Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	WIS Team
Website:	https://wis.com
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 3rd, 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

Project files

Name	Code Review and Security Analysis Report for WIS Protocol Smart Contracts
Platform	**BSC / Solidity**
File 1	MasterShef.sol
File 1 MD5 Hash	85D1761B9896AF269EB89F3105D9A8DE
File 1 Online Code	https://bscscan.com/address/0x4553e8629e307bb145c240435af7582f129b939b#code
File 2	SyrupBar.sol
File 2 MD5 Hash	EB1741F15C2ECEB13DF1BF73FF44FD16
File 2 Online Code	https://bscscan.com/address/0x1d03726335f5aa1f26039a7714384cde8176b45b#code
File 3	WisToken.sol
File 3 MD5 Hash	C36413E927F3CE7333E3D0949BCDF860
File 3 Online Code	https://bscscan.com/address/0x64e9ec7757184e598b304446cf415deed7b690f2#code
Audit Date	June 3rd, 2021

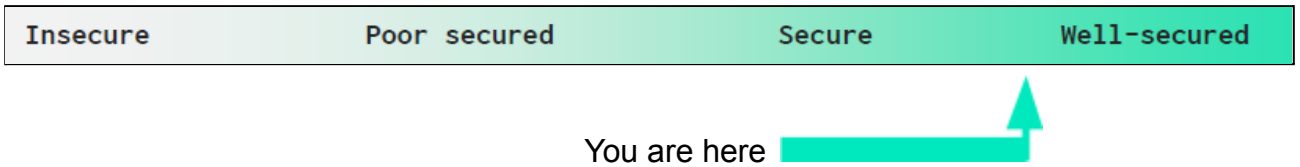
Introduction

We were contracted by the WIS team to perform the Security audit of the WIS Protocol 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 3rd, 2021.

The Audit type was Standard Audit. Which means this audit is concluded based on Standard audit scope. This document outlines all the findings as well as an AS-IS overview of the smart contract codes.

Executive Summary

According to the **standard** audit assessment, Customer`s solidity smart contracts are **Well secured**. These contracts also have 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. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 1 high, 0 medium, 2 low and some very low level issues in smart contracts which are described in the audit finding section. We also revised the code and these issues are fixed / acknowledged by the WIS team.

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	Passed
	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
	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	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 3 smart contracts. These smart contracts also contain Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in the WIS protocol 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 WIS protocol.

The WIS 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 WIS smart contracts code in the form of the files. 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://wis.com 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

WIS Protocol is a group of smart contracts, having functionality like swapping, minting, distributing tokens, etc. Following are the main components of the core smart contract.

MasterChef.sol

(1) Interface

(a) IBEP20

(b) IMigratorChef

(2) Inherited contracts

(a) Context

(b) Ownable

(3) Struct

(a) UserInfo: Information about UserInfo.

(b) PoolInfo: Information about PoolInfo.

(4) Usages

(a) using SafeMath for uint256;

(b) using SafeBEP20 for IBEP20;

(5) Events

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

(b) event Deposit(address indexed user, uint256 indexed pid, uint256 amount);

(c) event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);

(d) event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256amount);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	updateMultiplier	write	access only Owner	No Issue
2	poolLength	read	Passed	No Issue
3	addToWhitelist	write	access only Owner	No Issue
4	add	write	Input validation missing	LP Token must not be added twice
5	set	write	access only Owner	No Issue
6	updateStakingPool	write	Infinite loop possibility	Array length must be limited
7	getMultiplier	read	Passed	No Issue
8	pendingCake	read	Passed	No Issue
9	massUpdatePools	write	Infinite loop possibility	Array length must be limited
10	updatePool	write	Passed	No Issue
11	deposit	write	Passed	No Issue
12	withdraw	write	Passed	No Issue
13	enterStaking	write	Passed	No Issue
14	leaveStaking	write	Passed	No Issue
15	emergencyWithdraw	write	Passed	No Issue
16	safeCakeTransfer	write	Passed	No Issue
17	dev	write	Passed	No Issue

SyrupBar.sol

(1) Interface

(a) IBEP20

(2) Inherited contracts

(a) Ownable

(b) Context

(c) BEP20

(d) IBEP20

(e) WisToken

(3) Struct

(a) Checkpoint: Information about Checkpoint.

(4) Usages

(a) using SafeMath for uint256;

(b) using Address for address;

(c) using SafeBEP20 for IBEP20;

(5) Events

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

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

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

(d) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

(e) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

(f) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

(g) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	Owner is the MasterChef	No Issue
2	burn	write	Owner is the MasterChef	No Issue
3	safeCakeTransfer	write	Owner is the MasterChef	No Issue
4	delegates	external	Passed	No Issue
5	delegate	external	Passed	No Issue
6	delegateBySig	external	Handle signatures carefully	No Issue
7	getCurrentVotes	external	Passed	No Issue
8	getPriorVotes	external	Infinite loop possibility	Keep array length limited
9	_delegate	write	Passed	No Issue
10	_moveDelegates	write	Passed	No Issue
11	_writeCheckpoint	write	Passed	No Issue
12	safe32	write	Passed	No Issue
13	getChainId	write	Passed	No Issue
14	owner	read	Passed	No Issue
15	renounceOwnership	write	Owner is the MasterChef	No Issue
16	transferOwnership	write	Owner is the MasterChef	No Issue
17	_transferOwnership	write	Passed	No Issue
18	_msgSender	read	Passed	No Issue
19	_msgData	read	Passed	No Issue

WisToken.sol

(1) Interface

(a) IBEP20

(2) Inherited contracts

(a) Ownable

(b) BEP20

(c) IBEP20

(d) Context

(3) Struct

(a) Checkpoint: Information about Checkpoint.

