Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	RBH
Platform:	Binance Smart Chain
Language:	Solidity
Date:	July 12th, 2021

Table of contents

Email: audit@EtherAuthority.io

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

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the RBH team to perform the Security audit of the RBH 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 July 9th, 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

RobinHood (RBH) BEP20 Token in Binance Smart Chain Mainnet. Token is implemented as an BEP20 smart contract with address.

RobinHoodSwap is a yield farming ecosystem running on the Binance Smart Chain (BSC). It runs on RobinHoodSwap Tokens or RBH.

The objective of RobinHoodSwap is to allow users to utilize blockchain and DeFi to earn rewards while contributing to society by giving back to charitable organizations. It is indeed a platform where prosperity meets charity to create a better society. As a user farm, rewards are generated which are then attributed to give back to charity.

Audit scope

Name	Code Review and Security Analysis Report for RBH protocol Smart Contracts
Platform	**BSC / Solidity**
File 1	RbhReferral.sol
Smart Contract Online Code	https://bscscan.com/address/0x4f8012de3dcf0ba7089cee58b8e9124da70d0e81#code
File 1 MD5 Hash	0D19D66A267938470B6A08D0015BEE5A
File 2	RobinHoodFarmTimelock.sol
Smart Contract Online Code	https://bscscan.com/address/0x9832f55f3bda656100ecdaf02c6afe5535f715c3#code
File 2 MD5 Hash	BBEDF910C019B8FD4212D02F520B5192
File 3	MasterChef.sol
Smart Contract Online Code	https://bscscan.com/address/0xd4dc714a68638ffc5ec24441fe37e9dda677467a#code
File 3 MD5 Hash	BE7EC331DDB31637E2CE8D82C4841983
File 4	RobinHood.sol
Smart Contract Online Code	https://bscscan.com/address/0xd5779f2f9d7d239228e4e78bc78f50768661a081#code
File 4 MD5 Hash	C589F75ACC13DB1CDC5E60E6A2EB118C
Audit Date	July 12th, 2021

PS: There are some external libraries and other contracts inherited, which are not included in the audit scope and thus are not audited.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
File 1: RbhReferral.sol • recordReferral: The Operator can store referral addresses. • recordReferralCommission: The Operator owner can set referral commission.	**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: RobinHoodFarmTimelock.sol • Grace Period: 14 days • Minimum Delay: 6 hours • Maximum Delay: 30 days	**YES, This is valid.Owner Can change time period.**
File 3: MasterChef.sol • Max referral commission rate: 10%. • 25% tokens can be harvested if limit exceeds • The Owner can access functions like: setMaxHarvestLimit, setMaxHarvestThreshold, add,set ,setDevFee, updateEmissionRate, setRbhReferral, setReferralCommissionRate,etc.	**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 4: RobinHood.sol • Name: RobinhoodSwap • Symbol: RBH • Decimals: 18	**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.**

• setAntiWhaleEnabled: The Operator can set antiWhale status. • setExcludedFromAntiWhale: The Operator can set Exclude or include an address from antiWhale. • transferOperator:The Operator can Transfers operator of the contract to a new account (`newOperator`).	**Because if the private key is compromised, then it will create problems.**

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 8 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Moderated
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Moderated
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

These audit scope have 4 smart contracts. These smart contracts also contain Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in the RBH 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 RBH Protocol.

The 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 RBH 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.

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

RbhReferral.sol

(1) Interface

(a) IBEP20

(b) IRbhReferral

(2) Inherited contracts

(a) Context

(b) IRbhReferral

(c) Ownable

(3) Usages

(a) using SafeBEP20 for IBEP20;

(4) Events

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

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

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

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

(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	external	access only Owner	No Issue
6	drainBEP20Token	external	Function input parameters lack of check	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

RobinHoodFarmTimelock.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	setDelay	write	Passed	No Issue
2	acceptAdmin	write	Passed	No Issue
3	setPendingAdmin	write	Passed	No Issue
4	queueTransaction	write	Passed	No Issue
5	cancelTransaction	write	Passed	No Issue
6	executeTransaction	write	Deprecated function use	Refer Audit Findings
7	getBlockTimestamp	internal	Passed	No Issue
8	includeInReward	write	Infinite loops possibility at multiple places:	Refer Audit Findings

MasterChef.sol

(1) Interface

(a) IRbhReferral

(b) IBEP20

(2) Inherited contracts

(a) Ownable

(b) ReentrancyGuard

(c) RobinHood

(d) BEP20

(e) Context

(f) IBEP20

(3) Struct

(a) UserInfo

(b) PoolInfo

(4) Usages

(a) using SafeMath for uint256;

(b) using SafeBEP20 for IBEP20;

(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	poolLength	external	Passed	No Issue
2	setMaxHarvestLimit	write	access only Owner	No Issue
3	setMaxHarvestThreshold	write	access only Owner	No Issue
4	add	write	Critical operation lacks event log	Refer Audit Findings
5	set	write	Range validation missing	Refer Audit Findings
6	getMultiplier	write	Passed	No Issue
7	pendingRbh	external	Passed	No Issue
8	massUpdatePools	write	Infinite loop	Refer Audit Findings
9	updatePool	write	Critical operation lacks event log	Refer Audit Findings
10	setDevFee	write	Function input parameters lack of check	Refer Audit Findings
11	deposit	write	Function input parameters lack of check	Refer Audit Findings
12	withdraw	write	Passed	No Issue
13	emergencyWithdraw	write	Passed	No Issue
14	payOrLockupPendingRbh	internal	Passed	No Issue
15	safeRbhTransfer	internal	Function input parameters lack of check	Refer Audit Findings
16	setDevAddress	write	Passed	No Issue
17	setFeeAddress	write	Passed	No Issue
18	updateEmissionRate	write	access only Owner	No Issue
19	setRbhReferral	write	access only Owner	No Issue
20	payReferralCommission	internal	Passed	No Issue
21	nonReentrant	modifier	Passed	No Issue
22	owner	read	Passed	No Issue
23	onlyOwner	modifier	Passed	No Issue
24	renounceOwnership	write	access only Owner	No Issue
25	transferOwnership	write	access only Owner	No Issue

RobinHood.sol

(1) Interface

(a) IBEP20

(2) Inherited contracts

(a) BEP20

(b) Ownable

(c) Context

(3) Struct

(a) Checkpoint

(4) Events

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

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

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

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

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

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	onlyOperator	modifier	Passed	No Issue
2	antiWhale	modifier	Passed	No Issue
3	mint	write	Function input parameters lack of check	Refer Audit Findings
4	transfer	internal	Passed	No Issue
5	maxTransferAmount	read	Passed	No Issue
6	isExcludedFromAntiWhale	read	Passed	No Issue
7	setAntiWhaleEnabled	write	access only Operator	No Issue
8	updateMaxTransferAmountRate	write	access only Operator	No Issue
9	setExcludedFromAntiWhale	write	access only Operator	No Issue
10	operator	read	Passed	No Issue

11	transferOperator	write	access only Operator	No Issue
12	delegates	external	Passed	No Issue
13	delegate	external	Passed	No Issue
14	delegateBySig	external	Passed	No Issue
15	getCurrentVotes	external	Passed	No Issue
16	getPriorVotes	external	Passed	No Issue
17	_delegate	internal	Passed	No Issue
18	_moveDelegates	internal	Passed	No Issue
19	_writeCheckpoint	internal	Passed	No Issue
20	safe32	internal	Passed	No Issue
21	getChainId	internal	Passed	No Issue
22	getOwner	read	Passed	No Issue
23	name	read	Passed	No Issue
24	decimals	read	Passed	No Issue
25	symbol	read	Passed	No Issue
26	totalSupply	read	Passed	No Issue
27	balanceOf	read	Passed	No Issue
28	transfer	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	approve	write	Passed	No Issue
31	transferFrom	write	Passed	No Issue
32	increaseAllowance	write	Passed	No Issue
33	decreaseAllowance	write	Passed	No Issue
34	mint	write	access only Owner	No Issue
35	_transfer	internal	Passed	No Issue
36	_mint	internal	Passed	No Issue
37	_burn	internal	Passed	No Issue
38	_approve	internal	Passed	No Issue
39	_burnFrom	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

RobinHood.sol

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check:

```
function mint(address _to, uint256 _amount) public onlyOwner {
    if(totalSupply().add(_amount)<= MAX_SUPPLY){
        _mint(_to, _amount);
        _moveDelegates(address(0), _delegates[_to], _amount);

    }
}
```

In the mint function, the parameter amount is not checked.

