Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Privacy Swap
Website:	https://privacyswap.finance
Platform:	Binance Smart Chain
Language:	Solidity
Date:	June 22nd, 2021

Table of contents

Email: audit@EtherAuthority.io

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

We were contracted by the Privacy Swap team to perform the Security audit of the Privacy Swap protocol smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on June 22nd, 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

PrivacySwap is a yield farming ecosystem running on the Binance Smart Chain (BSC). It is the brainchild of a group of cybersecurity professionals who understands the merit of cryptocurrency, blockchain, and cybersecurity.

Audit scope

Name	Code Review and Security Analysis Report for Privacy Swap protocol Smart Contracts
Platform	**BSC / Solidity**
File 1	PrvReferral.sol
Smart Contract Online Code	https://testnet.bscscan.com/address/0x68255509b303e3440670e6eebd07f653cfff2265#code
File 1 MD5 Hash	2CCBDB832F5D1AB01E0DEC1B27895EA5
File 2	PrivacySwap.sol
Smart Contract Online Code	https://testnet.bscscan.com/address/0xa5d0b52ec985951eaacb5c13993a375803051ac1#code
File 2 MD5 Hash	90740AF0BD22CBEE40A71525C74CAB7A
File 3	MasterChef.sol
Smart Contract Online Code	https://testnet.bscscan.com/address/0x3e962b1B141Cb51cc432CdD0F88Ada8398ac7867#code
File 3 MD5 Hash	B1347B7B64272F7C10D1E7A4F8308BC2
Audit Date	June 22nd, 2021

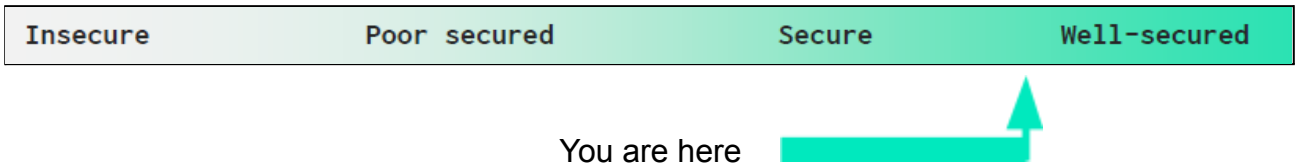
Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1: PrvReferral.sol • The owner can update the status of the operator and also the owner can drain tokens that are sent here by mistake. • The Operator can store ReferralCommission and Referral records.	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**
File 2: PrivacySwap.sol Symbol: PRV Decimals: 18 MaxSupply: 30000000 • mint: Owner can create `_amount` token to `_to`.	**YES, This is valid. Owner must be a MasterChef smart contract.**
File 3: MasterChef.sol • Max harvest interval: 14 days. • default1stComission: 1% • default2ndComission: 0.5% • The Owner can add a new LP to the pool and also Update the given pool's PRV allocation point and deposit fee. • The Owner can update the prv referral contract address. • Pancake has to add hidden dummy pools in order to alter the emission, here Owner can make it simple and transparent to all. • The Owner can set	**YES, This is valid. The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

default1stComission and default2ndComission. ● The Owner can check Initialized or not. ● setCustomReferralComission: The Operator can set CustomReferralComission for the address. ● enableCustomDepositReferral: The Operator can enable CustomDepositReferral for the address.	

Audit Summary

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



We used various tools like MythX, Slither and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 2 low and some very low level issues.

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	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Moderated
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

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

The libraries in the Privacy 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 Privacy Swap Protocol.

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

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

Documentation

We were given Privacy Swap protocol 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 difficult to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Another source of information was its official website https://privacyswap.finance/ which provided rich information about the project architecture and tokenomics.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects. And their core code blocks are written well.

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

AS-IS overview

Privacy Swap protocols are smart contracts, having functionality like swap and liquidity, pool, etc.

PrvReferral.sol

(1) Interface

(a) IPrvReferral

(2) Inherited contracts

(a) Context

(b) Ownable

(3) Usages

(a) using SafeBEP20 for IBEP20;

(4) Events

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

(b) event ReferralRecorded(address indexed user, address indexed referrer);

(c) event ReferralCommissionRecorded(address indexed referrer, uint256 commission);

(d) event OperatorUpdated(address indexed operator, bool indexed status);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	onlyOperator	modifier	Passed	No Issue
2	recordReferral	write	access only Operator	No Issue
3	recordReferralCommission	write	access only Operator	No Issue
4	getReferrer	read	Passed	No Issue
5	updateOperator	write	access only Owner	No Issue

6	drainBEP20Token	write	Owner can drain tokens	Refer Audit Findings
7	owner	read	Passed	No Issue
8	onlyOwner	modifier	Passed	No Issue
9	renounceOwnership	write	access only Owner	No Issue
10	transferOwnership	write	access only Owner	No Issue

PrivacySwap.sol

(1) Interface

(a) IBEP20

(2) Inherited contracts

(a) Context

(b) Ownable

(c) IBEP20

(d) BEP20

(3) Usages

(a) using SafeMath for uint256;

(4) Struct

(a) Checkpoint

(5) Events

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

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

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	mint	write	Owner can mint unlimited tokens	Owner must be MasterChef
2	delegates	read	Passed	No Issue
3	delegate	external	Passed	No Issue
4	delegateBySig	external	Passed	No Issue
5	getCurrentVotes	external	Passed	No Issue
6	getPriorVotes	external	Passed	No Issue
7	_delegate	internal	Passed	No Issue
8	_moveDelegates	internal	Passed	No Issue
9	_writeCheckpoint	internal	Passed	No Issue
10	safe32	internal	Passed	No Issue
11	getChainId	internal	Passed	No Issue