(4) Usages

(a) using SafeMath for uint256;

(b) using Address for address;

(5) Events

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

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

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

(d) event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed toDelegate);

(e) event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	Owner is the MasterChef	No Issue
2	delegates	read	Passed	No Issue
3	delegate	external	Passed	No Issue
4	delegateBySig	external	Handle signatures carefully	No Issue
5	getCurrentVotes	external	Passed	No Issue
6	getPriorVotes	external	Infinite loop possibility	Keep array length limited
7	_delegate	write	Passed	No Issue
8	_moveDelegates	write	Passed	No Issue
9	_writeCheckpoint	write	Passed	No Issue
10	safe32	write	Passed	No Issue
11	getChainId	write	Passed	No Issue
12	getOwner	external	Passed	No Issue
13	name	read	Passed	No Issue
14	decimals	read	Passed	No Issue
15	symbol	read	Passed	No Issue
16	totalSupply	read	Passed	No Issue
17	balanceOf	read	Passed	No Issue
18	transfer	write	Passed	No Issue
19	allowance	read	Passed	No Issue
20	approve	write	Passed	No Issue
21	transferFrom	write	Passed	No Issue
22	increaseAllowance	write	Passed	No Issue
23	decreaseAllowance	write	Passed	No Issue
25	_transfer	internal	Passed	No Issue
26	_mint	internal	Passed	No Issue
27	_burn	internal	Passed	No Issue
28	_approve	internal	Passed	No Issue
29	_burnFrom	internal	Passed	No Issue
30	owner	read	Passed	No Issue
31	renounceOwnership	write	Owner is the MasterChef	No Issue
32	transferOwnership	write	Owner is the MasterChef	No Issue
33	_transferOwnership	internal	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial 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

(1) Flash Loan attack possibility:

An attacker can borrow a large sum from Flash Loan providers and abuse the smart contract in an unexpected way.

Resolution: This issue is fixed using antiHack logic, which prevents all the smart contract calls to critical functions except whitelisted smart contracts.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Input validation missing in MasterChef.sol

```
// 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, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accCakePerShare: 0
    }));
    updateStakingPool();
```

As mentioned in the comment, the token must never be added twice. So, there must be a condition to prevent that happening by mistake.

Resolution: we got confirmation from the WIS team that the extra care will be taken as this is the owner function.

(2) Infinite loops possibility at multiple places:

```
// 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);
    }
}
```

As seen in the AS-IS section, there are several places in the smart contracts, where array.length is used directly in the loops. It is recommended to put some kind of limits, so it does not go wild and create any scenario where it can hit the block gas limit.

Resolution: We got confirmation from the WIS team that the array will be provided as a limited length. And this will be taken care of from the client side.

Very Low / Discussion / Best practices:

(1) Solidity version

```
pragma solidity 0.6.12;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.
Resolution: This issue is acknowledged.

(2) Event log must be fired in place where the stats are being changed. for example:

- updateMultiplier function in MasterChef.sol
- add function in MasterChef.sol
- set function in MasterChef.sol
- dev function in MasterChef.sol

Resolution: This issue is acknowledged.

(3) Consider specifying function visibility to “external” instead of “public”, if that function is not being called internally. It will save some gas as well.
https://ethereum.stackexchange.com/questions/32353/what-is-the-difference-between-an-internal-external-and-public-private-function/32464

Centralization

These smart contracts have 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:

- renounceOwnership: Owner can give up ownership completely.
- updateMultiplier: can be accessed by the MasterChef Owner.
- add: can be accessed by the MasterChef Owner.
- set: can be accessed by the MasterChef Owner.

Conclusion

We were given contract codes. And we have used all possible tests based on giving 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 "**Well Secured**".

Email: audit@EtherAuthority.io

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).

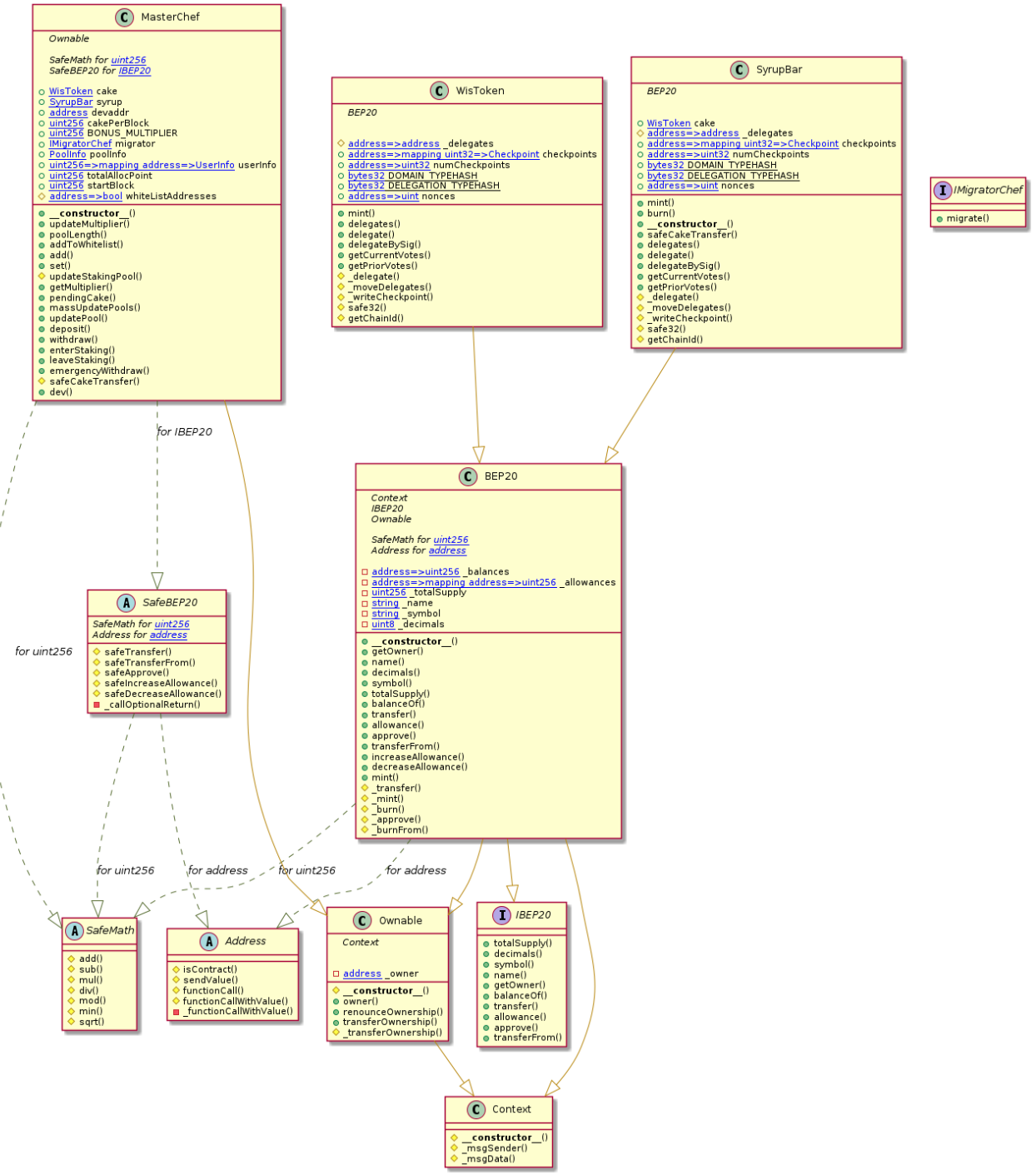
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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

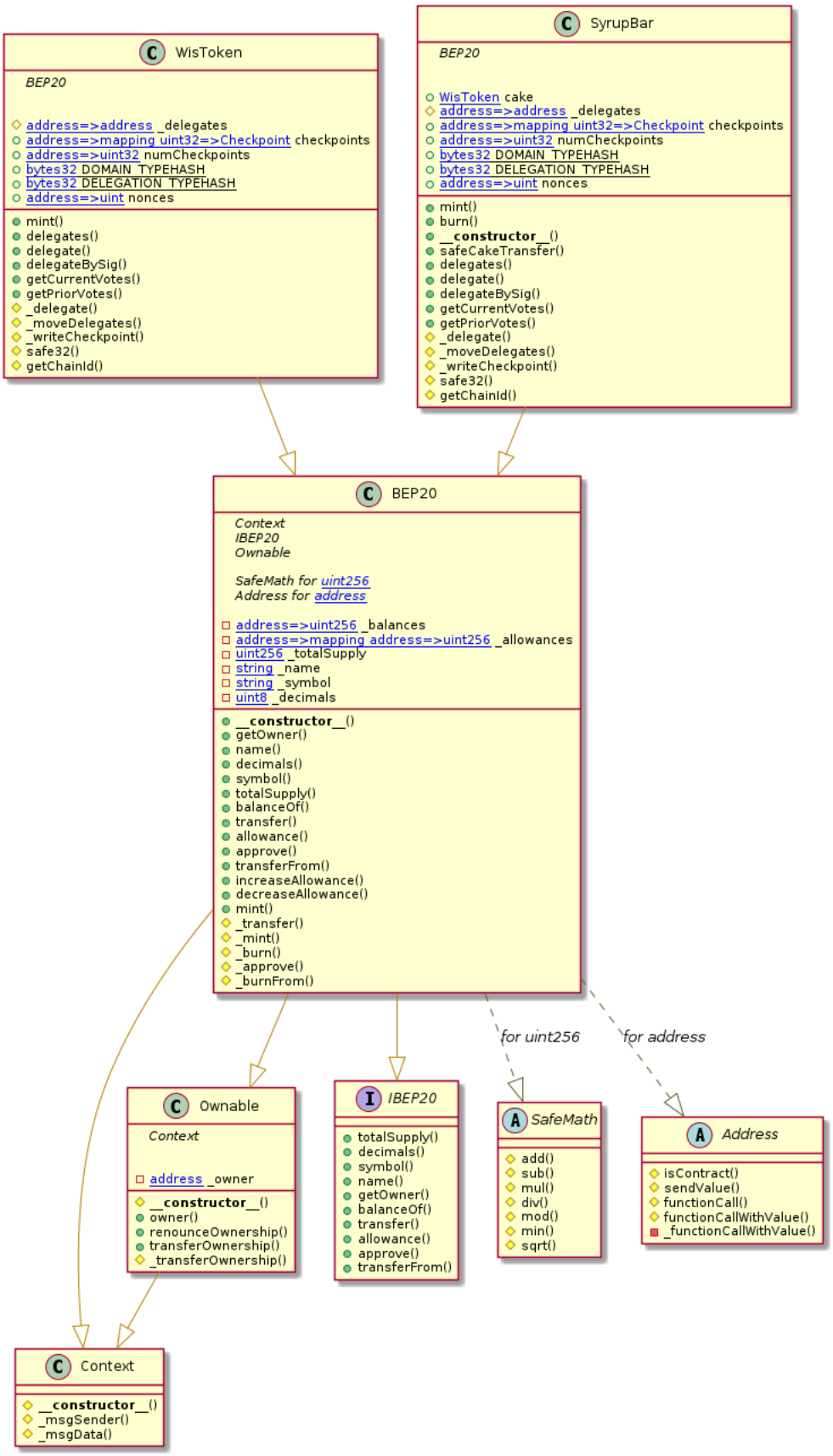
Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

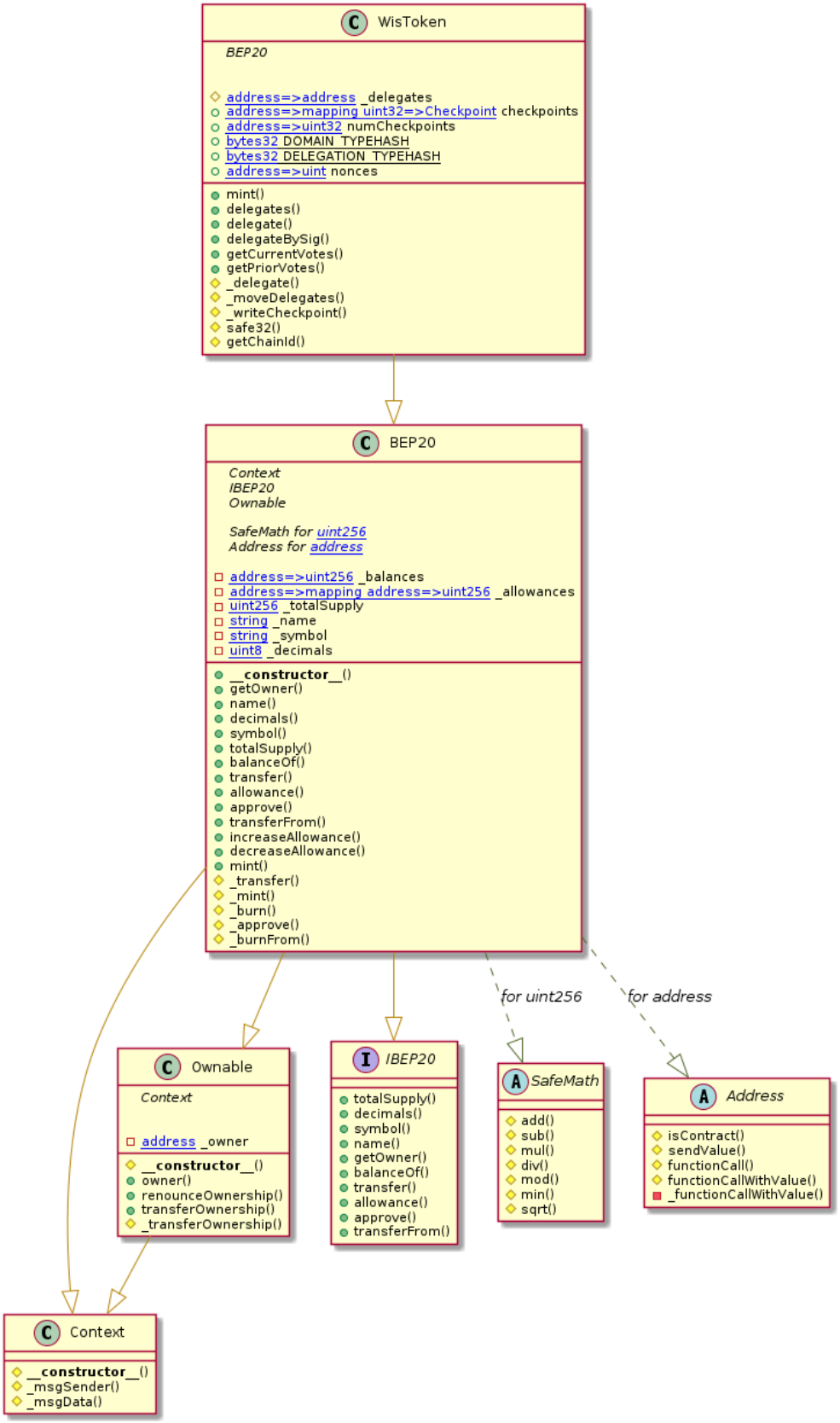
Code Flow Diagram - MasterChef



Code Flow Diagram - SyrupBar



Code Flow Diagram - WIS Token



Slither Results Log

Slither log >> MasterChef.sol