Resolution: Please put validation/check for amount.

(2) Infinite loop:

```
uint32 lower = 0;
uint32 upper = nCheckpoints - 1;
while (upper > lower) {
    uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
    Checkpoint memory cp = checkpoints[account][center];
    if (cp.fromBlock == blockNumber) {
        return cp.votes;
    } else if (cp.fromBlock < blockNumber) {
        lower = center;
    } else {
        upper = center - 1;
    }
}
return checkpoints[account][lower].votes;
```

in the getPriorVotes function, if the upper value is too high than lower, then it will consume a lot of gas.

Resolution: nCheckpoints should be kept limited.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.6.12;
```

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

Resolution: Please use 0.8.6 which is the latest version.

(2) Make variables constant:

```
uint16 public constant MAXIMUM_TRANSFER_TAX_RATE = 1000;

bool public isAntiWhaleEnabled = false;
uint16 public maxTransferAmountRate = 50;
// Addresses that excluded from antiWhale
mapping(address => bool) private _excludedFromAntiWhale;
uint256 public MAX_SUPPLY = 32000000 *10**18;
```

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

Resolution: Declare those variables as constant. Just put a constant keyword. And define constant in constructor

(3) Use SPDX License Identifier:

```
RobinHood.sol: Warning: SPDX
license identifier not provided
in source file. Before
publishing, consider adding a
comment containing "SPDX-License-
Identifier: <SPDX-License>" to
each source file. Use "SPDX-
License-Identifier: UNLICENSED"
for non-open-source code. Please
see https://spdx.org for more
information.
```

SPDX license identifier not provided in source file.

Resolution: Please add SPDX identifier.

RbhReferral.sol

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Function input parameters lack of check:

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

In the drainBEP20Token function , the amount should be > 0.

Resolution: Please put validation for the amount.

(2) 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: if the private key of Owner is compromised, then it will create problems.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.6.12;
```

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

Resolution: Please use 0.8.6 which is the latest version.

(2) Missing event log:

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

Emit/ Event log is not written for drainBEP20Token.

Resolution: please add emit for all listed functions.

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) Range validation missing:

```
// Update the given pool's RobinHoodSWAP allocation point and deposit fee. Can only be called by the owner.
function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP,bool _withUpdate) public onlyOwner {
    require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
    if (_withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
    poolInfo[_pid].allocPoint = _allocPoint;
    poolInfo[_pid].depositFeeBP = _depositFeeBP;
}
```

In set function, state variables are arbitrarily large or small causing potential risks.

Resolution: Please set range for pid & _allocPoint .

(2) Function input parameters lack of check:

```
function safeRbhTransfer(address _to, uint256 _amount) internal {
    uint256 rbhBal = rbh.balanceOf(address(this));
    if (_amount > rbhBal) {
        rbh.transfer(_to, rbhBal);
    } else {
        rbh.transfer(_to, _amount);
    }
}
```

```
// Deposit LP tokens to MasterChef for RobinHoodSWAP allocation.
function deposit(uint256 _pid, uint256 _amount, address _referrer) public  {

    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
```

```
function setDevFee(uint256 _fee) public onlyOwner{
    devFee  = _fee;
}
```

In pendingRbh , check input parameter values for setDevFee , updatePool,deposit, withdraw,emergencyWithdraw , safeRbhTransfer functions.

Resolution: Please put validation for input parameters fee,amount,pid.

(3) Infinite loop:

```
// 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.

Resolution: The number of pools should be limited.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity 0.6.12;
```

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

Resolution: Please use 0.8.6 which is the latest version.

(2) Missing event log:

The missing event makes it difficult to track off-chain changes. An event should be emitted for significant transactions calling the following functions:

- add
- set
- updatePool

Resolution: We recommend emitting an event to log in following functions:

- add
- set
- updatePool

RobinHoodFarmTimelock.sol

Critical

No critical severity vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Deprecated function use:

```
function executeTransaction(address target, uint value, string memory signature, bytes memory data, uint eta)
public payable returns (bytes memory) {
    require(msg.sender == admin, "Timelock::executeTransaction: Call must come from admin.");
     bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
    require(queuedTransactions[txHash], "Timelock::executeTransaction: Transaction hasn't been queued.");
    require(getBlockTimestamp() >= eta, "Timelock::executeTransaction: Transaction hasn't surpassed time lock.");
    require(getBlockTimestamp() <= eta.add(GRACE_PERIOD), "Timelock::executeTransaction: Transaction is stale.");
    queuedTransactions[txHash] = false;
    bytes memory callData;

    if (bytes(signature).length == 0) {
        callData = data;
    } else {
        callData = abi.encodePacked(bytes4(keccak256(bytes(signature))), data);
    }
    // solium-disable-next-line security/no-call-value
    (bool success, bytes memory returnData) = target.call.value(value)(callData);
    require(success, "Timelock::executeTransaction: Transaction execution reverted.");
    emit ExecuteTransaction(txHash, target, value, signature, data, eta);
    return returnData;
}
```

deprecated function used in executeTransaction() function.

Resolution: We suggest not using a deprecated function.. Use {value:..} instead for this.

Very Low / Discussion / Best practices:

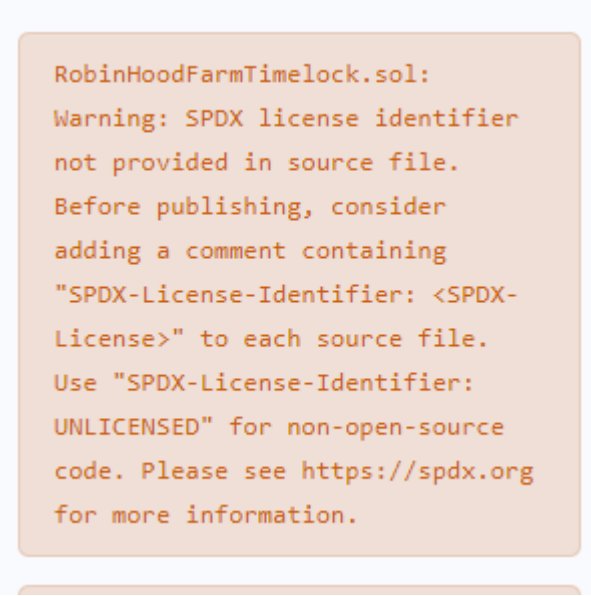
(1) Use latest solidity version:

```
pragma solidity 0.6.12;
```

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

Resolution: Please use 0.8.6 which is the latest version.

(2) Use SPDX License Identifier:



SPDX license identifier not provided in source file.

Resolution: Please add SPDX 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:

- drainBEP20Token: The RbhReferral owner can drain tokens that are sent here by mistake.
- updateOperator: The RbhReferral owner can update the status of the operator.
- setMaxHarvestLimit: The MasterChef owner can set maxHarvestLimit limit.
- setMaxHarvestThreshold: The MasterChef owner set maxHarvestThreshold.
- add: The MasterChef owner can add a new lp to the pool.
- set: The MasterChef owner can update the given pool's RobinHoodSWAP allocation point and deposit fee.
- setDevFee: The MasterChef owner can set the dev fee.
- updateEmissionRate: The MasterChef owner can update Pancake has to add hidden dummy pools in order to alter the emission, here we make it simple and transparent to all.
- setRbhReferral: The MasterChef owner can update the rbh referral contract address.
- setReferralCommissionRate: The MasterChef owner can update referral commission rate.
- 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.

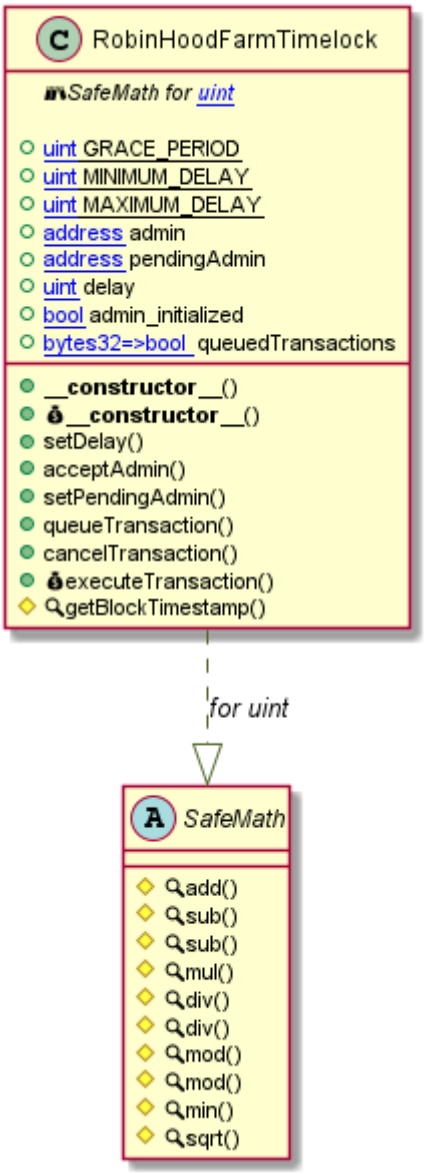
Technical Disclaimer

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.

