Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Brownie Swap
Website:	https://brownieswap.io
Platform:	Binance Smart Chain
Language:	Solidity
Date:	September 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

Introduction

EtherAuthority was contracted by the Brownie Swap team to perform the Security audit of the Brownie Swap smart contracts 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 3rd, 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

.

Audit scope

Name	Code Review and Security Analysis Report for Brownie Swap Smart Contracts
Platform	**BSC / Solidity**
File 1	BrownieToken.sol
File 1 MD5 Hash	D2A876DC025F2B5C733E3386A1A7F5AD
File 2	Multicall.sol
File 2 MD5 Hash	FD6FA3E278926ED4FFCF8E9620DB918C
File 3	Timelock.sol
File 3 MD5 Hash	B62208B71930D65576A66819545253EA
File 4	MasterChef.sol
File 4 MD5 Hash	08C67D2EF41113E1C9E5E160BE4E5900
Audit Date	September 3rd, 2021

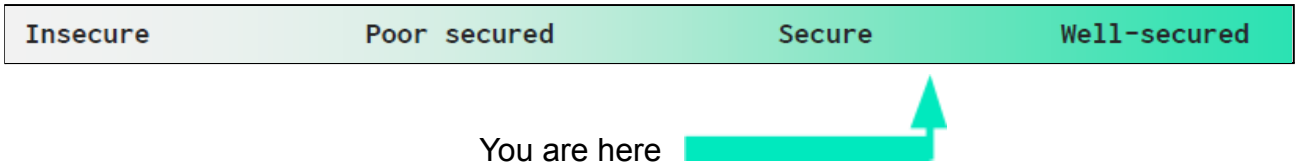
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1: BrownieToken.sol • Name: Brownie Token • Symbol: BRWN • Decimals: 18 • Initial Supply: 0 • Burn rate: 25% of tax • Transfer tax: 0% • Max transfer tax rate: 0% • Min amount to liquify: 50 Tokens • Minting limits: No limit	**YES, This is valid. The owner can change rate percentages.** **Ownership is sent to the masterChef smart contract, so this is safe from unlimited minting problems.**
File 2: Multicall.sol • The Multicall contracts can Aggregate results from multiple read-only function calls.	**YES, This is valid.**
File 3: Timelock.sol • Grace Period: 14 days • Minimum Delay: 6 hours • Maximum Delay: 30 days	**YES, This is valid.**
File 4: MasterChef.sol • Bonus Multiplier: 1 • Max harvest interval: 14 days • Max referral commission rate: 10% • Referral commission rate: 1% • 10% of Brownie rewards are minted for devs (which are splitted into 10% into vault and 90% to dev address)	**YES, This is valid. Master chef contract owner can change referral commission rates.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. This protocol has many owner functions. And they must be executed as per the business plan.



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 9 low and some very low level issues.
These issues are acknowledged as part of the business plan.

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	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
	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**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 4 smart contract files. Smart contracts also contain Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in Brownie Swap 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 Brownie Protocol.

The Brownie Protocol 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 in the smart contracts.

Documentation

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

As mentioned above, some code parts are **not well** commented. So it is 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://www.brownieswap.io 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

BrownieToken.sol

(1) Interface

(a) IBEP20

(b) IBrownieReferral

(c) IUniswapV2Router01

(d) IUniswapV2Router02

(e) IUniswapV2Pair

(f) IUniswapV2Factory

(2) Inherited contracts

(a) BEP20

(b) Context

(c) Ownable

(d) ReentrancyGuard

(3) Usages

(a) using SafeERC20 for IERC20;

(b) using Address for address;

(c) using SafeMath for uint;

(4) Struct

(a) Checkpoint

(5) Events

(a) event OperatorTransferred(address indexed previousOperator, address indexed newOperator);

(b) event TransferTaxRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);

(c) event BurnRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);

(d) event MaxTransferAmountRateUpdated(address indexed operator, uint256 previousRate, uint256 newRate);

(e) event SwapAndLiquifyEnabledUpdated(address indexed operator, bool enabled);

(f) event MinAmountToLiquifyUpdated(address indexed operator, uint256 previousAmount, uint256 newAmount);

(g) event PancakeSwapRouterUpdated(address indexed operator, address indexed router, address indexed pair);

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

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

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

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	onlyOperator	modifier	Passed	No Issue
2	antiWhale	modifier	Passed	No Issue
3	lockTheSwap	modifier	Passed	No Issue
4	transferTaxFree	modifier	Passed	No Issue
5	constructor	write	Passed	No Issue
6	mint	write	Unlimited minting	Ownership sent to masterChef
7	_transfer	internal	Passed	No Issue
8	swapAndLiquify	write	Passed	No Issue
9	swapTokensForEth	write	Passed	No Issue
10	addLiquidity	write	Passed	No Issue
11	maxTransferAmount	read	Passed	No Issue
12	isExcludedFromAntiWhale	read	Passed	No Issue
13	receive	external	Passed	No Issue
14	updateTransferTaxRate	write	Passed	No Issue
15	updateBurnRate	write	access only Operator	No Issue
16	updateMaxTransferAmountRate	write	access only Operator	No Issue
17	updateMinAmountToLiquify	write	access only Operator	No Issue
18	setExcludedFromAntiWhale	write	access only Operator	No Issue
19	updateSwapAndLiquifyEnabled	write	access only Operator	No Issue
20	updatePancakeSwapRouter	write	access only Operator	No Issue

21	operator	read	Passed	No Issue
22	transferOperator	write	access only Operator	No Issue
23	delegates	external	Passed	No Issue
24	delegate	external	Passed	No Issue
25	delegateBySig	external	Handle sig carefully	No Issue
26	getCurrentVotes	external	Passed	No Issue
27	getPriorVotes	external	Infinite loop possibility	array length limited
28	_delegate	internal	Passed	No Issue
29	_moveDelegates	internal	Passed	No Issue
30	_writeCheckpoint	internal	Passed	No Issue
31	safe32	internal	Passed	No Issue
32	getChainId	internal	Passed	No Issue
33	getOwner	external	Passed	No Issue
34	name	read	Passed	No Issue
35	decimals	read	Passed	No Issue
36	symbol	read	Passed	No Issue
37	totalSupply	read	Passed	No Issue
38	balanceOf	read	Passed	No Issue
39	transfer	write	Passed	No Issue
40	allowance	read	Passed	No Issue
41	approve	write	Passed	No Issue
42	transferFrom	write	Passed	No Issue
43	increaseAllowance	write	Passed	No Issue
44	decreaseAllowance	write	Passed	No Issue
45	mint	write	access only Owner	No Issue
46	_transfer	internal	Passed	No Issue
47	_mint	internal	Passed	No Issue
48	_burn	internal	Passed	No Issue
49	_approve	internal	Passed	No Issue
50	_burnFrom	internal	Passed	No Issue

Multicall.sol

(1) Functions

Sl.	Functions	Type	Observation	Conclusion
1	aggregate	write	Passed	No Issue
2	getEthBalance	read	Passed	No Issue
3	getBlockHash	read	Passed	No Issue
4	getLastBlockHash	read	Passed	No Issue
5	getCurrentBlockTimestamp	read	Passed	No Issue
6	getCurrentBlockDifficulty	read	Passed	No Issue
7	getCurrentBlockGasLimit	read	Passed	No Issue

8	getCurrentBlockCoinbase	read	Passed	No Issue

Timelock.sol

(1) Usages

(a) using SafeMath for uint;

(2) Events

(a) event NewAdmin(address indexed newAdmin);

(b) event NewPendingAdmin(address indexed newPendingAdmin);

(c) event NewDelay(uint indexed newDelay);

(d) event CancelTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

(e) event ExecuteTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

(f) event QueueTransaction(bytes32 indexed txHash, address indexed target, uint value, string signature, bytes data, uint eta);

(3) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	receive	external	Passed	No Issue
3	setDelay	write	Passed	No Issue
4	acceptAdmin	write	Passed	No Issue
5	setPendingAdmin	write	Passed	No Issue
6	queueTransaction	write	Passed	No Issue
7	cancelTransaction	write	Passed	No Issue
8	executeTransaction	write	Passed	No Issue
9	getBlockTimestamp	internal	Passed	No Issue

MasterChef.sol

(1) Interface

(a) IBEP20

(b) IBrownieReferral

(c) IUniswapV2Router01

(d) IUniswapV2Router02

(e) IUniswapV2Pair

(f) IUniswapV2Factory

(2) Inherited contracts

(a) Ownable

(b) ReentrancyGuard

(c) Context

(3) Usages

(a) using SafeMath for uint256;

(b) using SafeBEP20 for IBEP20;

(4) Struct

(a) UserInfo

(b) PoolInfo

(5) Events

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

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

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

(d) event EmissionRateUpdated(address indexed caller, uint256 previousAmount, uint256 newAmount);

(e) event ReferralCommissionPaid(address indexed user, address indexed referrer, uint256 commissionAmount);

(f) event RewardLockedUp(address indexed user, uint256 indexed pid, uint256 amountLockedUp);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	poolLength	external	Passed	No Issue
3	add	write	Owner must not add duplicate LP tokens.	Refer audit findings
4	set	write	access only Owner	No Issue
5	getMultiplier	write	Passed	No Issue
6	pendingBrownie	external	Passed	No Issue
7	canHarvest	read	Passed	No Issue
8	massUpdatePools	write	Infinite loop possibility	keep array length limited
9	deposit	write	Passed	No Issue
10	updatePool	write	Passed	No Issue
11	withdraw	write	Passed	No Issue
12	emergencyWithdraw	write	Passed	No Issue
13	payOrLockupPendingBrownie	internal	Passed	No Issue
14	safeBrownieTransfer	internal	Passed	No Issue
15	setDevAddress	write	Passed	No Issue
16	setFeeAddress	write	Passed	No Issue
17	setCommunityVault	write	Passed	No Issue
18	updateEmissionRate	write	access only Owner	No Issue
19	setBrownieReferral	write	access only Owner	No Issue
20	setReferralCommissionRate	write	access only Owner	No Issue
21	payReferralCommission	internal	Passed	No Issue
22	nonReentrant	modifier	Passed	No Issue
23	owner	read	Passed	No Issue
24	onlyOwner	modifier	Passed	No Issue
25	renounceOwnership	write	access only Owner	No Issue
26	transferOwnership	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

BrownieToken.sol

Critical

No Critical severity vulnerabilities were found.