MasterChef.sol

(1) Interface

(a) IBEP20

(b) IMasterChef

(2) Inherited contracts

(a) Context

(b) Ownable

(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);

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

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	poolLength	external	Passed	No Issue
2	setReferalComissions	write	access only Owner	No Issue
3	initMasterChef	write	access only Owner	No Issue
4	add	write	access only Owner	No Issue
5	set	write	access only Owner	No Issue
6	massUpdatePools	write	Infinite loop possibility	Refer Audit Findings
7	getMultiplier	write	Passed	No Issue
8	pendingPrv	external	Passed	No Issue
9	updatePool	write	Passed	No Issue
10	deposit	write	Passed	No Issue
11	withdraw	write	Passed	No Issue
12	emergencyWithdraw	write	Passed	No Issue
13	payOrLockupPendingPrv	internal	Passed	No Issue
14	safePrvTransfer	internal	Passed	No Issue
15	setDevAddress	write	Passed	No Issue
16	setFeeAddress	write	Passed	No Issue
17	updateEmissionRate	write	access only Owner	No Issue
18	setPrvReferral	write	access only Owner	No Issue
19	changeOperator	write	Passed	No Issue
20	setCustomReferralComission	write	Passed	No Issue
21	enableCustomDepositReferral	write	Passed	No Issue
22	payComissionOnDeposit	internal	Passed	No Issue
23	payReferralCommission	internal	Passed	No Issue

Severity Definitions

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

Audit Findings

PrvRefferal.sol

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Owner can drain tokens:

```
// Owner can drain tokens that are sent here by mistake
function drainBEP20Token(IBEP20 _token, uint256 _amount, address _to) external onlyOwner {
    _token.safeTransfer(_to, _amount);
}
```

The owner can drain tokens from the contract. If this is part of the plan, then it's ok.

Resolution: This issue is acknowledged.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.6.12;
```

```
pragma solidity ^0.6.0;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: Please use 0.8.5 which is the latest version.

PrivacySwap.sol

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

No Low severity vulnerabilities were found.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity >=0.4.0;
```

```
pragma solidity ^0.6.12;
```

```
pragma solidity ^0.6.0;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: Please use 0.8.5 which is the latest version.

(2) Only owner can mint token:

```
function mint(uint256 amount) public onlyOwner returns (bool) {
    _mint(_msgSender(), amount);
    return true;
}
```

Only the owner can mint new tokens, users can't mint tokens.

Resolution: Please make sure the owner of this smart contract is MasterChef smart contract.

(3) Use SPDX License Identifier:

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.

Resolution: "Use ""// SPDX-License-Identifier: MIT License"" for non-open-source code. Please see https://spdx.org for more information."

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) 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);
    }
}
```

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

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.6.12;
```

```
pragma solidity ^0.6.0;
```

```
pragma solidity >=0.4.0;
```

Use the latest solidity version while contract deployment to prevent any compiler version level bugs.
Resolution: Please use 0.8.5 which is the latest version.

(2) Multiple SPDX license identifiers found in the source file:
Resolution: Use only 1 SPDX license identifier.

Centralization

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

- updateOperator: PreReferral wallet owner can Update the status of the operator.
- drainBEP20Token: PreReferral wallet owner can drain tokens that are sent here by mistake.
- setReferalComissions: MasterChef wallet owner can set default1stComission and default2ndComission.
- initMasterChef: MasterChef can check Already Initialized or not
- Add: Add a new lp to the pool by the MasterChef wallet.
- set: Update the given pool's PRV allocation point and deposit fee by the MasterChef wallet.
- setPrvReferral: Update the prv referral contract address by the MasterChef wallet.
- updateEmissionRate: Pancake has to add hidden dummy pools in order to alter the emission, here MasterChef wallet makes it simple and transparent to all.
- mint: Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).