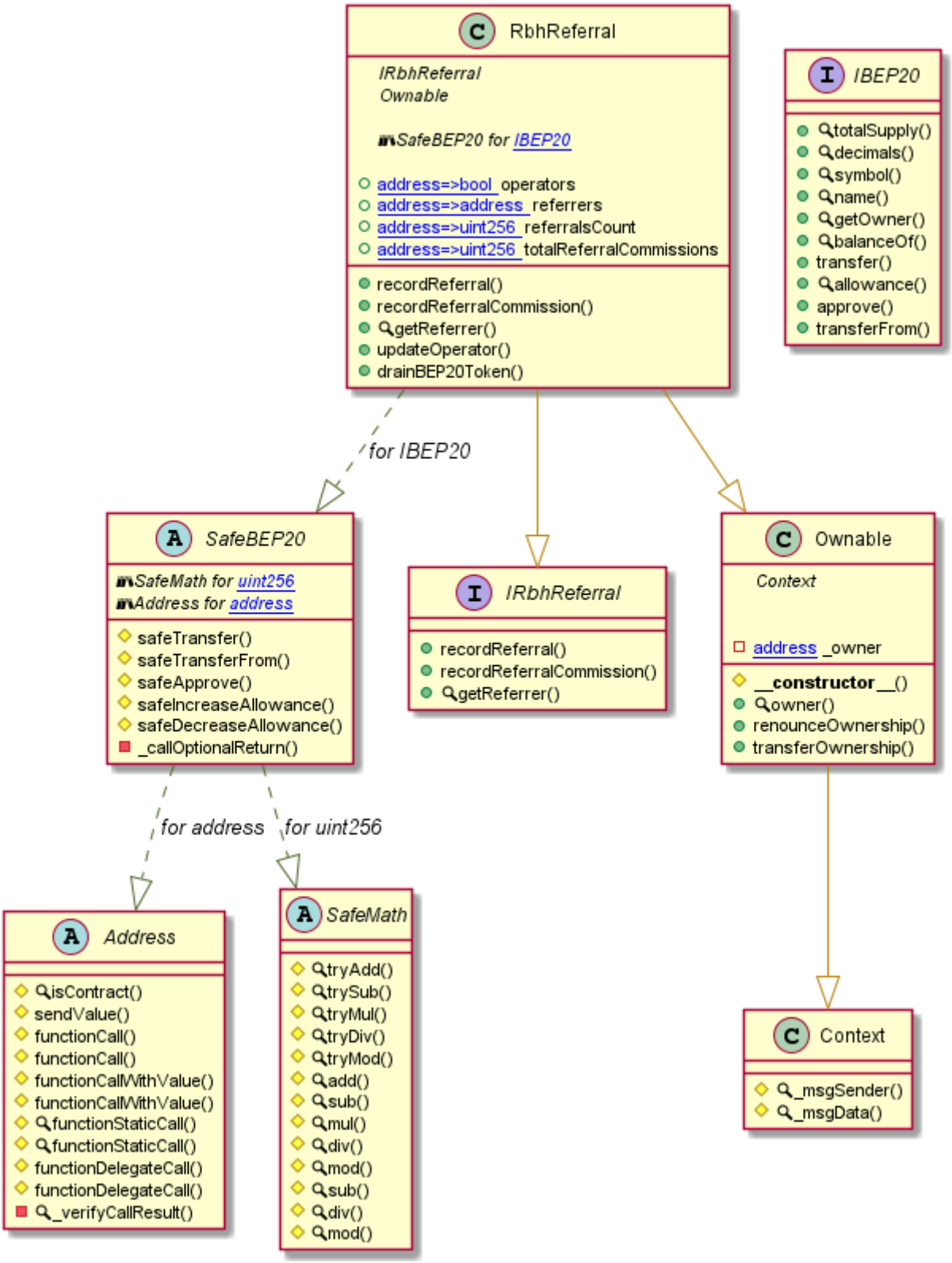
Appendix

Code Flow Diagram - RBH Protocol

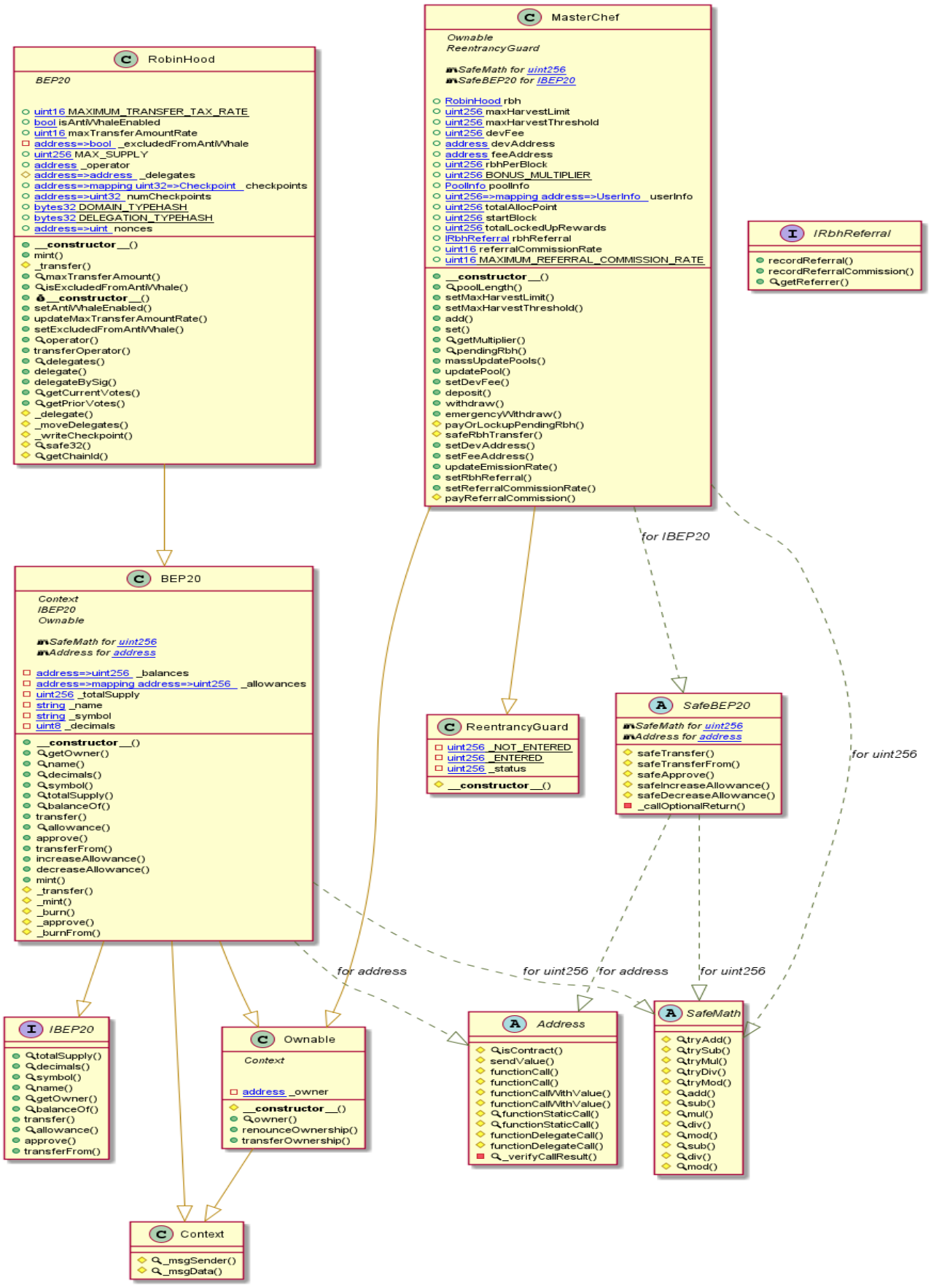
RobinHoodFarmTimelock Diagram



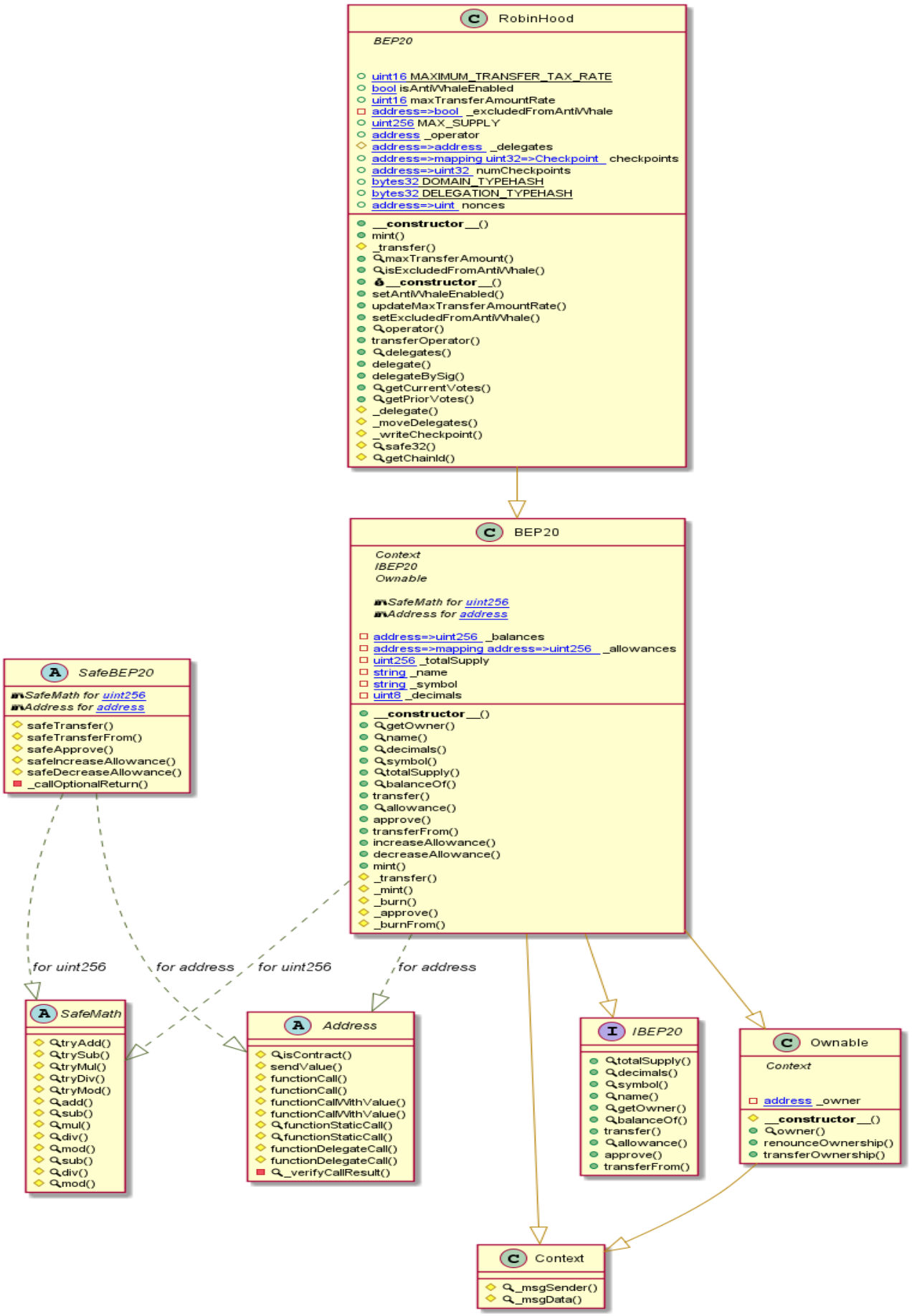
RbhReferral Diagram



MasterChef Diagram



RobinHood Diagram



Slither Results Log

Slither log >> RobinHoodFarmTimelock.sol

INFO:Detectors:
RobinHoodFarmTimelock.constructor(address,uint256).admin_ (RobinHoodFarmTimelock.sol#206) lacks a zero-check on :
- admin = admin_ (RobinHoodFarmTimelock.sol#210)
RobinHoodFarmTimelock.setPendingAdmin(address).pendingAdmin_ (RobinHoodFarmTimelock.sol#235) lacks a zero-check on :
- pendingAdmin = pendingAdmin_ (RobinHoodFarmTimelock.sol#243)
RobinHoodFarmTimelock.executeTransaction(address,uint256,string,bytes,uint256).target (RobinHoodFarmTimelock.sol#268) lacks a zero-check on :
- (success,returnData) = target.call.value(value)(callData) (RobinHoodFarmTimelock.sol#287)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in RobinHoodFarmTimelock.executeTransaction(address,uint256,string,bytes,uint256) (RobinHoodFarmTimelock.sol#268-293):
External calls:
- (success,returnData) = target.call.value(value)(callData) (RobinHoodFarmTimelock.sol#287)
Event emitted after the call(s):
- ExecuteTransaction(txHash,target,value,signature,data,eta) (RobinHoodFarmTimelock.sol#290)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
RobinHoodFarmTimelock.queueTransaction(address,uint256,string,bytes,uint256) (RobinHoodFarmTimelock.sol#248-257) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(eta >= getBlockTimestamp().add(delay),Timelock::queueTransaction: Estimated execution block must satisfy delay.) (RobinHoodFarmTimelock.sol#250)
RobinHoodFarmTimelock.executeTransaction(address,uint256,string,bytes,uint256) (RobinHoodFarmTimelock.sol#268-293) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(getBlockTimestamp() >= eta,Timelock::executeTransaction: Transaction hasn't surpassed time lock.) (RobinHoodFarmTimelock.sol#273)
- require(bool,string)(getBlockTimestamp() <= eta.add(GRACE_PERIOD),Timelock::executeTransaction: Transaction is stale.) (RobinHoodFarmTimelock.sol#274)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
SafeMath.div(uint256,uint256) (RobinHoodFarmTimelock.sol#100-102) is never used and should be removed