High

No Critical severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Unlimited Brownie tokens minting

```
function _mint(address account, uint256 amount) internal {
    require(account != address(0), "BEP20: mint to the zero address");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
    emit Transfer(address(0), account, amount);
```

Owner of this contract can mint unlimited tokens. Having unlimited minting is considered inappropriate for tokenomics. So, we advise either to put some minting limits, or transfer the ownership to the masterChef smart contract.

Update: The ownership is transferred to masterChef contract. so, this issue is Fixed.

(2) Centralized risk in addLiquidity function

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(pancakeSwapRouter), tokenAmount);

    // add the liquidity
    pancakeSwapRouter.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        operator(),
        block.timestamp
    );
```

This operator wallet will accumulate all the liquidity over time. If the private key of that wallet is compromised, then it will create problems. So we suggest using any delegated smart contract or the same token contract itself, having governance logic.

(3) Handle signature with extra care

The function delegateBySig() at line number #1642 will enable anyone to execute this function if he has the signature of other users. This works great in normal scenarios, but this feature can be abused in phishing scams. So, we advise users to handle the signatures with extra care.

(4) Infinite loop possibility

```
    while (upper > lower) {
        uint32 center = upper - (upper - lower) / 2; // ceil, avoiding
        Checkpoint memory cp = checkpoints[account][center];
        if (cp.fromBlock == blockNumber) {
            return cp.votes;
        } else if (cp.fromBlock < blockNumber) {
            lower = center;
        } else {
            upper = center - 1;
        }
```

There is a potential intensive loop at line number #1730. This works well when there are less records to iterate. But if the array elements increase, then it will cost more gas, or even it can hit the block's gas limit. So, we suggest to keep the array elements limited, or adjust the logic without loops.

(5) Missing events:

```
function setExcludedFromAntiWhale(address _account, bool _excluded) public
    _excludedFromAntiWhale[_account] = _excluded;
}
```

Every function which changes the smart contract state, having significant impact, must emit an event. Suggest to place events which will be helpful to the clients and UIs.

(6) Gas savings

```
    _approve(address(this), address(pancakeSwapRouter), tokenAmount);
```

This approval is happening twice when tokens are transferred. This increases the gas cost to users. So, we suggest removing this line from swapTokensForEth and addLiquidity functions. And provide maximum approval to the pancake router contract when the router contract is set.

Very Low / Informational / Best practices:

(1) Input parameter validation missing

```
function updateMinAmountToLiquify(uint256 _minAmount) public onlyOperator {
    emit MinAmountToLiquifyUpdated(msg.sender, minAmountToLiquify, _minAmount);
    minAmountToLiquify = _minAmount;
}
```

The function like updateMinAmountToLiquify should validate the input amount. Ideally, there should be a range (minimum or maximum amount the owner can input). This will prevent any human errors, and it will give confidence to users that the owner can not input unintended amounts.

(2) Make variables constant

```
string private _name;
string private _symbol;
uint8 private _decimals;
```

These variable's values will be unchanged. So, please make it constant. It will save some gas. Just put a constant keyword.

(3) Consider using latest solidity version

Compiler Version	**v0.6.12+commit.27d51765**

Although this does not create major security vulnerabilities, the latest solidity version has lots of improvements, so it's recommended to use the latest solidity version, which is 0.8.7 at the time of this audit.

(4) All functions which are not called internally, must be declared as external. It is more efficient as sometimes it saves some gas.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

MasterChef.sol

Critical

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Input validation missing

```
    // 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
    function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, uint256
onlyOwner {
        require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
        require(_harvestInterval <= MAXIMUM_HARVEST_INTERVAL, "add: invalid harvest i
        if (_withUpdate) {
            massUpdatePools();
```

As said in the comment, it is advisable that the owner should never add the same LP token more than once. To prevent any human error scenario, we suggest putting one validation condition which will only allow the execution if the LP token does not exist.

(2) Missing events: All significant state changing functions must emit an event, such as:

- add
- set
- setDevAddress
- setFeeAddress
- setCommunityVault
- updateEmissionRate
- setBrownieReferral
- setReferralCommissionRate

(3) Infinite loop possibility

```
// 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);
```

In the massUpdatePools() function, 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. Or, the owner must add limited pools to prevent this scenario.

Very Low / Informational / Best practices:

(1) Input parameter validation missing:

The function should validate the input amount. Ideally, there should be a range (minimum or maximum amount the owner can input). This will prevent any human errors, and it will give confidence to users that the owner can not input unintended amounts. For example:

- setBrownieReferral
- updateEmissionRate

(2) Unnecessary code:

Masterchef smart contract contains token contract code, which is unnecessary. It makes contract code lengthy. Following contracts and interfaces can be removed:

- BEP20
- IUniswapV2Router01
- IUniswapV2Router02
- IUniswapV2Pair
- IUniswapV2Factory
- BrownieToken

And this can be simply replaced by a short interface, which can be used to interact with Brownie Token smart contract:

```
interface BrownieToken{
    function transfer(address to, uint256 amount) external returns(bool);
    function transferFrom(address from, address to, uint256 amount) external returns(bool);
    function mint(address user, uint256 amount) external returns(uint256);
    function transferTaxRate() external returns(uint256);
    function balanceOf(address user) external returns(uint256);
}
```

Timelock .sol and MultiCall.sol

These smart contracts do not have any High, medium, low vulnerabilities. Except those contracts can be deployed using the latest solidity version.

Centralization

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

- mint: Brownie token contract owner can mint unlimited tokens. We suggest the ownership must be transferred to the masterChef contract.
- add: The MasterChef owner can add a new LP to the pool.
- set: The MasterChef owner can update the given pool's BROWNIE allocation point and deposit fee.
- updateEmissionRate: The MasterChef owner can update the emission rate.
- setBrownieReferral: Update the Brownie Referral contract address by the MasterChef owner.
- setReferralCommissionRate: Update referral commission rate by the MasterChef owner.

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, but they are acknowledged as part of the business plan. 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 As-is overview section of the report.

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

Security state of the reviewed contract, based on standard audit procedure scope, is **"secured".**

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.

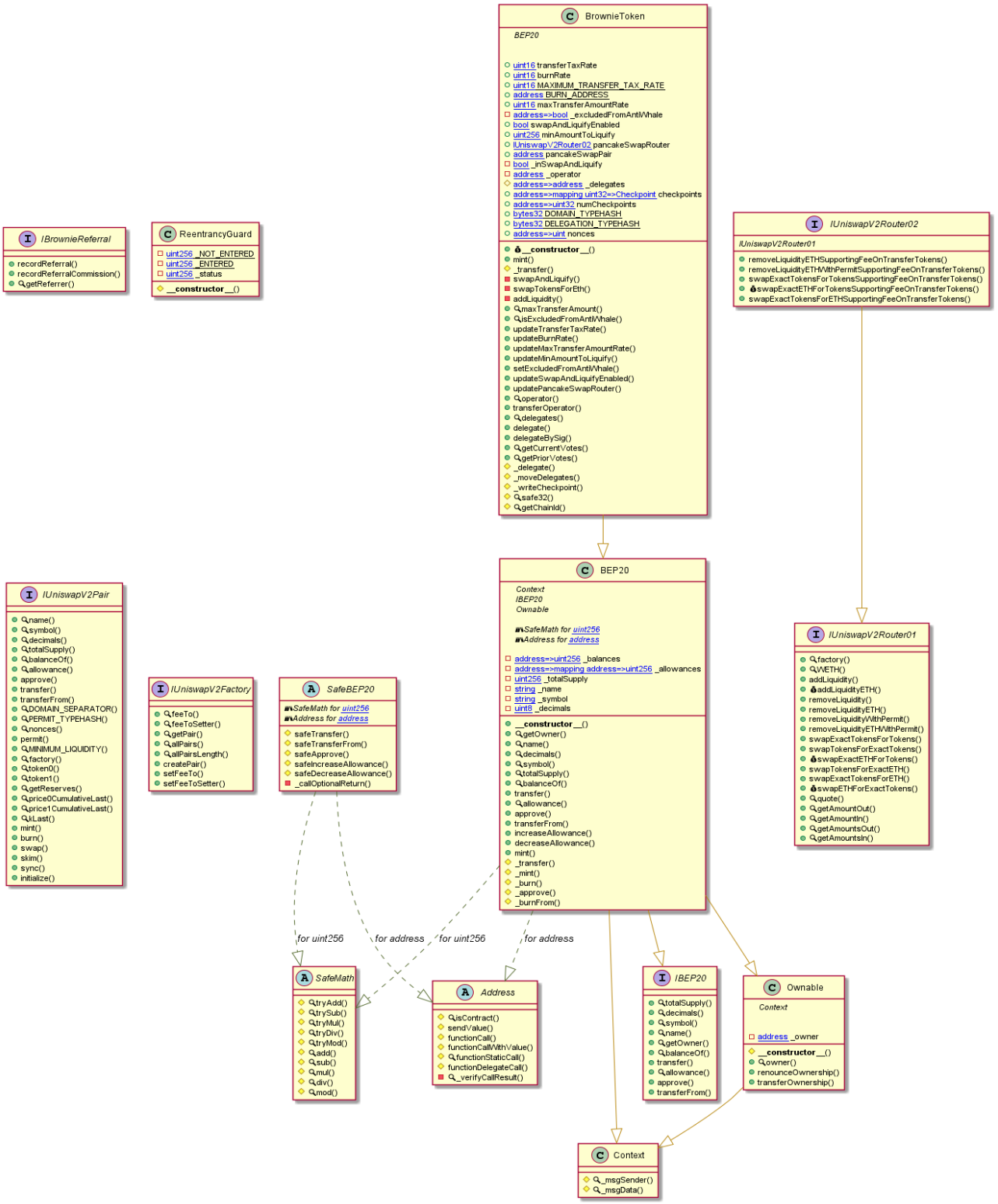
Technical Disclaimer

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.

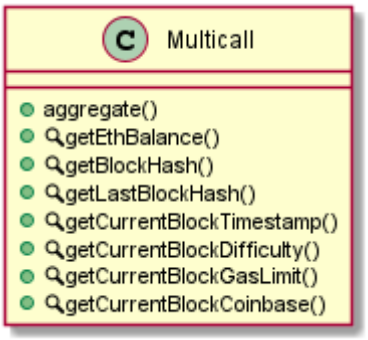
Appendix

Code Flow Diagram - Brownie Protocol

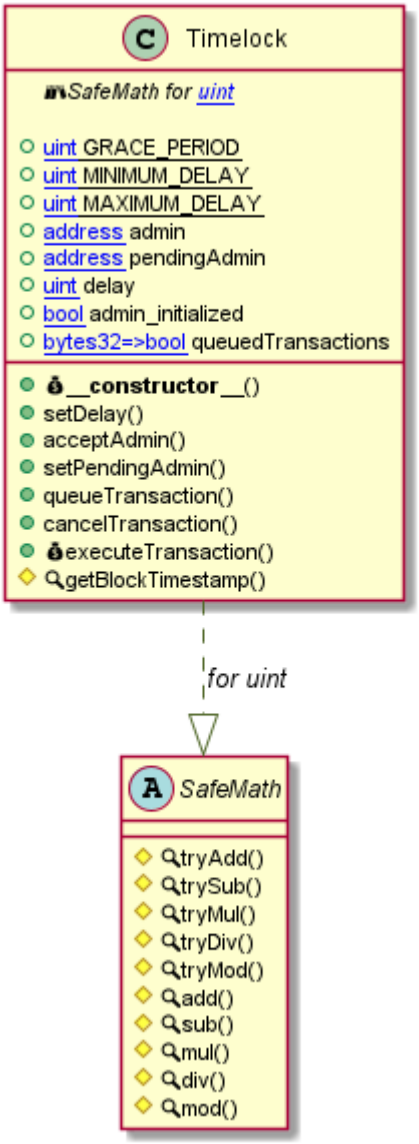
BrownieToken Diagram



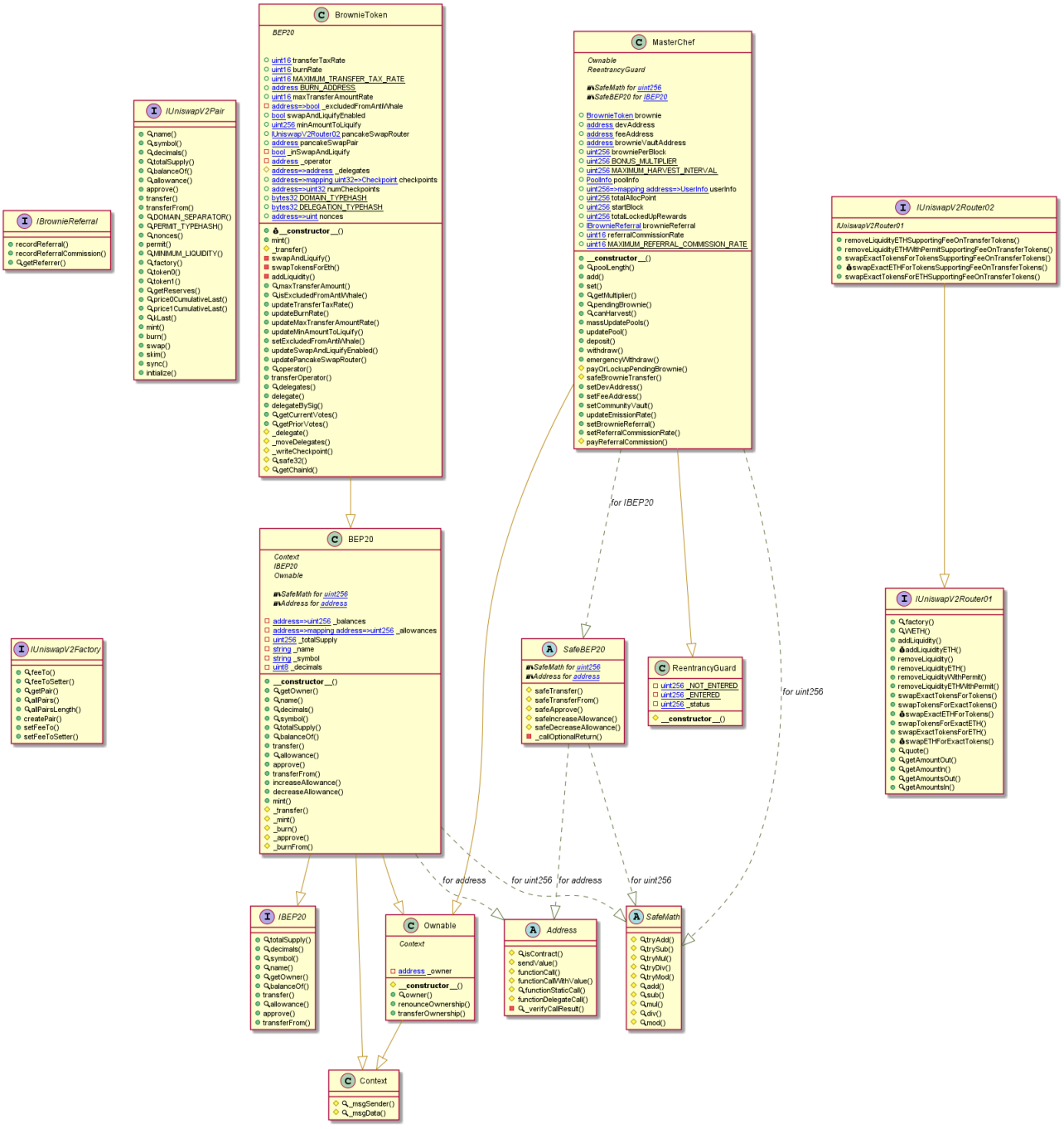
Multicall Diagram



Timelock Diagram



MasterChef Diagram



Slither Results Log

Slither log >> BrownieToken.sol

```
INFO:Detectors:
BrownieToken.addLiquidity(uint256,uint256) (BrownieToken.sol#1450-1463) sends eth to arbitrary user
        Dangerous calls:
        - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieToken.sol
#1455-1462)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in BrownieToken._transfer(address,address,uint256) (BrownieToken.sol#1364-1395):
        External calls:
        - swapAndLiquify() (BrownieToken.sol#1374)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(BrownieToken.sol#1440-1446)
        External calls sending eth:
        - swapAndLiquify() (BrownieToken.sol#1374)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        State variables written after the call(s):
        - super._transfer(sender,recipient,amount) (BrownieToken.sol#1378)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (BrownieToken.sol#987)
                - _balances[recipient] = _balances[recipient].add(amount) (BrownieToken.sol#988)
        - super._transfer(sender,BURN_ADDRESS,burnAmount) (BrownieToken.sol#1390)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (BrownieToken.sol#987)
                - _balances[recipient] = _balances[recipient].add(amount) (BrownieToken.sol#988)
        - super._transfer(sender,address(this),liquidityAmount) (BrownieToken.sol#1391)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (BrownieToken.sol#987)
                - _balances[recipient] = _balances[recipient].add(amount) (BrownieToken.sol#988)
        - super._transfer(sender,recipient,sendAmount) (BrownieToken.sol#1392)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (BrownieToken.sol#987)
                - _balances[recipient] = _balances[recipient].add(amount) (BrownieToken.sol#988)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
BrownieToken._transfer(address,address,uint256) (BrownieToken.sol#1364-1395) performs a multiplication on the result of a division:
        -taxAmount = amount.mul(transferTaxRate).div(10000) (BrownieToken.sol#1381)
        -burnAmount = taxAmount.mul(burnRate).div(100) (BrownieToken.sol#1382)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
```

```
INFO:Detectors:
BrownieToken._writeCheckpoint(address,uint32,uint256,uint256) (BrownieToken.sol#1759-1777) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (BrownieToken.sol#1769)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BrownieToken.addLiquidity(uint256,uint256) (BrownieToken.sol#1450-1463) ignores return value by pancakeSwapRouter.addLiquidityETH{value:
ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieToken.sol#1455-1462)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
BEP20.constructor(string,string).name (BrownieToken.sol#806) shadows:
        - BEP20.name() (BrownieToken.sol#822-824) (function)
        - IBEP20.name() (BrownieToken.sol#28) (function)
BEP20.constructor(string,string).symbol (BrownieToken.sol#806) shadows:
        - BEP20.symbol() (BrownieToken.sol#836-838) (function)
        - IBEP20.symbol() (BrownieToken.sol#23) (function)
BEP20.allowance(address,address).owner (BrownieToken.sol#870) shadows:
        - Ownable.owner() (BrownieToken.sol#664-666) (function)
BEP20._approve(address,address,uint256).owner (BrownieToken.sol#1042) shadows:
        - Ownable.owner() (BrownieToken.sol#664-666) (function)
BrownieToken.swapAndLiquify().maxTransferAmount (BrownieToken.sol#1400) shadows:
        - BrownieToken.maxTransferAmount() (BrownieToken.sol#1468-1470) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in BrownieToken.swapAndLiquify() (BrownieToken.sol#1398-1428):
        External calls:
        - swapTokensForEth(half) (BrownieToken.sol#1418)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(BrownieToken.sol#1440-1446)
        - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
                - _allowances[owner][spender] = amount (BrownieToken.sol#1049)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in BrownieToken._transfer(address,address,uint256) (BrownieToken.sol#1364-1395):
        External calls:
        - swapAndLiquify() (BrownieToken.sol#1374)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(BrownieToken.sol#1440-1446)
        External calls sending eth:
        - swapAndLiquify() (BrownieToken.sol#1374)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (BrownieToken.sol#989)
                - super._transfer(sender,recipient,amount) (BrownieToken.sol#1378)
        - Transfer(sender,recipient,amount) (BrownieToken.sol#989)
                - super._transfer(sender,address(this),liquidityAmount) (BrownieToken.sol#1391)
        - Transfer(sender,recipient,amount) (BrownieToken.sol#989)
                - super._transfer(sender,BURN_ADDRESS,burnAmount) (BrownieToken.sol#1390)
```