```
INFO:Detectors:
SyrupBar.safeCakeTransfer(address,uint256) (MasterChef.sol#1221-1228) ignores return value by cake.transfer(_to,cakeBal) (MasterChef.sol#1224)
SyrupBar.safeCakeTransfer(address,uint256) (MasterChef.sol#1221-1228) ignores return value by cake.transfer(_to,_amount) (MasterChef.sol#1226)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
MasterChef.pendingCake(uint256,address) (MasterChef.sol#1632-1643) performs a multiplication on the result of a division:
	-cakeReward = multiplier.mul(cakePerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#1639)
	-accCakePerShare = accCakePerShare.add(cakeReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1640)
MasterChef.updatePool(uint256) (MasterChef.sol#1655-1671) performs a multiplication on the result of a division:
	-cakeReward = multiplier.mul(cakePerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#1666)
	-pool.accCakePerShare = pool.accCakePerShare.add(cakeReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1669)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
WisToken._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1165-1183) uses a dangerous strict equality:
	- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (MasterChef.sol#1175)
SyrupBar._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1429-1447) uses a dangerous strict equality:
	- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (MasterChef.sol#1439)
MasterChef.updatePool(uint256) (MasterChef.sol#1655-1671) uses a dangerous strict equality:
	- lpSupply == 0 (MasterChef.sol#1661)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef.add(uint256,IBEP20,bool) (MasterChef.sol#1585-1598):
	External calls:
	- massUpdatePools() (MasterChef.sol#1587)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	State variables written after the call(s):
	- poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0)) (MasterChef.sol#1591-1596)
	- updateStakingPool() (MasterChef.sol#1597)
		- poolInfo[0].allocPoint = points (MasterChef.sol#1622)
	- totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChef.sol#1590)
	- updateStakingPool() (MasterChef.sol#1597)
		- totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(points) (MasterChef.sol#1621)
Reentrancy in MasterChef.deposit(uint256,uint256) (MasterChef.sol#1674-1693):
	External calls:
	- updatePool(_pid) (MasterChef.sol#1680)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1684)
		- syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
	- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1688)
	State variables written after the call(s):
	- user.amount = user.amount.add(_amount) (MasterChef.sol#1689)
	- user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12) (MasterChef.sol#1691)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1759-1766):
	External calls:
	- pool.lpToken.safeTransfer(address(msg.sender),user.amount) (MasterChef.sol#1762)
	State variables written after the call(s):
	- user.amount = 0 (MasterChef.sol#1764)
	- user.rewardDebt = 0 (MasterChef.sol#1765)
Reentrancy in MasterChef.enterStaking(uint256) (MasterChef.sol#1717-1735):
	External calls:
	- updatePool(0) (MasterChef.sol#1720)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1724)
```