SafeMath.div(uint256,uint256,string) (RobinHoodFarmTimelock.sol#116-126) is never used and should be removed
SafeMath.min(uint256,uint256) (RobinHoodFarmTimelock.sol#165-167) is never used and should be removed
SafeMath.mod(uint256,uint256) (RobinHoodFarmTimelock.sol#140-142) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (RobinHoodFarmTimelock.sol#156-163) is never used and should be removed
SafeMath.mul(uint256,uint256) (RobinHoodFarmTimelock.sol#74-86) is never used and should be removed
SafeMath.sqrt(uint256) (RobinHoodFarmTimelock.sol#170-181) is never used and should be removed
SafeMath.sub(uint256,uint256) (RobinHoodFarmTimelock.sol#39-41) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (RobinHoodFarmTimelock.sol#53-62) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in RobinHoodFarmTimelock.executeTransaction(address,uint256,string,bytes,uint256) (RobinHoodFarmTimelock.sol#268-293):
- (success,returnData) = target.call.value(value)(callData) (RobinHoodFarmTimelock.sol#287)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:

Variable RobinHoodFarmTimelock.admin_initialized (RobinHoodFarmTimelock.sol#201) 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:
- RobinHoodFarmTimelock.setDelay(uint256) (RobinHoodFarmTimelock.sol#218-225)
acceptAdmin() should be declared external:
- RobinHoodFarmTimelock.acceptAdmin() (RobinHoodFarmTimelock.sol#227-233)
setPendingAdmin(address) should be declared external:
- RobinHoodFarmTimelock.setPendingAdmin(address) (RobinHoodFarmTimelock.sol#235-246)
queueTransaction(address,uint256,string,bytes,uint256) should be declared external:
- RobinHoodFarmTimelock.queueTransaction(address,uint256,string,bytes,uint256)
(RobinHoodFarmTimelock.sol#248-257)
cancelTransaction(address,uint256,string,bytes,uint256) should be declared external:
- RobinHoodFarmTimelock.cancelTransaction(address,uint256,string,bytes,uint256)
(RobinHoodFarmTimelock.sol#259-266)
executeTransaction(address,uint256,string,bytes,uint256) should be declared external:
- RobinHoodFarmTimelock.executeTransaction(address,uint256,string,bytes,uint256)
(RobinHoodFarmTimelock.sol#268-293)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RobinHoodFarmTimelock.sol analyzed (2 contracts with 75 detectors), 23 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Slither log >> RbhReferral.sol

