Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Stroopwafel Token
Website:	stroopwafel.finance
Platform:	Binance Smart Chain
Language:	Solidity
Date:	September 27th, 2021

Table of contents

Introduction

EtherAuthority was contracted by the Stroopwafel team to perform the Security audit of the Stroopwafel Token and master chef 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 September 27th, 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

StroopWafel Finance is a decentralized exchange running on Binance Smart Chain which integrates the power of NFTs into DeFi. The Stroopwafel Token (SWAFEL) is a BEP20 standard token and is at the backbone of the StroopWafel ecosystem.

Audit scope

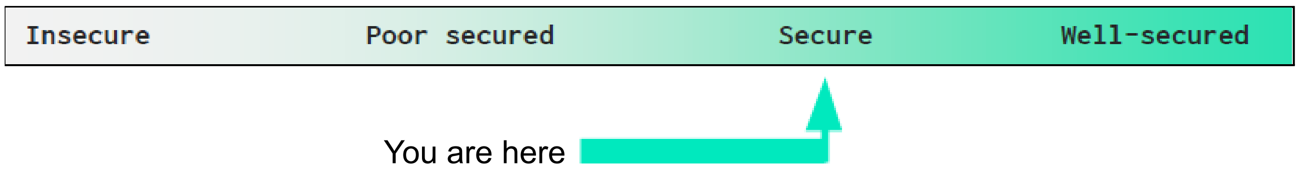
Name	Code Review and Security Analysis Report for Stroopwafel Token and Master Chef Smart Contract
Platform	**BSC / Solidity**
SWAFEL Contract	https://bscscan.com/address/0x5D39953964E141C4ffc78149280C65347ab80DdC#code
Master Chef Contract	https://bscscan.com/address/0x9ab01FE39E778F2FDb38f85929A50dcF3cE99766#code
Audit Date	September 27th, 2021

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
StroopwafelToken.sol ● Name: Stroopwafel Token ● Symbol: SWAFEL ● Decimals: 18 ● Chain: Binance Smart Chain (BEP-20) ● Max Supply: 1,800,000 ● Current owner: Presale contract	**YES, This is valid.** **The owner of this token contract is the Presale contract. And we got confirmation from the Stroopwafel team that the Ownership will be transferred to the Master Chef after presale.**
MasterChef.sol ● Bonus Multiplier: 1 ● Mints SWAFEL tokens as needed ● MasterChef owner is Timelock contract ● Migrator code: Non existence	**YES, This is valid.** **Timelock contract provides users with an extra layer of safety by providing buffer time of minimum 6 hours before any owner functions can come into effect.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Technically Secured"**. The Master Chef contract contains owner control, which does not make it fully decentralized. However, time locked control makes it a safer approach.



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 3 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	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	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	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: PASSED

Code Quality

This audit scope has two smart contracts. Smart contracts also contain Libraries, Smart contracts, inherits and Interfaces. These are compact and well written contracts.

The libraries in Stroopwafel 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.

The Stroopwafel 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 **not well** commented on smart contracts.

Documentation

We were given a Stroopwafel smart contracts code in the form of a BSCScan web links, which are mentioned above in the table.

As mentioned above, code parts are **not well** commented. So it is not 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://stroopwafel.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

SWAFEL Token

Sl.	Functions	Type	Observation	Conclusion
1	constructor	read	Passed	No Issue
2	mint	write	access only Owner	No Issue
3	delegates	external	Passed	No Issue
4	delegate	external	Passed	No Issue
5	delegateBySig	external	Handle signatures securely	No Issue
6	getCurrentVotes	external	Passed	No Issue
7	getPriorVotes	external	Infinite Loop possibility	Refer audit finding section
8	_delegate	internal	Passed	No Issue
9	_moveDelegates	internal	Passed	No Issue
10	_writeCheckpoint	internal	Passed	No Issue
11	safe32	internal	Passed	No Issue
12	getChainId	internal	Passed	No Issue
13	getOwner	external	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	read	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	mint	write	access only Owner	No Issue
26	_transfer	internal	Passed	No Issue
27	_mint	internal	Passed	No Issue
28	_burn	internal	Passed	No Issue
29	_approve	internal	Passed	No Issue
30	_burnFrom	internal	Passed	No Issue

Master Chef Contract

1	poolLength	external	Passed	No Issue
2	add	write	Input validation missing	Refer audit finding section
3	set	write	access only Owner	No Issue
4	getMultiplier	read	Passed	No Issue
5	pendingSTROOPWAFEL	external	Passed	No Issue
6	massUpdatePools	write	Infinite Loop possibility	Refer audit finding section
7	updatePool	write	Passed	No Issue
8	deposit	write	Passed	No Issue
9	withdraw	write	Passed	No Issue
10	emergencyWithdraw	write	Passed	No Issue
11	safeSTROOPWAFELTransfer	internal	Passed	No Issue
12	dev	write	Passed	No Issue
13	setFeeAddress	write	Passed	No Issue
14	updateEmissionRate	write	access only Owner	No Issue

Severity Definitions

Risk Level	**Description**
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Audit Findings

Critical

No Critical severity vulnerabilities were found.

High

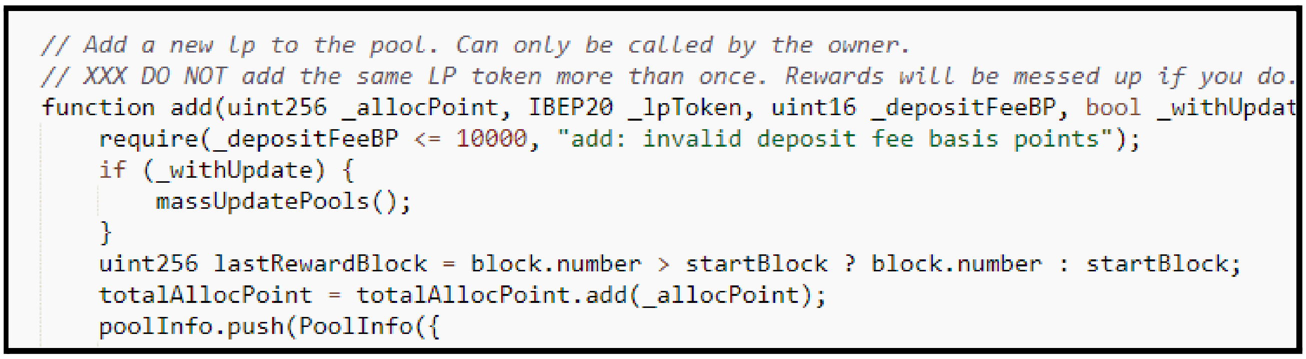
No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Duplicate LP tokens may create discrepancy - Master Chef contract

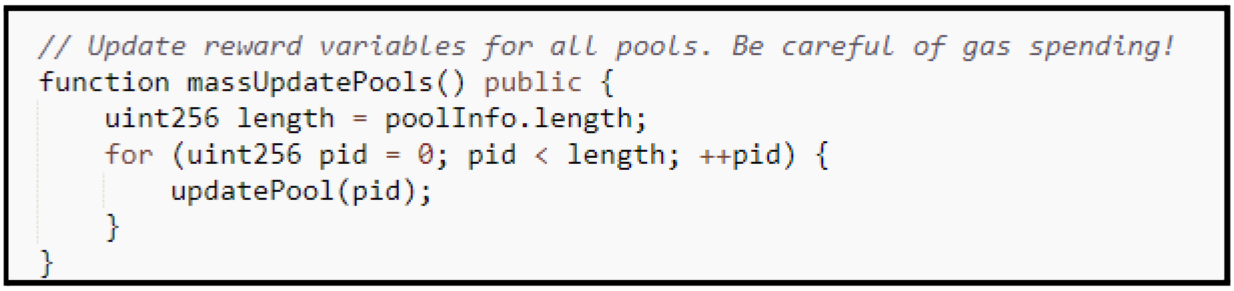


```
// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool _withUpdat
    require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
```

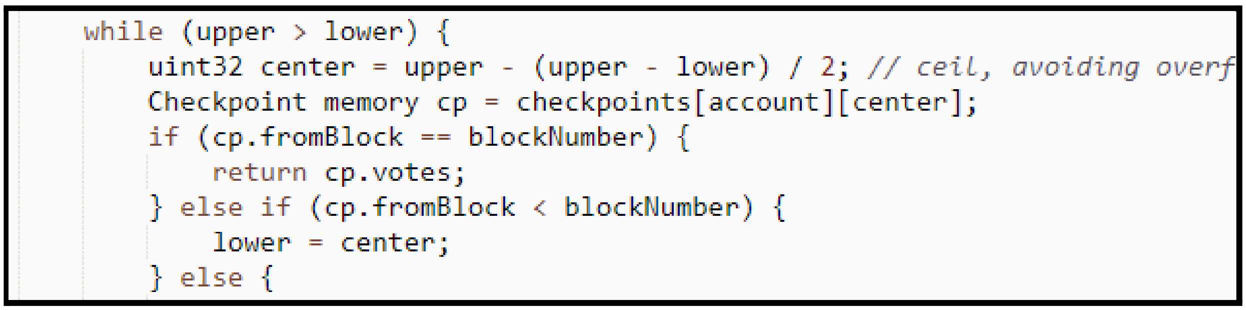
As per the comment, the add function must not allow the owner to add the same LP token more than once.