```
        - syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
    - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1728)
    State variables written after the call(s):
    - user.amount = user.amount.add(_amount) (MasterChef.sol#1729)
    - user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12) (MasterChef.sol#1731)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1738-1756):
    External calls:
    - updatePool(0) (MasterChef.sol#1743)
        - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
        - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    - safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1746)
        - syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
    State variables written after the call(s):
    - user.amount = user.amount.sub(_amount) (MasterChef.sol#1749)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1738-1756):
    External calls:
    - updatePool(0) (MasterChef.sol#1743)
        - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
        - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    - safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1746)
        - syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
    - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1750)
    State variables written after the call(s):
    - user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12) (MasterChef.sol#1752)
Reentrancy in MasterChef.set(uint256,uint256,bool) (MasterChef.sol#1601-1611):
    External calls:
    - massUpdatePools() (MasterChef.sol#1603)
        - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
        - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    State variables written after the call(s):
    - poolInfo[_pid].allocPoint = _allocPoint (MasterChef.sol#1607)
    - updateStakingPool() (MasterChef.sol#1609)
        - poolInfo[0].allocPoint = points (MasterChef.sol#1622)
    - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (MasterChef.sol#1605)
    - updateStakingPool() (MasterChef.sol#1609)
        - totalAllocPoint = totalAllocPoint.sub(poolInfo[0].allocPoint).add(points) (MasterChef.sol#1621)
Reentrancy in MasterChef.updatePool(uint256) (MasterChef.sol#1655-1671):
    External calls:
    - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
    - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    State variables written after the call(s):
    - pool.accCakePerShare = pool.accCakePerShare.add(cakeReward.mul(1e12).div(lpSupply))
(MasterChef.sol#1669)
    - pool.lastRewardBlock = block.number (MasterChef.sol#1670)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1696-1714):
    External calls:
    - updatePool(_pid) (MasterChef.sol#1703)
        - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
        - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    - safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1706)
        - syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
    State variables written after the call(s):
    - user.amount = user.amount.sub(_amount) (MasterChef.sol#1709)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1696-1714):
    External calls:
    - updatePool(_pid) (MasterChef.sol#1703)
        - cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
        - cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
    - safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1706)
        - syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
    - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1710)
    State variables written after the call(s):
    - user.rewardDebt = user.amount.mul(pool.accCakePerShare).div(1e12) (MasterChef.sol#1712)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
BEP20.constructor(string,string).name (MasterChef.sol#695) shadows:
    - BEP20.name() (MasterChef.sol#711-713) (function)
    - IBEP20.name() (MasterChef.sol#219) (function)
BEP20.constructor(string,string).symbol (MasterChef.sol#695) shadows:
    - BEP20.symbol() (MasterChef.sol#725-727) (function)
    - IBEP20.symbol() (MasterChef.sol#214) (function)
BEP20.allowance(address,address).owner (MasterChef.sol#759) shadows:
    - Ownable.owner() (MasterChef.sol#605-607) (function)
BEP20._approve(address,address,uint256).owner (MasterChef.sol#931) shadows:
    - Ownable.owner() (MasterChef.sol#605-607) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
```

MasterChef.constructor(WisToken,SyrupBar,address,uint256,uint256)._devaddr (MasterChef.sol#1542) lacks a zero-check on :
	- devaddr = _devaddr (MasterChef.sol#1548)
MasterChef.dev(address)._devaddr (MasterChef.sol#1774) lacks a zero-check on :
	- devaddr = _devaddr (MasterChef.sol#1776)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256) (MasterChef.sol#1674-1693):
	External calls:
	- updatePool(_pid) (MasterChef.sol#1680)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1684)
		- syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
	- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1688)
	Event emitted after the call(s):
	- Deposit(msg.sender,_pid,_amount) (MasterChef.sol#1692)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1759-1766):
	External calls:
	- pool.lpToken.safeTransfer(address(msg.sender),user.amount) (MasterChef.sol#1762)
	Event emitted after the call(s):
	- EmergencyWithdraw(msg.sender,_pid,user.amount) (MasterChef.sol#1763)
Reentrancy in MasterChef.enterStaking(uint256) (MasterChef.sol#1717-1735):
	External calls:
	- updatePool(0) (MasterChef.sol#1720)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1724)
		- syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
	- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1728)
	- syrup.mint(msg.sender,_amount) (MasterChef.sol#1733)
	Event emitted after the call(s):
	- Deposit(msg.sender,0,_amount) (MasterChef.sol#1734)
Reentrancy in MasterChef.leaveStaking(uint256) (MasterChef.sol#1738-1756):
	External calls:
	- updatePool(0) (MasterChef.sol#1743)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1746)
		- syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
	- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1750)
	- syrup.burn(msg.sender,_amount) (MasterChef.sol#1754)
	Event emitted after the call(s):
	- Withdraw(msg.sender,0,_amount) (MasterChef.sol#1755)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1696-1714):
	External calls:
	- updatePool(_pid) (MasterChef.sol#1703)
		- cake.mint(devaddr,cakeReward.div(10)) (MasterChef.sol#1667)
		- cake.mint(address(syrup),cakeReward) (MasterChef.sol#1668)
	- safeCakeTransfer(msg.sender,pending) (MasterChef.sol#1706)
		- syrup.safeCakeTransfer(_to,_amount) (MasterChef.sol#1770)
	- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1710)
	Event emitted after the call(s):
	- Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#1713)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
WisToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1031-1072) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now <= expiry,CAKE::delegateBySig: signature expired) (MasterChef.sol#1070)
SyrupBar.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1295-1336) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now <= expiry,CAKE::delegateBySig: signature expired) (MasterChef.sol#1334)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (MasterChef.sol#317-328) uses assembly
	- INLINE ASM (MasterChef.sol#324-326)
Address._functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#425-451) uses assembly
	- INLINE ASM (MasterChef.sol#443-446)
WisToken.getChainId() (MasterChef.sol#1190-1194) uses assembly
	- INLINE ASM (MasterChef.sol#1192)
SyrupBar.getChainId() (MasterChef.sol#1454-1458) uses assembly
	- INLINE ASM (MasterChef.sol#1456)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
MasterChef.antiHack() (MasterChef.sol#1578-1581) compares to a boolean constant:

-require(bool,string)(whiteListAddresses[msg.sender] == true || msg.sender == tx.origin,AntiHack: caller is not whitelisted) (MasterChef.sol#1579)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address.functionCall(address,bytes) (MasterChef.sol#372-374) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MasterChef.sol#401-407) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#415-423) is never used and should be removed
Address.sendValue(address,uint256) (MasterChef.sol#346-352) is never used and should be removed
BEP20._burnFrom(address,uint256) (MasterChef.sol#948-955) is never used and should be removed
Context._msgData() (MasterChef.sol#569-572) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (MasterChef.sol#492-506) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#517-527) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#508-515) is never used and should be removed
SafeMath.min(uint256,uint256) (MasterChef.sol#180-182) is never used and should be removed
SafeMath.mod(uint256,uint256) (MasterChef.sol#155-157) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MasterChef.sol#171-178) is never used and should be removed
SafeMath.sqrt(uint256) (MasterChef.sol#185-196) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (MasterChef.sol#346-352):
- (success) = recipient.call{value: amount}() (MasterChef.sol#350)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#425-451):
- (success,returndata) = target.call{value: weiValue}(data) (MasterChef.sol#434)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter WisToken.mint(address,uint256)._to (MasterChef.sol#961) is not in mixedCase
Parameter WisToken.mint(address,uint256)._amount (MasterChef.sol#961) is not in mixedCase
Variable WisToken._delegates (MasterChef.sol#973) is not in mixedCase
Parameter SyrupBar.mint(address,uint256)._to (MasterChef.sol#1200) is not in mixedCase
Parameter SyrupBar.mint(address,uint256)._amount (MasterChef.sol#1200) is not in mixedCase
Parameter SyrupBar.burn(address,uint256)._from (MasterChef.sol#1205) is not in mixedCase
Parameter SyrupBar.burn(address,uint256)._amount (MasterChef.sol#1205) is not in mixedCase
Parameter SyrupBar.safeCakeTransfer(address,uint256)._to (MasterChef.sol#1221) is not in mixedCase
Parameter SyrupBar.safeCakeTransfer(address,uint256)._amount (MasterChef.sol#1221) is not in mixedCase
Variable SyrupBar._delegates (MasterChef.sol#1237) is not in mixedCase
Parameter MasterChef.addToWhitelist(address,bool)._address (MasterChef.sol#1573) is not in mixedCase
Parameter MasterChef.addToWhitelist(address,bool)._isWhitelisted (MasterChef.sol#1573) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,bool)._allocPoint (MasterChef.sol#1585) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,bool)._lpToken (MasterChef.sol#1585) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,bool)._withUpdate (MasterChef.sol#1585) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,bool)._pid (MasterChef.sol#1601) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,bool)._allocPoint (MasterChef.sol#1601) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,bool)._withUpdate (MasterChef.sol#1601) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (MasterChef.sol#1627) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (MasterChef.sol#1627) is not in mixedCase
Parameter MasterChef.pendingCake(uint256,address)._pid (MasterChef.sol#1632) is not in mixedCase
Parameter MasterChef.pendingCake(uint256,address)._user (MasterChef.sol#1632) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (MasterChef.sol#1655) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._pid (MasterChef.sol#1674) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256)._amount (MasterChef.sol#1674) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (MasterChef.sol#1696) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (MasterChef.sol#1696) is not in mixedCase
Parameter MasterChef.enterStaking(uint256)._amount (MasterChef.sol#1717) is not in mixedCase
Parameter MasterChef.leaveStaking(uint256)._amount (MasterChef.sol#1738) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (MasterChef.sol#1759) is not in mixedCase
Parameter MasterChef.safeCakeTransfer(address,uint256)._to (MasterChef.sol#1769) is not in mixedCase
Parameter MasterChef.safeCakeTransfer(address,uint256)._amount (MasterChef.sol#1769) is not in mixedCase
Parameter MasterChef.dev(address)._devaddr (MasterChef.sol#1774) is not in mixedCase
Variable MasterChef.BONUS_MULTIPLIER (MasterChef.sol#1520) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (MasterChef.sol#570)" inContext (MasterChef.sol#560-573)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (MasterChef.sol#624-627)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (MasterChef.sol#633-635)
decimals() should be declared external:

- BEP20.decimals() (MasterChef.sol#718-720)
symbol() should be declared external:
- BEP20.symbol() (MasterChef.sol#725-727)
totalSupply() should be declared external:
- BEP20.totalSupply() (MasterChef.sol#732-734)
transfer(address,uint256) should be declared external:
- BEP20.transfer(address,uint256) (MasterChef.sol#751-754)
allowance(address,address) should be declared external:
- BEP20.allowance(address,address) (MasterChef.sol#759-761)
approve(address,uint256) should be declared external:
- BEP20.approve(address,uint256) (MasterChef.sol#770-773)
transferFrom(address,address,uint256) should be declared external:
- BEP20.transferFrom(address,address,uint256) (MasterChef.sol#787-799)
increaseAllowance(address,uint256) should be declared external:
- BEP20.increaseAllowance(address,uint256) (MasterChef.sol#813-816)
