

Audit Report September, 2021

For



Poly Nuts

Contents

Overview	01
Techniques and Methods	02
Issue Categories	03
Manual Testing Results	05
Functional Testing	09
Automated Testing	11
Closing Summary	16
Disclaimer	17

Scope of the Audit

The scope of this audit was to analyze and document the PolyNuts smart contract codebase for quality, security, and correctness.

Checked Vulnerabilities

We have scanned the smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered:

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Mythril, Slither, SmartCheck, Surya, Solhint.

Issue Categories

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

Risk-level	Description
High	A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.
Medium	The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.
Low	Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.
Informational	These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Number of issues per severity

Type	High	Medium	Low	Informational
Open	0	0	0	0
Acknowledged	0	2	2	3
Closed	0	0	0	0

Introduction

During the period of **September 06, 2021 to September 10, 2021** - QuillAudits Team performed a security audit for PolyNuts smart contracts.

The code for the audit was taken from following the official link:

DeezNutsMasterChef:

[https://polygonscan.com/
address/0xA98E88C974361015723E889CdCF63cd8D8004765#code](https://polygonscan.com/address/0xA98E88C974361015723E889CdCF63cd8D8004765#code)

NUTSToken:

[https://polygonscan.com/
address/0xf28CA9d779CA6A1945A85Ac50bdB7Dd5D0038c1F#code](https://polygonscan.com/address/0xf28CA9d779CA6A1945A85Ac50bdB7Dd5D0038c1F#code)



Issues Found

A. Contract - DeezNutsMasterChef

High severity issues

No issues were found.

Medium severity issues

1. Centralization Risks

Description

The role owner has the authority to :

- add new liquidity pool
- set/modify the settings of a pool
- modify dev and fee address
- update Nuts emission rate
- set/modify referral commission rate
- update the start block

It has to be noted that the owner can change the eulerTxFee at any time. The eulerTxFee variable is used to take into consideration the transaction fee charged by the Euler Token contract that goes to the burn address. Initially, it is set to 100 which is equal to the 1% charged by the token contract.

If it is increased, then some of that fee will go to the staking contract. If it is decreased to a number below 100, then it can be a loss for the staking contract.

Remediation

We advise the client to handle the governance account carefully to avoid any potential hack. We also advise the client to consider the following solutions:

1. Time-lock with reasonable latency for community awareness on privileged operations;
2. Multisig with community-voted 3rd-party independent co-signers;
3. DAO or Governance module increasing transparency and community involvement;

Status: Acknowledged by the Auditee

Low level severity issues

2. User's Lockdown period resets after withdraw()

Line	Code
1437-1448	<pre> function updateStartBlock(uint256 _newStartBlock) external onlyOwner { require(block.number < startBlock, "cannot change start block if farm has already started"); require(block.number < _newStartBlock, "cannot set start block in the past"); uint256 length = poolInfo.length; for (uint256 pid = 0; pid < length; ++pid) { PoolInfo storage pool = poolInfo[pid]; pool.lastRewardBlock = _newStartBlock; } startBlock = _newStartBlock; emit UpdateStartBlock(startBlock); } </pre>
1292-1297	<pre> function massUpdatePools() public { uint256 length = poolInfo.length; for (uint256 pid = 0; pid < length; ++pid) { updatePool(pid); } } </pre>

Description

The function `updateStartBlock()` and `massUpdatePools()`, has loops which may run out of gas if a significant amount of pools are added to the contract.

Remediation

Keep a check on the number of pools added.

Status: Acknowledged by the Auditee

3. Deposits and withdrawals will revert when token maximum supply is reached

Description

When the maximum supply is reached, the `_mint` function will revert, causing `updatePool()` to fail. This means both deposit and withdrawals will not work.

Remediation

Once max supply has been minted, simply setting allocPoints to 0 is sufficient to resolve this issue.