Resolution: We suggest using some conditions to not allow the user to add the same LP token more than once.

(2) Infinite loop possibility - Master Chef and SWAFEL contracts



```
// Update reward variables for all pools. Be careful of gas spending!
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}
```



```
    while (upper > lower) {
        uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overf
        Checkpoint memory cp = checkpoints[account][center];
        if (cp.fromBlock == blockNumber) {
            return cp.votes;
        } else if (cp.fromBlock < blockNumber) {
            lower = center;
        } else {
```

In the master chef contract, if the size of pools increases, then it will consume more gas. And because there are no limits on adding pools, there is a minor possibility of infinite loops.

The same way as getting prior votes in the SWAFEL token contract, there is a possibility of high gas-consuming loops.

Resolution: We recommend keeping array length limited to prevent such high gas consuming code execution.

(3) Critical state changing functions must fire an event to properly coordinate with the clients. Following functions in the master chef contract must emit events:

- add
- set
- updatePool
- dev
- setFeeAddress
- updateEmissionRate

Very Low / Informational / Best practices:

(1) Consider using the latest solidity version

v0.6.12+commit.27d51765

We recommend using the latest solidity version to prevent the compiler level bugs. The solidity version at the time of audit is: 0.8.7

(2) Visibility external over public:

It is recommended to specify function visibility as external instead of public, if that function is not called from the contract internally. This is considered a gas saver.
https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

Centralized Risk

The SWAFEL token contract is currently owned by a pre-sale contract.. We got confirmation from the Stroopwafel team that the ownership will be transferred to the master chef contract after the presale.

The Master Chef contract owner has tremendous power to access the fund, which creates centralized risk. And to manage this risk, the Stroopwafel team has implemented the timelock contract. Time lock contract allows buffer time of minimum 6 hours for any owner function to come into effect.

If any user is not agreeing to any owner actions, they can withdraw their funds within 6 hours. This is an approach for centralized risk mitigation. However, users are responsible to watch the owner's functions in case they want to withdraw their funds.

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 issues are not critical ones. So, **it’s good to go to 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 the 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 **“Technically 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).

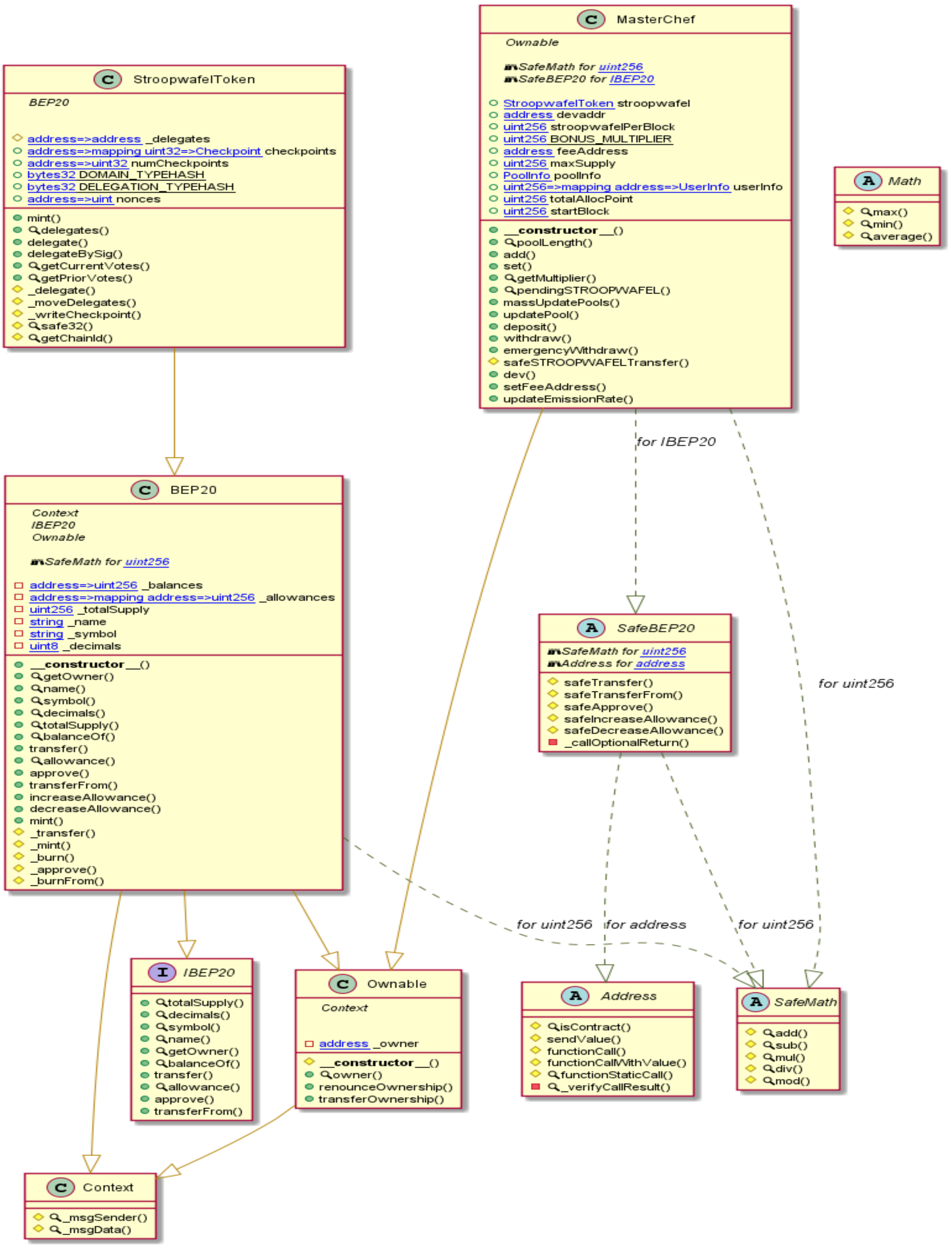
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