```
        - Transfer(sender,recipient,amount) (BrownieToken.sol#989)
                - super._transfer(sender,recipient,sendAmount) (BrownieToken.sol#1392)
Reentrancy in BrownieToken.swapAndLiquify() (BrownieToken.sol#1398-1428):
        External calls:
        - swapTokensForEth(half) (BrownieToken.sol#1418)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(BrownieToken.sol#1440-1446)
        - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (BrownieT
oken.sol#1455-1462)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (BrownieToken.sol#1050)
                - addLiquidity(otherHalf,newBalance) (BrownieToken.sol#1424)
        - SwapAndLiquify(half,newBalance,otherHalf) (BrownieToken.sol#1426)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
BrownieToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (BrownieToken.sol#1625-1666) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,BROWNIE::delegateBySig: signature expired) (BrownieToken.sol#1664)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (BrownieToken.sol#336-345) uses assembly
        - INLINE ASM (BrownieToken.sol#343)
Address._verifyCallResult(bool,bytes,string) (BrownieToken.sol#481-498) uses assembly
        - INLINE ASM (BrownieToken.sol#490-493)
BrownieToken.getChainId() (BrownieToken.sol#1784-1788) uses assembly
        - INLINE ASM (BrownieToken.sol#1786)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
BrownieToken._transfer(address,address,uint256) (BrownieToken.sol#1364-1395) compares to a boolean constant:
        -swapAndLiquifyEnabled == true && _inSwapAndLiquify == false && address(pancakeSwapRouter) != address(0) && pancakeSwapPair != ad
dress(0) && sender != pancakeSwapPair && sender != owner() (BrownieToken.sol#1367-1372)
BrownieToken.antiWhale(address,address,uint256) (BrownieToken.sol#1319-1329) compares to a boolean constant:
        -_excludedFromAntiWhale[sender] == false && _excludedFromAntiWhale[recipient] == false (BrownieToken.sol#1322-1323)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address._verifyCallResult(bool,bytes,string) (BrownieToken.sol#481-498) is never used and should be removed
Address.functionCall(address,bytes) (BrownieToken.sol#389-391) is never used and should be removed
Address.functionCall(address,bytes,string) (BrownieToken.sol#399-401) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (BrownieToken.sol#414-416) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (BrownieToken.sol#424-431) is never used and should be removed
Address.functionDelegateCall(address,bytes) (BrownieToken.sol#463-465) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (BrownieToken.sol#473-479) is never used and should be removed
Address.functionStaticCall(address,bytes) (BrownieToken.sol#439-441) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (BrownieToken.sol#449-455) is never used and should be removed
Address.isContract(address) (BrownieToken.sol#336-345) is never used and should be removed
Address.sendValue(address,uint256) (BrownieToken.sol#363-369) is never used and should be removed
BEP20._burn(address,uint256) (BrownieToken.sol#1020-1026) is never used and should be removed
BEP20._burnFrom(address,uint256) (BrownieToken.sol#1059-1066) is never used and should be removed
BrownieToken._transfer(address,address,uint256) (BrownieToken.sol#1364-1395) is never used and should be removed
BrownieToken.addLiquidity(uint256,uint256) (BrownieToken.sol#1450-1463) is never used and should be removed
BrownieToken.swapAndLiquify() (BrownieToken.sol#1398-1428) is never used and should be removed
BrownieToken.swapTokensForEth(uint256) (BrownieToken.sol#1431-1447) is never used and should be removed
Context._msgData() (BrownieToken.sol#629-632) is never used and should be removed
SafeBEP20._callOptionalReturn(IBEP20,bytes) (BrownieToken.sol#581-592) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (BrownieToken.sol#538-552) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (BrownieToken.sol#563-573) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (BrownieToken.sol#554-561) is never used and should be removed
SafeBEP20.safeTransfer(IBEP20,address,uint256) (BrownieToken.sol#514-520) is never used and should be removed
SafeBEP20.safeTransferFrom(IBEP20,address,address,uint256) (BrownieToken.sol#522-529) is never used and should be removed
SafeMath.div(uint256,uint256,string) (BrownieToken.sol#289-292) is never used and should be removed
SafeMath.mod(uint256,uint256) (BrownieToken.sol#251-254) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (BrownieToken.sol#309-312) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (BrownieToken.sol#123-127) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (BrownieToken.sol#159-162) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (BrownieToken.sol#169-172) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (BrownieToken.sol#144-152) is never used and should be removed
SafeMath.trySub(uint256,uint256) (BrownieToken.sol#134-137) 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) (BrownieToken.sol#363-369):
        - (success) = recipient.call{value: amount}() (BrownieToken.sol#367)
```

```
        - (success) = recipient.call{value: amount}() (BrownieToken.sol#367)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (BrownieToken.sol#424-431):
        - (success,returndata) = target.call{value: value}(data) (BrownieToken.sol#429)
Low level call in Address.functionStaticCall(address,bytes,string) (BrownieToken.sol#449-455):
        - (success,returndata) = target.staticcall(data) (BrownieToken.sol#453)
Low level call in Address.functionDelegateCall(address,bytes,string) (BrownieToken.sol#473-479):
        - (success,returndata) = target.delegatecall(data) (BrownieToken.sol#477)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Router01.WETH() (BrownieToken.sol#1072) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (BrownieToken.sol#1222) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (BrownieToken.sol#1223) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (BrownieToken.sol#1240) is not in mixedCase
Parameter BrownieToken.mint(address,uint256)._to (BrownieToken.sol#1358) is not in mixedCase
Parameter BrownieToken.mint(address,uint256)._amount (BrownieToken.sol#1358) is not in mixedCase
Parameter BrownieToken.isExcludedFromAntiWhale(address)._account (BrownieToken.sol#1475) is not in mixedCase
Parameter BrownieToken.updateTransferTaxRate(uint16)._transferTaxRate (BrownieToken.sol#1486) is not in mixedCase
Parameter BrownieToken.updateBurnRate(uint16)._burnRate (BrownieToken.sol#1496) is not in mixedCase
Parameter BrownieToken.updateMaxTransferAmountRate(uint16)._maxTransferAmountRate (BrownieToken.sol#1506) is not in mixedCase
Parameter BrownieToken.updateMinAmountToLiquify(uint256)._minAmount (BrownieToken.sol#1516) is not in mixedCase
Parameter BrownieToken.setExcludedFromAntiWhale(address,bool)._account (BrownieToken.sol#1525) is not in mixedCase
Parameter BrownieToken.setExcludedFromAntiWhale(address,bool)._excluded (BrownieToken.sol#1525) is not in mixedCase
Parameter BrownieToken.updateSwapAndLiquifyEnabled(bool)._enabled (BrownieToken.sol#1533) is not in mixedCase
Parameter BrownieToken.updatePancakeSwapRouter(address)._router (BrownieToken.sol#1542) is not in mixedCase
Variable BrownieToken._delegates (BrownieToken.sol#1567) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (BrownieToken.sol#630)" inContext (BrownieToken.sol#624-633)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (BrownieToken.so
l#1077) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired
 (BrownieToken.sol#1078)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
BrownieToken.slitherConstructorConstantVariables() (BrownieToken.sol#1276-1790) uses literals with too many digits:
        - BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD (BrownieToken.sol#1284)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (BrownieToken.sol#683-686)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (BrownieToken.sol#692-696)
decimals() should be declared external:
        - BEP20.decimals() (BrownieToken.sol#829-831)
symbol() should be declared external:
        - BEP20.symbol() (BrownieToken.sol#836-838)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (BrownieToken.sol#862-865)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (BrownieToken.sol#870-872)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (BrownieToken.sol#881-884)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (BrownieToken.sol#898-910)
increaseAllowance(address,uint256) should be declared external:
        - BEP20.increaseAllowance(address,uint256) (BrownieToken.sol#924-927)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20.decreaseAllowance(address,uint256) (BrownieToken.sol#943-950)
mint(uint256) should be declared external:
        - BEP20.mint(uint256) (BrownieToken.sol#960-963)
mint(address,uint256) should be declared external:
        - BrownieToken.mint(address,uint256) (BrownieToken.sol#1358-1361)
isExcludedFromAntiWhale(address) should be declared external:
        - BrownieToken.isExcludedFromAntiWhale(address) (BrownieToken.sol#1475-1477)
updateTransferTaxRate(uint16) should be declared external:
        - BrownieToken.updateTransferTaxRate(uint16) (BrownieToken.sol#1486-1490)
```

```
updateBurnRate(uint16) should be declared external:
        - BrownieToken.updateBurnRate(uint16) (BrownieToken.sol#1496-1500)
updateMaxTransferAmountRate(uint16) should be declared external:
        - BrownieToken.updateMaxTransferAmountRate(uint16) (BrownieToken.sol#1506-1510)
updateMinAmountToLiquify(uint256) should be declared external:
        - BrownieToken.updateMinAmountToLiquify(uint256) (BrownieToken.sol#1516-1519)
setExcludedFromAntiWhale(address,bool) should be declared external:
        - BrownieToken.setExcludedFromAntiWhale(address,bool) (BrownieToken.sol#1525-1527)
updateSwapAndLiquifyEnabled(bool) should be declared external:
        - BrownieToken.updateSwapAndLiquifyEnabled(bool) (BrownieToken.sol#1533-1536)
updatePancakeSwapRouter(address) should be declared external:
        - BrownieToken.updatePancakeSwapRouter(address) (BrownieToken.sol#1542-1547)
transferOperator(address) should be declared external:
        - BrownieToken.transferOperator(address) (BrownieToken.sol#1560-1564)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:BrownieToken.sol analyzed (14 contracts with 75 detectors), 95 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Multicall.sol

```
INFO:Detectors:
Multicall.aggregate(Multicall.Call[]) (Multicall.sol#20-28) has external calls inside a loop: (success,ret) = calls[i].target.call(calls[
i].callData) (Multicall.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Pragma version>=0.5.0 (Multicall.sol#6) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Multicall.aggregate(Multicall.Call[]) (Multicall.sol#20-28):
        - (success,ret) = calls[i].target.call(calls[i].callData) (Multicall.sol#24)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
aggregate(Multicall.Call[]) should be declared external:
        - Multicall.aggregate(Multicall.Call[]) (Multicall.sol#20-28)
getEthBalance(address) should be declared external:
        - Multicall.getEthBalance(address) (Multicall.sol#31-33)
getBlockHash(uint256) should be declared external:
        - Multicall.getBlockHash(uint256) (Multicall.sol#35-37)
getLastBlockHash() should be declared external:
        - Multicall.getLastBlockHash() (Multicall.sol#39-41)
getCurrentBlockTimestamp() should be declared external:
        - Multicall.getCurrentBlockTimestamp() (Multicall.sol#43-45)
getCurrentBlockDifficulty() should be declared external:
        - Multicall.getCurrentBlockDifficulty() (Multicall.sol#47-49)
getCurrentBlockGasLimit() should be declared external:
        - Multicall.getCurrentBlockGasLimit() (Multicall.sol#51-53)
getCurrentBlockCoinbase() should be declared external:
        - Multicall.getCurrentBlockCoinbase() (Multicall.sol#55-57)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Multicall.sol analyzed (1 contracts with 75 detectors), 11 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> Timelock.sol