Conclusion

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

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

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

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

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

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

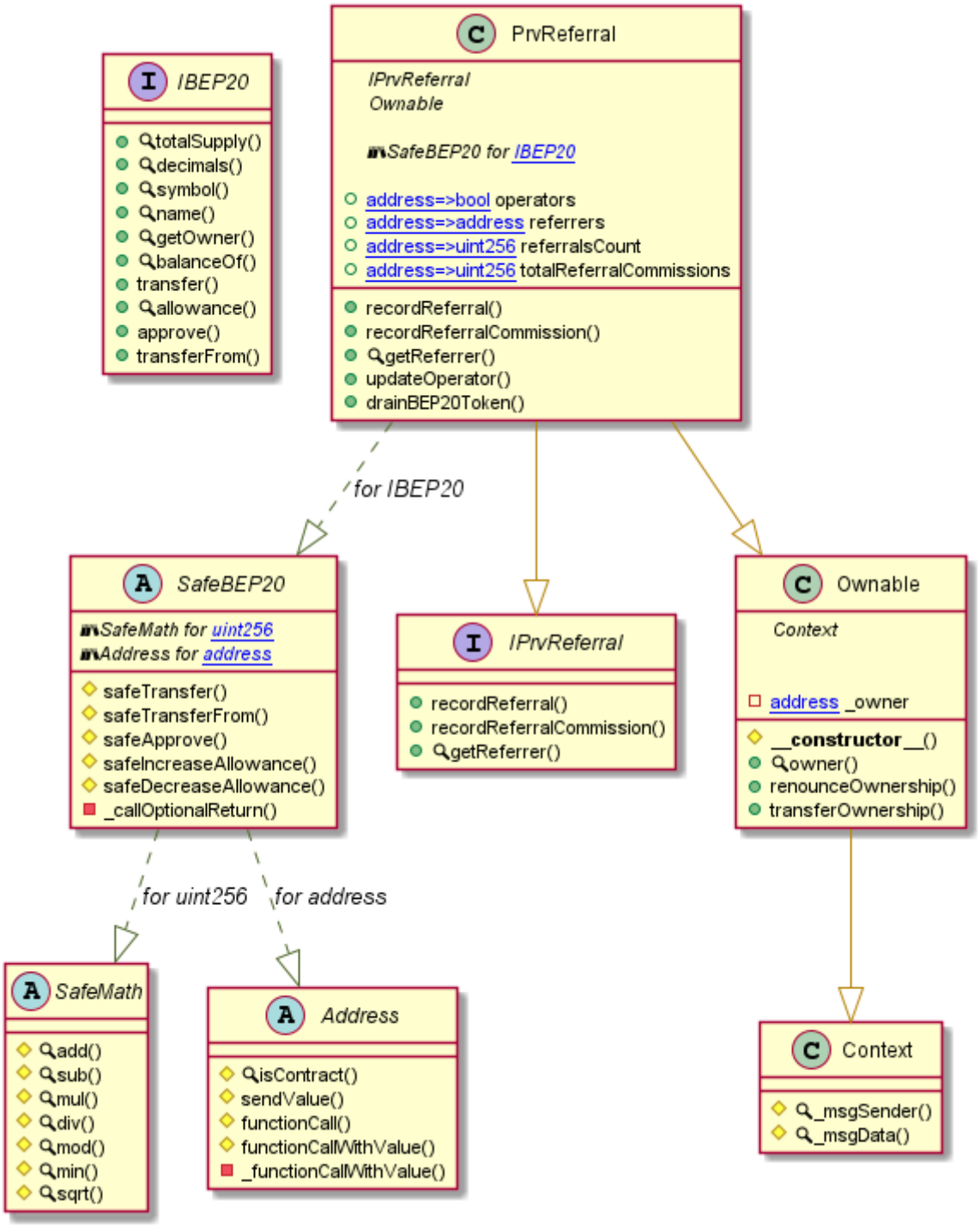
Technical Disclaimer

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

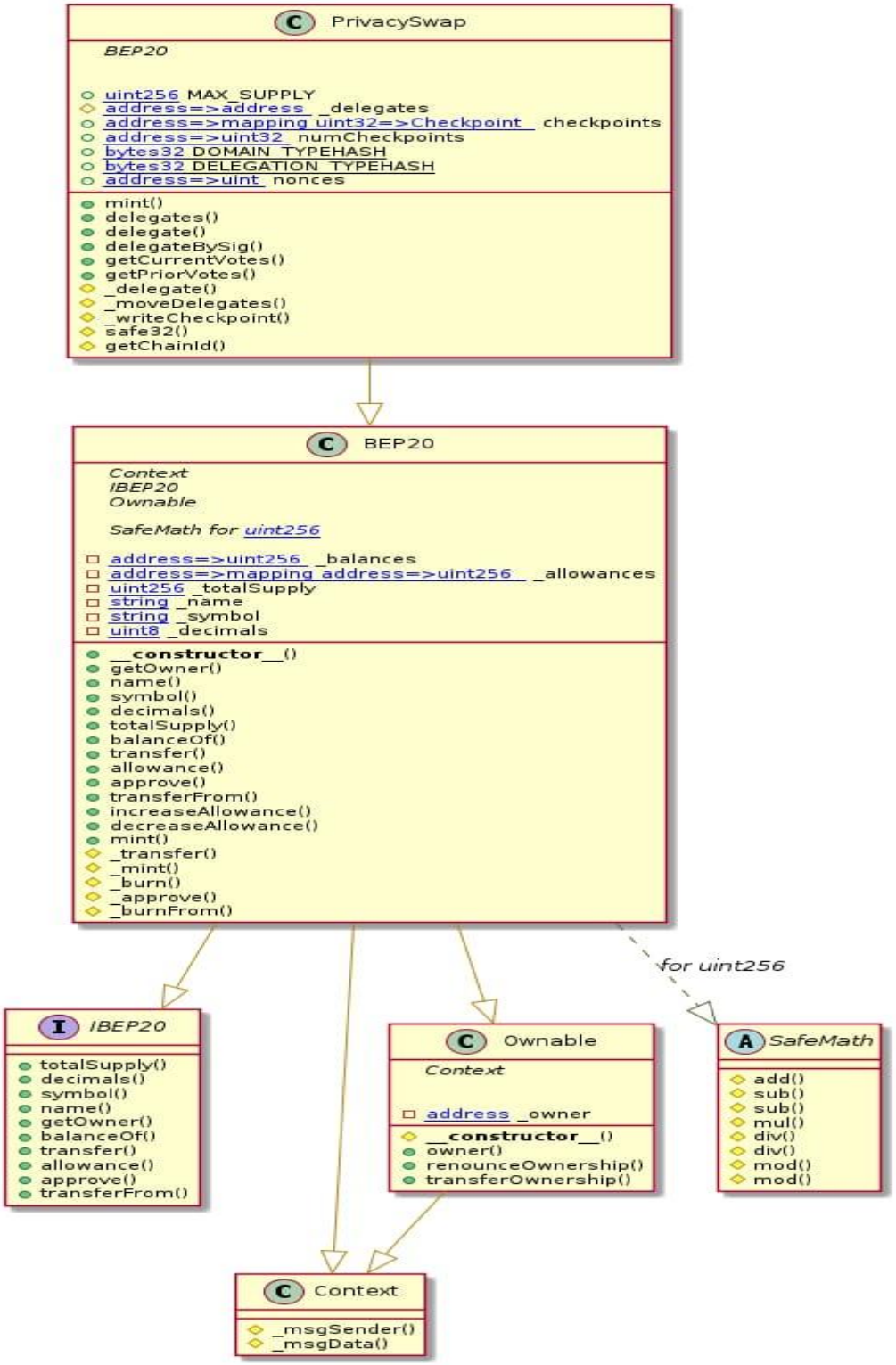
Appendix

Code Flow Diagram - Privacy Swap Protocol

PrvReferral Diagram

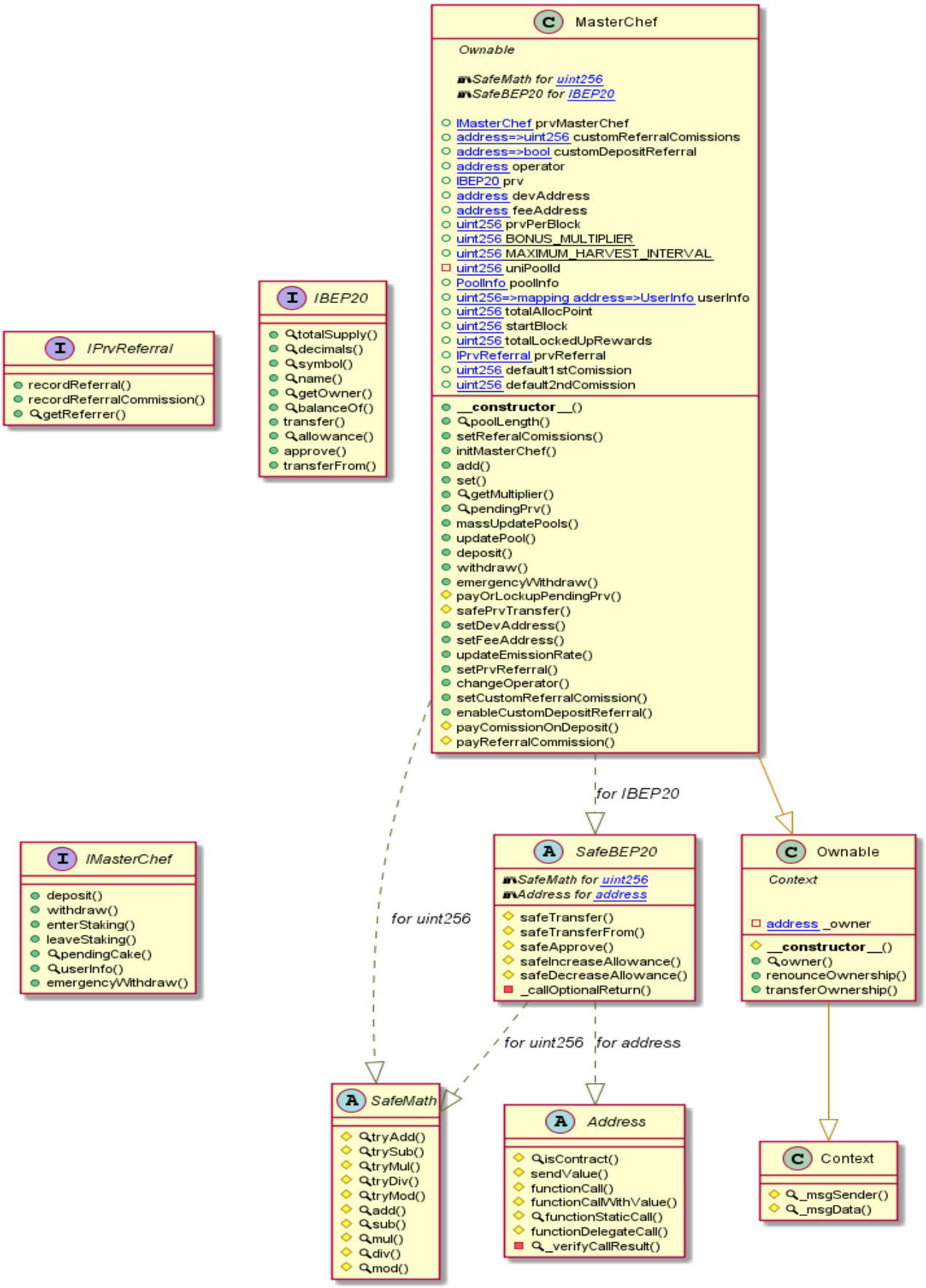


PrivacySwap Diagram



Email: audit@EtherAuthority.io

MasterChef Diagram



Slither Results Log

Slither log >> PrvReferral.sol

INFO:Detectors:
Address.isContract(address) (PrvReferral.sol#293-304) uses assembly
- INLINE ASM (PrvReferral.sol#300-302)
Address._functionCallWithValue(address,bytes,uint256,string) (PrvReferral.sol#401-427) uses assembly
- INLINE ASM (PrvReferral.sol#419-422)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
- Version used: ['^0.6.0', '^0.6.12']
- ^0.6.12 (PrvReferral.sol#7)
- ^0.6.0 (PrvReferral.sol#546)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (PrvReferral.sol#348-350) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PrvReferral.sol#377-383) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (PrvReferral.sol#391-399) is never used and should be removed
Address.sendValue(address,uint256) (PrvReferral.sol#322-328) is never used and should be removed
Context._msgData() (PrvReferral.sol#537-540) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (PrvReferral.sol#457-471) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (PrvReferral.sol#482-492) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (PrvReferral.sol#473-480) is never used and should be removed
SafeBEP20.safeTransferFrom(IBEP20,address,address,uint256) (PrvReferral.sol#441-448) is never used and should be removed
SafeMath.add(uint256,uint256) (PrvReferral.sol#114-119) is never used and should be removed
SafeMath.div(uint256,uint256) (PrvReferral.sol#192-194) is never used and should be removed
SafeMath.div(uint256,uint256,string) (PrvReferral.sol#208-218) is never used and should be removed
SafeMath.min(uint256,uint256) (PrvReferral.sol#257-259) is never used and should be removed