decreaseAllowance(address,uint256) should be declared external:
- BEP20.decreaseAllowance(address,uint256) (MasterChef.sol#832-839)
mint(uint256) should be declared external:
- BEP20.mint(uint256) (MasterChef.sol#849-852)
mint(address,uint256) should be declared external:
- WisToken.mint(address,uint256) (MasterChef.sol#961-964)
mint(address,uint256) should be declared external:
- SyrupBar.mint(address,uint256) (MasterChef.sol#1200-1203)
burn(address,uint256) should be declared external:
- SyrupBar.burn(address,uint256) (MasterChef.sol#1205-1208)
safeCakeTransfer(address,uint256) should be declared external:
- SyrupBar.safeCakeTransfer(address,uint256) (MasterChef.sol#1221-1228)
updateMultiplier(uint256) should be declared external:
- MasterChef.updateMultiplier(uint256) (MasterChef.sol#1564-1566)
addToWhitelist(address,bool) should be declared external:
- MasterChef.addToWhitelist(address,bool) (MasterChef.sol#1573-1575)
add(uint256,IBEP20,bool) should be declared external:
- MasterChef.add(uint256,IBEP20,bool) (MasterChef.sol#1585-1598)
set(uint256,uint256,bool) should be declared external:
- MasterChef.set(uint256,uint256,bool) (MasterChef.sol#1601-1611)
deposit(uint256,uint256) should be declared external:
- MasterChef.deposit(uint256,uint256) (MasterChef.sol#1674-1693)
withdraw(uint256,uint256) should be declared external:
- MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1696-1714)
enterStaking(uint256) should be declared external:
- MasterChef.enterStaking(uint256) (MasterChef.sol#1717-1735)
leaveStaking(uint256) should be declared external:
- MasterChef.leaveStaking(uint256) (MasterChef.sol#1738-1756)
emergencyWithdraw(uint256) should be declared external:
- MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1759-1766)
dev(address) should be declared external:
- MasterChef.dev(address) (MasterChef.sol#1774-1777)

Slither log >> SyrupBar.sol

SyrupBar.safeCakeTransfer(address,uint256) (SyrupBar.sol#1122-1129) ignores return value by cake.transfer(_to,cakeBal) (SyrupBar.sol#1125)
SyrupBar.safeCakeTransfer(address,uint256) (SyrupBar.sol#1122-1129) ignores return value by cake.transfer(_to,_amount) (SyrupBar.sol#1127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
WisToken._writeCheckpoint(address,uint32,uint256,uint256) (SyrupBar.sol#1066-1084) uses a dangerous strict equality:
- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (SyrupBar.sol#1076)
SyrupBar._writeCheckpoint(address,uint32,uint256,uint256) (SyrupBar.sol#1330-1348) uses a dangerous strict equality:
- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (SyrupBar.sol#1340)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BEP20.constructor(string,string).name (SyrupBar.sol#596) shadows:
- BEP20.name() (SyrupBar.sol#612-614) (function)
- IBEP20.name() (SyrupBar.sol#126) (function)
BEP20.constructor(string,string).symbol (SyrupBar.sol#596) shadows:
- BEP20.symbol() (SyrupBar.sol#626-628) (function)
- IBEP20.symbol() (SyrupBar.sol#121) (function)
BEP20.allowance(address,address).owner (SyrupBar.sol#660) shadows:
- Ownable.owner() (SyrupBar.sol#64-66) (function)
BEP20._approve(address,address,uint256).owner (SyrupBar.sol#832) shadows:

- Ownable.owner() (SyrupBar.sol#64-66) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
WisToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (SyrupBar.sol#932-973) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,CAKE::delegateBySig: signature expired) (SyrupBar.sol#971)
SyrupBar.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (SyrupBar.sol#1196-1237) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,CAKE::delegateBySig: signature expired) (SyrupBar.sol#1235)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (SyrupBar.sol#411-422) uses assembly
- INLINE ASM (SyrupBar.sol#418-420)
Address._functionCallWithValue(address,bytes,uint256,string) (SyrupBar.sol#519-545) uses assembly
- INLINE ASM (SyrupBar.sol#537-540)
WisToken.getChainId() (SyrupBar.sol#1091-1095) uses assembly
- INLINE ASM (SyrupBar.sol#1093)
SyrupBar.getChainId() (SyrupBar.sol#1355-1359) uses assembly
- INLINE ASM (SyrupBar.sol#1357)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (SyrupBar.sol#519-545) is never used and should be removed
Address.functionCall(address,bytes) (SyrupBar.sol#466-468) is never used and should be removed
Address.functionCall(address,bytes,string) (SyrupBar.sol#476-482) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (SyrupBar.sol#495-501) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (SyrupBar.sol#509-517) is never used and should be removed
Address.isContract(address) (SyrupBar.sol#411-422) is never used and should be removed
Address.sendValue(address,uint256) (SyrupBar.sol#440-446) is never used and should be removed
BEP20._burnFrom(address,uint256) (SyrupBar.sol#849-856) is never used and should be removed
Context._msgData() (SyrupBar.sol#28-31) is never used and should be removed
SafeMath.div(uint256,uint256) (SyrupBar.sol#305-307) is never used and should be removed
SafeMath.div(uint256,uint256,string) (SyrupBar.sol#321-331) is never used and should be removed
SafeMath.min(uint256,uint256) (SyrupBar.sol#370-372) is never used and should be removed
SafeMath.mod(uint256,uint256) (SyrupBar.sol#345-347) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (SyrupBar.sol#361-368) is never used and should be removed
SafeMath.mul(uint256,uint256) (SyrupBar.sol#279-291) is never used and should be removed
SafeMath.sqrt(uint256) (SyrupBar.sol#375-386) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (SyrupBar.sol#440-446):
- (success) = recipient.call{value: amount}() (SyrupBar.sol#444)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (SyrupBar.sol#519-545):
- (success,returndata) = target.call{value: weiValue}(data) (SyrupBar.sol#528)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter WisToken.mint(address,uint256)._to (SyrupBar.sol#862) is not in mixedCase
Parameter WisToken.mint(address,uint256)._amount (SyrupBar.sol#862) is not in mixedCase
Variable WisToken._delegates (SyrupBar.sol#874) is not in mixedCase
Parameter SyrupBar.mint(address,uint256)._to (SyrupBar.sol#1101) is not in mixedCase
Parameter SyrupBar.mint(address,uint256)._amount (SyrupBar.sol#1101) is not in mixedCase
Parameter SyrupBar.burn(address,uint256)._from (SyrupBar.sol#1106) is not in mixedCase
Parameter SyrupBar.burn(address,uint256)._amount (SyrupBar.sol#1106) is not in mixedCase
Parameter SyrupBar.safeCakeTransfer(address,uint256)._to (SyrupBar.sol#1122) is not in mixedCase
Parameter SyrupBar.safeCakeTransfer(address,uint256)._amount (SyrupBar.sol#1122) is not in mixedCase
Variable SyrupBar._delegates (SyrupBar.sol#1138) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (SyrupBar.sol#29)" inContext (SyrupBar.sol#19-32)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (SyrupBar.sol#83-86)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (SyrupBar.sol#92-94)