INFO:Detectors:
Address.isContract(address) (RbhReferral.sol#23-32) uses assembly
- INLINE ASM (RbhReferral.sol#30)
Address._verifyCallResult(bool,bytes,string) (RbhReferral.sol#168-185) uses assembly
- INLINE ASM (RbhReferral.sol#177-180)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (RbhReferral.sol#76-78) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (RbhReferral.sol#101-103) is never used and should be removed
Address.functionDelegateCall(address,bytes) (RbhReferral.sol#150-152) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (RbhReferral.sol#160-166) is never used and should be removed
Address.functionStaticCall(address,bytes) (RbhReferral.sol#126-128) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (RbhReferral.sol#136-142) is never used and should be removed
Address.sendValue(address,uint256) (RbhReferral.sol#50-56) is never used and should be removed
Context._msgData() (RbhReferral.sol#586-589) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (RbhReferral.sol#310-324) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (RbhReferral.sol#335-345) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (RbhReferral.sol#326-333) is never used and should be removed
SafeBEP20.safeTransferFrom(IBEP20,address,address,uint256) (RbhReferral.sol#294-301) is never used and should be removed
SafeMath.add(uint256,uint256) (RbhReferral.sol#433-437) is never used and should be removed
SafeMath.div(uint256,uint256) (RbhReferral.sol#483-486) is never used and should be removed
SafeMath.div(uint256,uint256,string) (RbhReferral.sol#538-541) is never used and should be removed
SafeMath.mod(uint256,uint256) (RbhReferral.sol#500-503) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (RbhReferral.sol#558-561) is never used and should be removed
SafeMath.mul(uint256,uint256) (RbhReferral.sol#464-469) is never used and should be removed
SafeMath.sub(uint256,uint256) (RbhReferral.sol#449-452) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (RbhReferral.sol#518-521) is never used and should be removed

SafeMath.tryAdd(uint256,uint256) (RbhReferral.sol#372-376) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (RbhReferral.sol#408-411) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (RbhReferral.sol#418-421) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (RbhReferral.sol#393-401) is never used and should be removed
SafeMath.trySub(uint256,uint256) (RbhReferral.sol#383-386) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (RbhReferral.sol#50-56):
- (success) = recipient.call{value: amount}() (RbhReferral.sol#54)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (RbhReferral.sol#111-118):
- (success,returndata) = target.call{value: value}(data) (RbhReferral.sol#116)
Low level call in Address.functionStaticCall(address,bytes,string) (RbhReferral.sol#136-142):
- (success,returndata) = target.staticcall(data) (RbhReferral.sol#140)
Low level call in Address.functionDelegateCall(address,bytes,string) (RbhReferral.sol#160-166):
- (success,returndata) = target.delegatecall(data) (RbhReferral.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter RbhReferral.recordReferral(address,address)._user (RbhReferral.sol#661) is not in mixedCase
Parameter RbhReferral.recordReferral(address,address)._referrer (RbhReferral.sol#661) is not in mixedCase
Parameter RbhReferral.recordReferralCommission(address,uint256)._referrer (RbhReferral.sol#673) is not in mixedCase
Parameter RbhReferral.recordReferralCommission(address,uint256)._commission (RbhReferral.sol#673) is not in mixedCase
Parameter RbhReferral.getReferrer(address)._user (RbhReferral.sol#681) is not in mixedCase
Parameter RbhReferral.updateOperator(address,bool)._operator (RbhReferral.sol#686) is not in mixedCase
Parameter RbhReferral.updateOperator(address,bool)._status (RbhReferral.sol#686) is not in mixedCase
Parameter RbhReferral.drainBEP20Token(IBEP20,uint256,address)._token (RbhReferral.sol#692) is not in mixedCase
Parameter RbhReferral.drainBEP20Token(IBEP20,uint256,address)._amount (RbhReferral.sol#692) is not in mixedCase
Parameter RbhReferral.drainBEP20Token(IBEP20,uint256,address)._to (RbhReferral.sol#692) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (RbhReferral.sol#587)" inContext (RbhReferral.sol#581-590)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
owner() should be declared external:
- Ownable.owner() (RbhReferral.sol#608-610)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (RbhReferral.sol#627-630)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (RbhReferral.sol#636-640)
recordReferral(address,address) should be declared external:
- RbhReferral.recordReferral(address,address) (RbhReferral.sol#661-671)
recordReferralCommission(address,uint256) should be declared external:
- RbhReferral.recordReferralCommission(address,uint256) (RbhReferral.sol#673-678)
getReferrer(address) should be declared external:
- RbhReferral.getReferrer(address) (RbhReferral.sol#681-683)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RbhReferral.sol analyzed (8 contracts with 75 detectors), 48 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Slither log >> MasterChef.sol

```
INFO:Detectors:
MasterChef.safeRbhTransfer(address,uint256) (MasterChef.sol#1705-1712) ignores return value by
rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
MasterChef.safeRbhTransfer(address,uint256) (MasterChef.sol#1705-1712) ignores return value by
rbh.transfer(_to,_amount) (MasterChef.sol#1710)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
MasterChef.pendingRbh(uint256,address) (MasterChef.sol#1572-1584) performs a multiplication on the
result of a division:
        -rbhReward = multiplier.mul(rbhPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
(MasterChef.sol#1579)
        -accRbhPerShare = accRbhPerShare.add(rbhReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1580)
MasterChef.updatePool(uint256) (MasterChef.sol#1596-1616) performs a multiplication on the result of a
division:
        -rbhReward = multiplier.mul(rbhPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
(MasterChef.sol#1609)
        -rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
MasterChef.updatePool(uint256) (MasterChef.sol#1596-1616) performs a multiplication on the result of a
division:
        -rbhReward = multiplier.mul(rbhPerBlock).mul(pool.allocPoint).div(totalAllocPoint)
(MasterChef.sol#1609)
        -pool.accRbhPerShare = pool.accRbhPerShare.add(rbhReward.mul(1e12).div(lpSupply))
(MasterChef.sol#1612)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
RobinHood._writeCheckpoint(address,uint32,uint256,uint256) (MasterChef.sol#1369-1387) uses a
dangerous strict equality:
        - nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber
(MasterChef.sol#1379)
MasterChef.updatePool(uint256) (MasterChef.sol#1596-1616) uses a dangerous strict equality:
        - lpSupply == 0 || pool.allocPoint == 0 (MasterChef.sol#1604)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Contract locking ether found:
        Contract RobinHood (MasterChef.sol#1048-1399) has payable functions:
         - RobinHood.receive() (MasterChef.sol#1130)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Reentrancy in MasterChef.add(uint256,IBEP20,uint16,bool) (MasterChef.sol#1527-1553):
        External calls:
        - massUpdatePools() (MasterChef.sol#1542)
                - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
                - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
        State variables written after the call(s):
        - poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP))
(MasterChef.sol#1546-1552)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChef.sol#1545)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652):
        External calls:
        - updatePool(_pid) (MasterChef.sol#1632)
                - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
                - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
        - rbhReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1635)
        - payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
                - rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
                - rbh.transfer(_to,_amount) (MasterChef.sol#1710)
                - rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
        State variables written after the call(s):
```

- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
    - user.rewardLockedUp = 0 (MasterChef.sol#1694)

Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652):
External calls:
- updatePool(_pid) (MasterChef.sol#1632)
    - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
    - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- rbhReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1635)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
    - rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
    - rbh.transfer(_to,_amount) (MasterChef.sol#1710)
    - rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1640)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#1644)
State variables written after the call(s):
- user.amount = user.amount.add(_amount).sub(depositFee) (MasterChef.sol#1645)

Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652):
External calls:
- updatePool(_pid) (MasterChef.sol#1632)
    - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
    - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- rbhReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1635)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
    - rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
    - rbh.transfer(_to,_amount) (MasterChef.sol#1710)
    - rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1640)
State variables written after the call(s):
- user.amount = user.amount.add(_amount) (MasterChef.sol#1647)

