NEBULA
SOLUTIONS

Deliverable: Extensive Audit Report

Coinopolis

Smart Contract Review

Security Report

May 2021

Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out inthis publication are elaborated in the specific for only project.

The eNebula Solutions does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personacting on the Company's behalf may be held responsible for the use that may be made of the information contained herein.

Report Summary

Title	**Coinopolis Smart Contract Pre-Launch Audit**		
Project Owner	Coinopolis		
Type	Public		
Reviewed by	Vatsal Raychura	**Revision date**	31/05/2021
Approved by	eNebula Solutions Private Limited	**Approval date**	31/05/2021
		Nº Pages	**26**

Overview

Background

The Coinopolis requested that eNebula Solutions perform a Extensive Smart Contract audit of the CoinopolisContracts.

Project Dates

The following is the project schedule for this review and report:

- **May 31**: Smart Contract Review Completed *(Completed)*
- **May 31**: Delivery of Smart Contract Audit Report *(Completed)*

Review Team

The following eNebula Solutions team member participated in this review:

- Vatsal Raychura, Security Researcher and Engineer

Coverage

Target Specification and Revision

For this audit, we performed research, investigation, and review of the smart contract of Coinopolis.

The following documentation repositories were considered in-scope for the review:

- Ether1Project: https://github.com/mattarad/CoinopolisContracts/blob/main/MasterChef.sol

Main Areas of Security Concern

Our investigation mainly focused on the following security areas:

- Considerations for Auth
- Considerations for CSRF
- Considerations for Command Injection
- Considerations for Cookies
- Considerations for Cryptography
- Considerations for DoS
- Considerations for File access
- Considerations for HTTP
- Considerations for Input Validation
- Considerations for Insecure Modules Libraries
- Considerations for Insecure Storage
- Considerations for Malicious Code
- Considerations for Mass Assignment

- Considerations for Regex
- Considerations for Routes
- Considerations for SQL Injection
- Considerations for SSL
- Considerations for Unexpected Behavior
- Considerations for Visibility
- Considerations for XSS
- Others.

Introduction

Given the opportunity to review the CoinopolisContracts related smart contract source code, we in the report outline our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts is ready to launch there are no issues found related to business logic, security or performance.

About Coinopolis: -

Item	Description
Issuer	Coinopolis
Website	https://coinopolis.io/
Type	CoinopolisContracts
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	May 31, 2021

The Full List of Check Items:

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security

Advanced DeFi Scrutiny	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

Common Weakness Enumeration (CWE) Classifications Used in This Audit:

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not securitysoftware.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values,Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code,or if the application does not handle all possible return/statuscodes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.

Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. Theymay not directly introduce a vulnerability, but indicate theproduct has not been carefully developed or maintained.

Findings

Summary

Here is a summary of our findings after analyzing the Coinopolis implementation. During the first phase of our audit, we studied the smart contract source code and ran our in-house static code analyzer through the Specific tool. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

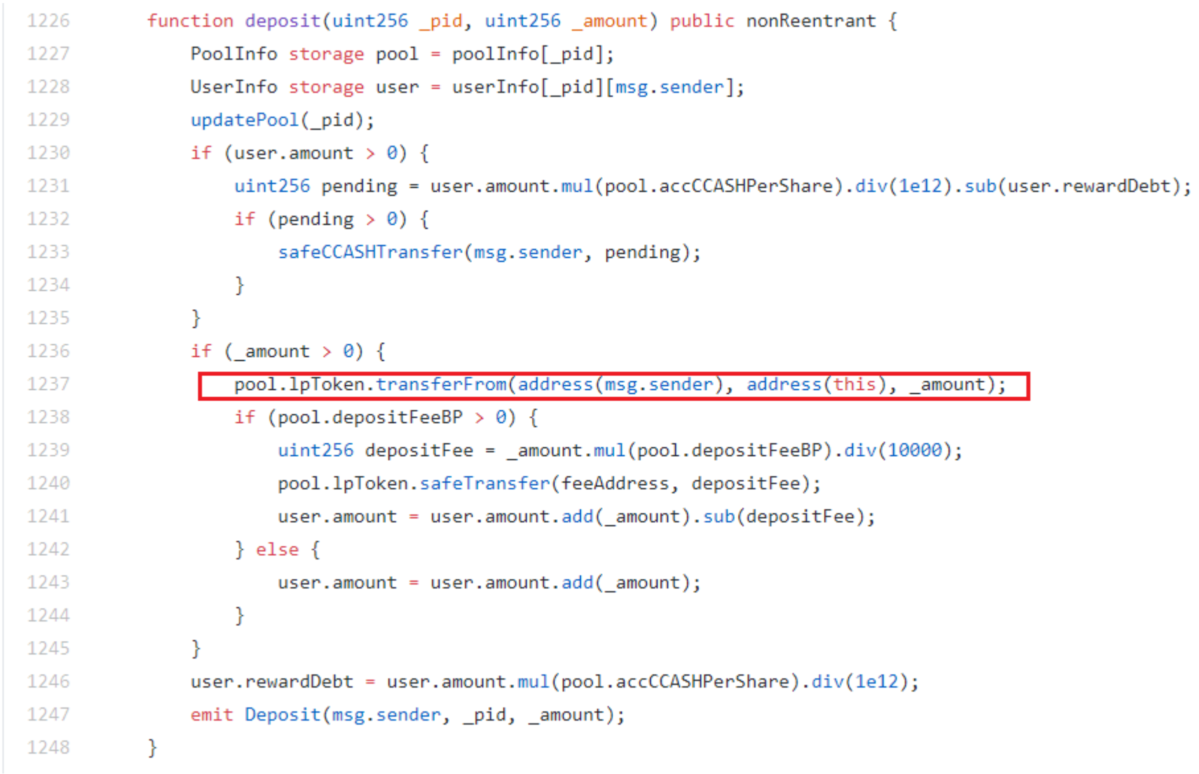
Severity	# of Findings	
Critical	0	
High	1	
Medium	11	
Low	6	
Total	18	

We have so far identified that there are no potential issues with severity of Critical, High, Medium, or even Low. Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by common recommendations.