decimals() should be declared external:
- BEP20.decimals() (SyrupBar.sol#619-621)
symbol() should be declared external:
- BEP20.symbol() (SyrupBar.sol#626-628)
totalSupply() should be declared external:
- BEP20.totalSupply() (SyrupBar.sol#633-635)
transfer(address,uint256) should be declared external:
- BEP20.transfer(address,uint256) (SyrupBar.sol#652-655)

allowance(address,address) should be declared external:
	- BEP20.allowance(address,address) (SyrupBar.sol#660-662)
approve(address,uint256) should be declared external:
	- BEP20.approve(address,uint256) (SyrupBar.sol#671-674)
transferFrom(address,address,uint256) should be declared external:
	- BEP20.transferFrom(address,address,uint256) (SyrupBar.sol#688-700)
increaseAllowance(address,uint256) should be declared external:
	- BEP20.increaseAllowance(address,uint256) (SyrupBar.sol#714-717)
decreaseAllowance(address,uint256) should be declared external:
	- BEP20.decreaseAllowance(address,uint256) (SyrupBar.sol#733-740)
mint(uint256) should be declared external:
	- BEP20.mint(uint256) (SyrupBar.sol#750-753)
mint(address,uint256) should be declared external:
	- WisToken.mint(address,uint256) (SyrupBar.sol#862-865)
mint(address,uint256) should be declared external:
	- SyrupBar.mint(address,uint256) (SyrupBar.sol#1101-1104)
burn(address,uint256) should be declared external:
	- SyrupBar.burn(address,uint256) (SyrupBar.sol#1106-1109)
safeCakeTransfer(address,uint256) should be declared external:
	- SyrupBar.safeCakeTransfer(address,uint256) (SyrupBar.sol#1122-1129)

Slither log >> WisToken.sol

WisToken._writeCheckpoint(address,uint32,uint256,uint256) (WisToken.sol#1071-1089) uses a dangerous strict equality:
	- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (WisToken.sol#1081)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BEP20.constructor(string,string).name (WisToken.sol#601) shadows:
	- BEP20.name() (WisToken.sol#617-619) (function)
	- IBEP20.name() (WisToken.sol#131) (function)
BEP20.constructor(string,string).symbol (WisToken.sol#601) shadows:
	- BEP20.symbol() (WisToken.sol#631-633) (function)
	- IBEP20.symbol() (WisToken.sol#126) (function)
BEP20.allowance(address,address).owner (WisToken.sol#665) shadows:
	- Ownable.owner() (WisToken.sol#69-71) (function)
BEP20._approve(address,address,uint256).owner (WisToken.sol#837) shadows:
	- Ownable.owner() (WisToken.sol#69-71) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
WisToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (WisToken.sol#937-978) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now <= expiry,CAKE::delegateBySig: signature expired) (WisToken.sol#976)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (WisToken.sol#416-427) uses assembly
	- INLINE ASM (WisToken.sol#423-425)
Address._functionCallWithValue(address,bytes,uint256,string) (WisToken.sol#524-550) uses assembly
	- INLINE ASM (WisToken.sol#542-545)
WisToken.getChainId() (WisToken.sol#1096-1100) uses assembly
	- INLINE ASM (WisToken.sol#1098)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (WisToken.sol#524-550) is never used and should be removed
Address.functionCall(address,bytes) (WisToken.sol#471-473) is never used and should be removed
Address.functionCall(address,bytes,string) (WisToken.sol#481-487) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (WisToken.sol#500-506) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (WisToken.sol#514-522) is never used and should be removed
Address.isContract(address) (WisToken.sol#416-427) is never used and should be removed
Address.sendValue(address,uint256) (WisToken.sol#445-451) is never used and should be removed
BEP20._burn(address,uint256) (WisToken.sol#815-821) is never used and should be removed
BEP20._burnFrom(address,uint256) (WisToken.sol#854-861) is never used and should be removed
Context._msgData() (WisToken.sol#33-36) is never used and should be removed
SafeMath.div(uint256,uint256) (WisToken.sol#310-312) is never used and should be removed
SafeMath.div(uint256,uint256,string) (WisToken.sol#326-336) is never used and should be removed
SafeMath.min(uint256,uint256) (WisToken.sol#375-377) is never used and should be removed
SafeMath.mod(uint256,uint256) (WisToken.sol#350-352) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (WisToken.sol#366-373) is never used and should be removed
SafeMath.mul(uint256,uint256) (WisToken.sol#284-296) is never used and should be removed
SafeMath.sqrt(uint256) (WisToken.sol#380-391) is never used and should be removed

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (WisToken.sol#445-451):
	- (success) = recipient.call{value: amount}() (WisToken.sol#449)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (WisToken.sol#524-550):
	- (success,returndata) = target.call{value: weiValue}(data) (WisToken.sol#533)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter WisToken.mint(address,uint256)._to (WisToken.sol#867) is not in mixedCase
Parameter WisToken.mint(address,uint256)._amount (WisToken.sol#867) is not in mixedCase
Variable WisToken._delegates (WisToken.sol#879) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (WisToken.sol#34)" inContext (WisToken.sol#24-37)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (WisToken.sol#88-91)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (WisToken.sol#97-99)
decimals() should be declared external:
	- BEP20.decimals() (WisToken.sol#624-626)
symbol() should be declared external:
	- BEP20.symbol() (WisToken.sol#631-633)
totalSupply() should be declared external:
	- BEP20.totalSupply() (WisToken.sol#638-640)
transfer(address,uint256) should be declared external:
	- BEP20.transfer(address,uint256) (WisToken.sol#657-660)
allowance(address,address) should be declared external:
	- BEP20.allowance(address,address) (WisToken.sol#665-667)
approve(address,uint256) should be declared external:
	- BEP20.approve(address,uint256) (WisToken.sol#676-679)
transferFrom(address,address,uint256) should be declared external:
	- BEP20.transferFrom(address,address,uint256) (WisToken.sol#693-705)
increaseAllowance(address,uint256) should be declared external:
	- BEP20.increaseAllowance(address,uint256) (WisToken.sol#719-722)
decreaseAllowance(address,uint256) should be declared external:
	- BEP20.decreaseAllowance(address,uint256) (WisToken.sol#738-745)
mint(uint256) should be declared external:
	- BEP20.mint(uint256) (WisToken.sol#755-758)
mint(address,uint256) should be declared external:
	- WisToken.mint(address,uint256) (WisToken.sol#867-870)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:WisToken.sol analyzed (7 contracts with 75 detectors), 45 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io