Reentrancy in MasterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#1556-1564):
External calls:
- massUpdatePools() (MasterChef.sol#1559)
    - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
    - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
State variables written after the call(s):
- poolInfo[_pid].allocPoint = _allocPoint (MasterChef.sol#1562)
- poolInfo[_pid].depositFeeBP = _depositFeeBP (MasterChef.sol#1563)
- totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (MasterChef.sol#1561)

Reentrancy in MasterChef.updateEmissionRate(uint256) (MasterChef.sol#1728-1732):
External calls:
- massUpdatePools() (MasterChef.sol#1729)
    - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
    - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
State variables written after the call(s):
- rbhPerBlock = _rbhPerBlock (MasterChef.sol#1731)

Reentrancy in MasterChef.updatePool(uint256) (MasterChef.sol#1596-1616):
External calls:
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
State variables written after the call(s):
- pool.accRbhPerShare = pool.accRbhPerShare.add(rbhReward.mul(1e12).div(lpSupply)) (MasterChef.sol#1612)
- pool.lastRewardBlock = block.number (MasterChef.sol#1613)

Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1655-1667):
External calls:
- updatePool(_pid) (MasterChef.sol#1659)
    - rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
    - rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)
    - rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
    - rbh.transfer(_to,_amount) (MasterChef.sol#1710)
    - rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
State variables written after the call(s):
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)
    - user.rewardLockedUp = 0 (MasterChef.sol#1694)

- user.amount = user.amount.sub(_amount) (MasterChef.sol#1662)
Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1655-1667):
External calls:
- updatePool(_pid) (MasterChef.sol#1659)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1663)
State variables written after the call(s):
- user.rewardDebt = user.amount.mul(pool.accRbhPerShare).div(1e12) (MasterChef.sol#1665)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
BEP20.constructor(string,string).name (MasterChef.sol#785) shadows:
- BEP20.name() (MasterChef.sol#801-803) (function)
- IBEP20.name() (MasterChef.sol#528) (function)
BEP20.constructor(string,string).symbol (MasterChef.sol#785) shadows:
- BEP20.symbol() (MasterChef.sol#815-817) (function)
- IBEP20.symbol() (MasterChef.sol#523) (function)
BEP20.allowance(address,address).owner (MasterChef.sol#849) shadows:
- Ownable.owner() (MasterChef.sol#644-646) (function)
BEP20._approve(address,address,uint256).owner (MasterChef.sol#1021) shadows:
- Ownable.owner() (MasterChef.sol#644-646) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652):
External calls:
- updatePool(_pid) (MasterChef.sol#1632)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- rbhReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1635)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#1753)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
Reentrancy in MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652):
External calls:
- updatePool(_pid) (MasterChef.sol#1632)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- rbhReferral.recordReferral(msg.sender,_referrer) (MasterChef.sol#1635)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1638)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1640)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#1644)
Event emitted after the call(s):
- Deposit(msg.sender,_pid,_amount) (MasterChef.sol#1651)
Reentrancy in MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1670-1679):
External calls:
- pool.lpToken.safeTransfer(address(msg.sender),amount) (MasterChef.sol#1677)
Event emitted after the call(s):
- EmergencyWithdraw(msg.sender,_pid,amount) (MasterChef.sol#1678)
Reentrancy in MasterChef.payOrLockupPendingRbh(uint256) (MasterChef.sol#1682-1702):
External calls:
- safeRbhTransfer(msg.sender,totalRewards) (MasterChef.sol#1697)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- payReferralCommission(msg.sender,totalRewards) (MasterChef.sol#1699)

- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#1753)
- payReferralCommission(msg.sender,totalRewards) (MasterChef.sol#1699)

Reentrancy in MasterChef.payReferralCommission(address,uint256) (MasterChef.sol#1746-1756):
External calls:
- safeRbhTransfer(referrer,commissionAmount) (MasterChef.sol#1751)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#1753)

Reentrancy in MasterChef.updateEmissionRate(uint256) (MasterChef.sol#1728-1732):
External calls:
- massUpdatePools() (MasterChef.sol#1729)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)

Event emitted after the call(s):
- EmissionRateUpdated(msg.sender,rbhPerBlock,_rbhPerBlock) (MasterChef.sol#1730)

Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1655-1667):
External calls:
- updatePool(_pid) (MasterChef.sol#1659)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)

Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer,commissionAmount) (MasterChef.sol#1753)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)

Reentrancy in MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1655-1667):
External calls:
- updatePool(_pid) (MasterChef.sol#1659)
- rbh.mint(devAddress,rbhReward.mul(devFee).div(100)) (MasterChef.sol#1610)
- rbh.mint(address(this),rbhReward) (MasterChef.sol#1611)
- payOrLockupPendingRbh(_pid) (MasterChef.sol#1660)
- rbh.transfer(_to,rbhBal) (MasterChef.sol#1708)
- rbh.transfer(_to,_amount) (MasterChef.sol#1710)
- rbhReferral.recordReferralCommission(referrer,commissionAmount) (MasterChef.sol#1752)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1663)

Event emitted after the call(s):
- Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#1666)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
RobinHood.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (MasterChef.sol#1235-1276) uses timestamp for comparisons

Dangerous comparisons:
- require(bool,string)(now <= expiry,ROBINHOOD::delegateBySig: signature expired) (MasterChef.sol#1274)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (MasterChef.sol#22-31) uses assembly
- INLINE ASM (MasterChef.sol#29)

Address._verifyCallResult(bool,bytes,string) (MasterChef.sol#167-184) uses assembly
- INLINE ASM (MasterChef.sol#176-179)

RobinHood.getChainId() (MasterChef.sol#1394-1398) uses assembly
- INLINE ASM (MasterChef.sol#1396)

Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
RobinHood.antiWhale(address,address,uint256) (MasterChef.sol#1072-1082) compares to a boolean constant:

-maxTransferAmount() > 0 && isAntiWhaleEnabled == true (MasterChef.sol#1073)

RobinHood.antiWhale(address,address,uint256) (MasterChef.sol#1072-1082) compares to a boolean constant:
	-_excludedFromAntiWhale[sender] == false && _excludedFromAntiWhale[recipient] == false (MasterChef.sol#1075-1076)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address.functionCall(address,bytes) (MasterChef.sol#75-77) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (MasterChef.sol#100-102) is never used and should be removed
Address.functionDelegateCall(address,bytes) (MasterChef.sol#149-151) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#159-165) is never used and should be removed
Address.functionStaticCall(address,bytes) (MasterChef.sol#125-127) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (MasterChef.sol#135-141) is never used and should be removed
Address.sendValue(address,uint256) (MasterChef.sol#49-55) is never used and should be removed
BEP20._burn(address,uint256) (MasterChef.sol#999-1005) is never used and should be removed
BEP20._burnFrom(address,uint256) (MasterChef.sol#1038-1045) is never used and should be removed
Context._msgData() (MasterChef.sol#609-612) is never used and should be removed
RobinHood._transfer(address,address,uint256) (MasterChef.sol#1107-1112) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (MasterChef.sol#435-449) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#460-470) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (MasterChef.sol#451-458) is never used and should be removed
SafeMath.div(uint256,uint256,string) (MasterChef.sol#372-375) is never used and should be removed
SafeMath.mod(uint256,uint256) (MasterChef.sol#334-337) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (MasterChef.sol#392-395) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (MasterChef.sol#206-210) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (MasterChef.sol#242-245) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (MasterChef.sol#252-255) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (MasterChef.sol#227-235) is never used and should be removed
SafeMath.trySub(uint256,uint256) (MasterChef.sol#217-220) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (MasterChef.sol#49-55):
	- (success) = recipient.call{value: amount}() (MasterChef.sol#53)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (MasterChef.sol#110-117):
	- (success,returndata) = target.call{value: value}(data) (MasterChef.sol#115)
Low level call in Address.functionStaticCall(address,bytes,string) (MasterChef.sol#135-141):
	- (success,returndata) = target.staticcall(data) (MasterChef.sol#139)
Low level call in Address.functionDelegateCall(address,bytes,string) (MasterChef.sol#159-165):
	- (success,returndata) = target.delegatecall(data) (MasterChef.sol#163)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter RobinHood.mint(address,uint256)._to (MasterChef.sol#1099) is not in mixedCase
Parameter RobinHood.mint(address,uint256)._amount (MasterChef.sol#1099) is not in mixedCase
Parameter RobinHood.isExcludedFromAntiWhale(address)._account (MasterChef.sol#1125) is not in mixedCase
Parameter RobinHood.updateMaxTransferAmountRate(uint16)._maxTransferAmountRate (MasterChef.sol#1140) is not in mixedCase
Parameter RobinHood.setExcludedFromAntiWhale(address,bool)._account (MasterChef.sol#1151) is not in mixedCase
Parameter RobinHood.setExcludedFromAntiWhale(address,bool)._excluded (MasterChef.sol#1151) is not in mixedCase
Variable RobinHood.MAX_SUPPLY (MasterChef.sol#1056) is not in mixedCase
Variable RobinHood._operator (MasterChef.sol#1060) is not in mixedCase
Variable RobinHood._delegates (MasterChef.sol#1175) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._allocPoint (MasterChef.sol#1527) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._lpToken (MasterChef.sol#1527) is not in mixedCase

Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._depositFeeBP (MasterChef.sol#1527) is not in mixedCase
Parameter MasterChef.add(uint256,IBEP20,uint16,bool)._withUpdate (MasterChef.sol#1527) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._pid (MasterChef.sol#1556) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._allocPoint (MasterChef.sol#1556) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._depositFeeBP (MasterChef.sol#1556) is not in mixedCase
Parameter MasterChef.set(uint256,uint256,uint16,bool)._withUpdate (MasterChef.sol#1556) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._from (MasterChef.sol#1567) is not in mixedCase
Parameter MasterChef.getMultiplier(uint256,uint256)._to (MasterChef.sol#1567) is not in mixedCase
Parameter MasterChef.pendingRbh(uint256,address)._pid (MasterChef.sol#1572) is not in mixedCase
Parameter MasterChef.pendingRbh(uint256,address)._user (MasterChef.sol#1572) is not in mixedCase
Parameter MasterChef.updatePool(uint256)._pid (MasterChef.sol#1596) is not in mixedCase
Parameter MasterChef.setDevFee(uint256)._fee (MasterChef.sol#1621) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._pid (MasterChef.sol#1627) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._amount (MasterChef.sol#1627) is not in mixedCase
Parameter MasterChef.deposit(uint256,uint256,address)._referrer (MasterChef.sol#1627) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._pid (MasterChef.sol#1655) is not in mixedCase
Parameter MasterChef.withdraw(uint256,uint256)._amount (MasterChef.sol#1655) is not in mixedCase
Parameter MasterChef.emergencyWithdraw(uint256)._pid (MasterChef.sol#1670) is not in mixedCase
Parameter MasterChef.payOrLockupPendingRbh(uint256)._pid (MasterChef.sol#1682) is not in mixedCase
Parameter MasterChef.safeRbhTransfer(address,uint256)._to (MasterChef.sol#1705) is not in mixedCase
Parameter MasterChef.safeRbhTransfer(address,uint256)._amount (MasterChef.sol#1705) is not in mixedCase
Parameter MasterChef.setDevAddress(address)._devAddress (MasterChef.sol#1715) is not in mixedCase
Parameter MasterChef.setFeeAddress(address)._feeAddress (MasterChef.sol#1721) is not in mixedCase
Parameter MasterChef.updateEmissionRate(uint256)._rbhPerBlock (MasterChef.sol#1728) is not in mixedCase
Parameter MasterChef.setRbhReferral(IRbhReferral)._rbhReferral (MasterChef.sol#1735) is not in mixedCase
Parameter MasterChef.setReferralCommissionRate(uint16)._referralCommissionRate (MasterChef.sol#1740) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._user (MasterChef.sol#1746) is not in mixedCase
Parameter MasterChef.payReferralCommission(address,uint256)._pending (MasterChef.sol#1746) 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#610)" inContext (MasterChef.sol#604-613)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
RobinHood.slitherConstructorVariables() (MasterChef.sol#1048-1399) uses literals with too many digits:
	- MAX_SUPPLY = 32000000 * 10 ** 18 (MasterChef.sol#1056)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
RobinHood.MAX_SUPPLY (MasterChef.sol#1056) 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() (MasterChef.sol#663-666)
transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (MasterChef.sol#672-676)
decimals() should be declared external:
	- BEP20.decimals() (MasterChef.sol#808-810)
symbol() should be declared external:
	- BEP20.symbol() (MasterChef.sol#815-817)
transfer(address,uint256) should be declared external:
	- BEP20.transfer(address,uint256) (MasterChef.sol#841-844)

allowance(address,address) should be declared external:
	- BEP20.allowance(address,address) (MasterChef.sol#849-851)
approve(address,uint256) should be declared external:
	- BEP20.approve(address,uint256) (MasterChef.sol#860-863)
transferFrom(address,address,uint256) should be declared external:
	- BEP20.transferFrom(address,address,uint256) (MasterChef.sol#877-889)
increaseAllowance(address,uint256) should be declared external:
	- BEP20.increaseAllowance(address,uint256) (MasterChef.sol#903-906)
decreaseAllowance(address,uint256) should be declared external:
	- BEP20.decreaseAllowance(address,uint256) (MasterChef.sol#922-929)
mint(uint256) should be declared external:
	- BEP20.mint(uint256) (MasterChef.sol#939-942)
mint(address,uint256) should be declared external:
	- RobinHood.mint(address,uint256) (MasterChef.sol#1099-1105)
isExcludedFromAntiWhale(address) should be declared external:
	- RobinHood.isExcludedFromAntiWhale(address) (MasterChef.sol#1125-1127)
setAntiWhaleEnabled(bool) should be declared external:
	- RobinHood.setAntiWhaleEnabled(bool) (MasterChef.sol#1134-1136)
updateMaxTransferAmountRate(uint16) should be declared external:
	- RobinHood.updateMaxTransferAmountRate(uint16) (MasterChef.sol#1140-1144)
setExcludedFromAntiWhale(address,bool) should be declared external:
	- RobinHood.setExcludedFromAntiWhale(address,bool) (MasterChef.sol#1151-1153)
operator() should be declared external:
	- RobinHood.operator() (MasterChef.sol#1161-1163)
transferOperator(address) should be declared external:
	- RobinHood.transferOperator(address) (MasterChef.sol#1169-1173)
setMaxHarvestLimit(uint256) should be declared external:
	- MasterChef.setMaxHarvestLimit(uint256) (MasterChef.sol#1512-1514)
setMaxHarvestThreshold(uint256) should be declared external:
	- MasterChef.setMaxHarvestThreshold(uint256) (MasterChef.sol#1517-1519)
add(uint256,IBEP20,uint16,bool) should be declared external:
	- MasterChef.add(uint256,IBEP20,uint16,bool) (MasterChef.sol#1527-1553)
set(uint256,uint256,uint16,bool) should be declared external:
	- MasterChef.set(uint256,uint256,uint16,bool) (MasterChef.sol#1556-1564)
setDevFee(uint256) should be declared external:
	- MasterChef.setDevFee(uint256) (MasterChef.sol#1621-1623)
deposit(uint256,uint256,address) should be declared external:
	- MasterChef.deposit(uint256,uint256,address) (MasterChef.sol#1627-1652)
withdraw(uint256,uint256) should be declared external:
	- MasterChef.withdraw(uint256,uint256) (MasterChef.sol#1655-1667)
emergencyWithdraw(uint256) should be declared external:
	- MasterChef.emergencyWithdraw(uint256) (MasterChef.sol#1670-1679)
setDevAddress(address) should be declared external:
	- MasterChef.setDevAddress(address) (MasterChef.sol#1715-1719)
setFeeAddress(address) should be declared external:
	- MasterChef.setFeeAddress(address) (MasterChef.sol#1721-1725)
updateEmissionRate(uint256) should be declared external:
	- MasterChef.updateEmissionRate(uint256) (MasterChef.sol#1728-1732)
setRbhReferral(IRbhReferral) should be declared external:
	- MasterChef.setRbhReferral(IRbhReferral) (MasterChef.sol#1735-1737)
setReferralCommissionRate(uint16) should be declared external:
	- MasterChef.setReferralCommissionRate(uint16) (MasterChef.sol#1740-1743)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:MasterChef.sol analyzed (11 contracts with 75 detectors), 134 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Slither log >> RobinHood.sol