Detailed Results

Basic Coding Bugs

1. Unchecked Transfer
    - Severity: High
    - Result: Found
    - Affected file: MasterChef.sol
    - Description: The return value of an external file transfer/transferFrom call is not checked.
    - POC:

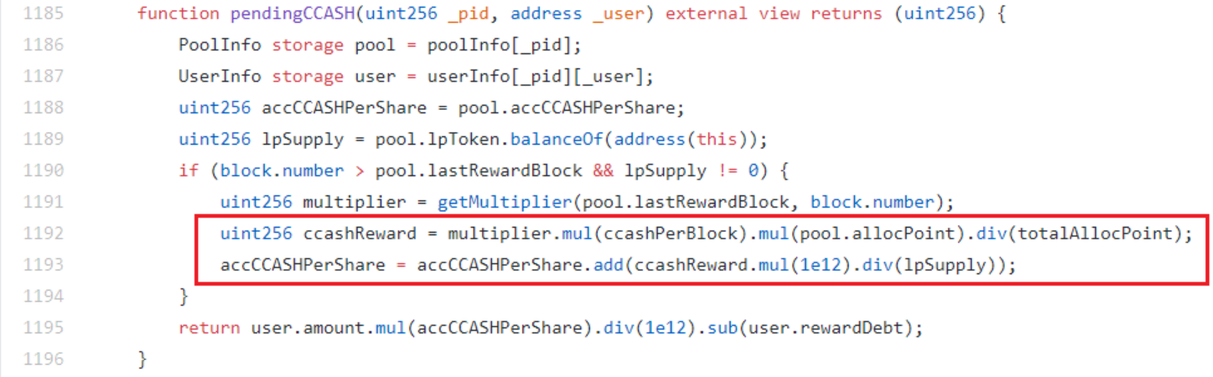


```
    function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
            if (pending > 0) {
                safeCCASHTransfer(msg.sender, pending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.transferFrom(address(msg.sender), address(this), _amount);
            if (pool.depositFeeBP > 0) {
                uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
                pool.lpToken.safeTransfer(feeAddress, depositFee);
                user.amount = user.amount.add(_amount).sub(depositFee);
            } else {
                user.amount = user.amount.add(_amount);
            }
        }
        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }
```

- Recommendation: Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

2. Divide before Multiply

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

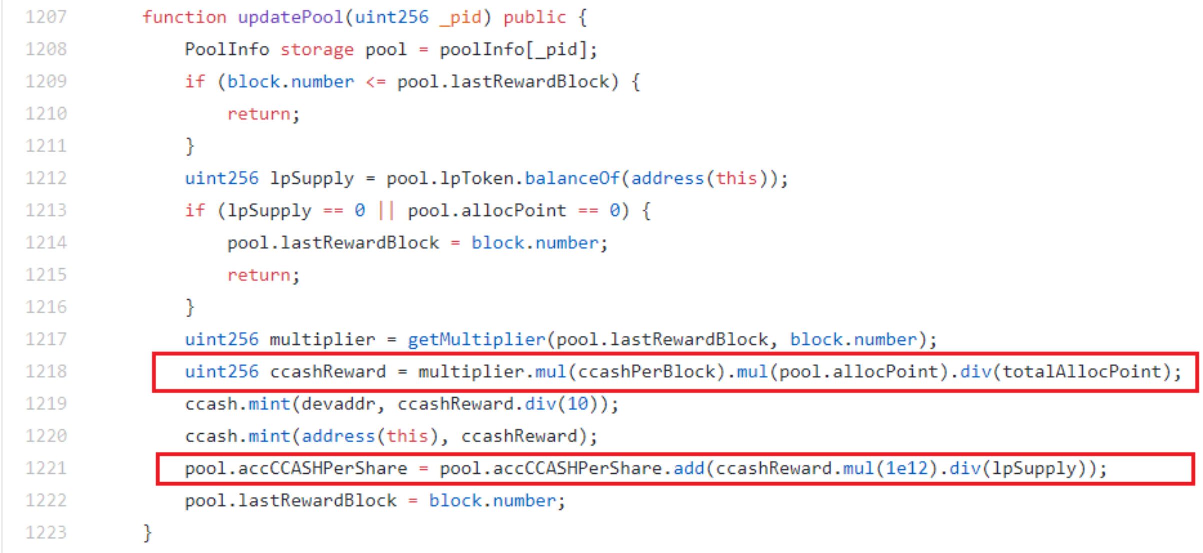


```
    function pendingCCASH(uint256 _pid, address _user) external view returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accCCASHPerShare = pool.accCCASHPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (block.number > pool.lastRewardBlock && lpSupply != 0) {
            uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
            uint256 ccashReward = multiplier.mul(ccashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
            accCCASHPerShare = accCCASHPerShare.add(ccashReward.mul(1e12).div(lpSupply));
        }
        return user.amount.mul(accCCASHPerShare).div(1e12).sub(user.rewardDebt);
    }
```

- Recommendation: Consider ordering multiplication before division.

3. Divide before Multiply

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Solidity integer division might truncate. As a result, performing multiplication before division can sometimes avoid loss of precision.
- POC:

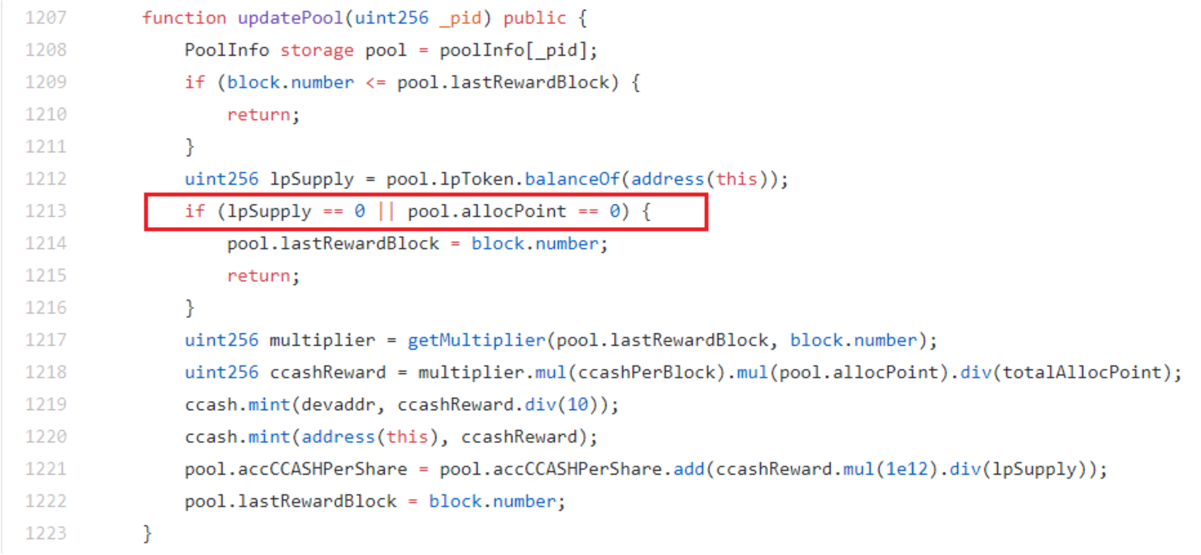


```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0 || pool.allocPoint == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
        uint256 ccashReward = multiplier.mul(ccashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        ccash.mint(devaddr, ccashReward.div(10));
        ccash.mint(address(this), ccashReward);
        pool.accCCASHPerShare = pool.accCCASHPerShare.add(ccashReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }
```

- Recommendation: Consider ordering multiplication before division.