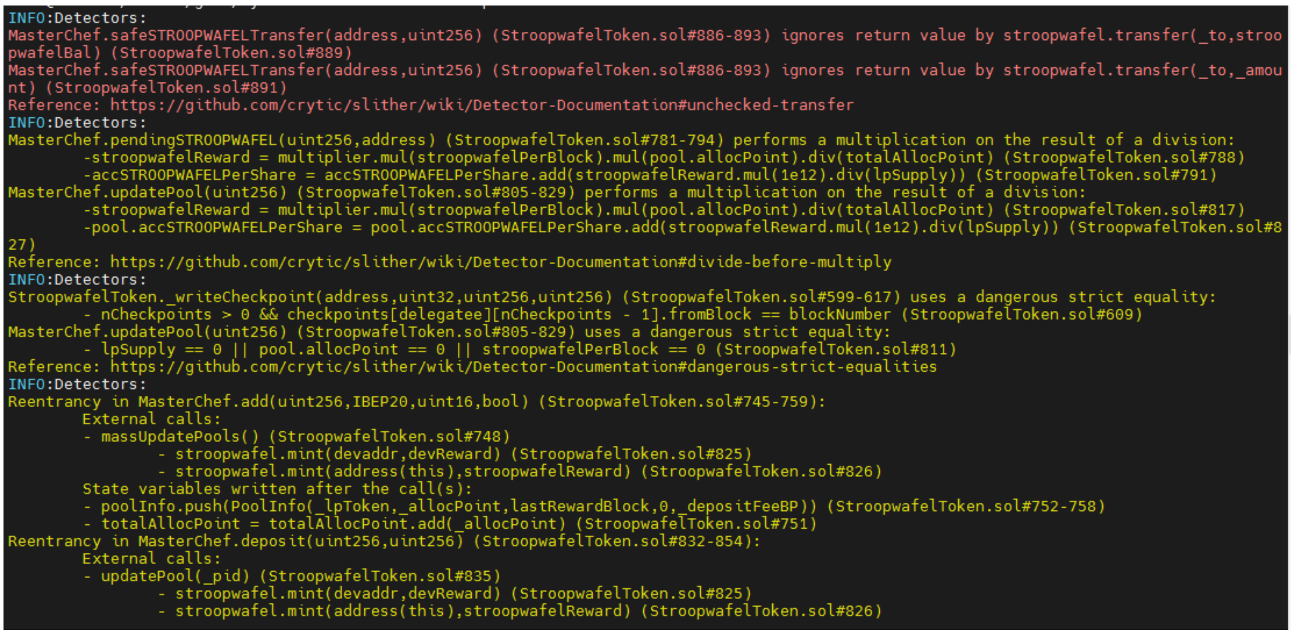
Appendix

Code Flow Diagram

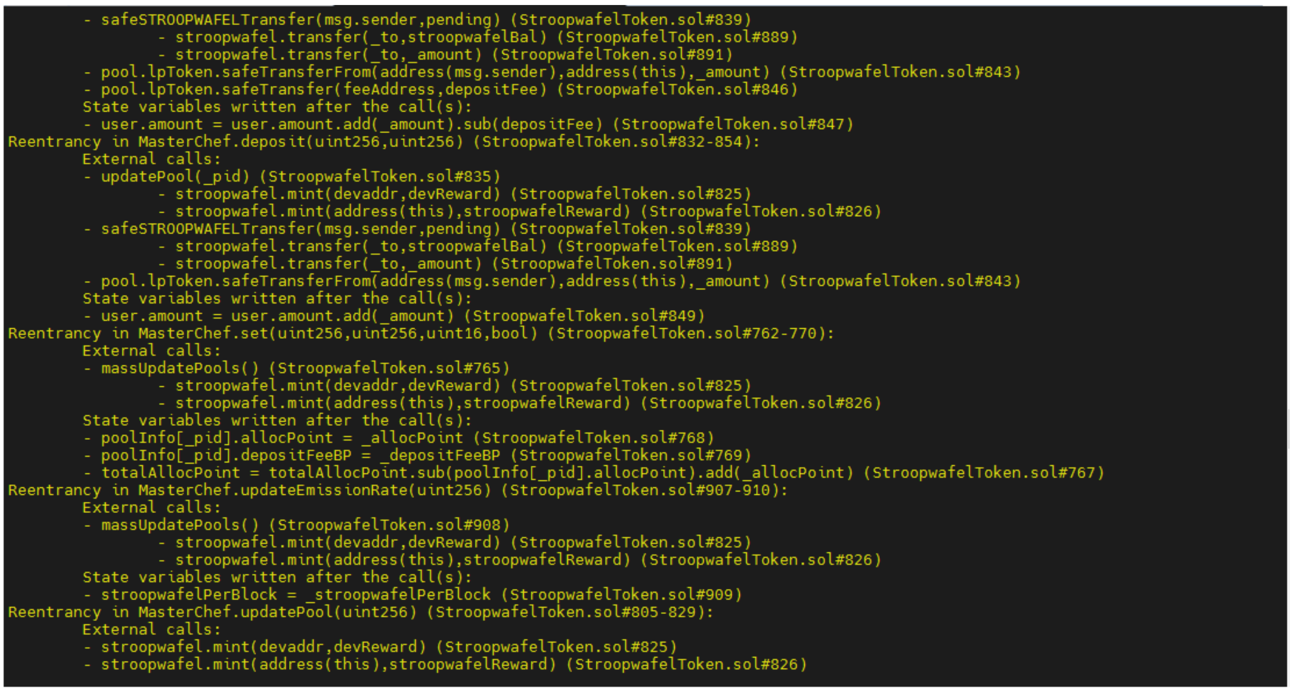


Slither Results Log

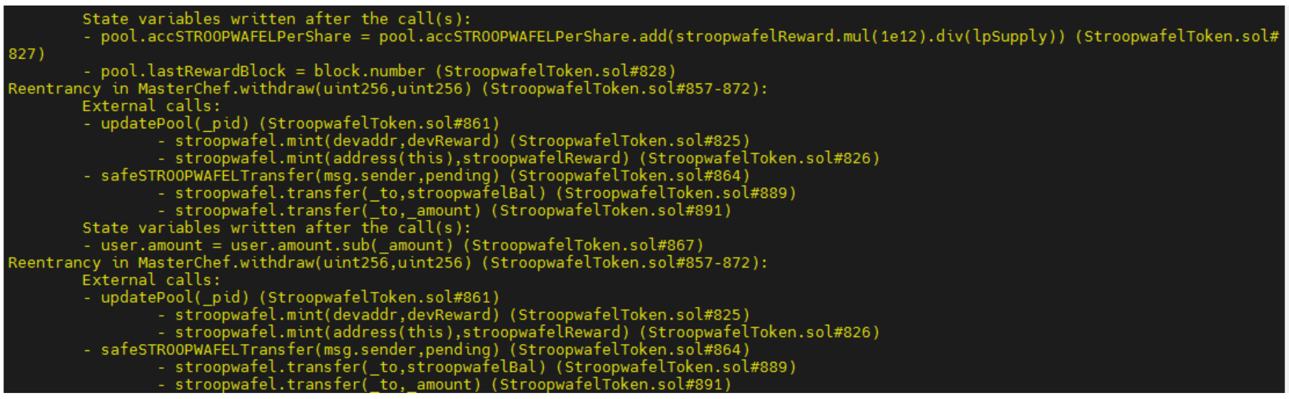
Slither log >> StroopwafelToken.sol



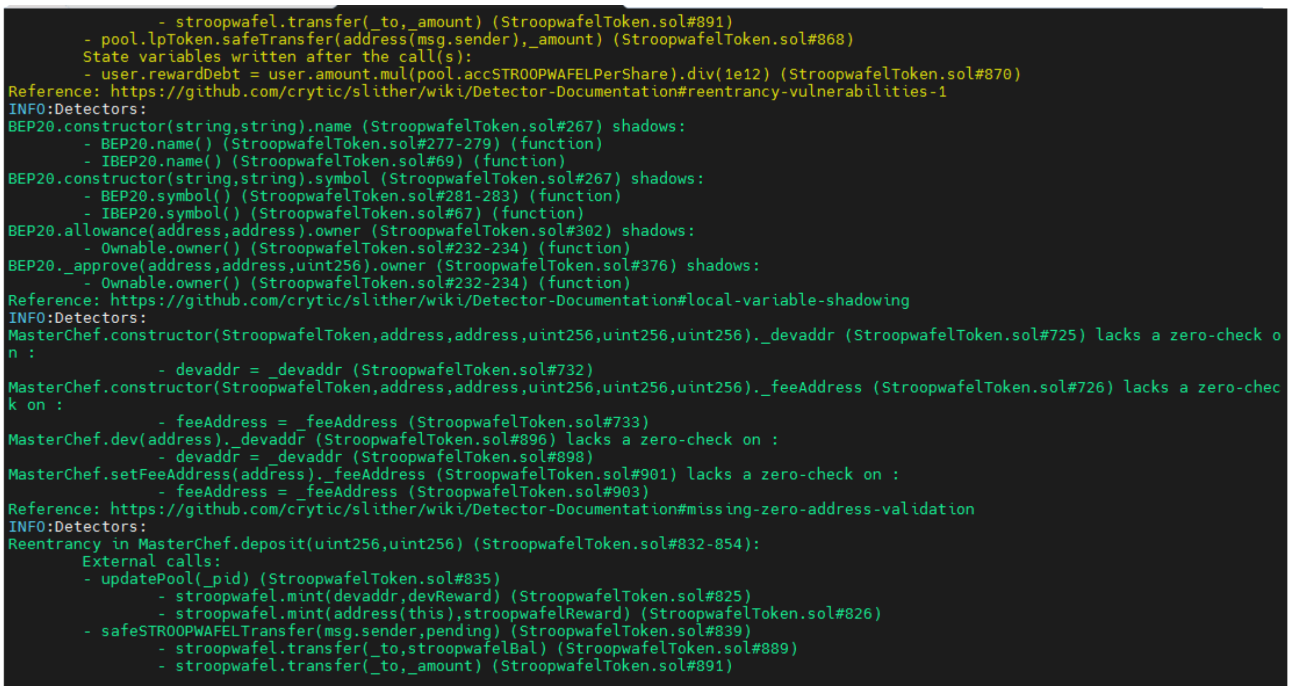
```
INFO:Detectors:
MasterChef.safeSTROOPWAFELTransfer(address,uint256) (StroopwafelToken.sol#886-893) ignores return value by stroopwafel.transfer(_to,stroo
pwafelBal) (StroopwafelToken.sol#889)
MasterChef.safeSTROOPWAFELTransfer(address,uint256) (StroopwafelToken.sol#886-893) ignores return value by stroopwafel.transfer(_to,_amou
nt) (StroopwafelToken.sol#891)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
MasterChef.pendingSTROOPWAFEL(uint256,address) (StroopwafelToken.sol#781-794) performs a multiplication on the result of a division:
        -stroopwafelReward = multiplier.mul(stroopwafelPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (StroopwafelToken.sol#788)
        -accSTROOPWAFELPerShare = accSTROOPWAFELPerShare.add(stroopwafelReward.mul(1e12).div(lpSupply)) (StroopwafelToken.sol#791)
MasterChef.updatePool(uint256) (StroopwafelToken.sol#805-829) performs a multiplication on the result of a division:
        -stroopwafelReward = multiplier.mul(stroopwafelPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (StroopwafelToken.sol#817)
        -pool.accSTROOPWAFELPerShare = pool.accSTROOPWAFELPerShare.add(stroopwafelReward.mul(1e12).div(lpSupply)) (StroopwafelToken.sol#8
27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
StroopwafelToken._writeCheckpoint(address,uint32,uint256,uint256) (StroopwafelToken.sol#599-617) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (StroopwafelToken.sol#609)
MasterChef.updatePool(uint256) (StroopwafelToken.sol#805-829) uses a dangerous strict equality:
        - lpSupply == 0 || pool.allocPoint == 0 || stroopwafelPerBlock == 0 (StroopwafelToken.sol#811)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef.add(uint256,IBEP20,uint16,bool) (StroopwafelToken.sol#745-759):
        External calls:
        - massUpdatePools() (StroopwafelToken.sol#748)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        State variables written after the call(s):
        - poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP)) (StroopwafelToken.sol#752-758)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (StroopwafelToken.sol#751)
Reentrancy in MasterChef.deposit(uint256,uint256) (StroopwafelToken.sol#832-854):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#835)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
```