Status: Acknowledged by the Auditee

Informational Issues

4. Remove unused events

Line	Code
1213	event SetReferralAddress(address indexed user, IReferral indexed newAddress);

Description

The unused events can be removed from the code to save deployment costs.

Remediation

We recommend removing the unused event.

Status: Acknowledged by the Auditee

5. Public function that could be declared external

Description

The following public functions that are never called by the contract should be declared external to save gas:

- claim
- deposit
- withdraw
- emergencyWithdraw

Remediation

Use the external attribute for functions never called from the contract. -

Status: Acknowledged by the Auditee

B. Contract - NUTSToken

High severity issues

No issues were found.

Medium severity issues

1. Use of safemath to avoid overflow

Description

In line 599 in `_mint` function, use of `safemath.add` function is highly recommended to avoid overflow.

```

597 function _mint(address account, uint256 amount) internal {
598     require(account != address(0), 'BEP20: mint to the zero address');
599     require((MAXCAP+amount)<=MAXCAPSUPPLY,"Max supply reached");
600
601     _totalSupply = _totalSupply.add(amount);
602     MAXCAP=MAXCAP.add(amount);
603     _balances[account] = _balances[account].add(amount);
604     emit Transfer(address(0), account, amount);
605 }
606

```

Status: Acknowledged by the Auditee

Low level severity issues

No issues were found.

Informational Issues

2. MAXCAP has no other use

Description

In line 599, Variable MAXCAP has no other use in the contract; the same functionality can be achieved by using `_totalSupply.add(amount)<=MAXCAPSUPPLY`

```

597 function _mint(address account, uint256 amount) internal {
598     require(account != address(0), 'BEP20: mint to the zero address');
599     require((MAXCAP+amount)<=MAXCAPSUPPLY,"Max supply reached");
600
601     _totalSupply = _totalSupply.add(amount);
602     MAXCAP=MAXCAP.add(amount);
603     _balances[account] = _balances[account].add(amount);
604     emit Transfer(address(0), account, amount);
605 }
606

```

Status: Acknowledged by the Auditee

Functional Tests

DeezNutsMasterChef

Function Names	Testing results
add()	Passed
set()	Passed
pendingNuts()	Passed
updatePool()	Passed
deposit()	Passed
withdraw()	Passed
emergencyWithdraw()	Passed
setDevAddress()	Passed
setFeeAddress()	Passed
updateEmissionRate()	Passed
setReferralCommissionRate()	Passed
updateStartBlock()	Passed

NUTSToken

Function Names	Testing results
constructor	PASS
getOwner	PASS
decimals	PASS
symbol	PASS
name	PASS
totalSupply	PASS
balanceOf	PASS
Transfer	PASS
allowance	PASS
approve	PASS
transferFrom	PASS
increaseAllowance	PASS
decreaseAllowance	PASS
mint	FAIL
transferOwnership	PASS

Automated Testing

Slither

DeezNutsMasterChef

INFO:Detectors:

DeezNutsMasterChef.pendingNuts(uint256,address) (DeezNutsMasterChef.sol#1278-1289) performs a multiplication on the result of a division:

- NutsReward = multiplier.mul(NutsPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (DeezNutsMasterChef.sol#1285)
- accNutsPerShare = accNutsPerShare.add(NutsReward.mul(1e18).div(lpSupply)) (DeezNutsMasterChef.sol#1286)

DeezNutsMasterChef.updatePool(uint256) (DeezNutsMasterChef.sol#1300-1316) performs a multiplication on the result of a division:

- NutsReward = multiplier.mul(NutsPerBlock).mul(pool.allocPoint).div(totalAllocPoint) (DeezNutsMasterChef.sol#1311)
- pool.accNutsPerShare = pool.accNutsPerShare.add(NutsReward.mul(1e18).div(lpSupply)) (DeezNutsMasterChef.sol#1314)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply>

INFO:Detectors:

NUTS._writeCheckpoint(address,uint32,uint256,uint256) (DeezNutsMasterChef.sol#1095-1113) uses a dangerous strict equality:

- nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber (DeezNutsMasterChef.sol#1105)

DeezNutsMasterChef.updatePool(uint256) (DeezNutsMasterChef.sol#1300-1316) uses a dangerous strict equality:

- lpSupply == 0 || pool.allocPoint == 0 (DeezNutsMasterChef.sol#1306)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities>

INFO:Detectors:

Reentrancy in DeezNutsMasterChef.deposit(uint256,uint256,address) (DeezNutsMasterChef.sol#1319-1350):

External calls:

- updatePool(_pid) (DeezNutsMasterChef.sol#1322)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
 - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- referral.recordReferral(msg.sender,_referrer) (DeezNutsMasterChef.sol#1324)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1329)
 - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
 - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1330)
 - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (DeezNutsMasterChef.sol#1335)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (DeezNutsMasterChef.sol#1339)

State variables written after the call(s):

- pool.lpSupply = pool.lpSupply.add(final_amount).sub(depositFee) (DeezNutsMasterChef.sol#1341)
- user.amount = user.amount.add(final_amount).sub(depositFee) (DeezNutsMasterChef.sol#1340)

Reentrancy in DeezNutsMasterChef.deposit(uint256,uint256,address) (DeezNutsMasterChef.sol#1319-1350):

External calls:

- updatePool(_pid) (DeezNutsMasterChef.sol#1322)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
 - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- referral.recordReferral(msg.sender,_referrer) (DeezNutsMasterChef.sol#1324)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1329)
 - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
 - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1330)
 - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (DeezNutsMasterChef.sol#1335)

State variables written after the call(s):

- pool.lpSupply = pool.lpSupply.add(final_amount) (DeezNutsMasterChef.sol#1345)
- user.amount = user.amount.add(final_amount) (DeezNutsMasterChef.sol#1344)

Reentrancy in DeezNutsMasterChef.updateEmissionRate(uint256) (DeezNutsMasterChef.sol#1409-1414):

External calls:

- massUpdatePools() (DeezNutsMasterChef.sol#1411)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)

```

- Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
- Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
State variables written after the call(s):
- NutsPerBlock = _NutsPerBlock (DeezNutsMasterChef.sol#1412)
Reentrancy in DeezNutsMasterChef.updatePool(uint256) (DeezNutsMasterChef.sol#1300-1316):
External calls:
- Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
- Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
State variables written after the call(s):
- pool.accNutsPerShare = pool.accNutsPerShare.add(NutsReward.mul(1e18).div(lpSupply)) (DeezNutsMasterChef.sol#1314)
- pool.lastRewardBlock = block.number (DeezNutsMasterChef.sol#1315)
Reentrancy in DeezNutsMasterChef.withdraw(uint256,uint256) (DeezNutsMasterChef.sol#1353-1370):
External calls:
- updatePool(_pid) (DeezNutsMasterChef.sol#1357)
- Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
- Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1360)
- transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
- transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1361)
- Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
State variables written after the call(s):
- user.amount = user.amount.sub(_amount) (DeezNutsMasterChef.sol#1364)
Reentrancy in DeezNutsMasterChef.withdraw(uint256,uint256) (DeezNutsMasterChef.sol#1353-1370):
External calls:
- updatePool(_pid) (DeezNutsMasterChef.sol#1357)
- Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
- Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1360)
- transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
- transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1361)
- Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (DeezNutsMasterChef.sol#1365)
State variables written after the call(s):
- pool.lpSupply = pool.lpSupply.sub(_amount) (DeezNutsMasterChef.sol#1366)
- user.rewardDebt = user.amount.mul(pool.accNutsPerShare).div(1e18) (DeezNutsMasterChef.sol#1368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
DeezNutsMasterChef.add(uint256,IERC20,uint16) (DeezNutsMasterChef.sol#1245-1261) ignores return value by _lpToken.balanceOf(address(this)) (DeezNutsMasterChef.sol#1247)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
ERC20.constructor(string,string).name (DeezNutsMasterChef.sol#510) shadows:
- ERC20.name() (DeezNutsMasterChef.sol#519-521) (function)
ERC20.constructor(string,string).symbol (DeezNutsMasterChef.sol#510) shadows:
- ERC20.symbol() (DeezNutsMasterChef.sol#527-529) (function)
ERC20.allowance(address,address).owner (DeezNutsMasterChef.sol#578) shadows:
- Ownable.owner() (DeezNutsMasterChef.sol#59-61) (function)
ERC20._approve(address,address,uint256).owner (DeezNutsMasterChef.sol#726) shadows:
- Ownable.owner() (DeezNutsMasterChef.sol#59-61) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:

```

DeezNutsMasterChef.constructor(NUTS,uint256,address,address,IReferral)._devAddress (DeezNutsMasterChef.sol#1220) lacks a zero-check on :

- devAddress = _devAddress (DeezNutsMasterChef.sol#1228)

DeezNutsMasterChef.constructor(NUTS,uint256,address,address,IReferral)._feeAddress (DeezNutsMasterChef.sol#1221) lacks a zero-check on :

- feeAddress = _feeAddress (DeezNutsMasterChef.sol#1229)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

INFO:Detectors:

Reentrancy in DeezNutsMasterChef.deposit(uint256,uint256,address) (DeezNutsMasterChef.sol#1319-1350):

External calls:

- updatePool(_pid) (DeezNutsMasterChef.sol#1322)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
 - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- referral.recordReferral(msg.sender,_referrer) (DeezNutsMasterChef.sol#1324)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1329)
 - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
 - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1330)
 - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)

Event emitted after the call(s):

- ReferralCommissionPaid(_user,referrer,commissionAmount) (DeezNutsMasterChef.sol#1431)
 - payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1330)

Reentrancy in DeezNutsMasterChef.deposit(uint256,uint256,address) (DeezNutsMasterChef.sol#1319-1350):

External calls:

- updatePool(_pid) (DeezNutsMasterChef.sol#1322)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)

- updatePool(_pid) (DeezNutsMasterChef.sol#1322)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
 - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- referral.recordReferral(msg.sender,_referrer) (DeezNutsMasterChef.sol#1324)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1329)
 - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
 - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1330)
 - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount) (DeezNutsMasterChef.sol#1335)
- pool.lpToken.safeTransfer(feeAddress,depositFee) (DeezNutsMasterChef.sol#1339)

Event emitted after the call(s):

- Deposit(msg.sender,_pid,_amount) (DeezNutsMasterChef.sol#1349)

Reentrancy in DeezNutsMasterChef.emergencyWithdraw(uint256) (DeezNutsMasterChef.sol#1373-1381):

External calls:

- pool.lpToken.safeTransfer(address(msg.sender),amount) (DeezNutsMasterChef.sol#1379)

Event emitted after the call(s):

- EmergencyWithdraw(msg.sender,_pid,amount) (DeezNutsMasterChef.sol#1380)

Reentrancy in DeezNutsMasterChef.payReferralCommission(address,uint256) (DeezNutsMasterChef.sol#1424-1434):

External calls:

- Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)

Event emitted after the call(s):

- ReferralCommissionPaid(_user,referrer,commissionAmount) (DeezNutsMasterChef.sol#1431)

Reentrancy in DeezNutsMasterChef.updateEmissionRate(uint256) (DeezNutsMasterChef.sol#1409-1414):

External calls:

- massUpdatePools() (DeezNutsMasterChef.sol#1411)
 - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)


```

- Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
- Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
Event emitted after the call(s):
- UpdateEmissionRate(msg.sender,_NutsPerBlock) (DeezNutsMasterChef.sol#1413)
Reentrancy in DeezNutsMasterChef.withdraw(uint256,uint256) (DeezNutsMasterChef.sol#1353-1370):
External calls:
- updatePool(_pid) (DeezNutsMasterChef.sol#1357)
  - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
  - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1360)
  - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
  - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1361)
  - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
Event emitted after the call(s):
- ReferralCommissionPaid(_user,referrer,commissionAmount) (DeezNutsMasterChef.sol#1431)
  - payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1361)
Reentrancy in DeezNutsMasterChef.withdraw(uint256,uint256) (DeezNutsMasterChef.sol#1353-1370):
External calls:
- updatePool(_pid) (DeezNutsMasterChef.sol#1357)
  - Nuts.mint(devAddress,NutsReward.div(10)) (DeezNutsMasterChef.sol#1312)
  - Nuts.mint(address(this),NutsReward) (DeezNutsMasterChef.sol#1313)
- safeNutsTransfer(msg.sender,pending) (DeezNutsMasterChef.sol#1360)
  - transferSuccess = Nuts.transfer(_to,NutsBal) (DeezNutsMasterChef.sol#1388)
  - transferSuccess = Nuts.transfer(_to,_amount) (DeezNutsMasterChef.sol#1390)
- payReferralCommission(msg.sender,pending) (DeezNutsMasterChef.sol#1361)
  - Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- Nuts.mint(referrer,commissionAmount) (DeezNutsMasterChef.sol#1430)
- pool.lpToken.safeTransfer(address(msg.sender),_amount) (DeezNutsMasterChef.sol#1365)
Event emitted after the call(s):
- Withdraw(msg.sender,_pid,_amount) (DeezNutsMasterChef.sol#1369)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
NUTS.delegateBySig(address,uint256,uint256,uint8,bytes32,bytes32) (DeezNutsMasterChef.sol#961-1002) uses timestamp for comparisons
Dangerous comparisons:
- require(bool,string)(now <= expiry,TOKEN::delegateBySig: signature expired) (DeezNutsMasterChef.sol#1000)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (DeezNutsMasterChef.sol#271-280) uses assembly
- INLINE ASM (DeezNutsMasterChef.sol#278)
Address._functionCallWithValue(address,bytes,uint256,string) (DeezNutsMasterChef.sol#364-385) uses assembly
- INLINE ASM (DeezNutsMasterChef.sol#377-380)
NUTS.getChainId() (DeezNutsMasterChef.sol#1120-1124) uses assembly
- INLINE ASM (DeezNutsMasterChef.sol#1122)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
DeezNutsMasterChef.nonDuplicated(IERC20) (DeezNutsMasterChef.sol#1239-1242) compares to a boolean constant:
-require(bool,string)(poolExistence[_lpToken] == false,nonDuplicated: duplicated) (DeezNutsMasterChef.sol#1240)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Different versions of Solidity is used:
- Version used: ['0.6.12', '>=0.4.22<0.9.0']
- 0.6.12 (DeezNutsMasterChef.sol#7)
- >=0.4.22<0.9.0 (Migrations.sol#2)