4. Dangerous strict equalities

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Use the strict equalities that can be easily manipulated by an attacker.
- POC:



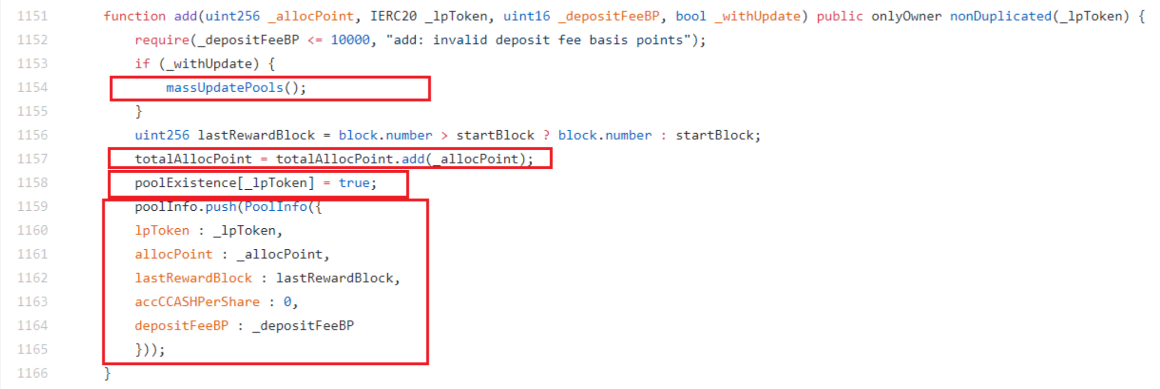
```
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0 || pool.allocPoint == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
        uint256 ccashReward = multiplier.mul(ccashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        ccash.mint(devaddr, ccashReward.div(10));
        ccash.mint(address(this), ccashReward);
        pool.accCCASHPerShare = pool.accCCASHPerShare.add(ccashReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }
```

- Recommendation: Don’t use strict equality to determine if an account has enough Ether or tokens.

5. Reentrancy vulnerabilities

    - Severity: Medium
    - Result: Found
    - Affected file: MasterChef.sol
    - Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
    - POC:
      External calls:
        - massUpdatePools() (MasterChef.sol#1154)
            - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
            - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)

      State variables written after the call(s):
        - poolExistence[_lpToken] = true (MasterChef.sol#1158)
        - poolInfo.push(PoolInfo(_lpToken,_allocPoint,lastRewardBlock,0,_depositFeeBP)) (MasterChef.sol#1159-1165)
        - totalAllocPoint = totalAllocPoint.add(_allocPoint) (MasterChef.sol#1157)



```
1151    function add(uint256 _allocPoint, IERC20 _lpToken, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner nonDuplicated(_lpToken) {
1152        require(_depositFeeBP <= 10000, "add: invalid deposit fee basis points");
1153        if (_withUpdate) {
1154            massUpdatePools();
1155        }
1156        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
1157        totalAllocPoint = totalAllocPoint.add(_allocPoint);
1158        poolExistence[_lpToken] = true;
1159        poolInfo.push(PoolInfo({
1160        lpToken : _lpToken,
1161        allocPoint : _allocPoint,
1162        lastRewardBlock : lastRewardBlock,
1163        accCCASHPerShare : 0,
1164        depositFeeBP : _depositFeeBP
1165        }));
1166    }
```

    - Recommendation: Apply the check-effects-interactions pattern.

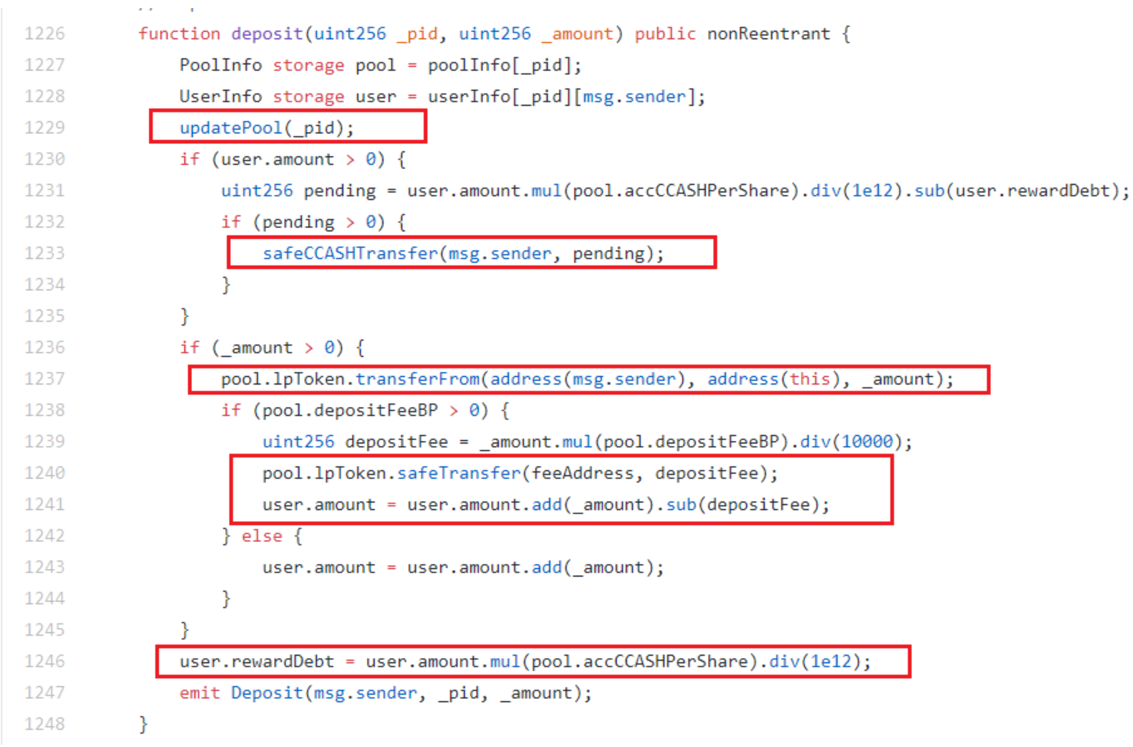
6. Reentrancy vulnerabilities

    - Severity: Medium
    - Result: Found
    - Affected file: MasterChef.sol
    - Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
    - POC:
      External calls:
        - updatePool(_pid) (MasterChef.sol#1229)
            - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
            - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
        - safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1233)
            - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
            - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)

- pool.lpToken.transferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1237)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#1240)

State variables written after the call(s):

- user.amount = user.amount.add(_amount).sub(depositFee) (MasterChef.sol#1241)
- user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12) (MasterChef.sol#1246)



```
1226    function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
1227        PoolInfo storage pool = poolInfo[_pid];
1228        UserInfo storage user = userInfo[_pid][msg.sender];
1229        updatePool(_pid);
1230        if (user.amount > 0) {
1231            uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
1232            if (pending > 0) {
1233                safeCCASHTransfer(msg.sender, pending);
1234            }
1235        }
1236        if (_amount > 0) {
1237            pool.lpToken.transferFrom(address(msg.sender), address(this), _amount);
1238            if (pool.depositFeeBP > 0) {
1239                uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1240                pool.lpToken.safeTransfer(feeAddress, depositFee);
1241                user.amount = user.amount.add(_amount).sub(depositFee);
1242            } else {
1243                user.amount = user.amount.add(_amount);
1244            }
1245        }
1246        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
1247        emit Deposit(msg.sender, _pid, _amount);
1248    }
```

- Recommendation: Apply the check-effects-interactions pattern.

