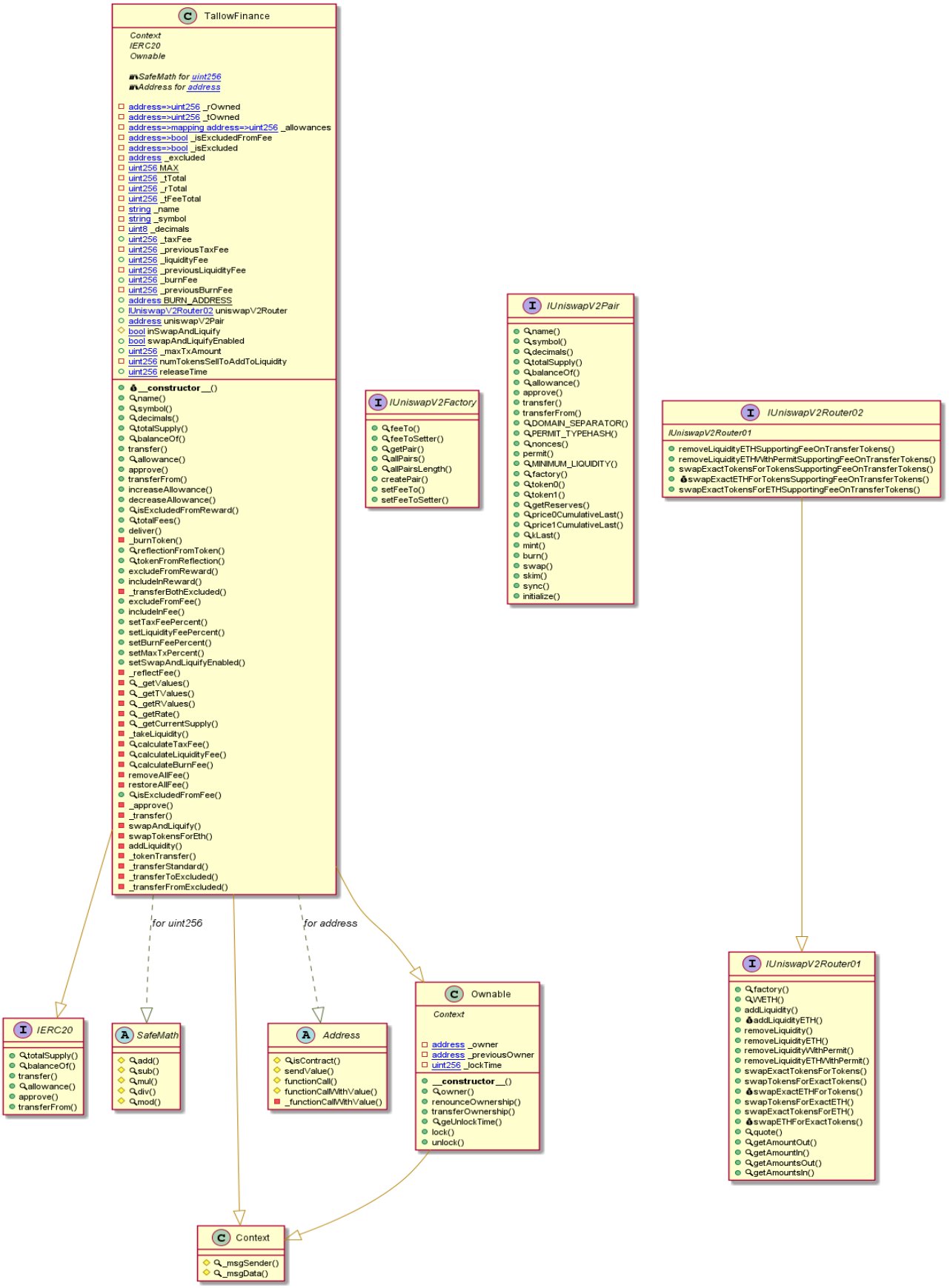
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither log >> TallowFinance.sol

```
INFO:Detectors:
Reentrancy in TallowFinance._transfer(address,address,uint256) (TallowFinance.sol#1000-1044):
	External calls:
	- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (TallowFinance.sol#1073-1079)
	External calls sending eth:
	- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
	State variables written after the call(s):
	- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
		- _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (TallowFinance.sol#948)
		- _rOwned[BURN_ADDRESS] = _rOwned[BURN_ADDRESS] + rBurn (TallowFinance.sol#808)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (TallowFinance.sol#1126)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (TallowFinance.sol#1136)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (TallowFinance.sol#856)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (TallowFinance.sol#1148)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (TallowFinance.sol#1127)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (TallowFinance.sol#1149)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (TallowFinance.sol#1138)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (TallowFinance.sol#858)
	- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
		- _rTotal = _rTotal.sub(rFee) (TallowFinance.sol#901)
	- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
		- _tOwned[BURN_ADDRESS] = _tOwned[BURN_ADDRESS] + tBurn (TallowFinance.sol#810)
		- _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity) (TallowFinance.sol#950)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (TallowFinance.sol#855)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (TallowFinance.sol#1147)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (TallowFinance.sol#1137)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (TallowFinance.sol#857)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
TallowFinance.addLiquidity(uint256,uint256) (TallowFinance.sol#1086-1101) ignores return value by
uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
TallowFinance.allowance(address,address).owner (TallowFinance.sol#762) shadows:
	- Ownable.owner() (TallowFinance.sol#409-411) (function)
TallowFinance._approve(address,address,uint256).owner (TallowFinance.sol#992) shadows:
	- Ownable.owner() (TallowFinance.sol#409-411) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in TallowFinance._transfer(address,address,uint256) (TallowFinance.sol#1000-1044):
	External calls:
	- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
		- uniswapV2Router.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
		-
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(t
his),block.timestamp) (TallowFinance.sol#1073-1079)
	External calls sending eth:
```

- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)

State variables written after the call(s):
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _burnFee = _previousBurnFee (TallowFinance.sol#985)
  - _burnFee = 0 (TallowFinance.sol#979)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _liquidityFee = _previousLiquidityFee (TallowFinance.sol#984)
  - _liquidityFee = 0 (TallowFinance.sol#978)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _previousBurnFee = _burnFee (TallowFinance.sol#975)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _previousLiquidityFee = _liquidityFee (TallowFinance.sol#974)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _previousTaxFee = _taxFee (TallowFinance.sol#973)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _tFeeTotal = _tFeeTotal.add(tFee) (TallowFinance.sol#902)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
  - _taxFee = _previousTaxFee (TallowFinance.sol#983)
  - _taxFee = 0 (TallowFinance.sol#977)

Reentrancy in TallowFinance.constructor() (TallowFinance.sol#724-734):

External calls:
- uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (TallowFinance.sol#728-729)

State variables written after the call(s):
- _isExcludedFromFee[owner()] = true (TallowFinance.sol#731)
- _isExcludedFromFee[address(this)] = true (TallowFinance.sol#732)
- uniswapV2Router = _uniswapV2Router (TallowFinance.sol#730)

Reentrancy in TallowFinance.swapAndLiquify(uint256) (TallowFinance.sol#1046-1062):

External calls:
- swapTokensForEth(half) (TallowFinance.sol#1056)
  - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (TallowFinance.sol#1073-1079)
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)

External calls sending eth:
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)

State variables written after the call(s):
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
  - _allowances[owner][spender] = amount (TallowFinance.sol#996)

Reentrancy in TallowFinance.transferFrom(address,address,uint256) (TallowFinance.sol#771-775):

External calls:
- _transfer(sender,recipient,amount) (TallowFinance.sol#772)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
  - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (TallowFinance.sol#1073-1079)

External calls sending eth:
- _transfer(sender,recipient,amount) (TallowFinance.sol#772)
  - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)

State variables written after the call(s):
- _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (TallowFinance.sol#773)
  - _allowances[owner][spender] = amount (TallowFinance.sol#996)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

INFO:Detectors:

Reentrancy in TallowFinance._transfer(address,address,uint256) (TallowFinance.sol#1000-1044):