INFO:Detectors:
RobinHood._writeCheckpoint(address,uint32,uint256,uint256) (RobinHood.sol#1235-1253) uses a dangerous strict equality:
	- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (RobinHood.sol#1245)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Contract locking ether found:
	Contract RobinHood (RobinHood.sol#914-1266) has payable functions:
	 - RobinHood.receive() (RobinHood.sol#996)
	But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
BEP20.constructor(string,string).name (RobinHood.sol#651) shadows:
	- BEP20.name() (RobinHood.sol#667-669) (function)
	- IBEP20.name() (RobinHood.sol#553) (function)
BEP20.constructor(string,string).symbol (RobinHood.sol#651) shadows:
	- BEP20.symbol() (RobinHood.sol#681-683) (function)
	- IBEP20.symbol() (RobinHood.sol#548) (function)
BEP20.allowance(address,address).owner (RobinHood.sol#715) shadows:
	- Ownable.owner() (RobinHood.sol#413-415) (function)
BEP20._approve(address,address,uint256).owner (RobinHood.sol#887) shadows:
	- Ownable.owner() (RobinHood.sol#413-415) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
RobinHood.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (RobinHood.sol#1101-1142) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now <= expiry,ROBINHOOD::delegateBySig: signature expired) (RobinHood.sol#1140)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (RobinHood.sol#23-32) uses assembly
	- INLINE ASM (RobinHood.sol#30)
Address._verifyCallResult(bool,bytes,string) (RobinHood.sol#168-185) uses assembly
	- INLINE ASM (RobinHood.sol#177-180)
RobinHood.getChainId() (RobinHood.sol#1260-1264) uses assembly
	- INLINE ASM (RobinHood.sol#1262)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
RobinHood.antiWhale(address,address,uint256) (RobinHood.sol#938-948) compares to a boolean constant:
	-maxTransferAmount() > 0 && isAntiWhaleEnabled == true (RobinHood.sol#939)
RobinHood.antiWhale(address,address,uint256) (RobinHood.sol#938-948) compares to a boolean constant:
	-_excludedFromAntiWhale[sender] == false && _excludedFromAntiWhale[recipient] == false (RobinHood.sol#941-942)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address._verifyCallResult(bool,bytes,string) (RobinHood.sol#168-185) is never used and should be removed
Address.functionCall(address,bytes) (RobinHood.sol#76-78) is never used and should be removed
Address.functionCall(address,bytes,string) (RobinHood.sol#86-88) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (RobinHood.sol#101-103) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (RobinHood.sol#111-118) is never used and should be removed
Address.functionDelegateCall(address,bytes) (RobinHood.sol#150-152) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (RobinHood.sol#160-166) is never used and should be removed
Address.functionStaticCall(address,bytes) (RobinHood.sol#126-128) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (RobinHood.sol#136-142) is never used and should be removed
Address.isContract(address) (RobinHood.sol#23-32) is never used and should be removed
Address.sendValue(address,uint256) (RobinHood.sol#50-56) is never used and should be removed
BEP20._burn(address,uint256) (RobinHood.sol#865-871) is never used and should be removed

BEP20._burnFrom(address,uint256) (RobinHood.sol#904-911) is never used and should be removed
Context._msgData() (RobinHood.sol#192-195) is never used and should be removed
RobinHood._transfer(address,address,uint256) (RobinHood.sol#973-978) is never used and should be removed
SafeBEP20._callOptionalReturn(IBEP20,bytes) (RobinHood.sol#519-530) is never used and should be removed
SafeBEP20.safeApprove(IBEP20,address,uint256) (RobinHood.sol#476-490) is never used and should be removed
SafeBEP20.safeDecreaseAllowance(IBEP20,address,uint256) (RobinHood.sol#501-511) is never used and should be removed
SafeBEP20.safeIncreaseAllowance(IBEP20,address,uint256) (RobinHood.sol#492-499) is never used and should be removed
SafeBEP20.safeTransfer(IBEP20,address,uint256) (RobinHood.sol#452-458) is never used and should be removed
SafeBEP20.safeTransferFrom(IBEP20,address,address,uint256) (RobinHood.sol#460-467) is never used and should be removed
SafeMath.div(uint256,uint256,string) (RobinHood.sol#370-373) is never used and should be removed
SafeMath.mod(uint256,uint256) (RobinHood.sol#332-335) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (RobinHood.sol#390-393) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (RobinHood.sol#204-208) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (RobinHood.sol#240-243) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (RobinHood.sol#250-253) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (RobinHood.sol#225-233) is never used and should be removed
SafeMath.trySub(uint256,uint256) (RobinHood.sol#215-218) 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) (RobinHood.sol#50-56):
	- (success) = recipient.call{value: amount}() (RobinHood.sol#54)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (RobinHood.sol#111-118):
	- (success,returndata) = target.call{value: value}(data) (RobinHood.sol#116)
Low level call in Address.functionStaticCall(address,bytes,string) (RobinHood.sol#136-142):
	- (success,returndata) = target.staticcall(data) (RobinHood.sol#140)
Low level call in Address.functionDelegateCall(address,bytes,string) (RobinHood.sol#160-166):
	- (success,returndata) = target.delegatecall(data) (RobinHood.sol#164)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter RobinHood.mint(address,uint256)._to (RobinHood.sol#965) is not in mixedCase
Parameter RobinHood.mint(address,uint256)._amount (RobinHood.sol#965) is not in mixedCase
Parameter RobinHood.isExcludedFromAntiWhale(address)._account (RobinHood.sol#991) is not in mixedCase
Parameter RobinHood.updateMaxTransferAmountRate(uint16)._maxTransferAmountRate (RobinHood.sol#1006) is not in mixedCase
Parameter RobinHood.setExcludedFromAntiWhale(address,bool)._account (RobinHood.sol#1017) is not in mixedCase
Parameter RobinHood.setExcludedFromAntiWhale(address,bool)._excluded (RobinHood.sol#1017) is not in mixedCase
Variable RobinHood.MAX_SUPPLY (RobinHood.sol#922) is not in mixedCase
Variable RobinHood._operator (RobinHood.sol#926) is not in mixedCase
Variable RobinHood._delegates (RobinHood.sol#1041) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (RobinHood.sol#193)" inContext (RobinHood.sol#187-196)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
RobinHood.slitherConstructorVariables() (RobinHood.sol#914-1266) uses literals with too many digits:
	- MAX_SUPPLY = 32000000 * 10 ** 18 (RobinHood.sol#922)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
RobinHood.MAX_SUPPLY (RobinHood.sol#922) 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() (RobinHood.sol#432-435)

transferOwnership(address) should be declared external:
	- Ownable.transferOwnership(address) (RobinHood.sol#441-445)
decimals() should be declared external:
	- BEP20.decimals() (RobinHood.sol#674-676)
symbol() should be declared external:
	- BEP20.symbol() (RobinHood.sol#681-683)
transfer(address,uint256) should be declared external:
	- BEP20.transfer(address,uint256) (RobinHood.sol#707-710)
allowance(address,address) should be declared external:
	- BEP20.allowance(address,address) (RobinHood.sol#715-717)
approve(address,uint256) should be declared external:
	- BEP20.approve(address,uint256) (RobinHood.sol#726-729)
transferFrom(address,address,uint256) should be declared external:
	- BEP20.transferFrom(address,address,uint256) (RobinHood.sol#743-755)
increaseAllowance(address,uint256) should be declared external:
	- BEP20.increaseAllowance(address,uint256) (RobinHood.sol#769-772)
decreaseAllowance(address,uint256) should be declared external:
	- BEP20.decreaseAllowance(address,uint256) (RobinHood.sol#788-795)
mint(uint256) should be declared external:
	- BEP20.mint(uint256) (RobinHood.sol#805-808)
mint(address,uint256) should be declared external:
	- RobinHood.mint(address,uint256) (RobinHood.sol#965-971)
isExcludedFromAntiWhale(address) should be declared external:
	- RobinHood.isExcludedFromAntiWhale(address) (RobinHood.sol#991-993)
setAntiWhaleEnabled(bool) should be declared external:
	- RobinHood.setAntiWhaleEnabled(bool) (RobinHood.sol#1000-1002)
updateMaxTransferAmountRate(uint16) should be declared external:
	- RobinHood.updateMaxTransferAmountRate(uint16) (RobinHood.sol#1006-1010)
setExcludedFromAntiWhale(address,bool) should be declared external:
	- RobinHood.setExcludedFromAntiWhale(address,bool) (RobinHood.sol#1017-1019)
operator() should be declared external:
	- RobinHood.operator() (RobinHood.sol#1027-1029)
transferOperator(address) should be declared external:
	- RobinHood.transferOperator(address) (RobinHood.sol#1035-1039)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RobinHood.sol analyzed (8 contracts with 75 detectors), 75 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io