7. Reentrancy vulnerabilities

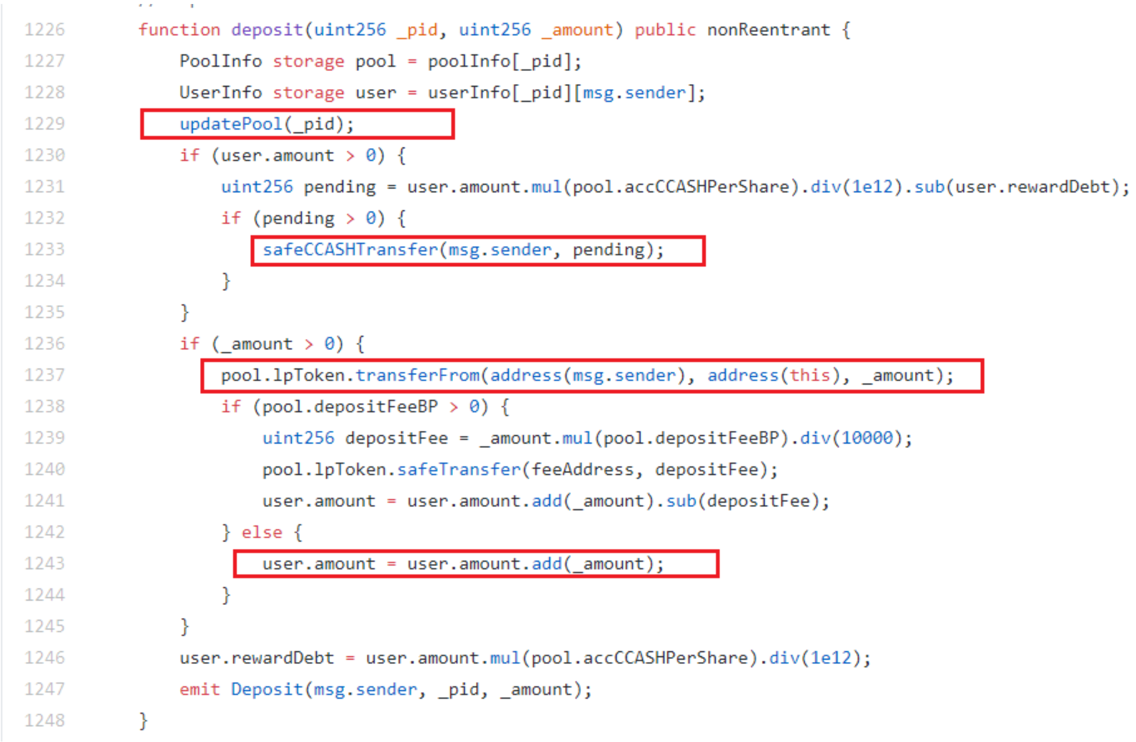
- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- POC:

External calls:

- updatePool(_pid) (MasterChef.sol#1229)
  - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
  - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
- safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1233)
  - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
  - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)
- pool.lpToken.transferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1237)

State variables written after the call(s):

- user.amount = user.amount.add(_amount) (MasterChef.sol#1243)



```
1226    function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
1227        PoolInfo storage pool = poolInfo[_pid];
1228        UserInfo storage user = userInfo[_pid][msg.sender];
1229        updatePool(_pid);
1230        if (user.amount > 0) {
1231            uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
1232            if (pending > 0) {
1233                safeCCASHTransfer(msg.sender, pending);
1234            }
1235        }
1236        if (_amount > 0) {
1237            pool.lpToken.transferFrom(address(msg.sender), address(this), _amount);
1238            if (pool.depositFeeBP > 0) {
1239                uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1240                pool.lpToken.safeTransfer(feeAddress, depositFee);
1241                user.amount = user.amount.add(_amount).sub(depositFee);
1242            } else {
1243                user.amount = user.amount.add(_amount);
1244            }
1245        }
1246        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
1247        emit Deposit(msg.sender, _pid, _amount);
1248    }
```

- ○ Recommendation: Apply the check-effects-interactions pattern.

8. Reentrancy vulnerabilities

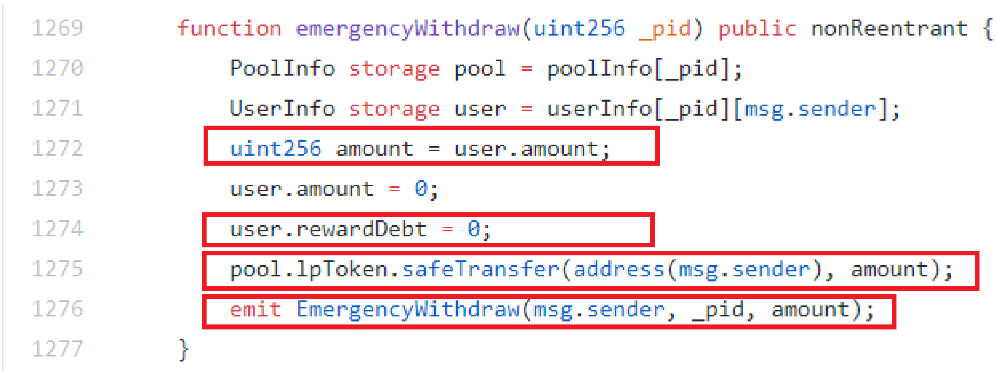
- ○ Severity: Medium
- ○ Result: Found
- ○ Affected file: MasterChef.sol
- ○ Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- ○ POC:

  External calls:

  - massUpdatePools() (MasterChef.sol#1172)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)

  State variables written after the call(s):

  - poolInfo[_pid].allocPoint = _allocPoint (MasterChef.sol#1175)
  - poolInfo[_pid].depositFeeBP = _depositFeeBP (MasterChef.sol#1176)
  - totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint) (MasterChef.sol#1174)



```
    function emergencyWithdraw(uint256 _pid) public nonReentrant {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }
```

- Recommendation: Apply the check-effects-interactions pattern.

9. Reentrancy vulnerabilities

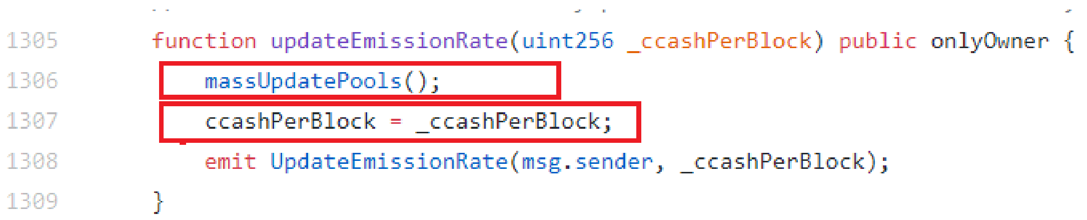
- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- POC:

External calls:

- massUpdatePools() (MasterChef.sol#1306)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)

State variables written after the call(s):

- ccashPerBlock = _ccashPerBlock (MasterChef.sol#1307)



```
1305    function updateEmissionRate(uint256 _ccashPerBlock) public onlyOwner {
1306        massUpdatePools();
1307        ccashPerBlock = _ccashPerBlock;
1308        emit UpdateEmissionRate(msg.sender, _ccashPerBlock);
1309    }
```

- Recommendation: Apply the check-effects-interactions pattern.