```



```

- >=0.4.22<0.9.0 (Migrations.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (DeezNutsMasterChef.sol#324-326) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (DeezNutsMasterChef.sol#349-351) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (DeezNutsMasterChef.sol#359-362) is never used and should be removed
Address.sendValue(address,uint256) (DeezNutsMasterChef.sol#298-304) is never used and should be removed
Context._msgData() (DeezNutsMasterChef.sol#24-27) is never used and should be removed
ERC20._burn(address,uint256) (DeezNutsMasterChef.sol#703-711) is never used and should be removed
ERC20._setupDecimals(uint8) (DeezNutsMasterChef.sol#741-743) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (DeezNutsMasterChef.sol#790-799) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (DeezNutsMasterChef.sol#806-809) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (DeezNutsMasterChef.sol#801-804) is never used and should be removed
SafeMath.mod(uint256,uint256) (DeezNutsMasterChef.sol#228-230) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (DeezNutsMasterChef.sol#244-247) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (DeezNutsMasterChef.sol#298-304):
- (success) = recipient.call{value: amount}() (DeezNutsMasterChef.sol#302)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (DeezNutsMasterChef.sol#364-385):
- (success,returndata) = target.call{value: weiValue}(data) (DeezNutsMasterChef.sol#368)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter NUTS.mint(address,uint256)._to (DeezNutsMasterChef.sol#891) is not in mixedCase

INFO:Detectors:
owner() should be declared external:
- Ownable.owner() (DeezNutsMasterChef.sol#59-61)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (DeezNutsMasterChef.sol#78-81)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (DeezNutsMasterChef.sol#87-91)
symbol() should be declared external:
- ERC20.symbol() (DeezNutsMasterChef.sol#527-529)
decimals() should be declared external:
- ERC20.decimals() (DeezNutsMasterChef.sol#544-546)
totalSupply() should be declared external:
- ERC20.totalSupply() (DeezNutsMasterChef.sol#551-553)
transfer(address,uint256) should be declared external:
- ERC20.transfer(address,uint256) (DeezNutsMasterChef.sol#570-573)
allowance(address,address) should be declared external:
- ERC20.allowance(address,address) (DeezNutsMasterChef.sol#578-580)
approve(address,uint256) should be declared external:
- ERC20.approve(address,uint256) (DeezNutsMasterChef.sol#589-592)
transferFrom(address,address,uint256) should be declared external:
- ERC20.transferFrom(address,address,uint256) (DeezNutsMasterChef.sol#606-610)
increaseAllowance(address,uint256) should be declared external:
- ERC20.increaseAllowance(address,uint256) (DeezNutsMasterChef.sol#624-627)
decreaseAllowance(address,uint256) should be declared external:
- ERC20.decreaseAllowance(address,uint256) (DeezNutsMasterChef.sol#643-646)

deposit(uint256,uint256,address) should be declared external:
- DeezNutsMasterChef.deposit(uint256,uint256,address) (DeezNutsMasterChef.sol#1319-1350)
withdraw(uint256,uint256) should be declared external:
- DeezNutsMasterChef.withdraw(uint256,uint256) (DeezNutsMasterChef.sol#1353-1370)
emergencyWithdraw(uint256) should be declared external:
- DeezNutsMasterChef.emergencyWithdraw(uint256) (DeezNutsMasterChef.sol#1373-1381)
setCompleted(uint256) should be declared external:
- Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (12 contracts with 75 detectors), 95 result(s) found

```

POLYNuts.sol

INFO:Detectors:

```
BEP20.constructor(string,string).name (contracts/Greeter.sol#408) shadows:
- BEP20.name() (contracts/Greeter.sol#424-426) (function)
- IBEP20.name() (contracts/Greeter.sol#188) (function)
BEP20.constructor(string,string).symbol (contracts/Greeter.sol#408) shadows:
- BEP20.symbol() (contracts/Greeter.sol#432-434) (function)
- IBEP20.symbol() (contracts/Greeter.sol#183) (function)
BEP20.allowance(address,address).owner (contracts/Greeter.sol#478) shadows:
- Ownable.owner() (contracts/Greeter.sol#317-319) (function)
BEP20.approve(address,address,uint256).owner (contracts/Greeter.sol#639) shadows:
- Ownable.owner() (contracts/Greeter.sol#317-319) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
```

INFO:Detectors:

Different versions of Solidity is used:

- Version used: ['0.6.12', '>=0.4.0', '>=0.6.0<0.8.0', '>=0.6.12']
- >=0.6.0<0.8.0 (contracts/Greeter.sol#7)
- >=0.6.12 (contracts/Greeter.sol#167)
- >=0.6.0<0.8.0 (contracts/Greeter.sol#261)
- >=0.6.0<0.8.0 (contracts/Greeter.sol#286)
- >=0.4.0 (contracts/Greeter.sol#354)
- 0.6.12 (contracts/Greeter.sol#663)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used>