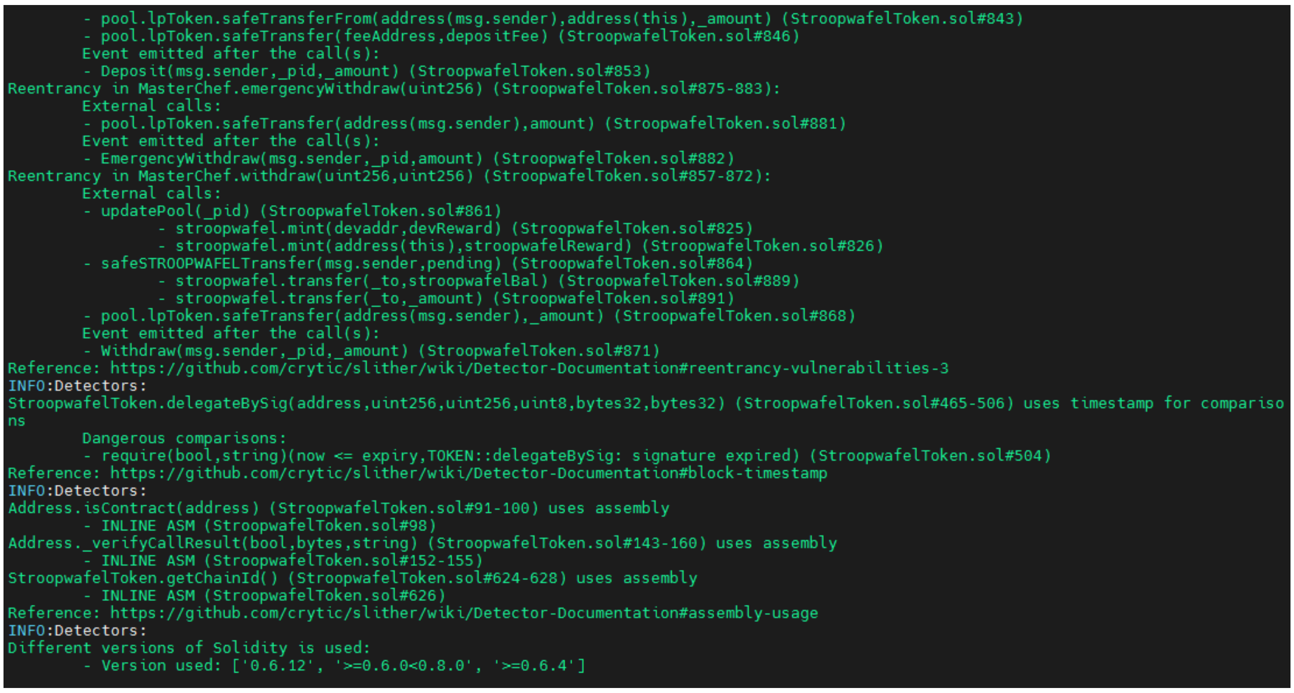
```
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#839)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (StroopwafelToken.sol#843)
        - pool.lpToken.safeTransfer(feeAddress,depositFee) (StroopwafelToken.sol#846)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount).sub(depositFee) (StroopwafelToken.sol#847)
Reentrancy in MasterChef.deposit(uint256,uint256) (StroopwafelToken.sol#832-854):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#835)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#839)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (StroopwafelToken.sol#843)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount) (StroopwafelToken.sol#849)
Reentrancy in MasterChef.set(uint256,uint256,uint16,bool) (StroopwafelToken.sol#762-770):
        External calls:
        - massUpdatePools() (StroopwafelToken.sol#765)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        State variables written after the call(s):
        - poolInfo[_pid].allocPoint = _allocPoint (StroopwafelToken.sol#768)
        - poolInfo[_pid].depositFeeBP = _depositFeeBP (StroopwafelToken.sol#769)
        - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (StroopwafelToken.sol#767)
Reentrancy in MasterChef.updateEmissionRate(uint256) (StroopwafelToken.sol#907-910):
        External calls:
        - massUpdatePools() (StroopwafelToken.sol#908)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        State variables written after the call(s):
        - stroopwafelPerBlock = _stroopwafelPerBlock (StroopwafelToken.sol#909)
Reentrancy in MasterChef.updatePool(uint256) (StroopwafelToken.sol#805-829):
        External calls:
        - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
        - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
```



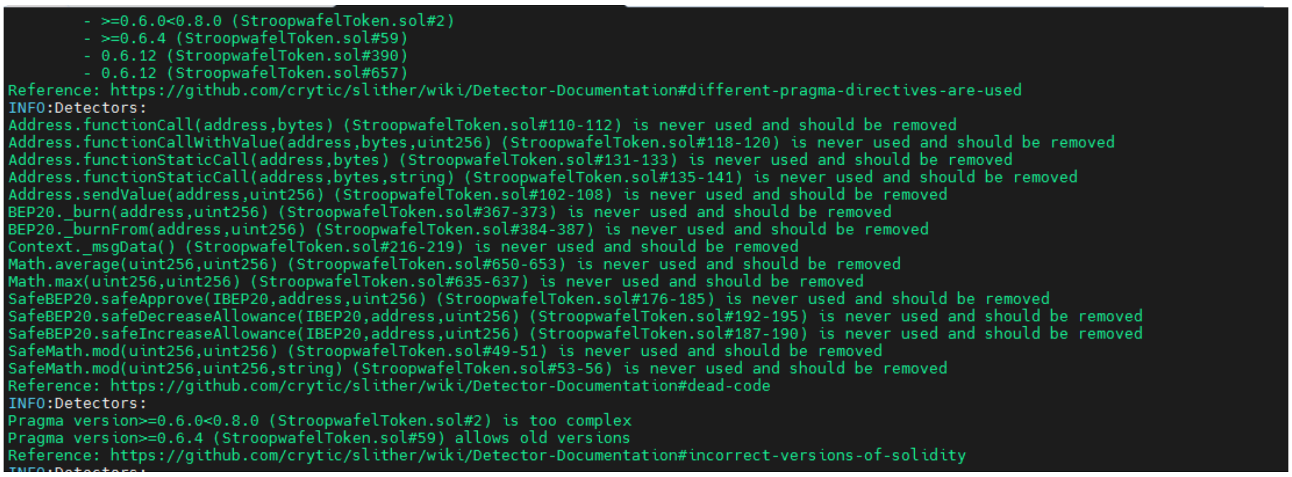
```
        State variables written after the call(s):
        - pool.accSTROOPWAFELPerShare = pool.accSTROOPWAFELPerShare.add(stroopwafelReward.mul(1e12).div(lpSupply)) (StroopwafelToken.sol#
827)
        - pool.lastRewardBlock = block.number (StroopwafelToken.sol#828)
Reentrancy in MasterChef.withdraw(uint256,uint256) (StroopwafelToken.sol#857-872):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#861)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#864)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
        State variables written after the call(s):
        - user.amount = user.amount.sub(_amount) (StroopwafelToken.sol#867)
Reentrancy in MasterChef.withdraw(uint256,uint256) (StroopwafelToken.sol#857-872):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#861)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#864)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
```