```
INFO:Detectors:
Timelock.constructor(address,uint256).admin_ (Timelock.sol#244) lacks a zero-check on :
                - admin = admin_ (Timelock.sol#248)
Timelock.setPendingAdmin(address).pendingAdmin_ (Timelock.sol#273) lacks a zero-check on :
                - pendingAdmin = pendingAdmin_ (Timelock.sol#281)
Timelock.executeTransaction(address,uint256,string,bytes,uint256).target (Timelock.sol#306) lacks a zero-check on :
                - (success,returnData) = target.call.value(value)(callData) (Timelock.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
```

```
INFO:Detectors:
Reentrancy in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#306-331):
        External calls:
        - (success,returnData) = target.call.value(value)(callData) (Timelock.sol#325)
        Event emitted after the call(s):
        - ExecuteTransaction(txHash,target,value,signature,data,eta) (Timelock.sol#328)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Timelock.queueTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#286-295) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(eta >= getBlockTimestamp().add(delay),Timelock::queueTransaction: Estimated execution block must satisfy d
elay.) (Timelock.sol#288)
Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#306-331) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(getBlockTimestamp() >= eta,Timelock::executeTransaction: Transaction hasn't surpassed time lock.) (Timeloc
k.sol#311)
        - require(bool,string)(getBlockTimestamp() <= eta.add(GRACE_PERIOD),Timelock::executeTransaction: Transaction is stale.) (Timeloc
k.sol#312)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.6.0<0.8.0']
        - >=0.6.0<0.8.0 (Timelock.sol#7)
        - 0.6.12 (Timelock.sol#220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
SafeMath.div(uint256,uint256) (Timelock.sol#139-142) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Timelock.sol#194-197) is never used and should be removed
SafeMath.mod(uint256,uint256) (Timelock.sol#156-159) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Timelock.sol#214-217) is never used and should be removed
SafeMath.mul(uint256,uint256) (Timelock.sol#120-125) is never used and should be removed
SafeMath.sub(uint256,uint256) (Timelock.sol#105-108) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Timelock.sol#174-177) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (Timelock.sol#28-32) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (Timelock.sol#64-67) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (Timelock.sol#74-77) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (Timelock.sol#49-57) is never used and should be removed
SafeMath.trySub(uint256,uint256) (Timelock.sol#39-42) 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 (Timelock.sol#7) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#306-331):
        - (success,returnData) = target.call.value(value)(callData) (Timelock.sol#325)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Variable Timelock.admin_initialized (Timelock.sol#239) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
setDelay(uint256) should be declared external:
        - Timelock.setDelay(uint256) (Timelock.sol#256-263)
acceptAdmin() should be declared external:
        - Timelock.acceptAdmin() (Timelock.sol#265-271)
setPendingAdmin(address) should be declared external:
        - Timelock.setPendingAdmin(address) (Timelock.sol#273-284)
queueTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.queueTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#286-295)
cancelTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.cancelTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#297-304)
executeTransaction(address,uint256,string,bytes,uint256) should be declared external:
        - Timelock.executeTransaction(address,uint256,string,bytes,uint256) (Timelock.sol#306-331)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Timelock.sol analyzed (2 contracts with 75 detectors), 28 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Slither log >> MasterChef.sol

```
INFO:Detectors:
BrownieToken.addLiquidity(uint256,uint256) (MasterChef.sol#1467-1480) sends eth to arbitrary user
        Dangerous calls:
        - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterChef.sol#1
472-1479)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in BrownieToken._transfer(address,address,uint256) (MasterChef.sol#1381-1412):
        External calls:
        - swapAndLiquify() (MasterChef.sol#1391)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(MasterChef.sol#1457-1463)
        External calls sending eth:
        - swapAndLiquify() (MasterChef.sol#1391)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        State variables written after the call(s):
        - super._transfer(sender,recipient,amount) (MasterChef.sol#1395)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (MasterChef.sol#999)
                - _balances[recipient] = _balances[recipient].add(amount) (MasterChef.sol#1000)
        - super._transfer(sender,BURN_ADDRESS,burnAmount) (MasterChef.sol#1407)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (MasterChef.sol#999)
                - _balances[recipient] = _balances[recipient].add(amount) (MasterChef.sol#1000)
        - super._transfer(sender,address(this),liquidityAmount) (MasterChef.sol#1408)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (MasterChef.sol#999)
                - _balances[recipient] = _balances[recipient].add(amount) (MasterChef.sol#1000)
        - super._transfer(sender,recipient,sendAmount) (MasterChef.sol#1409)
                - _balances[sender] = _balances[sender].sub(amount,BEP20: transfer amount exceeds balance) (MasterChef.sol#999)
                - _balances[recipient] = _balances[recipient].add(amount) (MasterChef.sol#1000)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
MasterChef.safeBrownieTransfer(address,uint256) (MasterChef.sol#2078-2085) ignores return value by brownie.transfer(_to,brownieBal) (Mast
erChef.sol#2081)
MasterChef.safeBrownieTransfer(address,uint256) (MasterChef.sol#2078-2085) ignores return value by brownie.transfer(_to,_amount) (MasterC
hef.sol#2083)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
BrownieToken._transfer(address,address,uint256) (MasterChef.sol#1381-1412) performs a multiplication on the result of a division:
        -taxAmount = amount.mul(transferTaxRate).div(10000) (MasterChef.sol#1398)
        -burnAmount = taxAmount.mul(burnRate).div(100) (MasterChef.sol#1399)
MasterChef.pendingBrownie(uint256,address) (MasterChef.sol#1942-1954) performs a multiplication on the result of a division:
        -brownieReward = multiplier.mul(brownniePerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#1949)
        -accBrowniePerShare = accBrowniePerShare.add(brownieReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1950)
MasterChef.updatePool(uint256) (MasterChef.sol#1971-1990) performs a multiplication on the result of a division:
        -brownieReward = multiplier.mul(brownniePerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#1982)
        -pool.accBrowniePerShare = pool.accBrowniePerShare.add(brownieReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1988)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
BrownieToken._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1782-1800) uses a dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (MasterChef.sol#1792)
MasterChef.updatePool(uint256) (MasterChef.sol#1971-1990) uses a dangerous strict equality:
        - lpSupply == 0 || pool.allocPoint == 0 (MasterChef.sol#1977)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef.add(uint256,IBEP20,uint16,uint256,bool) (MasterChef.sol#1905-1921):
        External calls:
        - massUpdatePools() (MasterChef.sol#1909)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        State variables written after the call(s):
        - poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP,_harvestInterval)) (MasterChef.sol#1913-1920)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChef.sol#1912)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1993-2017):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1996)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - brownieReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1998)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
```

```
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        State variables written after the call(s):
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (MasterChef.sol#2053)
                - user.rewardLockedUp = 0 (MasterChef.sol#2063)
                - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (MasterChef.sol#2064)
                - user.rewardLockedUp = user.rewardLockedUp.add(pending) (MasterChef.sol#2071)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1993-2017):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1996)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - brownieReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1998)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#2002)
        - pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#2009)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount).sub(depositFee) (MasterChef.sol#2010)
        - user.rewardDebt = user.amount.mul(pool.accBrowniePerShare).div(1e12) (MasterChef.sol#2015)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1993-2017):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1996)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - brownieReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1998)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
```

```
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#2002)
        State variables written after the call(s):
        - user.amount = user.amount.add(_amount) (MasterChef.sol#2012)
Reentrancy in MasterChef.set(uint256,uint256,uint16,uint256,bool) (MasterChef.sol#1924-1934):
        External calls:
        - massUpdatePools() (MasterChef.sol#1928)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        State variables written after the call(s):
        - poolInfo[_pid].allocPoint = _allocPoint (MasterChef.sol#1931)
        - poolInfo[_pid].depositFeeBP = _depositFeeBP (MasterChef.sol#1932)
        - poolInfo[_pid].harvestInterval = _harvestInterval (MasterChef.sol#1933)
        - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (MasterChef.sol#1930)
Reentrancy in MasterChef.updateEmissionRate(uint256) (MasterChef.sol#2109-2113):
        External calls:
        - massUpdatePools() (MasterChef.sol#2110)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        State variables written after the call(s):
        - browniePerBlock = _browniePerBlock (MasterChef.sol#2112)
Reentrancy in MasterChef.updatePool(uint256) (MasterChef.sol#1971-1990):
        External calls:
        - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
        - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
        - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        State variables written after the call(s):
        - pool.accBrowniePerShare = pool.accBrowniePerShare.add(brownieReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1988)
        - pool.lastRewardBlock = block.number (MasterChef.sol#1989)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#2020-2032):
```

```
        External calls:
        - updatePool(_pid) (MasterChef.sol#2024)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        State variables written after the call(s):
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
                - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (MasterChef.sol#2053)
                - user.rewardLockedUp = 0 (MasterChef.sol#2063)
                - user.nextHarvestUntil = block.timestamp.add(pool.harvestInterval) (MasterChef.sol#2064)
                - user.rewardLockedUp = user.rewardLockedUp.add(pending) (MasterChef.sol#2071)
        - user.amount = user.amount.sub(_amount) (MasterChef.sol#2027)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#2020-2032):