External calls:
- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (TallowFinance.sol#1073-1079)
External calls sending eth:
- swapAndLiquify(contractTokenBalance) (TallowFinance.sol#1031)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
Event emitted after the call(s):
- Transfer(sender,recipient,tTransferAmount) (TallowFinance.sol#1131)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
- Transfer(sender,recipient,tTransferAmount) (TallowFinance.sol#1142)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
- Transfer(sender,recipient,tTransferAmount) (TallowFinance.sol#1153)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
- Transfer(sender,recipient,tTransferAmount) (TallowFinance.sol#862)
- _tokenTransfer(from,to,amount,takeFee) (TallowFinance.sol#1043)
Reentrancy in TallowFinance.constructor() (TallowFinance.sol#724-734):
External calls:
- uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (TallowFinance.sol#728-729)
Event emitted after the call(s):
- Transfer(address(0),_msgSender(),_tTotal) (TallowFinance.sol#733)
Reentrancy in TallowFinance.swapAndLiquify(uint256) (TallowFinance.sol#1046-1062):
External calls:
- swapTokensForEth(half) (TallowFinance.sol#1056)
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (TallowFinance.sol#1073-1079)
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
External calls sending eth:
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
Event emitted after the call(s):
- Approval(owner,spender,amount) (TallowFinance.sol#997)
- addLiquidity(otherHalf,newBalance) (TallowFinance.sol#1060)
- SwapAndLiquify(half,newBalance,otherHalf) (TallowFinance.sol#1061)
Reentrancy in TallowFinance.transferFrom(address,address,uint256) (TallowFinance.sol#771-775):
External calls:
- _transfer(sender,recipient,amount) (TallowFinance.sol#772)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (TallowFinance.sol#1073-1079)
External calls sending eth:
- _transfer(sender,recipient,amount) (TallowFinance.sol#772)
- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(this),block.timestamp) (TallowFinance.sol#1093-1100)
Event emitted after the call(s):
- Approval(owner,spender,amount) (TallowFinance.sol#997)
- _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (TallowFinance.sol#773)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Ownable.unlock() (TallowFinance.sol#456-461) uses timestamp for comparisons
Dangerous comparisons:

- require(bool,string)(block.timestamp > _lockTime,Contract is locked until 7 days) (TallowFinance.sol#458)
TallowFinance.addLiquidity(uint256,uint256) (TallowFinance.sol#1086-1101) uses timestamp for comparisons
Dangerous comparisons:
- require(bool)(block.timestamp >= releaseTime) (TallowFinance.sol#1091)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (TallowFinance.sol#261-270) uses assembly
- INLINE ASM (TallowFinance.sol#268)
Address._functionCallWithValue(address,bytes,uint256,string) (TallowFinance.sol#354-375) uses assembly
- INLINE ASM (TallowFinance.sol#367-370)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (TallowFinance.sol#354-375) is never used and should be removed
Address.functionCall(address,bytes) (TallowFinance.sol#314-316) is never used and should be removed
Address.functionCall(address,bytes,string) (TallowFinance.sol#324-326) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (TallowFinance.sol#339-341) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (TallowFinance.sol#349-352) is never used and should be removed
Address.isContract(address) (TallowFinance.sol#261-270) is never used and should be removed
Address.sendValue(address,uint256) (TallowFinance.sol#288-294) is never used and should be removed
Context._msgData() (TallowFinance.sol#233-236) is never used and should be removed
SafeMath.mod(uint256,uint256) (TallowFinance.sol#206-208) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (TallowFinance.sol#222-225) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
TallowFinance._rTotal (TallowFinance.sol#681) is set pre-construction with a non-constant function or state variable:
- (MAX - (MAX % _tTotal))
TallowFinance._previousTaxFee (TallowFinance.sol#689) is set pre-construction with a non-constant function or state variable:
- _taxFee
TallowFinance._previousLiquidityFee (TallowFinance.sol#692) is set pre-construction with a non-constant function or state variable:
- _liquidityFee
TallowFinance._previousBurnFee (TallowFinance.sol#696) is set pre-construction with a non-constant function or state variable:
- _burnFee
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (TallowFinance.sol#288-294):
- (success) = recipient.call{value: amount}() (TallowFinance.sol#292)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (TallowFinance.sol#354-375):
- (success,returndata) = target.call{value: weiValue}(data) (TallowFinance.sol#358)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (TallowFinance.sol#495) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (TallowFinance.sol#496) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (TallowFinance.sol#513) is not in mixedCase
Function IUniswapV2Router01.WETH() (TallowFinance.sol#533) is not in mixedCase
Parameter TallowFinance.setSwapAndLiquifyEnabled(bool)._enabled (TallowFinance.sol#892) is not in mixedCase
Parameter TallowFinance.calculateTaxFee(uint256)._amount (TallowFinance.sol#953) is not in mixedCase
Parameter TallowFinance.calculateLiquidityFee(uint256)._amount (TallowFinance.sol#959) is not in mixedCase
Parameter TallowFinance.calculateBurnFee(uint256)._amount (TallowFinance.sol#965) is not in mixedCase
Variable TallowFinance._taxFee (TallowFinance.sol#688) is not in mixedCase
Variable TallowFinance._liquidityFee (TallowFinance.sol#691) is not in mixedCase
Variable TallowFinance._burnFee (TallowFinance.sol#695) is not in mixedCase
Variable TallowFinance._maxTxAmount (TallowFinance.sol#707) is not in mixedCase