SafeMath.mod(uint256,uint256) (PrvReferral.sol#232-234) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (PrvReferral.sol#248-255) is never used and should be removed
SafeMath.mul(uint256,uint256) (PrvReferral.sol#166-178) is never used and should be removed
SafeMath.sqrt(uint256) (PrvReferral.sol#262-273) is never used and should be removed
SafeMath.sub(uint256,uint256) (PrvReferral.sol#131-133) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (PrvReferral.sol#145-154) 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 (PrvReferral.sol#546) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (PrvReferral.sol#322-328):
- (success) = recipient.call{value: amount}() (PrvReferral.sol#326)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (PrvReferral.sol#401-427):
- (success,returndata) = target.call{value: weiValue}(data) (PrvReferral.sol#410)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter PrvReferral.recordReferral(address,address)._user (PrvReferral.sol#630) is not in mixedCase
Parameter PrvReferral.recordReferral(address,address)._referrer (PrvReferral.sol#630) is not in mixedCase
Parameter PrvReferral.recordReferralCommission(address,uint256)._referrer (PrvReferral.sol#642) is not in mixedCase
Parameter PrvReferral.recordReferralCommission(address,uint256)._commission (PrvReferral.sol#642) is not in mixedCase
Parameter PrvReferral.getReferrer(address)._user (PrvReferral.sol#650) is not in mixedCase
Parameter PrvReferral.updateOperator(address,bool)._operator (PrvReferral.sol#655) is not in mixedCase

Parameter PrvReferral.updateOperator(address,bool)._status (PrvReferral.sol#655) is not in mixedCase
Parameter PrvReferral.drainBEP20Token(IBEP20,uint256,address)._token (PrvReferral.sol#661) is not in mixedCase
Parameter PrvReferral.drainBEP20Token(IBEP20,uint256,address)._amount (PrvReferral.sol#661) is not in mixedCase
Parameter PrvReferral.drainBEP20Token(IBEP20,uint256,address)._to (PrvReferral.sol#661) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PrvReferral.sol#538)" inContext (PrvReferral.sol#532-541)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
owner() should be declared external:
- Ownable.owner() (PrvReferral.sol#577-579)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (PrvReferral.sol#596-599)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (PrvReferral.sol#605-609)
recordReferral(address,address) should be declared external:
- PrvReferral.recordReferral(address,address) (PrvReferral.sol#630-640)
recordReferralCommission(address,uint256) should be declared external:
- PrvReferral.recordReferralCommission(address,uint256) (PrvReferral.sol#642-647)
getReferrer(address) should be declared external:
- PrvReferral.getReferrer(address) (PrvReferral.sol#650-652)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PrvReferral.sol analyzed (8 contracts with 75 detectors), 42 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Slither log >> PrivacySwap.sol

INFO:Detectors:
PrivacySwap._writeCheckpoint(address,uint32,uint256,uint256) (PrivacySwap.sol#855-873) uses a dangerous strict equality:
- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (PrivacySwap.sol#865)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
BEP20.constructor(string,string).name (PrivacySwap.sol#401) shadows:
- BEP20.name() (PrivacySwap.sol#417-419) (function)
- IBEP20.name() (PrivacySwap.sol#27) (function)
BEP20.constructor(string,string).symbol (PrivacySwap.sol#401) shadows:
- BEP20.symbol() (PrivacySwap.sol#425-427) (function)
- IBEP20.symbol() (PrivacySwap.sol#22) (function)
BEP20.allowance(address,address).owner (PrivacySwap.sol#466) shadows:
- Ownable.owner() (PrivacySwap.sol#157-159) (function)
BEP20._approve(address,address,uint256).owner (PrivacySwap.sol#625) shadows:
- Ownable.owner() (PrivacySwap.sol#157-159) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PrivacySwap.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (PrivacySwap.sol#721-762) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,PRV::delegateBySig: signature expired) (PrivacySwap.sol#760)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
PrivacySwap.getChainId() (PrivacySwap.sol#880-884) uses assembly
- INLINE ASM (PrivacySwap.sol#882)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:

Different versions of Solidity is used:
- Version used: ['>=0.4.0', '^0.6.0', '^0.6.12']
- ^0.6.12 (PrivacySwap.sol#6)
- ^0.6.0 (PrivacySwap.sol#101)
- ^0.6.0 (PrivacySwap.sol#126)
- ^0.6.0 (PrivacySwap.sol#192)
- >=0.4.0 (PrivacySwap.sol#352)
- ^0.6.12 (PrivacySwap.sol#646)

Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
BEP20._burn(address,uint256) (PrivacySwap.sol#604-610) is never used and should be removed
BEP20._burnFrom(address,uint256) (PrivacySwap.sol#639-642) is never used and should be removed
Context._msgData() (PrivacySwap.sol#118-121) is never used and should be removed
SafeMath.div(uint256,uint256) (PrivacySwap.sol#292-294) is never used and should be removed
SafeMath.div(uint256,uint256,string) (PrivacySwap.sol#308-314) is never used and should be removed
SafeMath.mod(uint256,uint256) (PrivacySwap.sol#328-330) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (PrivacySwap.sol#344-347) is never used and should be removed
SafeMath.mul(uint256,uint256) (PrivacySwap.sol#266-278) 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 (PrivacySwap.sol#101) allows old versions
Pragma version^0.6.0 (PrivacySwap.sol#126) allows old versions
Pragma version^0.6.0 (PrivacySwap.sol#192) allows old versions
Pragma version>=0.4.0 (PrivacySwap.sol#352) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Parameter PrivacySwap.mint(address,uint256)._to (PrivacySwap.sol#657) is not in mixedCase
Parameter PrivacySwap.mint(address,uint256)._amount (PrivacySwap.sol#657) is not in mixedCase
Variable PrivacySwap.MAX_SUPPLY (PrivacySwap.sol#653) is not in mixedCase
Variable PrivacySwap._delegates (PrivacySwap.sol#663) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (PrivacySwap.sol#119)" inContext (PrivacySwap.sol#113-122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
PrivacySwap.slitherConstructorVariables() (PrivacySwap.sol#651-886) uses literals with too many digits:
- MAX_SUPPLY = 30000000 * 10 ** 18 (PrivacySwap.sol#653)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
PrivacySwap.MAX_SUPPLY (PrivacySwap.sol#653) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (PrivacySwap.sol#176-179)

transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (PrivacySwap.sol#185-189)

symbol() should be declared external:
- BEP20.symbol() (PrivacySwap.sol#425-427)

decimals() should be declared external:
- BEP20.decimals() (PrivacySwap.sol#432-434)

totalSupply() should be declared external:
- BEP20.totalSupply() (PrivacySwap.sol#439-441)

transfer(address,uint256) should be declared external:
- BEP20.transfer(address,uint256) (PrivacySwap.sol#458-461)

allowance(address,address) should be declared external:
- BEP20.allowance(address,address) (PrivacySwap.sol#466-468)

approve(address,uint256) should be declared external:
- BEP20.approve(address,uint256) (PrivacySwap.sol#477-480)

transferFrom(address,address,uint256) should be declared external:
- BEP20.transferFrom(address,address,uint256) (PrivacySwap.sol#494-502)

increaseAllowance(address,uint256) should be declared external:
- BEP20.increaseAllowance(address,uint256) (PrivacySwap.sol#516-519)

decreaseAllowance(address,uint256) should be declared external:
- BEP20.decreaseAllowance(address,uint256) (PrivacySwap.sol#535-538)
mint(uint256) should be declared external:
- BEP20.mint(uint256) (PrivacySwap.sol#548-551)
mint(address,uint256) should be declared external:
- PrivacySwap.mint(address,uint256) (PrivacySwap.sol#657-660)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PrivacySwap.sol analyzed (6 contracts with 75 detectors), 40 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Slither log >> MasterChef.sol

INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928):
External calls:
- prvReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#909)
- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
- prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
- prv.transfer(_to,prvBal) (MasterChef.sol#981)
- prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#915)
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
- returndata = address(token).functionCall(data,SafeBEP20: low-level call failed)
(MasterChef.sol#607)
- token.safeTransfer(refererer,commissionAmount) (MasterChef.sol#1038)
- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#920)
External calls sending eth:
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
State variables written after the call(s):
- user.amount = user.amount.add(_amount).sub(depositFee).sub(refComission) (MasterChef.sol#921)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928):
External calls:
- prvReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#909)
- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
- prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
- prv.transfer(_to,prvBal) (MasterChef.sol#981)
- prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#915)
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
- returndata = address(token).functionCall(data,SafeBEP20: low-level call failed)
(MasterChef.sol#607)
- token.safeTransfer(refererer,commissionAmount) (MasterChef.sol#1038)
- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
External calls sending eth:
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
State variables written after the call(s):
- user.amount = user.amount.add(_amount).sub(refComission) (MasterChef.sol#923)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
MasterChef.safePrvTransfer(address,uint256) (MasterChef.sol#977-985) ignores return value by
prv.transfer(_to,prvBal) (MasterChef.sol#981)
MasterChef.safePrvTransfer(address,uint256) (MasterChef.sol#977-985) ignores return value by
prv.transfer(_to,_amount) (MasterChef.sol#983)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer

```
INFO:Detectors:
MasterChef.pendingPrv(uint256,address) (MasterChef.sol#861-873) performs a multiplication on the result of a division:
	-prvReward = multiplier.mul(prvPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#868)
	-accPrvPerShare = accPrvPerShare.add(prvReward.mul(1e12).div(lpSupply)) (MasterChef.sol#869)
MasterChef.updatePool(uint256) (MasterChef.sol#886-901) performs a multiplication on the result of a division:
	-prvReward = multiplier.mul(prvPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (MasterChef.sol#897)
	-pool.accPrvPerShare = pool.accPrvPerShare.add(prvReward.mul(1e12).div(lpSupply)) (MasterChef.sol#899)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
MasterChef.updatePool(uint256) (MasterChef.sol#886-901) uses a dangerous strict equality:
	- lpSupply == 0 || pool.allocPoint == 0 (MasterChef.sol#892)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928):
	External calls:
	- prvReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#909)
	- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
		- prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
		- prv.transfer(_to,prvBal) (MasterChef.sol#981)
		- prv.transfer(_to,_amount) (MasterChef.sol#983)
		- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
		- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
	State variables written after the call(s):
	- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
		- user.rewardLockedUp = 0 (MasterChef.sol#967)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#931-943):
	External calls:
	- payOrLockupPendingPrv(_pid) (MasterChef.sol#936)
		- prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
		- prv.transfer(_to,prvBal) (MasterChef.sol#981)
		- prv.transfer(_to,_amount) (MasterChef.sol#983)
		- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
		- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
	State variables written after the call(s):
	- user.amount = user.amount.sub(_amount) (MasterChef.sol#938)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#931-943):
	External calls:
	- payOrLockupPendingPrv(_pid) (MasterChef.sol#936)
		- prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
		- prv.transfer(_to,prvBal) (MasterChef.sol#981)
		- prv.transfer(_to,_amount) (MasterChef.sol#983)
		- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
		- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
	- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#939)
	State variables written after the call(s):
	- user.rewardDebt = user.amount.mul(pool.accPrvPerShare).div(1e12) (MasterChef.sol#941)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
MasterChef.changeOperator(address).addr (MasterChef.sol#1013) lacks a zero-check on :
		- operator = addr (MasterChef.sol#1015)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in MasterChef.initMasterChef(IMasterChef,IBEP20,uint256) (MasterChef.sol#818-824):
	External calls:
	- uniqueStakingContract.safeApprove(address(prvMasterChef),uint256(- 1)) (MasterChef.sol#821)
	- prvMasterChef.deposit(poolId,1 ** 1e18) (MasterChef.sol#822)
	State variables written after the call(s):
	- uniPoolId = poolId (MasterChef.sol#823)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928):
	External calls:
```

- prvReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#909)
- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
    - prv.transfer(_to,prvBal) (MasterChef.sol#981)
    - prv.transfer(_to,_amount) (MasterChef.sol#983)
    - prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
    - prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer1st,commissionAmount1) (MasterChef.sol#1068)
    - payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
- ReferralCommissionPaid(_user,referrer2nd,commissionAmount2) (MasterChef.sol#1078)
    - payOrLockupPendingPrv(_pid) (MasterChef.sol#913)

Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928):

External calls:
- prvReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#909)
- payOrLockupPendingPrv(_pid) (MasterChef.sol#913)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
    - prv.transfer(_to,prvBal) (MasterChef.sol#981)
    - prv.transfer(_to,_amount) (MasterChef.sol#983)
    - prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
    - prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#915)
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
    - returndata = address(token).functionCall(data,SafeBEP20: low-level call failed) (MasterChef.sol#607)
    - token.safeTransfer(refererer,commissionAmount) (MasterChef.sol#1038)
    - (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#920)

External calls sending eth:
- refComission = payComissionOnDeposit(pool.lpToken,_amount,msg.sender) (MasterChef.sol#916)
    - (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)

Event emitted after the call(s):
- Deposit(msg.sender,_pid,_amount) (MasterChef.sol#927)

Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#946-955):

External calls:
- pool.lpToken.safeTransfer(address(msg.sender),amount) (MasterChef.sol#953)

Event emitted after the call(s):
- EmergencyWithdraw(msg.sender,_pid,amount) (MasterChef.sol#954)

Reentrancy in MasterChef.payReferralCommission(address,uint256) (MasterChef.sol#1048-1082):

External calls:
- safePrvTransfer(referrer1st,commissionAmount1) (MasterChef.sol#1066)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
    - prv.transfer(_to,prvBal) (MasterChef.sol#981)
    - prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer1st,commissionAmount1) (MasterChef.sol#1068)

Reentrancy in MasterChef.payReferralCommission(address,uint256) (MasterChef.sol#1048-1082):

External calls:
- safePrvTransfer(referrer1st,commissionAmount1) (MasterChef.sol#1066)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
    - prv.transfer(_to,prvBal) (MasterChef.sol#981)
    - prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
- safePrvTransfer(referrer2nd,commissionAmount2) (MasterChef.sol#1076)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)
    - prv.transfer(_to,prvBal) (MasterChef.sol#981)
    - prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer2nd,commissionAmount2) (MasterChef.sol#1078)

Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#931-943):

External calls:
- payOrLockupPendingPrv(_pid) (MasterChef.sol#936)
    - prvMasterChef.deposit(uniPoolId,0) (MasterChef.sol#978)

- prv.transfer(_to,prvBal) (MasterChef.sol#981)
- prv.transfer(_to,_amount) (MasterChef.sol#983)
- prvReferral.recordReferralCommission(referrer1st,commissionAmount1) (MasterChef.sol#1067)
- prvReferral.recordReferralCommission(referrer2nd,commissionAmount2) (MasterChef.sol#1077)

- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#939)

Event emitted after the call(s):

- Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#942)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Address.isContract(address) (MasterChef.sol#355-364) uses assembly
- INLINE ASM (MasterChef.sol#362)

Address._verifyCallResult(bool,bytes,string) (MasterChef.sol#500-517) uses assembly
- INLINE ASM (MasterChef.sol#509-512)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
MasterChef.payComissionOnDeposit(IBEP20,uint256,address) (MasterChef.sol#1033-1042) compares to a boolean constant:
-customDepositReferral[refererer] == true && customReferralComissions[refererer] > 0 (MasterChef.sol#1036)
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', '>=0.6.0<0.8.0', '>=0.6.2<0.8.0', '^0.6.0']
- 0.6.12 (MasterChef.sol#3)
- >=0.4.0 (MasterChef.sol#22)
- >=0.6.0<0.8.0 (MasterChef.sol#119)
- >=0.6.2<0.8.0 (MasterChef.sol#332)
- ^0.6.0 (MasterChef.sol#520)
- 0.6.12 (MasterChef.sol#616)
- ^0.6.0 (MasterChef.sol#641)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (MasterChef.sol#408-410) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MasterChef.sol#433-435) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MasterChef.sol#482-484) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#492-498) is never used and should be removed
Address.functionStaticCall(address,bytes) (MasterChef.sol#458-460) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MasterChef.sol#468-474) is never used and should be removed
Address.sendValue(address,uint256) (MasterChef.sol#382-388) is never used and should be removed
Context._msgData() (MasterChef.sol#633-636) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#584-594) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#575-582) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MasterChef.sol#306-309) is never used and should be removed
SafeMath.mod(uint256,uint256) (MasterChef.sol#268-271) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MasterChef.sol#326-329) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (MasterChef.sol#286-289) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MasterChef.sol#140-144) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MasterChef.sol#176-179) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MasterChef.sol#186-189) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MasterChef.sol#161-169) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MasterChef.sol#151-154) 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#22) allows old versions
Pragma version>=0.6.0<0.8.0 (MasterChef.sol#119) is too complex
Pragma version>=0.6.2<0.8.0 (MasterChef.sol#332) is too complex
Pragma version^0.6.0 (MasterChef.sol#520) allows old versions
Pragma version^0.6.0 (MasterChef.sol#641) allows old versions