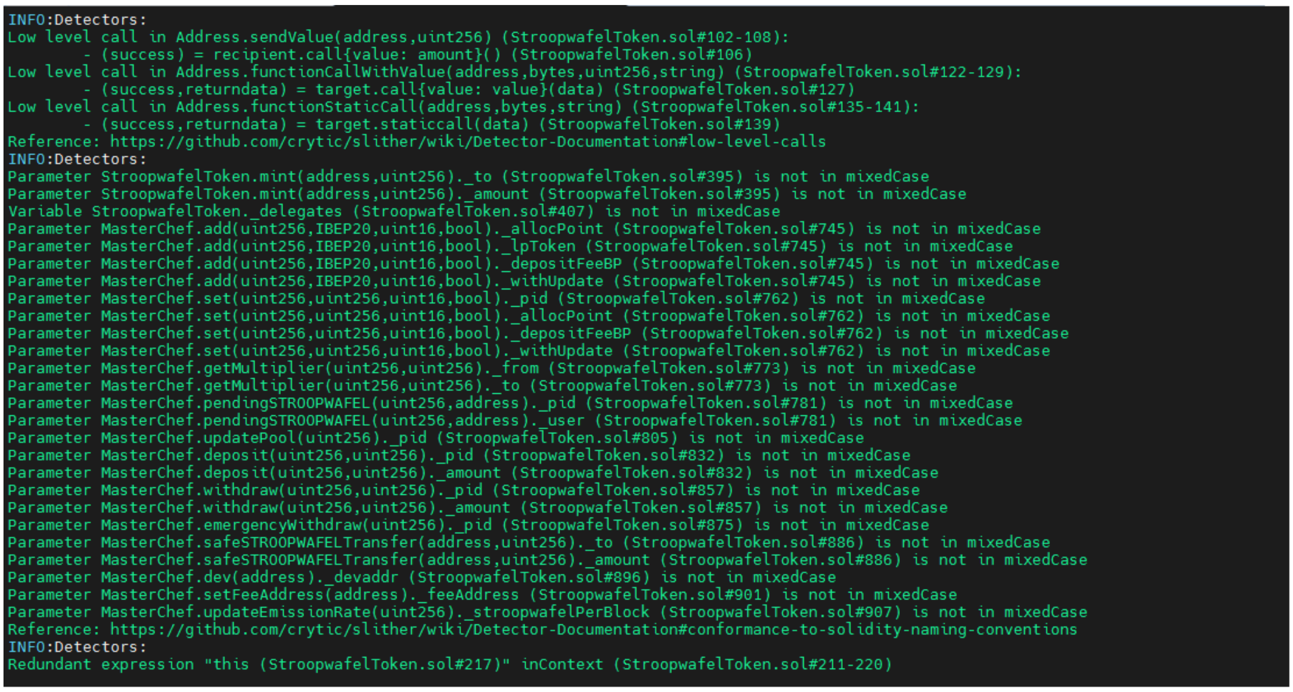
```
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (StroopwafelToken.sol#868)
        State variables written after the call(s):
        - user.rewardDebt = user.amount.mul(pool.accSTROOPWAFELPerShare).div(1e12) (StroopwafelToken.sol#870)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
BEP20.constructor(string,string).name (StroopwafelToken.sol#267) shadows:
        - BEP20.name() (StroopwafelToken.sol#277-279) (function)
        - IBEP20.name() (StroopwafelToken.sol#69) (function)
BEP20.constructor(string,string).symbol (StroopwafelToken.sol#267) shadows:
        - BEP20.symbol() (StroopwafelToken.sol#281-283) (function)
        - IBEP20.symbol() (StroopwafelToken.sol#67) (function)
BEP20.allowance(address,address).owner (StroopwafelToken.sol#302) shadows:
        - Ownable.owner() (StroopwafelToken.sol#232-234) (function)
BEP20._approve(address,address,uint256).owner (StroopwafelToken.sol#376) shadows:
        - Ownable.owner() (StroopwafelToken.sol#232-234) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MasterChef.constructor(StroopwafelToken,address,address,uint256,uint256,uint256)._devaddr (StroopwafelToken.sol#725) lacks a zero-check o
n :
                - devaddr = _devaddr (StroopwafelToken.sol#732)
MasterChef.constructor(StroopwafelToken,address,address,uint256,uint256,uint256)._feeAddress (StroopwafelToken.sol#726) lacks a zero-chec
k on :
                - feeAddress = _feeAddress (StroopwafelToken.sol#733)
MasterChef.dev(address)._devaddr (StroopwafelToken.sol#896) lacks a zero-check on :
                - devaddr = _devaddr (StroopwafelToken.sol#898)
MasterChef.setFeeAddress(address)._feeAddress (StroopwafelToken.sol#901) lacks a zero-check on :
                - feeAddress = _feeAddress (StroopwafelToken.sol#903)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256) (StroopwafelToken.sol#832-854):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#835)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#839)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
```



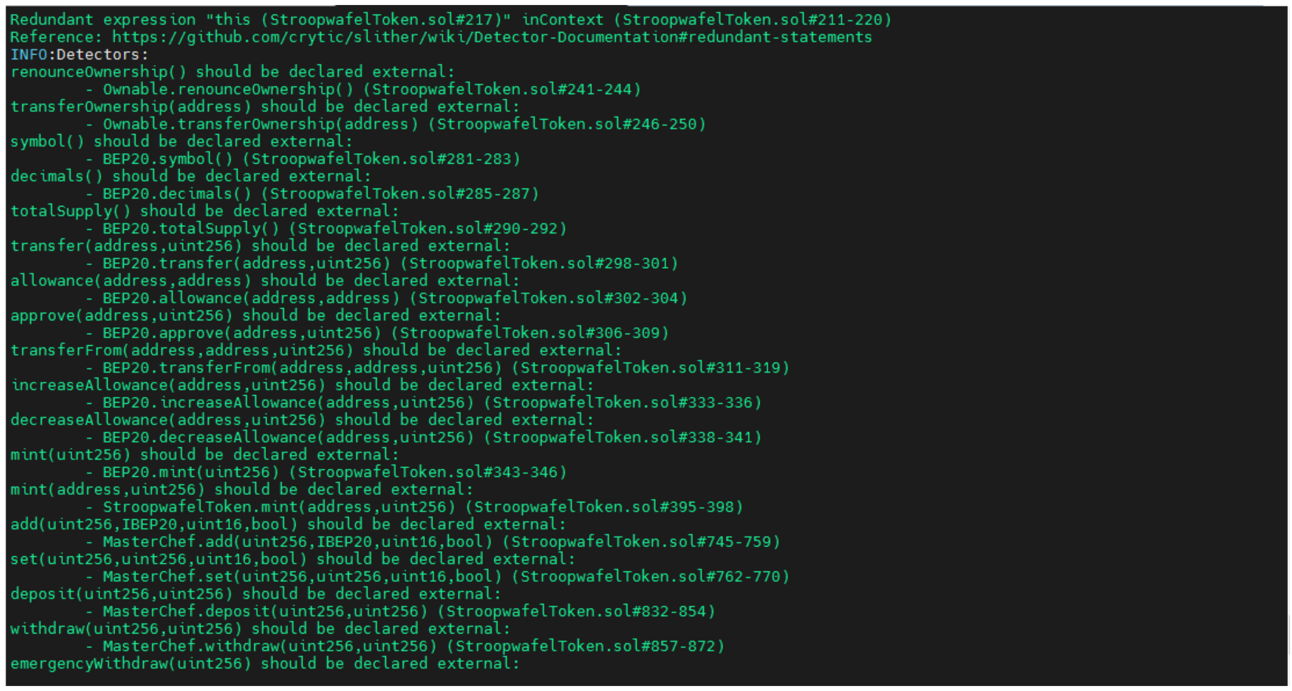
```
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (StroopwafelToken.sol#843)
        - pool.lpToken.safeTransfer(feeAddress,depositFee) (StroopwafelToken.sol#846)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (StroopwafelToken.sol#853)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (StroopwafelToken.sol#875-883):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),amount) (StroopwafelToken.sol#881)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,amount) (StroopwafelToken.sol#882)
Reentrancy in MasterChef.withdraw(uint256,uint256) (StroopwafelToken.sol#857-872):
        External calls:
        - updatePool(_pid) (StroopwafelToken.sol#861)
                - stroopwafel.mint(devaddr,devReward) (StroopwafelToken.sol#825)
                - stroopwafel.mint(address(this),stroopwafelReward) (StroopwafelToken.sol#826)
        - safeSTROOPWAFELTransfer(msg.sender,pending) (StroopwafelToken.sol#864)
                - stroopwafel.transfer(_to,stroopwafelBal) (StroopwafelToken.sol#889)
                - stroopwafel.transfer(_to,_amount) (StroopwafelToken.sol#891)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (StroopwafelToken.sol#868)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (StroopwafelToken.sol#871)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StroopwafelToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (StroopwafelToken.sol#465-506) uses timestamp for compariso
ns
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,TOKEN::delegateBySig: signature expired) (StroopwafelToken.sol#504)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (StroopwafelToken.sol#91-100) uses assembly
        - INLINE ASM (StroopwafelToken.sol#98)
Address._verifyCallResult(bool,bytes,string) (StroopwafelToken.sol#143-160) uses assembly
        - INLINE ASM (StroopwafelToken.sol#152-155)
StroopwafelToken.getChainId() (StroopwafelToken.sol#624-628) uses assembly
        - INLINE ASM (StroopwafelToken.sol#626)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.6.0<0.8.0', '>=0.6.4']
```



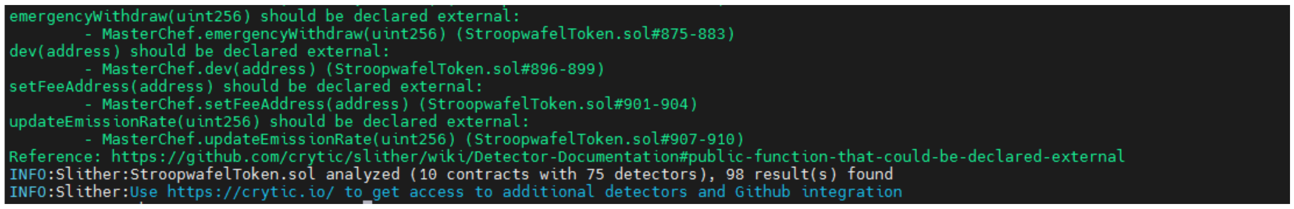
```
        - >=0.6.0<0.8.0 (StroopwafelToken.sol#2)
        - >=0.6.4 (StroopwafelToken.sol#59)
        - 0.6.12 (StroopwafelToken.sol#390)
        - 0.6.12 (StroopwafelToken.sol#657)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (StroopwafelToken.sol#110-112) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (StroopwafelToken.sol#118-120) is never used and should be removed
Address.functionStaticCall(address,bytes) (StroopwafelToken.sol#131-133) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (StroopwafelToken.sol#135-141) is never used and should be removed
Address.sendValue(address,uint256) (StroopwafelToken.sol#102-108) is never used and should be removed
BEP20._burn(address,uint256) (StroopwafelToken.sol#367-373) is never used and should be removed
BEP20._burnFrom(address,uint256) (StroopwafelToken.sol#384-387) is never used and should be removed
Context._msgData() (StroopwafelToken.sol#216-219) is never used and should be removed
Math.average(uint256,uint256) (StroopwafelToken.sol#650-653) is never used and should be removed
Math.max(uint256,uint256) (StroopwafelToken.sol#635-637) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (StroopwafelToken.sol#176-185) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (StroopwafelToken.sol#192-195) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (StroopwafelToken.sol#187-190) is never used and should be removed
SafeMath.mod(uint256,uint256) (StroopwafelToken.sol#49-51) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (StroopwafelToken.sol#53-56) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.6.0<0.8.0 (StroopwafelToken.sol#2) is too complex
Pragma version>=0.6.4 (StroopwafelToken.sol#59) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```