        External calls:
        - updatePool(_pid) (MasterChef.sol#2024)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#2028)
        State variables written after the call(s):
        - user.rewardDebt = user.amount.mul(pool.accBrowniePerShare).div(1e12) (MasterChef.sol#2030)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
BrownieToken.addLiquidity(uint256,uint256) (MasterChef.sol#1467-1480) ignores return value by pancakeSwapRouter.addLiquidityETH{value: et
hAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterChef.sol#1472-1479)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
BEP20.constructor(string,string).name (MasterChef.sol#818) shadows:
```

```
BEP20.constructor(string,string).name (MasterChef.sol#818) shadows:
        - BEP20.name() (MasterChef.sol#834-836) (function)
        - IBEP20.name() (MasterChef.sol#28) (function)
BEP20.constructor(string,string).symbol (MasterChef.sol#818) shadows:
        - BEP20.symbol() (MasterChef.sol#848-850) (function)
        - IBEP20.symbol() (MasterChef.sol#23) (function)
BEP20.allowance(address,address).owner (MasterChef.sol#882) shadows:
        - Ownable.owner() (MasterChef.sol#674-676) (function)
BEP20._approve(address,address,uint256).owner (MasterChef.sol#1054) shadows:
        - Ownable.owner() (MasterChef.sol#674-676) (function)
BrownieToken.swapAndLiquify().maxTransferAmount (MasterChef.sol#1417) shadows:
        - BrownieToken.maxTransferAmount() (MasterChef.sol#1485-1487) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
MasterChef.constructor(BrownieToken,uint256,uint256,address,address,address)._devaddr (MasterChef.sol#1886) lacks a zero-check on :
                - devAddress = _devaddr (MasterChef.sol#1894)
MasterChef.constructor(BrownieToken,uint256,uint256,address,address,address)._feeAddr (MasterChef.sol#1887) lacks a zero-check on :
                - feeAddress = _feeAddr (MasterChef.sol#1895)
MasterChef.constructor(BrownieToken,uint256,uint256,address,address,address)._brownieVaultAddr (MasterChef.sol#1888) lacks a zero-check o
n :
                - brownieVaultAddress = _brownieVaultAddr (MasterChef.sol#1896)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in BrownieToken.swapAndLiquify() (MasterChef.sol#1415-1445):
        External calls:
        - swapTokensForEth(half) (MasterChef.sol#1435)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(MasterChef.sol#1457-1463)
        - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        State variables written after the call(s):
        - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
                - _allowances[owner][spender] = amount (MasterChef.sol#1061)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in BrownieToken._transfer(address,address,uint256) (MasterChef.sol#1381-1412):
        External calls:
        - swapAndLiquify() (MasterChef.sol#1391)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(MasterChef.sol#1457-1463)
        External calls sending eth:
        - swapAndLiquify() (MasterChef.sol#1391)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        Event emitted after the call(s):
        - Transfer(sender,recipient,amount) (MasterChef.sol#1001)
                - super._transfer(sender,address(this),liquidityAmount) (MasterChef.sol#1408)
        - Transfer(sender,recipient,amount) (MasterChef.sol#1001)
                - super._transfer(sender,BURN_ADDRESS,burnAmount) (MasterChef.sol#1407)
        - Transfer(sender,recipient,amount) (MasterChef.sol#1001)
                - super._transfer(sender,recipient,sendAmount) (MasterChef.sol#1409)
        - Transfer(sender,recipient,amount) (MasterChef.sol#1001)
                - super._transfer(sender,recipient,amount) (MasterChef.sol#1395)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1993-2017):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1996)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - brownieReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1998)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
```

```
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        Event emitted after the call(s):
        - ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#2135)
                - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
        - RewardLockedUp(msg.sender,_pid,pending) (MasterChef.sol#2073)
                - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1993-2017):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1996)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - brownieReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1998)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2000)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        - pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#2002)
        - pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#2009)
        Event emitted after the call(s):
        - Deposit(msg.sender,_pid,_amount) (MasterChef.sol#2016)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#2035-2045):
        External calls:
        - pool.lpToken.safeTransfer(address(msg.sender),amount) (MasterChef.sol#2043)
        Event emitted after the call(s):
        - EmergencyWithdraw(msg.sender,_pid,amount) (MasterChef.sol#2044)
Reentrancy in MasterChef.payOrLockupPendingBrownie(uint256) (MasterChef.sol#2048-2075):
        External calls:
        - safeBrownieTransfer(msg.sender,totalRewards) (MasterChef.sol#2067)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
        - payReferralCommission(msg.sender,totalRewards) (MasterChef.sol#2068)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        Event emitted after the call(s):
        - ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#2135)
                - payReferralCommission(msg.sender,totalRewards) (MasterChef.sol#2068)
```

```
Reentrancy in MasterChef.payReferralCommission(address,uint256) (MasterChef.sol#2127-2138):
        External calls:
        - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
        - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        Event emitted after the call(s):
        - ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#2135)
Reentrancy in BrownieToken.swapAndLiquify() (MasterChef.sol#1415-1445):
        External calls:
        - swapTokensForEth(half) (MasterChef.sol#1435)
                - pancakeSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(MasterChef.sol#1457-1463)
        - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        External calls sending eth:
        - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
                - pancakeSwapRouter.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,operator(),block.timestamp) (MasterCh
ef.sol#1472-1479)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (MasterChef.sol#1062)
                - addLiquidity(otherHalf,newBalance) (MasterChef.sol#1441)
        - SwapAndLiquify(half,newBalance,otherHalf) (MasterChef.sol#1443)
Reentrancy in MasterChef.updateEmissionRate(uint256) (MasterChef.sol#2109-2113):
        External calls:
        - massUpdatePools() (MasterChef.sol#2110)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        Event emitted after the call(s):
        - EmissionRateUpdated(msg.sender,browniePerBlock,_browniePerBlock) (MasterChef.sol#2111)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#2020-2032):
        External calls:
        - updatePool(_pid) (MasterChef.sol#2024)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
```

```
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        Event emitted after the call(s):
        - ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#2135)
                - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
        - RewardLockedUp(msg.sender,_pid,pending) (MasterChef.sol#2073)
                - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#2020-2032):
        External calls:
        - updatePool(_pid) (MasterChef.sol#2024)
                - brownie.mint(address(this),brownieReward) (MasterChef.sol#1983)
                - brownie.mint(brownieVaultAddress,devReward.div(10)) (MasterChef.sol#1986)
                - brownie.mint(devAddress,devReward.sub(devReward.div(10))) (MasterChef.sol#1987)
        - payOrLockupPendingBrownie(_pid) (MasterChef.sol#2025)
                - brownie.transfer(_to,brownieBal) (MasterChef.sol#2081)
                - brownie.transfer(_to,_amount) (MasterChef.sol#2083)
                - brownie.mint(referrer,commissionAmount) (MasterChef.sol#2133)
                - brownieReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#2134)
        - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#2028)
        Event emitted after the call(s):
        - Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#2031)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
BrownieToken.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1648-1689) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(now <= expiry,BROWNIE::delegateBySig: signature expired) (MasterChef.sol#1687)
MasterChef.canHarvest(uint256,address) (MasterChef.sol#1957-1960) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp >= user.nextHarvestUntil (MasterChef.sol#1959)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Address.isContract(address) (MasterChef.sol#340-349) uses assembly
        - INLINE ASM (MasterChef.sol#347)
Address._verifyCallResult(bool,bytes,string) (MasterChef.sol#485-502) uses assembly
        - INLINE ASM (MasterChef.sol#494-497)
BrownieToken.getChainId() (MasterChef.sol#1807-1811) uses assembly
        - INLINE ASM (MasterChef.sol#1809)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
BrownieToken._transfer(address,address,uint256) (MasterChef.sol#1381-1412) compares to a boolean constant:
        -swapAndLiquifyEnabled == true && _inSwapAndLiquify == false && address(pancakeSwapRouter) != address(0) && pancakeSwapPair != ad
dress(0) && sender != pancakeSwapPair && sender != owner() (MasterChef.sol#1384-1389)
BrownieToken.antiWhale(address,address,uint256) (MasterChef.sol#1336-1346) compares to a boolean constant:
        -_excludedFromAntiWhale[sender] == false && _excludedFromAntiWhale[recipient] == false (MasterChef.sol#1339-1340)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.6.12', '>=0.4.0', '>=0.5.0', '>=0.6.0<0.8.0', '>=0.6.2', '>=0.6.2<0.8.0', '^0.6.0', '^0.6.12']
        - >=0.4.0 (MasterChef.sol#7)
        - >=0.6.0<0.8.0 (MasterChef.sol#104)
        - >=0.6.2<0.8.0 (MasterChef.sol#317)
        - ^0.6.0 (MasterChef.sol#505)
        - 0.6.12 (MasterChef.sol#601)
        - >=0.6.0<0.8.0 (MasterChef.sol#620)
        - >=0.6.0<0.8.0 (MasterChef.sol#643)
        - >=0.6.0<0.8.0 (MasterChef.sol#709)
        - >=0.4.0 (MasterChef.sol#770)
        - >=0.6.2 (MasterChef.sol#1081)
        - >=0.6.2 (MasterChef.sol#1177)
        - >=0.5.0 (MasterChef.sol#1220)
        - >=0.5.0 (MasterChef.sol#1273)
        - ^0.6.12 (MasterChef.sol#1291)
        - 0.6.12 (MasterChef.sol#1814)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (MasterChef.sol#393-395) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MasterChef.sol#418-420) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MasterChef.sol#467-469) is never used and should be removed
```

```
Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#477-483) is never used and should be removed
Address.functionStaticCall(address,bytes) (MasterChef.sol#443-445) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MasterChef.sol#453-459) is never used and should be removed
Address.sendValue(address,uint256) (MasterChef.sol#367-373) is never used and should be removed
BEP20._burn(address,uint256) (MasterChef.sol#1032-1038) is never used and should be removed
BEP20._burnFrom(address,uint256) (MasterChef.sol#1071-1078) is never used and should be removed
BrownieToken._transfer(address,address,uint256) (MasterChef.sol#1381-1412) is never used and should be removed
BrownieToken.addLiquidity(uint256,uint256) (MasterChef.sol#1467-1480) is never used and should be removed