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (MasterChef.sol#382-388):
- (success) = recipient.call{value: amount}() (MasterChef.sol#386)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#443-450):
- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#448)
Low level call in Address.functionStaticCall(address,bytes,string) (MasterChef.sol#468-474):
- (success,returndata) = target.staticcall(data) (MasterChef.sol#472)
Low level call in Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#492-498):
- (success,returndata) = target.delegatecall(data) (MasterChef.sol#496)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter MasterChef.initMasterChef(IMasterChef,IBEP20,uint256)._prvMasterChef (MasterChef.sol#818) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._allocPoint (MasterChef.sol#828) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._lpToken (MasterChef.sol#828) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._depositFeeBP (MasterChef.sol#828) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._withUpdate (MasterChef.sol#828) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._pid (MasterChef.sol#845) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._allocPoint (MasterChef.sol#845) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._depositFeeBP (MasterChef.sol#845) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._withUpdate (MasterChef.sol#845) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (MasterChef.sol#856) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (MasterChef.sol#856) is not in mixedCase
Parameter MasterChef.pendingPrv(uint256,address)._pid (MasterChef.sol#861) is not in mixedCase
Parameter MasterChef.pendingPrv(uint256,address)._user (MasterChef.sol#861) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (MasterChef.sol#886) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._pid (MasterChef.sol#904) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._amount (MasterChef.sol#904) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._referrer (MasterChef.sol#904) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (MasterChef.sol#931) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (MasterChef.sol#931) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (MasterChef.sol#946) is not in mixedCase
Parameter MasterChef.payOrLockupPendingPrv(uint256)._pid (MasterChef.sol#958) is not in mixedCase
Parameter MasterChef.safePrvTransfer(address,uint256)._to (MasterChef.sol#977) is not in mixedCase
Parameter MasterChef.safePrvTransfer(address,uint256)._amount (MasterChef.sol#977) is not in mixedCase
Parameter MasterChef.setDevAddress(address)._devAddress (MasterChef.sol#988) is not in mixedCase
Parameter MasterChef.setFeeAddress(address)._feeAddress (MasterChef.sol#994) is not in mixedCase
Parameter MasterChef.updateEmissionRate(uint256)._prvPerBlock (MasterChef.sol#1001) is not in mixedCase
Parameter MasterChef.setPrvReferral(IPrvReferral)._prvReferral (MasterChef.sol#1008) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._user (MasterChef.sol#1048) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._pending (MasterChef.sol#1048) 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#634)" inContext (MasterChef.sol#628-637)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable MasterChef.payReferralCommission(address,uint256).commissionAmount1 (MasterChef.sol#1059) is too similar to MasterChef.payReferralCommission(address,uint256).commissionAmount2 (MasterChef.sol#1071)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
owner() should be declared external:
- Ownable.owner() (MasterChef.sol#672-674)

renounceOwnership() should be declared external:
	- Ownable.renounceOwnership() (MasterChef.sol#691-694)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (MasterChef.sol#700-704)
setReferalComissions(uint256,uint256) should be declared external:
	- MasterChef.setReferalComissions(uint256,uint256) (MasterChef.sol#811-814)
initMasterChef(IMasterChef,IBEP20,uint256) should be declared external:
	- MasterChef.initMasterChef(IMasterChef,IBEP20,uint256) (MasterChef.sol#818-824)
add(uint256,IBEP20,uint16,bool) should be declared external:
	- MasterChef.add(uint256,IBEP20,uint16,bool) (MasterChef.sol#828-842)
set(uint256,uint256,uint16,bool) should be declared external:
	- MasterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#845-853)
deposit(uint256,uint256,address) should be declared external:
	- MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#904-928)
withdraw(uint256,uint256) should be declared external:
	- MasterChef.withdraw(uint256,uint256) (MasterChef.sol#931-943)
emergencyWithdraw(uint256) should be declared external:
	- MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#946-955)
setDevAddress(address) should be declared external:
	- MasterChef.setDevAddress(address) (MasterChef.sol#988-992)
setFeeAddress(address) should be declared external:
	- MasterChef.setFeeAddress(address) (MasterChef.sol#994-998)
updateEmissionRate(uint256) should be declared external:
	- MasterChef.updateEmissionRate(uint256) (MasterChef.sol#1001-1005)
setPrvReferral(IPrvReferral) should be declared external:
	- MasterChef.setPrvReferral(IPrvReferral) (MasterChef.sol#1008-1010)
changeOperator(address) should be declared external:
	- MasterChef.changeOperator(address) (MasterChef.sol#1013-1016)
setCustomReferralComission(address,uint256) should be declared external:
	- MasterChef.setCustomReferralComission(address,uint256) (MasterChef.sol#1019-1023)
enableCustomDepositReferral(address) should be declared external:
	- MasterChef.enableCustomDepositReferral(address) (MasterChef.sol#1025-1028)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MasterChef.sol analyzed (9 contracts with 75 detectors), 98 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io