10. Reentrancy vulnerabilities

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- POC:

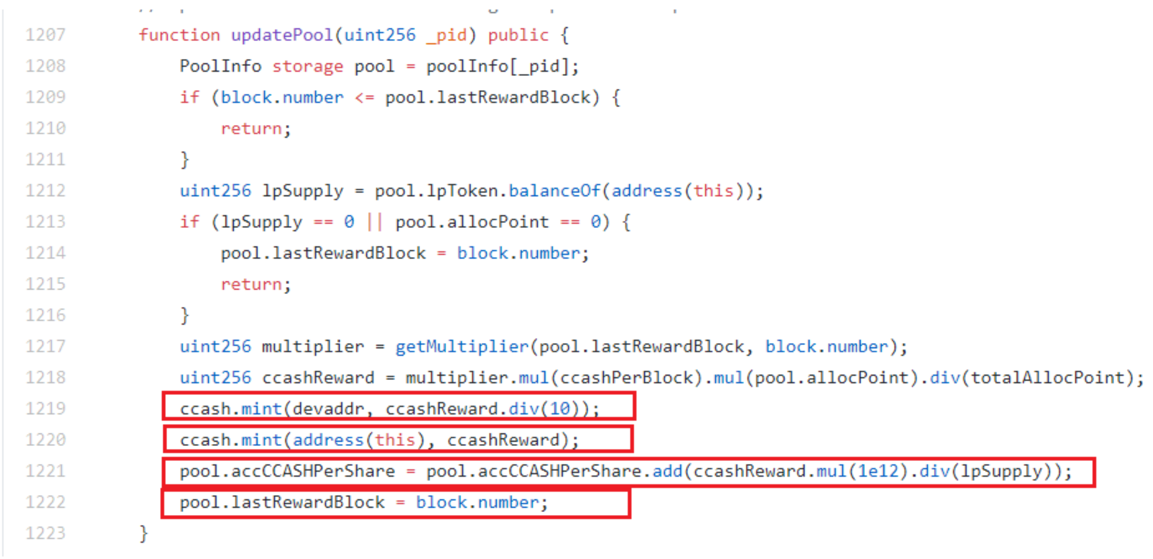
External calls:

- ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
- ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)

State variables written after the call(s):

- pool.accCCASHPerShare = pool.accCCASHPerShare.add(ccashReward.mul(1e12).div(lpSupply))

(MasterChef.sol#1221)
- pool.lastRewardBlock = block.number (MasterChef.sol#1222)



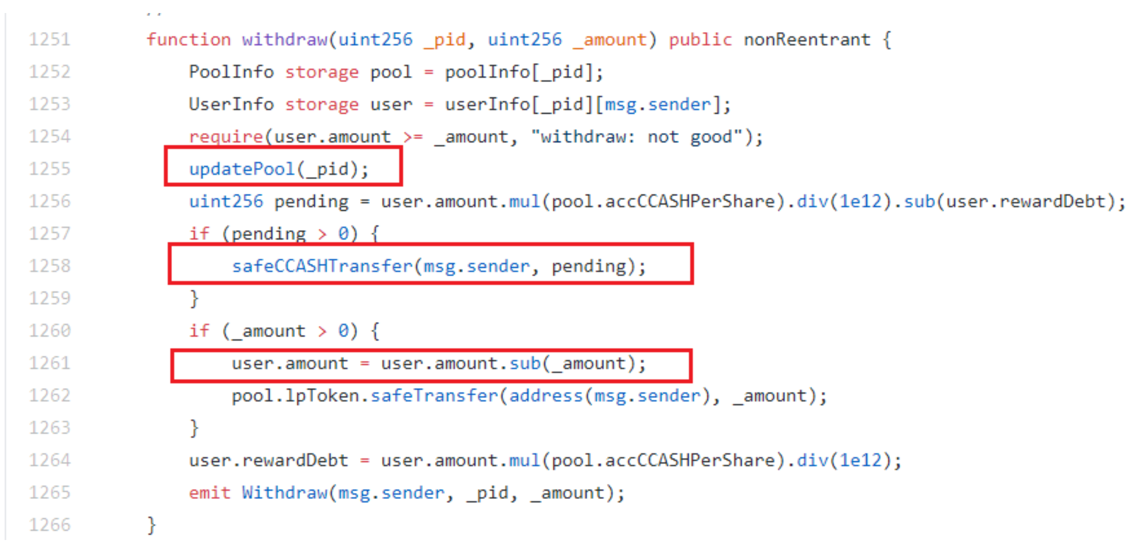
```
1207    function updatePool(uint256 _pid) public {
1208        PoolInfo storage pool = poolInfo[_pid];
1209        if (block.number <= pool.lastRewardBlock) {
1210            return;
1211        }
1212        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
1213        if (lpSupply == 0 || pool.allocPoint == 0) {
1214            pool.lastRewardBlock = block.number;
1215            return;
1216        }
1217        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
1218        uint256 ccashReward = multiplier.mul(ccashPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
1219        ccash.mint(devaddr, ccashReward.div(10));
1220        ccash.mint(address(this), ccashReward);
1221        pool.accCCASHPerShare = pool.accCCASHPerShare.add(ccashReward.mul(1e12).div(lpSupply));
1222        pool.lastRewardBlock = block.number;
1223    }
```

- Recommendation: Apply the check-effects-interactions pattern.

11. Reentrancy vulnerabilities

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- POC:
  External calls:
  - updatePool(_pid) (MasterChef.sol#1255)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
  - safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1258)
    - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
    - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)

  State variables written after the call(s):
  - user.amount = user.amount.sub(_amount) (MasterChef.sol#1261)