```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (StroopwafelToken.sol#102-108):
        - (success) = recipient.call{value: amount}() (StroopwafelToken.sol#106)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (StroopwafelToken.sol#122-129):
        - (success,returndata) = target.call{value: value}(data) (StroopwafelToken.sol#127)
Low level call in Address.functionStaticCall(address,bytes,string) (StroopwafelToken.sol#135-141):
        - (success,returndata) = target.staticcall(data) (StroopwafelToken.sol#139)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter StroopwafelToken.mint(address,uint256)._to (StroopwafelToken.sol#395) is not in mixedCase
Parameter StroopwafelToken.mint(address,uint256)._amount (StroopwafelToken.sol#395) is not in mixedCase
Variable StroopwafelToken._delegates (StroopwafelToken.sol#407) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._allocPoint (StroopwafelToken.sol#745) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._lpToken (StroopwafelToken.sol#745) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._depositFeeBP (StroopwafelToken.sol#745) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._withUpdate (StroopwafelToken.sol#745) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._pid (StroopwafelToken.sol#762) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._allocPoint (StroopwafelToken.sol#762) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._depositFeeBP (StroopwafelToken.sol#762) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._withUpdate (StroopwafelToken.sol#762) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (StroopwafelToken.sol#773) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (StroopwafelToken.sol#773) is not in mixedCase
Parameter MasterChef.pendingSTROOPWAFEL(uint256,address)._pid (StroopwafelToken.sol#781) is not in mixedCase
Parameter MasterChef.pendingSTROOPWAFEL(uint256,address)._user (StroopwafelToken.sol#781) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (StroopwafelToken.sol#805) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._pid (StroopwafelToken.sol#832) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._amount (StroopwafelToken.sol#832) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (StroopwafelToken.sol#857) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (StroopwafelToken.sol#857) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (StroopwafelToken.sol#875) is not in mixedCase
Parameter MasterChef.safeSTROOPWAFELTransfer(address,uint256)._to (StroopwafelToken.sol#886) is not in mixedCase
Parameter MasterChef.safeSTROOPWAFELTransfer(address,uint256)._amount (StroopwafelToken.sol#886) is not in mixedCase
Parameter MasterChef.dev(address)._devaddr (StroopwafelToken.sol#896) is not in mixedCase
Parameter MasterChef.setFeeAddress(address)._feeAddress (StroopwafelToken.sol#901) is not in mixedCase
Parameter MasterChef.updateEmissionRate(uint256)._stroopwafelPerBlock (StroopwafelToken.sol#907) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (StroopwafelToken.sol#217)" inContext (StroopwafelToken.sol#211-220)
```



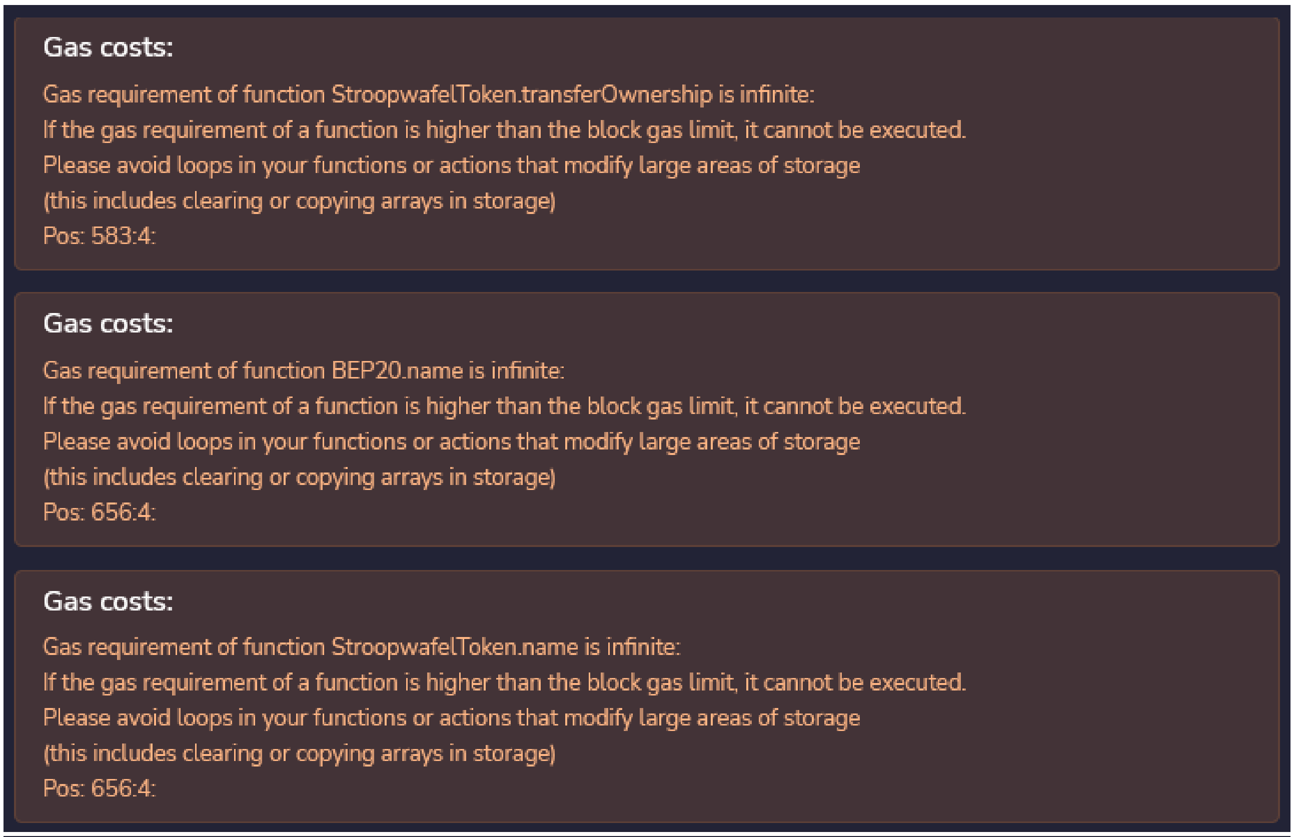
```
Redundant expression "this (StroopwafelToken.sol#217)" inContext (StroopwafelToken.sol#211-220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (StroopwafelToken.sol#241-244)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (StroopwafelToken.sol#246-250)
symbol() should be declared external:
        - BEP20.symbol() (StroopwafelToken.sol#281-283)
decimals() should be declared external:
        - BEP20.decimals() (StroopwafelToken.sol#285-287)
totalSupply() should be declared external:
        - BEP20.totalSupply() (StroopwafelToken.sol#290-292)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (StroopwafelToken.sol#298-301)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (StroopwafelToken.sol#302-304)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (StroopwafelToken.sol#306-309)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (StroopwafelToken.sol#311-319)
increaseAllowance(address,uint256) should be declared external:
        - BEP20.increaseAllowance(address,uint256) (StroopwafelToken.sol#333-336)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20.decreaseAllowance(address,uint256) (StroopwafelToken.sol#338-341)
mint(uint256) should be declared external:
        - BEP20.mint(uint256) (StroopwafelToken.sol#343-346)
mint(address,uint256) should be declared external:
        - StroopwafelToken.mint(address,uint256) (StroopwafelToken.sol#395-398)
add(uint256,IBEP20,uint16,bool) should be declared external:
        - MasterChef.add(uint256,IBEP20,uint16,bool) (StroopwafelToken.sol#745-759)
set(uint256,uint256,uint16,bool) should be declared external:
        - MasterChef.set(uint256,uint256,uint16,bool) (StroopwafelToken.sol#762-770)
deposit(uint256,uint256) should be declared external:
        - MasterChef.deposit(uint256,uint256) (StroopwafelToken.sol#832-854)
withdraw(uint256,uint256) should be declared external:
        - MasterChef.withdraw(uint256,uint256) (StroopwafelToken.sol#857-872)
emergencyWithdraw(uint256) should be declared external:
```