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (TallowFinance.sol#234)" inContext (TallowFinance.sol#228-237)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable
IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tADesired (TallowFinance.sol#538) is too similar to
IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amoun
tBDesired (TallowFinance.sol#539)
Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount
(TallowFinance.sol#1125) is too similar to TallowFinance._getValues(uint256).tTransferAmount
(TallowFinance.sol#906)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1135) is too similar to
TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount
(TallowFinance.sol#1125) is too similar to
TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1135) is too similar to
TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1135) is too similar to TallowFinance._getValues(uint256).tTransferAmount
(TallowFinance.sol#906)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to
TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to
TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to
TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to TallowFinance._getValues(uint256).tTransferAmount
(TallowFinance.sol#906)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1135) is too similar to TallowFinance._getTValues(uint256).tTransferAmount
(TallowFinance.sol#915)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to TallowFinance._getTValues(uint256).tTransferAmount
(TallowFinance.sol#915)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to
TallowFinance._getTValues(uint256).tTransferAmount (TallowFinance.sol#915)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount
(TallowFinance.sol#924) is too similar to TallowFinance._getTValues(uint256).tTransferAmount
(TallowFinance.sol#915)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#854) is too similar to TallowFinance._getTValues(uint256).tTransferAmount
(TallowFinance.sol#915)
Variable TallowFinance._transferFromExcluded(address,address,uint256).rTransferAmount
(TallowFinance.sol#1146) is too similar to
TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount
(TallowFinance.sol#1125) is too similar to
TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount
(TallowFinance.sol#924) is too similar to TallowFinance._getValues(uint256).tTransferAmount
(TallowFinance.sol#906)
Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount
(TallowFinance.sol#1125) is too similar to
TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)

Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount (TallowFinance.sol#1125) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._getValues(uint256).tTransferAmount (TallowFinance.sol#906)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to TallowFinance._getValues(uint256).tTransferAmount (TallowFinance.sol#906)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#1135) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._getTValues(uint256).tTransferAmount (TallowFinance.sol#915)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount (TallowFinance.sol#924) is too similar to TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#854) is too similar to TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._transferToExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#1135) is too similar to TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount (TallowFinance.sol#924) is too similar to TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#854) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#854) is too similar to TallowFinance._transferToExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1135)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount (TallowFinance.sol#924) is too similar to TallowFinance._transferStandard(address,address,uint256).tTransferAmount (TallowFinance.sol#1125)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance._getValues(uint256).rTransferAmount (TallowFinance.sol#907) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance._getRValues(uint256,uint256,uint256,uint256,uint256).rTransferAmount (TallowFinance.sol#924) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#854) is too similar to TallowFinance._transferFromExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#1146)
Variable TallowFinance.reflectionFromToken(uint256,bool).rTransferAmount (TallowFinance.sol#820) is too similar to TallowFinance._transferBothExcluded(address,address,uint256).tTransferAmount (TallowFinance.sol#854)
Variable TallowFinance._transferBothExcluded(address,address,uint256).rTransferAmount (TallowFinance.sol#854) is too similar to TallowFinance._getValues(uint256).tTransferAmount (TallowFinance.sol#906)
Variable TallowFinance._transferStandard(address,address,uint256).rTransferAmount (TallowFinance.sol#1125) is too similar to TallowFinance._getTValues(uint256).tTransferAmount (TallowFinance.sol#915)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
TallowFinance.slitherConstructorConstantVariables() (TallowFinance.sol#666-1160) uses literals with too many digits:
	- BURN_ADDRESS = 0x0000000000000000000000000000000000000000 (TallowFinance.sol#697)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
TallowFinance._decimals (TallowFinance.sol#686) should be constant
TallowFinance._name (TallowFinance.sol#684) should be constant
TallowFinance._symbol (TallowFinance.sol#685) should be constant
TallowFinance._tTotal (TallowFinance.sol#680) should be constant
TallowFinance.numTokensSellToAddToLiquidity (TallowFinance.sol#708) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (TallowFinance.sol#428-431)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (TallowFinance.sol#437-441)
geUnlockTime() should be declared external:
	- Ownable.geUnlockTime() (TallowFinance.sol#443-445)
lock(uint256) should be declared external:
	- Ownable.lock(uint256) (TallowFinance.sol#448-453)
unlock() should be declared external:
	- Ownable.unlock() (TallowFinance.sol#456-461)
name() should be declared external:
	- TallowFinance.name() (TallowFinance.sol#736-738)
symbol() should be declared external:
	- TallowFinance.symbol() (TallowFinance.sol#740-742)
decimals() should be declared external:
	- TallowFinance.decimals() (TallowFinance.sol#744-746)
totalSupply() should be declared external:
	- TallowFinance.totalSupply() (TallowFinance.sol#748-750)
transfer(address,uint256) should be declared external:
	- TallowFinance.transfer(address,uint256) (TallowFinance.sol#757-760)
allowance(address,address) should be declared external:
	- TallowFinance.allowance(address,address) (TallowFinance.sol#762-764)
approve(address,uint256) should be declared external:
	- TallowFinance.approve(address,uint256) (TallowFinance.sol#766-769)
transferFrom(address,address,uint256) should be declared external:
	- TallowFinance.transferFrom(address,address,uint256) (TallowFinance.sol#771-775)
increaseAllowance(address,uint256) should be declared external:
	- TallowFinance.increaseAllowance(address,uint256) (TallowFinance.sol#777-780)
decreaseAllowance(address,uint256) should be declared external:
	- TallowFinance.decreaseAllowance(address,uint256) (TallowFinance.sol#782-785)
isExcludedFromReward(address) should be declared external:
	- TallowFinance.isExcludedFromReward(address) (TallowFinance.sol#787-789)
totalFees() should be declared external:
	- TallowFinance.totalFees() (TallowFinance.sol#791-793)
deliver(uint256) should be declared external:
	- TallowFinance.deliver(uint256) (TallowFinance.sol#795-802)
reflectionFromToken(uint256,bool) should be declared external:
	- TallowFinance.reflectionFromToken(uint256,bool) (TallowFinance.sol#814-823)
excludeFromReward(address) should be declared external:
	- TallowFinance.excludeFromReward(address) (TallowFinance.sol#831-839)
excludeFromFee(address) should be declared external:
	- TallowFinance.excludeFromFee(address) (TallowFinance.sol#865-867)
includeInFee(address) should be declared external:
	- TallowFinance.includeInFee(address) (TallowFinance.sol#869-871)
setSwapAndLiquifyEnabled(bool) should be declared external:
	- TallowFinance.setSwapAndLiquifyEnabled(bool) (TallowFinance.sol#892-895)
isExcludedFromFee(address) should be declared external:
	- TallowFinance.isExcludedFromFee(address) (TallowFinance.sol#988-990)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Email: audit@EtherAuthority.io