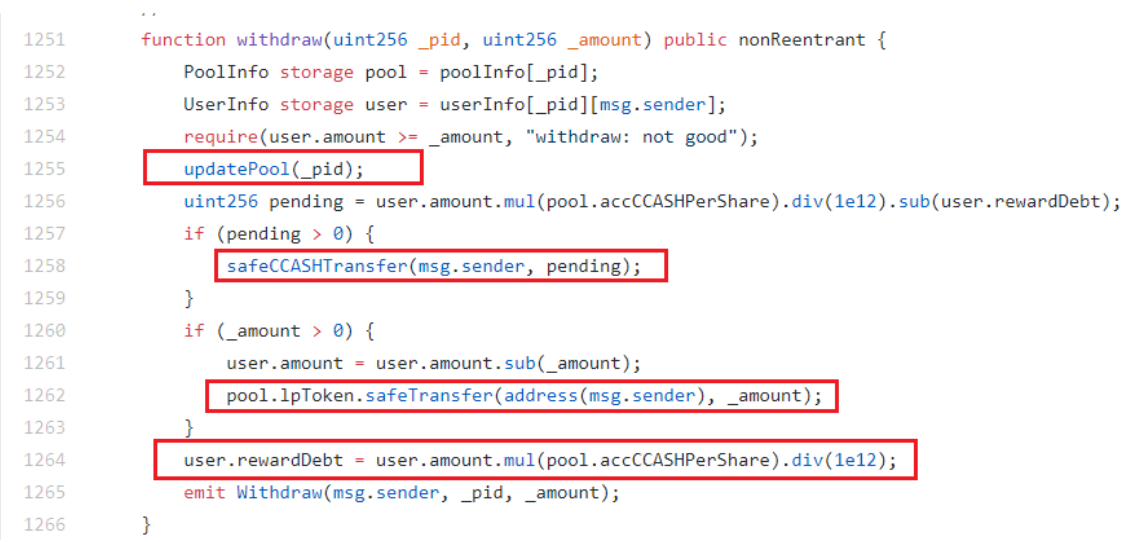
```
1251    function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
1252        PoolInfo storage pool = poolInfo[_pid];
1253        UserInfo storage user = userInfo[_pid][msg.sender];
1254        require(user.amount >= _amount, "withdraw: not good");
1255        updatePool(_pid);
1256        uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
1257        if (pending > 0) {
1258            safeCCASHTransfer(msg.sender, pending);
1259        }
1260        if (_amount > 0) {
1261            user.amount = user.amount.sub(_amount);
1262            pool.lpToken.safeTransfer(address(msg.sender), _amount);
1263        }
1264        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
1265        emit Withdraw(msg.sender, _pid, _amount);
1266    }
```

- Recommendation: Apply the check-effects-interactions pattern.

12. Reentrancy vulnerabilities

- Severity: Medium
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Do not report reentrancies that involve Ether (see reentrancy -eth).
- POC:
  External calls:
  - updatePool(_pid) (MasterChef.sol#1255)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
  - safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1258)
    - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
    - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)
  - pool.lpToken.safeTransfer(address(msg.sender),_amount) (MasterChef.sol#1262)

  State variables written after the call(s):
  - user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12) (MasterChef.sol#1264)

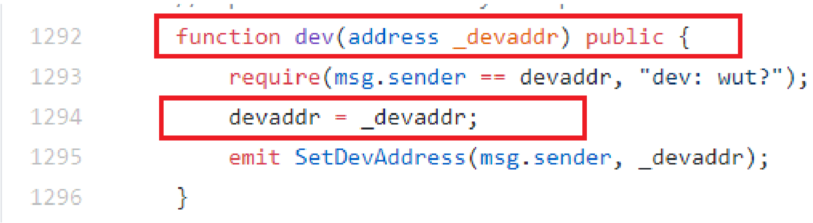


```
function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    require(user.amount >= _amount, "withdraw: not good");
    updatePool(_pid);
    uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
    if (pending > 0) {
        safeCCASHTransfer(msg.sender, pending);
    }
    if (_amount > 0) {
        user.amount = user.amount.sub(_amount);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
    }
    user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
    emit Withdraw(msg.sender, _pid, _amount);
}
```

- Recommendation: Apply the check-effects-interactions pattern.

13. Missing zero address validation

- Severity: Low
- Result: Found
- Affected file: MasterChef.sol
- Description: Detect missing zero address validation.
- POC:

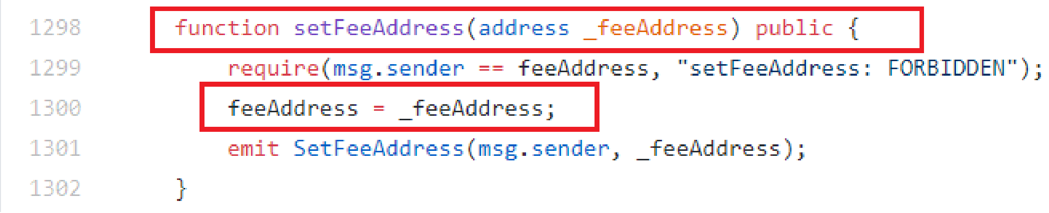


```
function dev(address _devaddr) public {
    require(msg.sender == devaddr, "dev: wut?");
    devaddr = _devaddr;
    emit SetDevAddress(msg.sender, _devaddr);
}
```

- Recommendation: Check that the address is not zero.

14. Missing zero address validation

- Severity: Low
- Result: Found
- Affected file: MasterChef.sol
- Description: Detect missing zero address validation.
- POC:



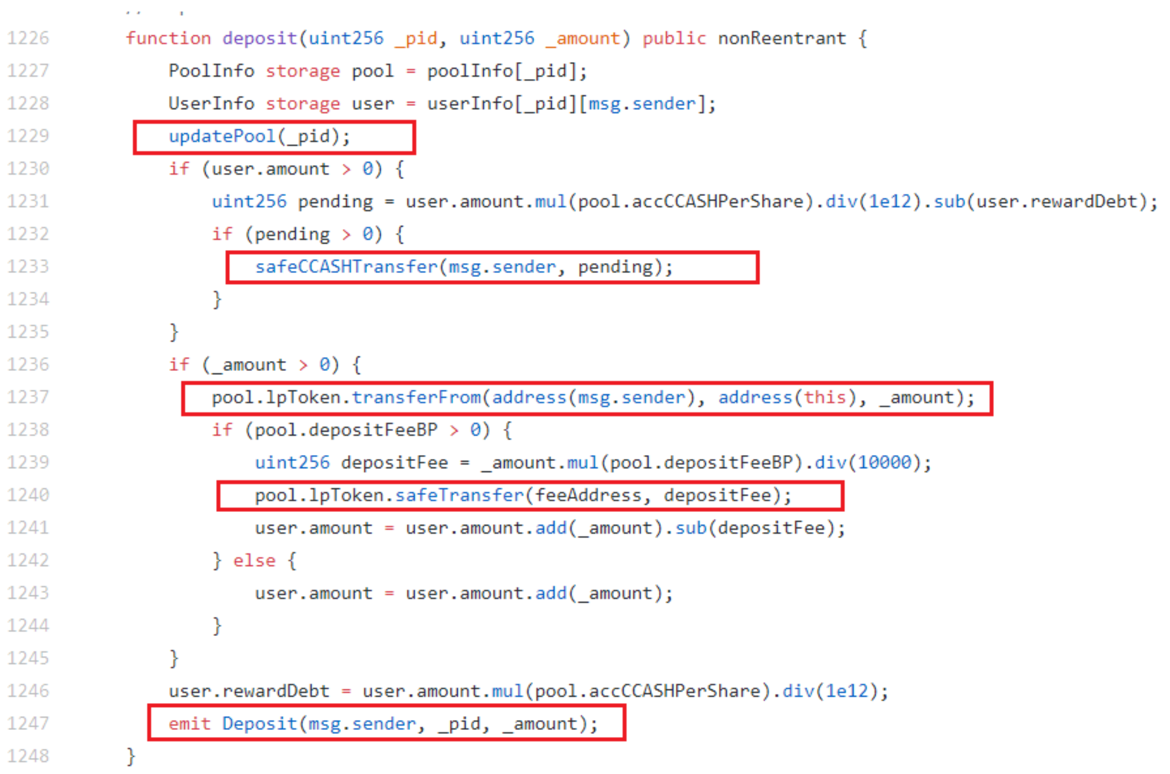
```
function setFeeAddress(address _feeAddress) public {
    require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
    feeAddress = _feeAddress;
    emit SetFeeAddress(msg.sender, _feeAddress);
}
```

- Recommendation: Check that the address is not zero.

15. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.
- POC:
  External calls:
  - updatePool(_pid) (MasterChef.sol#1229)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
  - safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1233)
    - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
    - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)
  - pool.lpToken.transferFrom(address(msg.sender),address(this),_amount) (MasterChef.sol#1237)
  - pool.lpToken.safeTransfer(feeAddress,depositFee) (MasterChef.sol#1240)

  Event emitted after the call(s):
  - Deposit(msg.sender,_pid,_amount) (MasterChef.sol#1247)



```
1226    function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
1227        PoolInfo storage pool = poolInfo[_pid];
1228        UserInfo storage user = userInfo[_pid][msg.sender];
1229        updatePool(_pid);
1230        if (user.amount > 0) {
1231            uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
1232            if (pending > 0) {
1233                safeCCASHTransfer(msg.sender, pending);
1234            }
1235        }
1236        if (_amount > 0) {
1237            pool.lpToken.transferFrom(address(msg.sender), address(this), _amount);
1238            if (pool.depositFeeBP > 0) {
1239                uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
1240                pool.lpToken.safeTransfer(feeAddress, depositFee);
1241                user.amount = user.amount.add(_amount).sub(depositFee);
1242            } else {
1243                user.amount = user.amount.add(_amount);
1244            }
1245        }
1246        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
1247        emit Deposit(msg.sender, _pid, _amount);
1248    }
```

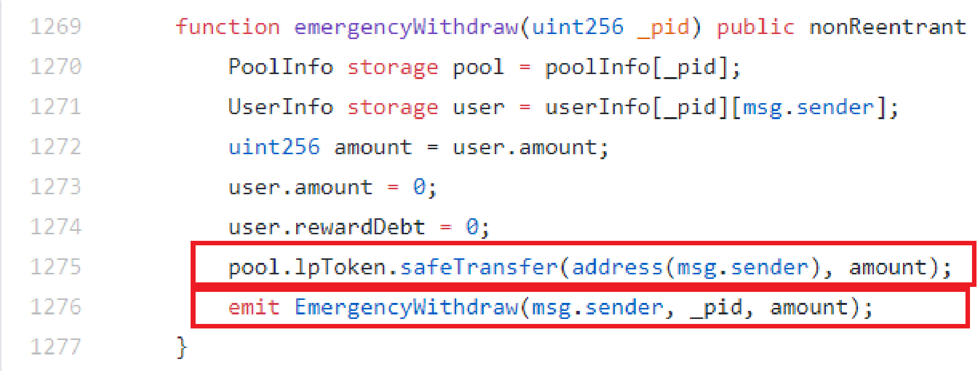
- Recommendation: Apply the check-effects-interactions pattern.

16. Reentrancy vulnerabilities

- Severity: Low

- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.
- POC:
  External calls:
  - pool.lpToken.safeTransfer(address(msg.sender),amount) (MasterChef.sol#1275)

  Event emitted after the call(s):
  - EmergencyWithdraw(msg.sender,_pid,amount) (MasterChef.sol#1276)



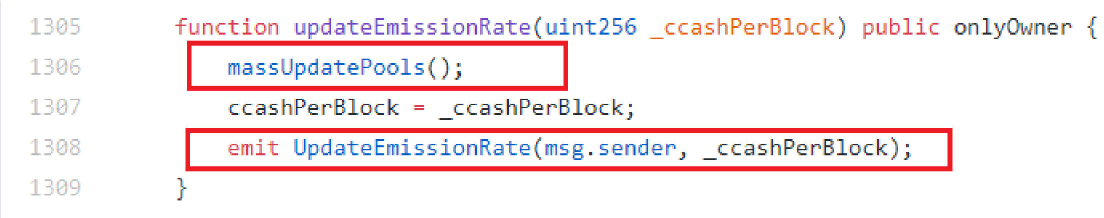
```
1269      function emergencyWithdraw(uint256 _pid) public nonReentrant {
1270          PoolInfo storage pool = poolInfo[_pid];
1271          UserInfo storage user = userInfo[_pid][msg.sender];
1272          uint256 amount = user.amount;
1273          user.amount = 0;
1274          user.rewardDebt = 0;
1275          pool.lpToken.safeTransfer(address(msg.sender), amount);
1276          emit EmergencyWithdraw(msg.sender, _pid, amount);
1277      }
```

- Recommendation: Apply the check-effects-interactions pattern.

17. Reentrancy vulnerabilities

- Severity: Low
- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.
- POC:
  External calls:
  - massUpdatePools() (MasterChef.sol#1306)
    - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
    - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)

  Event emitted after the call(s):
  - UpdateEmissionRate(msg.sender,_ccashPerBlock) (MasterChef.sol#1308)



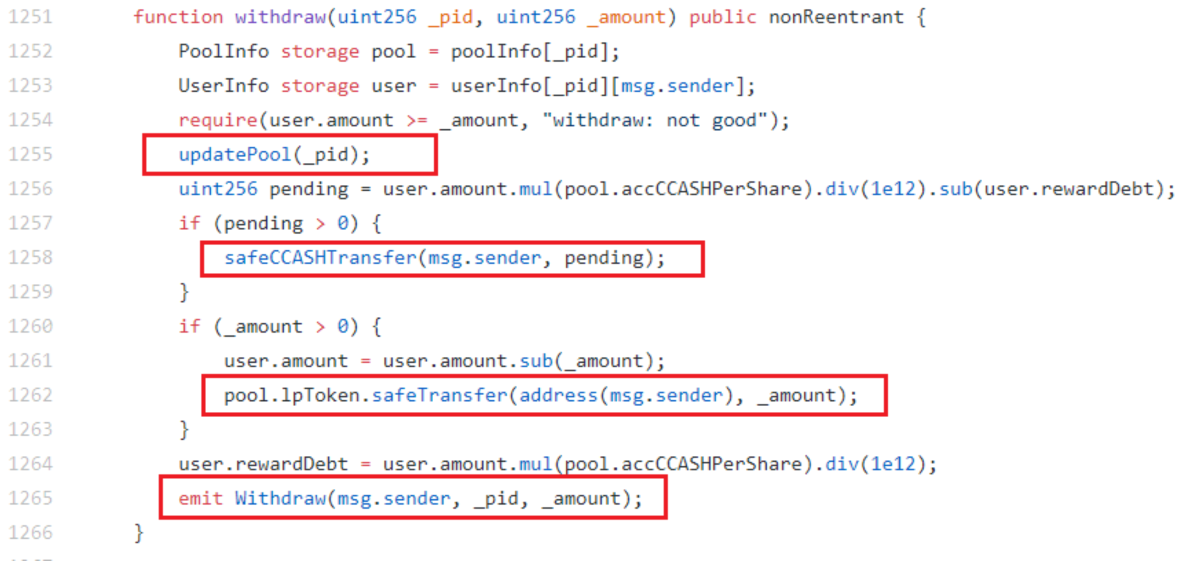
```
1305      function updateEmissionRate(uint256 _ccashPerBlock) public onlyOwner {
1306          massUpdatePools();
1307          ccashPerBlock = _ccashPerBlock;
1308          emit UpdateEmissionRate(msg.sender, _ccashPerBlock);
1309      }
```

- Recommendation: Apply the check-effects-interactions pattern.