```
emergencyWithdraw(uint256) should be declared external:
        - MasterChef.emergencyWithdraw(uint256) (StroopwafelToken.sol#875-883)
dev(address) should be declared external:
        - MasterChef.dev(address) (StroopwafelToken.sol#896-899)
setFeeAddress(address) should be declared external:
        - MasterChef.setFeeAddress(address) (StroopwafelToken.sol#901-904)
updateEmissionRate(uint256) should be declared external:
        - MasterChef.updateEmissionRate(uint256) (StroopwafelToken.sol#907-910)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:StroopwafelToken.sol analyzed (10 contracts with 75 detectors), 98 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

StroopwafelToken.sol



Gas costs:

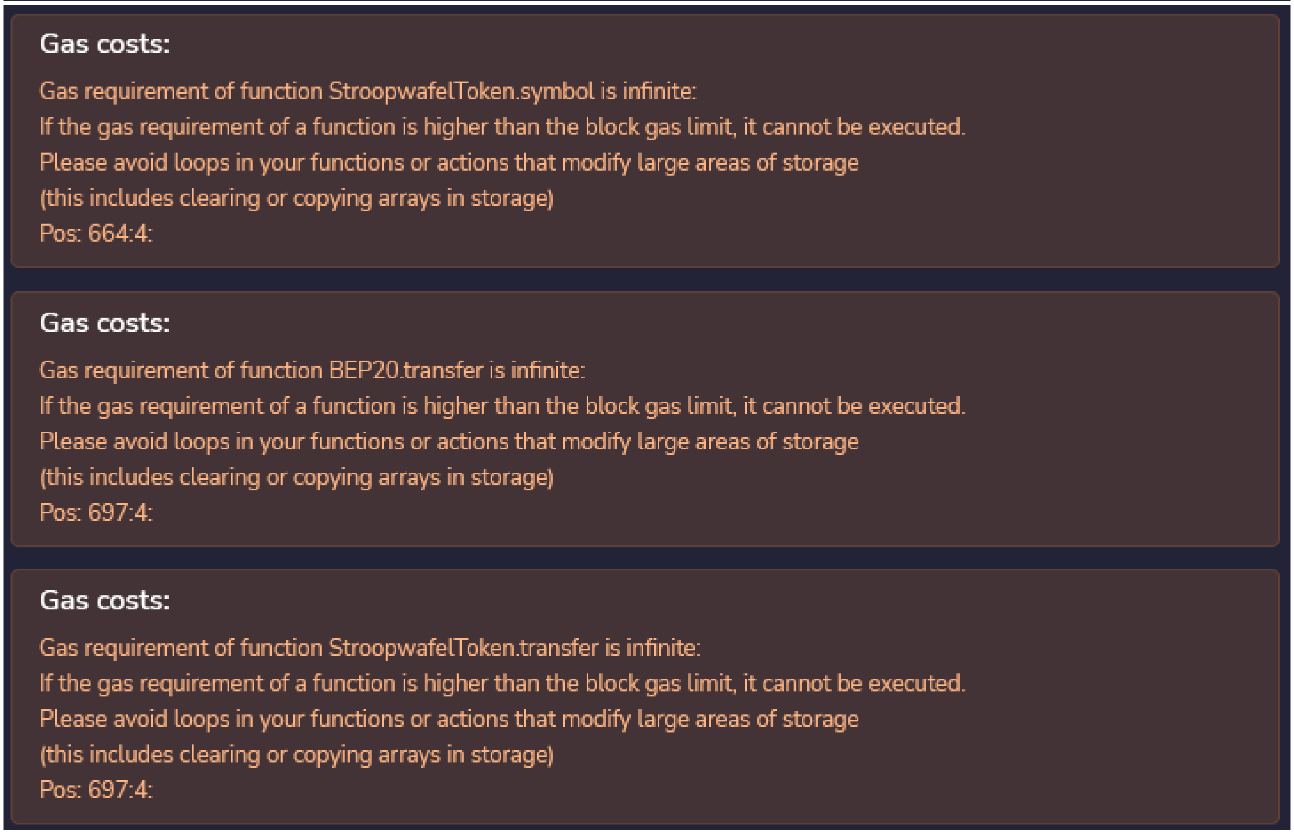
Gas requirement of function StroopwafelToken.transferOwnership is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 583:4:

Gas costs:

Gas requirement of function BEP20.name 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: 656:4:

Gas costs:

Gas requirement of function StroopwafelToken.name 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: 656:4:



Gas costs:

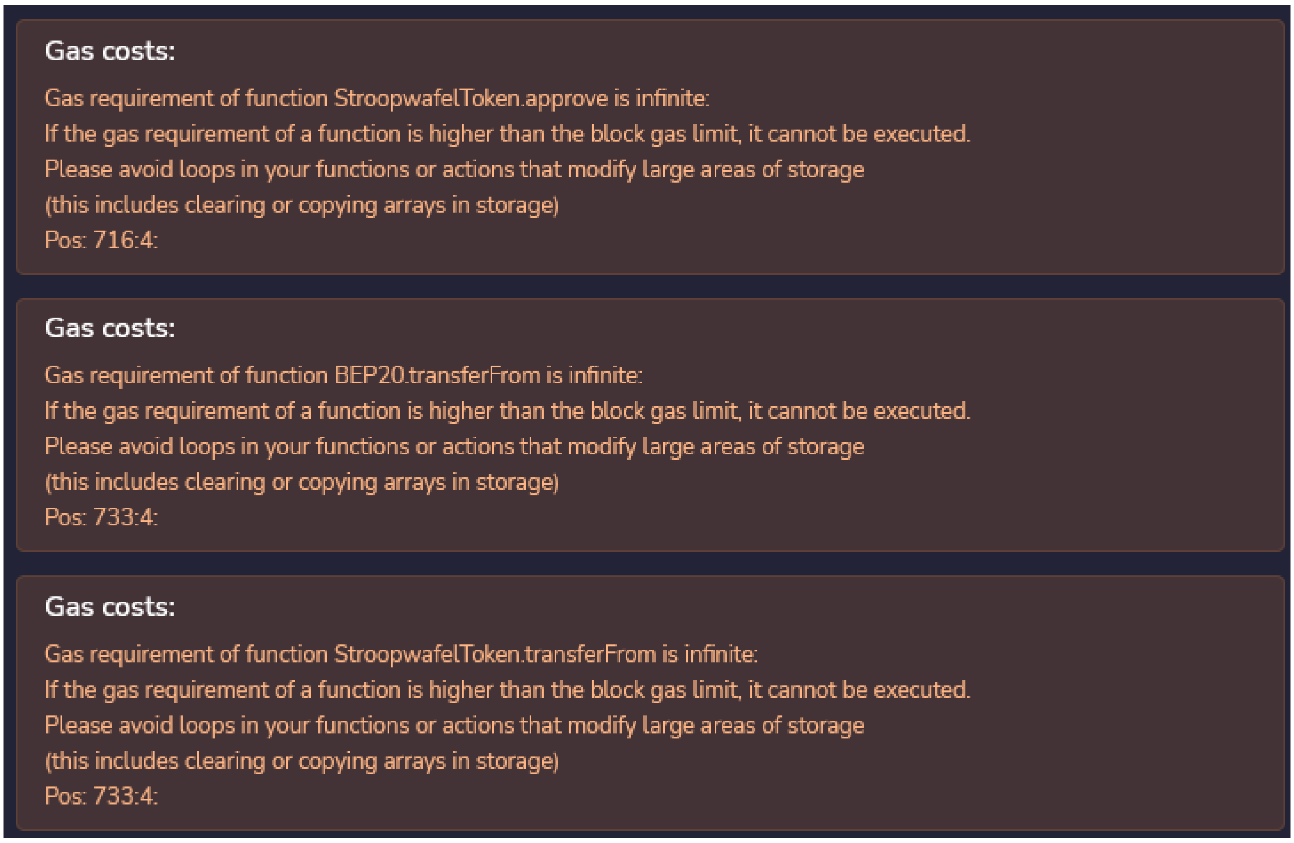
Gas requirement of function StroopwafelToken.symbol is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 664:4:

Gas costs:

Gas requirement of function BEP20.transfer 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: 697:4:

Gas costs:

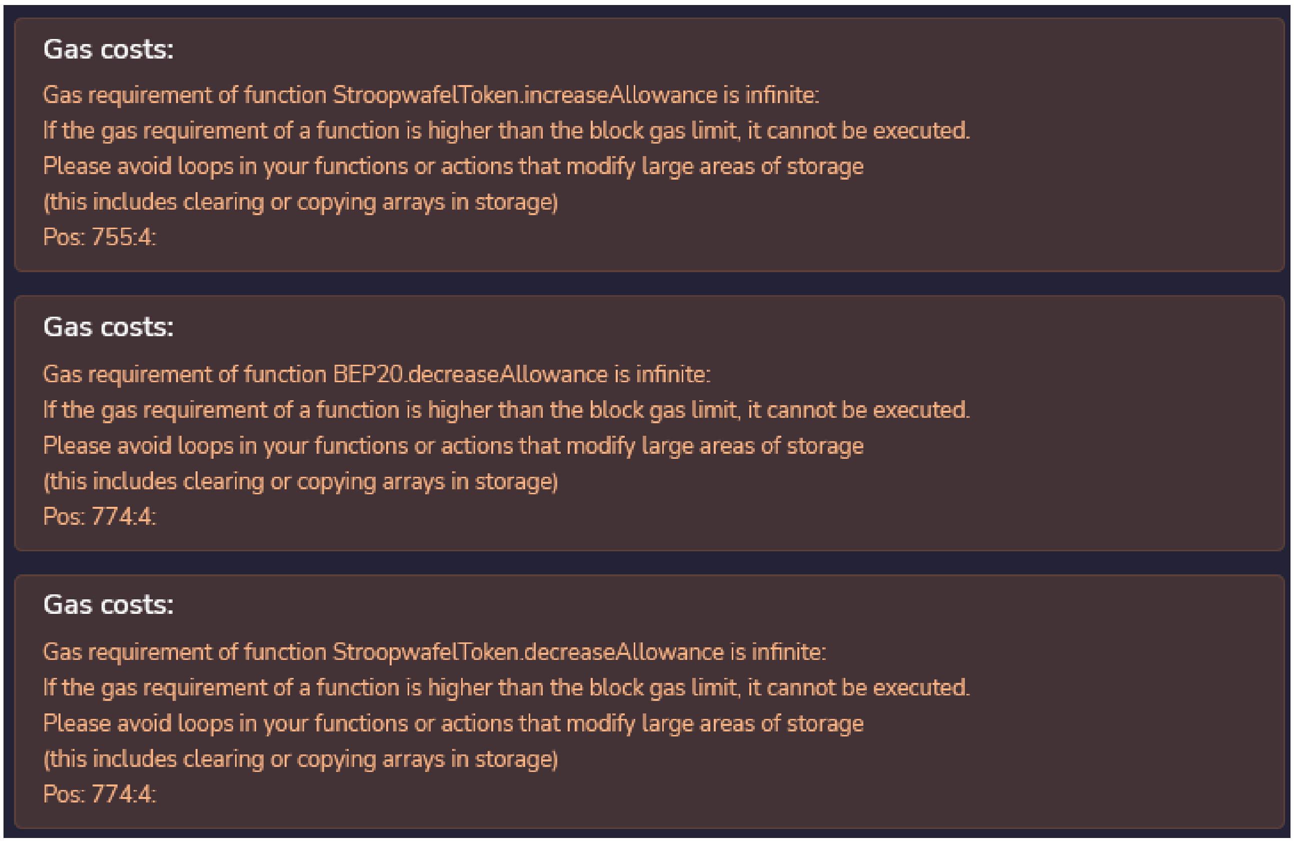
Gas requirement of function StroopwafelToken.transfer is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 697:4:



```
Gas costs:
Gas requirement of function StroopwafelToken.approve 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: 716:4:
```

```
Gas costs:
Gas requirement of function BEP20.transferFrom 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: 733:4:
```

```
Gas costs:
Gas requirement of function StroopwafelToken.transferFrom 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: 733:4:
```



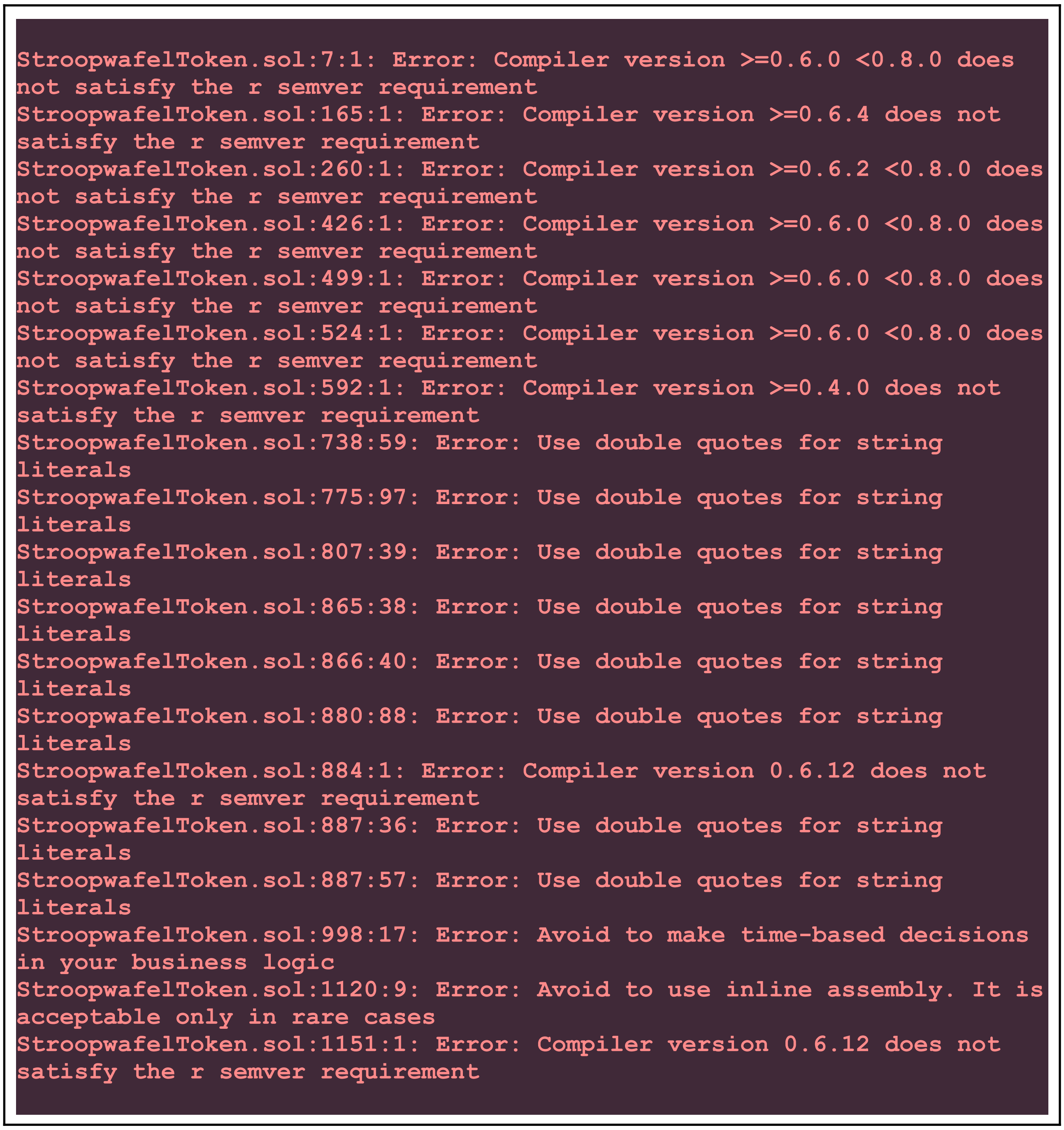
```
Gas costs:
Gas requirement of function StroopwafelToken.increaseAllowance is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)
Pos: 755:4:
```

```
Gas costs:
Gas requirement of function BEP20.decreaseAllowance 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: 774:4:
```

```
Gas costs:
Gas requirement of function StroopwafelToken.decreaseAllowance 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: 774:4:
```

Solhint Linter

StroopwafelToken.sol



```
StroopwafelToken.sol:7:1: Error: Compiler version >=0.6.0 <0.8.0 does
not satisfy the r semver requirement
StroopwafelToken.sol:165:1: Error: Compiler version >=0.6.4 does not
satisfy the r semver requirement
StroopwafelToken.sol:260:1: Error: Compiler version >=0.6.2 <0.8.0 does
not satisfy the r semver requirement
StroopwafelToken.sol:426:1: Error: Compiler version >=0.6.0 <0.8.0 does
not satisfy the r semver requirement
StroopwafelToken.sol:499:1: Error: Compiler version >=0.6.0 <0.8.0 does
not satisfy the r semver requirement
StroopwafelToken.sol:524:1: Error: Compiler version >=0.6.0 <0.8.0 does
not satisfy the r semver requirement
StroopwafelToken.sol:592:1: Error: Compiler version >=0.4.0 does not
satisfy the r semver requirement
StroopwafelToken.sol:738:59: Error: Use double quotes for string
literals
StroopwafelToken.sol:775:97: Error: Use double quotes for string
literals
StroopwafelToken.sol:807:39: Error: Use double quotes for string
literals
StroopwafelToken.sol:865:38: Error: Use double quotes for string
literals
StroopwafelToken.sol:866:40: Error: Use double quotes for string
literals
StroopwafelToken.sol:880:88: Error: Use double quotes for string
literals
StroopwafelToken.sol:884:1: Error: Compiler version 0.6.12 does not
satisfy the r semver requirement
StroopwafelToken.sol:887:36: Error: Use double quotes for string
literals
StroopwafelToken.sol:887:57: Error: Use double quotes for string
literals
StroopwafelToken.sol:998:17: Error: Avoid to make time-based decisions
in your business logic
StroopwafelToken.sol:1120:9: Error: Avoid to use inline assembly. It is
acceptable only in rare cases
StroopwafelToken.sol:1151:1: Error: Compiler version 0.6.12 does not
satisfy the r semver requirement
```

Software analysis result:

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

EtherAuthority