BrownieToken.swapAndLiquify() (MasterChef.sol#1415-1445) is never used and should be removed
BrownieToken.swapTokensForEth(uint256) (MasterChef.sol#1448-1464) is never used and should be removed
Context._msgData() (MasterChef.sol#637-640) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (MasterChef.sol#543-557) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#568-578) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#559-566) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MasterChef.sol#291-294) is never used and should be removed
SafeMath.mod(uint256,uint256) (MasterChef.sol#253-256) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MasterChef.sol#311-314) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MasterChef.sol#125-129) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MasterChef.sol#161-164) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MasterChef.sol#171-174) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MasterChef.sol#146-154) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MasterChef.sol#136-139) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.4.0 (MasterChef.sol#7) allows old versions
Pragma version>=0.6.0<0.8.0 (MasterChef.sol#104) is too complex
Pragma version>=0.6.2<0.8.0 (MasterChef.sol#317) is too complex
Pragma version^0.6.0 (MasterChef.sol#505) allows old versions
Pragma version>=0.6.0<0.8.0 (MasterChef.sol#620) is too complex
Pragma version>=0.6.0<0.8.0 (MasterChef.sol#643) is too complex
Pragma version>=0.6.0<0.8.0 (MasterChef.sol#709) is too complex
Pragma version>=0.4.0 (MasterChef.sol#770) allows old versions
Pragma version>=0.6.2 (MasterChef.sol#1081) allows old versions
Pragma version>=0.6.2 (MasterChef.sol#1177) allows old versions
Pragma version>=0.5.0 (MasterChef.sol#1220) allows old versions
Pragma version>=0.5.0 (MasterChef.sol#1273) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
```

```
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (MasterChef.sol#367-373):
        - (success) = recipient.call{value: amount}() (MasterChef.sol#371)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#428-435):
        - (success,returndata) = target.call{value: value}(data) (MasterChef.sol#433)
Low level call in Address.functionStaticCall(address,bytes,string) (MasterChef.sol#453-459):
        - (success,returndata) = target.staticcall(data) (MasterChef.sol#457)
Low level call in Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#477-483):
        - (success,returndata) = target.delegatecall(data) (MasterChef.sol#481)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IUniswapV2Router01.WETH() (MasterChef.sol#1085) is not in mixedCase
Function IUniswapV2Pair.DOMAIN_SEPARATOR() (MasterChef.sol#1237) is not in mixedCase
Function IUniswapV2Pair.PERMIT_TYPEHASH() (MasterChef.sol#1238) is not in mixedCase
Function IUniswapV2Pair.MINIMUM_LIQUIDITY() (MasterChef.sol#1255) is not in mixedCase
Parameter BrownieToken.mint(address,uint256)._to (MasterChef.sol#1375) is not in mixedCase
Parameter BrownieToken.mint(address,uint256)._amount (MasterChef.sol#1375) is not in mixedCase
Parameter BrownieToken.isExcludedFromAntiWhale(address)._account (MasterChef.sol#1492) is not in mixedCase
Parameter BrownieToken.updateTransferTaxRate(uint16)._transferTaxRate (MasterChef.sol#1503) is not in mixedCase
Parameter BrownieToken.updateBurnRate(uint16)._burnRate (MasterChef.sol#1513) is not in mixedCase
Parameter BrownieToken.updateMaxTransferAmountRate(uint16)._maxTransferAmountRate (MasterChef.sol#1523) is not in mixedCase
Parameter BrownieToken.updateMinAmountToLiquify(uint256)._minAmount (MasterChef.sol#1533) is not in mixedCase
Parameter BrownieToken.setExcludedFromAntiWhale(address,bool)._account (MasterChef.sol#1542) is not in mixedCase
Parameter BrownieToken.setExcludedFromAntiWhale(address,bool)._excluded (MasterChef.sol#1542) is not in mixedCase
Parameter BrownieToken.updateSwapAndLiquifyEnabled(bool)._enabled (MasterChef.sol#1550) is not in mixedCase
Parameter BrownieToken.updatePancakeSwapRouter(address)._router (MasterChef.sol#1559) is not in mixedCase
Variable BrownieToken._delegates (MasterChef.sol#1590) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,uint256,bool)._allocPoint (MasterChef.sol#1905) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,uint256,bool)._lpToken (MasterChef.sol#1905) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,uint256,bool)._depositFeeBP (MasterChef.sol#1905) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,uint256,bool)._harvestInterval (MasterChef.sol#1905) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,uint256,bool)._withUpdate (MasterChef.sol#1905) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._pid (MasterChef.sol#1924) is not in mixedCase
```

```
Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._depositFeeBP (MasterChef.sol#1924) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._harvestInterval (MasterChef.sol#1924) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,uint256,bool)._withUpdate (MasterChef.sol#1924) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (MasterChef.sol#1937) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (MasterChef.sol#1937) is not in mixedCase
Parameter MasterChef.pendingBrownie(uint256,address)._pid (MasterChef.sol#1942) is not in mixedCase
Parameter MasterChef.pendingBrownie(uint256,address)._user (MasterChef.sol#1942) is not in mixedCase
Parameter MasterChef.canHarvest(uint256,address)._pid (MasterChef.sol#1957) is not in mixedCase
Parameter MasterChef.canHarvest(uint256,address)._user (MasterChef.sol#1957) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (MasterChef.sol#1971) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._pid (MasterChef.sol#1993) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._amount (MasterChef.sol#1993) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._referrer (MasterChef.sol#1993) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (MasterChef.sol#2020) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (MasterChef.sol#2020) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (MasterChef.sol#2035) is not in mixedCase
Parameter MasterChef.payOrLockupPendingBrownie(uint256)._pid (MasterChef.sol#2048) is not in mixedCase
Parameter MasterChef.safeBrownieTransfer(address,uint256)._to (MasterChef.sol#2078) is not in mixedCase
Parameter MasterChef.safeBrownieTransfer(address,uint256)._amount (MasterChef.sol#2078) is not in mixedCase
Parameter MasterChef.setDevAddress(address)._devAddress (MasterChef.sol#2088) is not in mixedCase
Parameter MasterChef.setFeeAddress(address)._feeAddress (MasterChef.sol#2095) is not in mixedCase
Parameter MasterChef.setCommunityVault(address)._brownieVaultAddr (MasterChef.sol#2102) is not in mixedCase
Parameter MasterChef.updateEmissionRate(uint256)._browniePerBlock (MasterChef.sol#2109) is not in mixedCase
Parameter MasterChef.setBrownieReferral(IBrownieReferral)._brownieReferral (MasterChef.sol#2116) is not in mixedCase
Parameter MasterChef.setReferralCommissionRate(uint16)._referralCommissionRate (MasterChef.sol#2121) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._user (MasterChef.sol#2127) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._pending (MasterChef.sol#2127) 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#638)" inContext (MasterChef.sol#632-641)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountADesired (MasterChef.sol#
1090) is too similar to IUniswapV2Router01.addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256).amountBDesired (
MasterChef.sol#1091)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
BrownieToken.slitherConstructorConstantVariables() (MasterChef.sol#1293-1812) uses literals with too many digits:
```

```
        - BURN_ADDRESS = 0x000000000000000000000000000000000000dEaD (MasterChef.sol#1301)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (MasterChef.sol#693-696)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (MasterChef.sol#702-706)
decimals() should be declared external:
        - BEP20.decimals() (MasterChef.sol#841-843)
symbol() should be declared external:
        - BEP20.symbol() (MasterChef.sol#848-850)
transfer(address,uint256) should be declared external:
        - BEP20.transfer(address,uint256) (MasterChef.sol#874-877)
allowance(address,address) should be declared external:
        - BEP20.allowance(address,address) (MasterChef.sol#882-884)
approve(address,uint256) should be declared external:
        - BEP20.approve(address,uint256) (MasterChef.sol#893-896)
transferFrom(address,address,uint256) should be declared external:
        - BEP20.transferFrom(address,address,uint256) (MasterChef.sol#910-922)
increaseAllowance(address,uint256) should be declared external:
        - BEP20.increaseAllowance(address,uint256) (MasterChef.sol#936-939)
decreaseAllowance(address,uint256) should be declared external:
        - BEP20.decreaseAllowance(address,uint256) (MasterChef.sol#955-962)
mint(uint256) should be declared external:
        - BEP20.mint(uint256) (MasterChef.sol#972-975)
mint(address,uint256) should be declared external:
        - BrownieToken.mint(address,uint256) (MasterChef.sol#1375-1378)
isExcludedFromAntiWhale(address) should be declared external:
        - BrownieToken.isExcludedFromAntiWhale(address) (MasterChef.sol#1492-1494)
updateTransferTaxRate(uint16) should be declared external:
        - BrownieToken.updateTransferTaxRate(uint16) (MasterChef.sol#1503-1507)
updateBurnRate(uint16) should be declared external:
        - BrownieToken.updateBurnRate(uint16) (MasterChef.sol#1513-1517)
updateMaxTransferAmountRate(uint16) should be declared external:
        - BrownieToken.updateMaxTransferAmountRate(uint16) (MasterChef.sol#1523-1527)
updateMinAmountToLiquify(uint256) should be declared external:
        - BrownieToken.updateMinAmountToLiquify(uint256) (MasterChef.sol#1533-1536)
setExcludedFromAntiWhale(address,bool) should be declared external:
```

```
setExcludedFromAntiWhale(address,bool) should be declared external:
        - BrownieToken.setExcludedFromAntiWhale(address,bool) (MasterChef.sol#1542-1544)
updateSwapAndLiquifyEnabled(bool) should be declared external:
        - BrownieToken.updateSwapAndLiquifyEnabled(bool) (MasterChef.sol#1550-1553)
updatePancakeSwapRouter(address) should be declared external:
        - BrownieToken.updatePancakeSwapRouter(address) (MasterChef.sol#1559-1564)
transferOperator(address) should be declared external:
        - BrownieToken.transferOperator(address) (MasterChef.sol#1577-1581)
add(uint256,IBEP20,uint16,uint256,bool) should be declared external:
        - MasterChef.add(uint256,IBEP20,uint16,uint256,bool) (MasterChef.sol#1905-1921)
set(uint256,uint256,uint16,uint256,bool) should be declared external:
        - MasterChef.set(uint256,uint256,uint16,uint256,bool) (MasterChef.sol#1924-1934)
withdraw(uint256,uint256) should be declared external:
        - MasterChef.withdraw(uint256,uint256) (MasterChef.sol#2020-2032)
emergencyWithdraw(uint256) should be declared external:
        - MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#2035-2045)
setDevAddress(address) should be declared external:
        - MasterChef.setDevAddress(address) (MasterChef.sol#2088-2092)
setFeeAddress(address) should be declared external:
        - MasterChef.setFeeAddress(address) (MasterChef.sol#2095-2099)
setCommunityVault(address) should be declared external:
        - MasterChef.setCommunityVault(address) (MasterChef.sol#2102-2106)
updateEmissionRate(uint256) should be declared external:
        - MasterChef.updateEmissionRate(uint256) (MasterChef.sol#2109-2113)
setBrownieReferral(IBrownieReferral) should be declared external:
        - MasterChef.setBrownieReferral(IBrownieReferral) (MasterChef.sol#2116-2118)
setReferralCommissionRate(uint16) should be declared external:
        - MasterChef.setReferralCommissionRate(uint16) (MasterChef.sol#2121-2124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MasterChef.sol analyzed (15 contracts with 75 detectors), 171 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
root@server:/chetan/gaza/mycontracts#
```