18. Reentrancy vulnerabilities

- Severity: Low

- Result: Found
- Affected file: MasterChef.sol
- Description: Detection of the reentrancy bug. Only report reentrancies leading to out-of-order events.
- POC:
  External calls:
          - updatePool(_pid) (MasterChef.sol#1255)
                  - ccash.mint(devaddr,ccashReward.div(10)) (MasterChef.sol#1219)
                  - ccash.mint(address(this),ccashReward) (MasterChef.sol#1220)
          - safeCCASHTransfer(msg.sender,pending) (MasterChef.sol#1258)
                  - transferSuccess = ccash.transfer(_to,ccashBal) (MasterChef.sol#1284)
                  - transferSuccess = ccash.transfer(_to,_amount) (MasterChef.sol#1286)
          - pool.lpToken.safeTransfer(address(msg.sender),_amount)
  (MasterChef.sol#1262)
          Event emitted after the call(s):
          - Withdraw(msg.sender,_pid,_amount) (MasterChef.sol#1265)



```
1251    function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
1252        PoolInfo storage pool = poolInfo[_pid];
1253        UserInfo storage user = userInfo[_pid][msg.sender];
1254        require(user.amount >= _amount, "withdraw: not good");
1255        updatePool(_pid);
1256        uint256 pending = user.amount.mul(pool.accCCASHPerShare).div(1e12).sub(user.rewardDebt);
1257        if (pending > 0) {
1258            safeCCASHTransfer(msg.sender, pending);
1259        }
1260        if (_amount > 0) {
1261            user.amount = user.amount.sub(_amount);
1262            pool.lpToken.safeTransfer(address(msg.sender), _amount);
1263        }
1264        user.rewardDebt = user.amount.mul(pool.accCCASHPerShare).div(1e12);
1265        emit Withdraw(msg.sender, _pid, _amount);
1266    }
```

- Recommendation: Apply the check-effects-interactions pattern.

Basic Coding Bugs

19. Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

20. Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

21. Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

22. Overflows & Underflows

    - Description: Whether the contract has general overflow or underflow vulnerabilities
    - Result: Not found
    - Severity: Critical

23. Reentrancy

    - Description: Reentrancy is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
    - Result: Not found
    - Severity: Critical

24. Money-Giving Bug

    - Description: Whether the contract returns funds to an arbitrary address.
    - Result: Not found
    - Severity: High

25. Blackhole

    - Description: Whether the contract locks ETH indefinitely: merely in without out.
    - Result: Not found
    - Severity: High

26. Unauthorized Self-Destruct

    - Description: Whether the contract can be killed by any arbitrary address.
    - Result: Not found
    - Severity: Medium

27. Revert DoS

    - Description: Whether the contract is vulnerable to DoS attack because of unexpected revert.
    - Result: Not found
    - Severity: Medium

28. Unchecked External Call

    - Description: Whether the contract has any external call without checking the return value.
    - Result: Not found
    - Severity: Medium

29. Gasless Send

    - Description: Whether the contract is vulnerable to gasless send.

- Result: Not found
- Severity: Medium

30. Send Instead of Transfer

    - Description: Whether the contract uses send instead of transfer.
    - Result: Not found
    - Severity: Medium

31. Costly Loop

    - Description: Whether the contract has any costly loop which may lead to Out-Of-Gas exception.
    - Result: Not found
    - Severity: Medium

32. (Unsafe) Use of Untrusted Libraries

    - Description: Whether the contract use any suspicious libraries.
    - Result: Not found
    - Severity: Medium

33. (Unsafe) Use of Predictable Variables

    - Description: Whether the contract contains any randomness variable, but its value can be predicated.
    - Result: Not found
    - Severity: Medium

34. Transaction Ordering Dependence

    - Description: Whether the final state of the contract depends on the order of the transactions.
    - Result: Not found
    - Severity: Medium

35. Deprecated Uses

    - Description: Whether the contract use the deprecated tx.origin to perform the authorization.
    - Result: Not found
    - Severity: Medium

Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

As there are no security vulnerabilities, business logic issues or coding bugs found in first phase of these smart contracts, there are no detailed results to show.

Conclusion

In this audit, we thoroughly analyzed the Coinopolis documentation and implementation. The current code base is well organized but there are promptly some issues found in first phase of Pre-Launch Audit. Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

About eNebula Solutions

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.

The eNebula Solutions team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including incryptocurrency, blockchains, payments, and smart contracts. Additionally, the team can utilize varioustools to scan code & networks and build custom tools as necessary.

Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

For more information about our security consulting, please mail us at – contact@enebula.in.

Our Methodology

We wish to work with a clear method and build our reviews a cooperative effort. The goals of our security audits are to boost the standard of systems we tend to review and aim for adequate remediation to assist protect users. The subsequent is that the methodology suggested by synopsys (synopsys.com) we tend to use in our security code review audit method.

1. **Finalize the tool.** Select a static analysis tool that can perform code reviews of applications written in the programming languages you use. The tool should also be able to comprehend the underlying framework used by your software.
2. **Create the scanning infrastructure, and deploy the tool.** This step involves handling the licensing requirements, setting up access control and authorization, and procuring the resources required (e.g., servers and databases) to deploy the tool.
3. **Customize the tool.** Fine-tune the tool to suit the needs of the organization. For example, you might configure it to reduce false positives or find additional security vulnerabilities by writing new rules or updating existing ones. Integrate the tool into the build environment, create dashboards for tracking scan results, and build custom reports.
4. **Prioritize and onboard applications.** Once the tool is ready, onboard your applications. If you have a large number of applications, prioritize the high-risk applications to scan first. Eventually, all your applications should be onboarded and scanned regularly, with application scans synced with

release cycles, daily or monthly builds, or code check-ins.

5. **Analyze scan results.** This step involves triaging the results of the scan to remove false positives. Once the set of issues is finalized, they should be tracked and provided to the deployment teams for proper and timely remediation.

6. **Provide governance and training.** Proper governance ensures that your development teams are employing the scanning tools properly. The software security touchpoints should be present within the SDLC. SAST should be incorporated as part of your application development and deployment process.