INFO:Detectors:

```
BEP20.burn(address,uint256) (contracts/Greeter.sol#618-624) is never used and should be removed
BEP20.burnFrom(address,uint256) (contracts/Greeter.sol#653-656) is never used and should be removed
Context.msgData() (contracts/Greeter.sol#278-281) is never used and should be removed
SafeMath.div(uint256,uint256) (contracts/Greeter.sol#107-109) is never used and should be removed
SafeMath.div(uint256,uint256,string) (contracts/Greeter.sol#123-129) is never used and should be removed
SafeMath.mod(uint256,uint256) (contracts/Greeter.sol#143-145) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (contracts/Greeter.sol#159-162) is never used and should be removed
SafeMath.mul(uint256,uint256) (contracts/Greeter.sol#81-93) is never used and should be removed
SafeMath.sub(uint256,uint256) (contracts/Greeter.sol#50-52) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

INFO:Detectors:

```
Pragma version>=0.6.0<0.8.0 (contracts/Greeter.sol#7) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/Greeter.sol#261) is too complex
Pragma version>=0.6.0<0.8.0 (contracts/Greeter.sol#286) is too complex
Pragma version>=0.4.0 (contracts/Greeter.sol#354) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

INFO:Detectors:

Redundant expression "this (contracts/Greeter.sol#279)" inContext (contracts/Greeter.sol#273-282)


Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements>

INFO:Detectors:

```
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (contracts/Greeter.sol#336-339)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (contracts/Greeter.sol#345-349)
name() should be declared external:
- BEP20.name() (contracts/Greeter.sol#424-426)
symbol() should be declared external:
- BEP20.symbol() (contracts/Greeter.sol#432-434)
decimals() should be declared external:
- BEP20.decimals() (contracts/Greeter.sol#439-441)
totalSupply() should be declared external:
- BEP20.totalSupply() (contracts/Greeter.sol#446-448)
maxSupply() should be declared external:
- BEP20.maxSupply() (contracts/Greeter.sol#450-452)
balanceOf(address) should be declared external:
- BEP20.balanceOf(address) (contracts/Greeter.sol#458-460)
transfer(address,uint256) should be declared external:
- BEP20.transfer(address,uint256) (contracts/Greeter.sol#470-473)
allowance(address,address) should be declared external:
- BEP20.allowance(address,address) (contracts/Greeter.sol#478-480)
approve(address,uint256) should be declared external:
- BEP20.approve(address,uint256) (contracts/Greeter.sol#489-492)
transferFrom(address,address,uint256) should be declared external:
- BEP20.transferFrom(address,address,uint256) (contracts/Greeter.sol#506-514)
increaseAllowance(address,uint256) should be declared external:
- BEP20.increaseAllowance(address,uint256) (contracts/Greeter.sol#528-531)
decreaseAllowance(address,uint256) should be declared external:
- BEP20.decreaseAllowance(address,uint256) (contracts/Greeter.sol#547-550)
mint(uint256) should be declared external:
- BEP20.mint(uint256) (contracts/Greeter.sol#560-563)
mint(address,uint256) should be declared external:
- NUTSToken.mint(address,uint256) (contracts/Greeter.sol#668-670)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

INFO:Slither: analyzed (6 contracts with 75 detectors), 38 result(s) found

INFO:Slither: Use <https://crytic.io/> to get access to additional detectors and Github integration

 solc error: TypeError: solc er

SOLIDITY STATIC ANALYSIS

POLYNuts.sol

Gas & Economy

Gas costs:Gas requirement of function BEP20.transferOwnership is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 345:4:

Gas costs:Gas requirement of function NUTSToken.transferOwnership is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 345:4:

Gas costs:Gas requirement of function BEP20.name is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 424:4:

Gas costs:Gas requirement of function NUTSToken.name is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 424:4:

Gas costs:Gas requirement of function BEP20.symbol is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 432:4:

Gas costs:Gas requirement of function NUTSToken.symbol is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 432:4:

Gas costs:Gas requirement of function BEP20.transfer is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 470:4:

Gas costs:Gas requirement of function NUTSToken.transfer is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 470:4:

Gas costs:Gas requirement of function BEP20.approve is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 489:4:

Gas costs:Gas requirement of function NUTSToken.approve is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 489:4:

Gas costs:Gas requirement of function BEP20.transferFrom is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 506:4:

Gas costs:Gas requirement of function NUTSToken.transferFrom is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 506:4:

Gas costs:Gas requirement of function BEP20.increaseAllowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 528:4:

Gas costs:Gas requirement of function NUTSToken.increaseAllowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 528:4:

Gas costs:Gas requirement of function BEP20.decreaseAllowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 547:4:

Gas costs:Gas requirement of function NUTSToken.decreaseAllowance is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 547:4:

Gas costs:Gas requirement of function BEP20.mint is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 560:4:

Gas costs:Gas requirement of function NUTSToken.mint is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 560:4:

Gas costs:Gas requirement of function NUTSToken.mint is infinite:

If the gas requirement of a function is higher than the block gas limit, it cannot be executed.

Please avoid loops in your functions or actions that modify large areas of storage

(this includes clearing or copying arrays in storage)Pos: 668:4:

ERC

ERC20:ERC20 contract's "decimals" function should have "uint8" as return type[more](#)Pos: 178:4:

Miscellaneous

Constant/View/Pure functions:SafeMath.sub(uint256,uint256) : Is constant but potentially should not be.

Note: Modifiers are currently not considered by this static analysis.[more](#)Pos: 50:4:

Constant/View/Pure functions:SafeMath.div(uint256,uint256) : Is constant but potentially should not be. Note:

Modifiers are currently not considered by this static analysis.[more](#)Pos: 107:4:

Constant/View/Pure functions:SafeMath.mod(uint256,uint256) : Is constant but potentially should not be.

Note: Modifiers are currently not considered by this static analysis.[more](#)Pos: 143:4:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 598:16:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 599:34:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 601:40:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 602:26:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 603:18:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 603:39:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 603:52:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 604:34:

Similar variable names:BEP20._mint(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 604:43:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 619:16:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 621:18:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 621:39:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 621:52:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 622:40:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 623:22:

Similar variable names:BEP20._burn(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 623:43:

Similar variable names:BEP20._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 654:14:

Similar variable names:BEP20._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 654:23:

Similar variable names:BEP20._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 655:17:

Similar variable names:BEP20._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 655:52:

Similar variable names:BEP20._burnFrom(address,uint256) : Variables have very similar names "account" and "amount". Note: Modifiers are currently not considered by this static analysis.Pos: 655:79:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 35:8:

Guard conditions:Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 65:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 90:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 124:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 160:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 325:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 346:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 580:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 581:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 598:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 599:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 619:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 640:8:

Guard conditions: Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.[more](#)Pos: 641:8:

Data truncated: Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.Pos: 90:16:

Data truncated: Division of integer values yields an integer value again. That means e.g. $10 / 100 = 0$ instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.Pos: 125:20:

Result

1 major issue was found. Some false positive errors were reported by the tool. All the other issues have been categorized above, according to their level of severity.

Closing Summary

Overall, smart contracts are very well written and adhere to guidelines.

No instances of Integer Overflow and Underflow vulnerabilities or Back-Door Entry were found in the contract, but relying on other contracts might cause Reentrancy Vulnerability.

Several issues were discovered during the initial audit. It is recommended to kindly go through the above-mentioned details and fix the code accordingly.



Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the PolyNuts platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the PolyNuts Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.



Audit Report September, 2021

For



Poly Nuts



QuillAudits

📍 Canada, India, Singapore, United Kingdom

🌐 audits.quillhash.com

✉️ audits@quillhash.com