Solidity Static Analysis

BrownieToken.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: Cannot convert undefined or null to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: Cannot convert undefined or null to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: Cannot convert undefined or null to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: Cannot convert undefined or null to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: Cannot convert undefined or null to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: Cannot convert undefined or null to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: Cannot convert undefined or null to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: Cannot convert undefined or null to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: Cannot convert undefined or null to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: Cannot convert undefined or null to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: Cannot convert undefined or null to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: Cannot convert undefined or null to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: Cannot convert undefined or null to object
Pos: not available

No return:

INTERNAL ERROR in module No return: Cannot convert undefined or null to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: Cannot convert undefined or null to object
Pos: not available

String length:

INTERNAL ERROR in module String length: Cannot convert undefined or null to object
Pos: not available

Multicall.sol

Security

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.
more
Pos: 44:20:

Low level calls:

Use of "call": should be avoided whenever possible.
It can lead to unexpected behavior if return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.
more
Pos: 24:47:

Block hash:

Use of "blockhash": "blockhash(uint blockNumber)" is used to access the last 256 block hashes.
A miner computes the block hash by "summing up" the information in the current block mined.
By "summing up" the information cleverly, a miner can try to influence the outcome of a transaction in the current block.
This is especially easy if there are only a small number of equally likely outcomes.
Pos: 36:20:

Block hash:

Use of "blockhash": "blockhash(uint blockNumber)" is used to access the last 256 block hashes.
A miner computes the block hash by "summing up" the information in the current block mined.
By "summing up" the information cleverly, a miner can try to influence the outcome of a transaction in the current block.
This is especially easy if there are only a small number of equally likely outcomes.
Pos: 40:20:

Gas & Economy

Gas costs:

Gas requirement of function Multicall.aggregate 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: 20:4:

Gas costs:

Gas requirement of function Multicall.getLastBlockHash 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: 39:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.
Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 23:8:

Miscellaneous

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 25:12:

Timelock.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: Cannot convert undefined or null to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: Cannot convert undefined or null to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: Cannot convert undefined or null to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: Cannot convert undefined or null to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: Cannot convert undefined or null to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: Cannot convert undefined or null to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: Cannot convert undefined or null to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: Cannot convert undefined or null to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: Cannot convert undefined or null to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: Cannot convert undefined or null to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: Cannot convert undefined or null to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: Cannot convert undefined or null to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: Cannot convert undefined or null to object
Pos: not available

No return:

INTERNAL ERROR in module No return: Cannot convert undefined or null to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: Cannot convert undefined or null to object
Pos: not available

String length:

INTERNAL ERROR in module String length: Cannot convert undefined or null to object
Pos: not available

MasterChef.sol

Security

Transaction origin:

INTERNAL ERROR in module Transaction origin: Cannot convert undefined or null to object
Pos: not available

Check-effects-interaction:

INTERNAL ERROR in module Check-effects-interaction: Cannot convert undefined or null to object
Pos: not available

Inline assembly:

INTERNAL ERROR in module Inline assembly: Cannot convert undefined or null to object
Pos: not available

Block timestamp:

INTERNAL ERROR in module Block timestamp: Cannot convert undefined or null to object
Pos: not available

Low level calls:

INTERNAL ERROR in module Low level calls: Cannot convert undefined or null to object
Pos: not available

Selfdestruct:

INTERNAL ERROR in module Selfdestruct: Cannot convert undefined or null to object
Pos: not available

Gas & Economy

This on local calls:

INTERNAL ERROR in module This on local calls: Cannot convert undefined or null to object
Pos: not available

Delete dynamic array:

INTERNAL ERROR in module Delete dynamic array: Cannot convert undefined or null to object
Pos: not available

For loop over dynamic array:

INTERNAL ERROR in module For loop over dynamic array: Cannot convert undefined or null to object
Pos: not available

Ether transfer in loop:

INTERNAL ERROR in module Ether transfer in loop: Cannot convert undefined or null to object
Pos: not available

ERC

ERC20:

INTERNAL ERROR in module ERC20: Cannot convert undefined or null to object
Pos: not available

Miscellaneous

Constant/View/Pure functions:

INTERNAL ERROR in module Constant/View/Pure functions: Cannot convert undefined or null to object
Pos: not available

Similar variable names:

INTERNAL ERROR in module Similar variable names: Cannot convert undefined or null to object
Pos: not available

No return:

INTERNAL ERROR in module No return: Cannot convert undefined or null to object
Pos: not available

Guard conditions:

INTERNAL ERROR in module Guard conditions: Cannot convert undefined or null to object
Pos: not available

String length:

INTERNAL ERROR in module String length: Cannot convert undefined or null to object
Pos: not available

Solhint Linter

BrownieToken.sol

```
BrownieToken.sol:2:1: Error: Compiler version must be declared
BrownieToken.sol:970:5: Error: Function name must be in mixedCase
BrownieToken.sol:1116:5: Error: Function name must be in mixedCase
BrownieToken.sol:1117:5: Error: Function name must be in mixedCase
BrownieToken.sol:1134:5: Error: Function name must be in mixedCase
BrownieToken.sol:1335:13: Error: Avoid to make time-based decisions in
your business logic
BrownieToken.sol:1351:13: Error: Avoid to make time-based decisions in
your business logic
BrownieToken.sol:1370:32: Error: Code contains empty blocks
BrownieToken.sol:1554:17: Error: Avoid to make time-based decisions in
your business logic
BrownieToken.sol:1676:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
```

Multicall.sol

```
Multicall.sol:6:1: Error: Compiler version >=0.5.0 does not satisfy the
r semver requirement
Multicall.sol:24:48: Error: Avoid using low level calls.
Multicall.sol:44:21: Error: Avoid to make time-based decisions in your
business logic
```

Timelock.sol

```
Timelock.sol:7:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
Timelock.sol:220:1: Error: Compiler version 0.6.12 does not satisfy the
r semver requirement
Timelock.sol:239:17: Error: Variable name must be in mixedCase
Timelock.sol:254:32: Error: Code contains empty blocks
Timelock.sol:325:51: Error: Avoid to use ".call.value()()"
Timelock.sol:325:51: Error: Avoid using low level calls.
Timelock.sol:335:16: Error: Avoid to make time-based decisions in your
business logic
```

MasterChef.sol

```
MasterChef.sol:7:1: Error: Compiler version >=0.4.0 does not satisfy
the r semver requirement
MasterChef.sol:104:1: Error: Compiler version >=0.6.0 <0.8.0 does not
satisfy the r semver requirement
MasterChef.sol:317:1: Error: Compiler version >=0.6.2 <0.8.0 does not
satisfy the r semver requirement
MasterChef.sol:505:1: Error: Compiler version ^0.6.0 does not satisfy
the r semver requirement
MasterChef.sol:601:1: Error: Compiler version 0.6.12 does not satisfy
the r semver requirement
MasterChef.sol:1081:1: Error: Compiler version >=0.6.2 does not satisfy
the r semver requirement
MasterChef.sol:1085:5: Error: Function name must be in mixedCase
MasterChef.sol:1177:1: Error: Compiler version >=0.6.2 does not satisfy
the r semver requirement
MasterChef.sol:1220:1: Error: Compiler version >=0.5.0 does not satisfy
the r semver requirement
MasterChef.sol:1237:5: Error: Function name must be in mixedCase
MasterChef.sol:1238:5: Error: Function name must be in mixedCase
MasterChef.sol:1255:5: Error: Function name must be in mixedCase
MasterChef.sol:1273:1: Error: Compiler version >=0.5.0 does not satisfy
the r semver requirement
MasterChef.sol:1291:1: Error: Compiler version ^0.6.12 does not satisfy
the r semver requirement
MasterChef.sol:1462:13: Error: Avoid to make time-based decisions in
your business logic
MasterChef.sol:1809:9: Error: Avoid using inline assembly. It is
acceptable only in rare cases
MasterChef.sol:1814:1: Error: Compiler version 0.6.12 does not satisfy
the r semver requirement
MasterChef.sol:1959:16: Error: Avoid to make time-based decisions in
your business logic
MasterChef.sol:2053:37: Error: Avoid to make time-based decisions in
your business logic
MasterChef.sol:2064:41: Error: Avoid to make time-based decisions in
your business logic
```

Software analysis result:

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

Email: audit@EtherAuthority.io