QuillAudits

Audit Report
August, 2021

Contents

Scope of Audit

The scope of this audit was to analyze and document the PlentyCoin Token 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 analyzed 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.
SmartCheck.

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 security of smart contracts.

Code Review / Manual Analysis
Manual Analysis or review of code was done to identify new vulnerability or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of 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 possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit
Remix IDE, Truffle, Truffle Team, Ganache, Solhint, Mythril, Slither, SmartCheck.

Issue Categories

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

High severity issues

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 to be fixed before moving to a live environment.

Medium level severity issues

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 level severity issues

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 severity four issues which 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	1	1	3
Closed	2	0	1	4

Introduction

During the period of **August 12, 2021 to August 20, 2021** - QuillAudits Team performed a security audit for PlentyCoin smart contracts.

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

Note	Date	Link
Version 1	August	https://bscscan.com/address/0xa5e8a886b6bab60e1a2273b79caa72143ed8b418#code
Version 2	August	https://bscscan.com/address/0x625DAD4E02e4379f1920BBE2dd966400c5d97320#writeProxyContract

Issues Found – Code Review / Manual Testing

High severity issues

1. Transfer is from msg.sender instead of contract balance to _buyBackAddress

Line	Code
1749	function swapAndLiquify(uint256 contractTokenBalance) private lockTheSwap { // split the contract balance into halves uint256 half = contractTokenBalance.mul(4000).div(10000); uint256 rest = contractTokenBalance.sub(half).sub(half); transfer(_buyBackAddress, rest);

Description

In the function swapAndLiquify(), a call to the function transfer() is done to make a transfer from the contract address to the _buyBackAddress. But as the msg.sender for this transfer call will be the user doing the transaction, the amount will be deducted from that user's account.

If swapAndLiquify is ever executed then a huge amount of that user's tokens will be sent to the buyback Address. Otherwise there is a high chance that all the transactions will get reverted if contract balance > numTokensSellToAddToLiquidity, because then swapAndLiquify() will be called and the user won't have the balance to transfer 20% of the numTokensSellToAddToLiquidity to the buyback Address.

Remediation

Use the internal _transfer() function to do the transfer from contract balance.

Status: Fixed

In the Version 2, _transfer() function is used.

2. Infinite loop

Line	Code
1895-1917	function lock(bytes32 _reason, uint256 _amount, uint256 _time) public override onlyWhitelistAdmin returns (bool) { uint256 validUntil = block.timestamp.add(_time); // If tokens are already locked, then functions extendLock or // increaseLockAmount should be used to make any changes require(tokensLocked(msg.sender, _reason) == 0, "ERC1132: Already locked"); require(_amount != 0, "ERC1132: Zero Amount"); if (locked[msg.sender][_reason].amount == 0) lockReason[msg.sender].push(_reason); transfer(address(this), _amount); locked[msg.sender][_reason] = LockToken(_amount, validUntil, false); emit Locked(msg.sender, _reason, _amount, validUntil); return true; }
1927-1949	function transferFromWithLock(address _from, address _to, bytes32 _reason, uint256 _amount, uint256 _time) public returns (bool) { ... ... _lockedAmount.add(_amount); emit Locked(_to, _reason, _lockedAmount, validUntil); return true; }
2075-2090	function unlock(address _of) public override returns (uint256 unlockableTokens) { ... ... if (unlockableTokens > 0) { uint256 beforeBalance = balanceOf(_of); this.transfer(_of, unlockableTokens); _lockedAmount.sub((balanceOf(_of).sub(beforeBalance))); }

Description
_lockedAmount state variable is used to keep track of the amount of tokens locked in the contract balance. But in the lock() function, it's not updated after some tokens are locked. Due to this, there can be an underflow, while unlocking.

In the transferFromWithLock() function, the statement _lockedAmount.add(_amount); does not update the state variable.

In the unlock() function, the statement _lockedAmount.sub((balanceOf(_of).sub(beforeBalance))); does not update the variable.

In the increaseLockAmount() function, no statement to update the _lockedAmount was found.

Remediation
Update the _lockedAmount variable in the lock(), transferFromWithLock(),increaseLockAmount() and unlock() function.

Status: Fixed
In the Version 2, _lockedAmount variable was updated in all the above-mentioned functions.

Medium severity issues

3. Centralized risk in addLiquidity

Description
The role owner has the authority to

- update settings (transaction fees and addresses)
- manage the list containing contracts excluding from reward, fee, or max transaction limitation.
- withdraw ether from the contract at any point of time.
- unlock tokens for any user

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:

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

Status: Acknowledged by the Auditee
Comments from Auditee: “We are composing the DAO structure so for now, it's actually centralized.”

Low level severity issues

4. Locks cannot be extended or increased in amount

Line	Code
1926-1948	function transferFromWithLock(address _from, address _to, bytes32 _reason, uint256 _amount, uint256 _time) public returns (bool) { uint256 validUntil = block.timestamp.add(_time); require(isWhitelistAdmin(_from), "ERC1132: Not whitelisted"); require(tokensLocked(_to, _reason) == 0, "ERC1132: Already locked"); require(_amount != 0, "ERC1132: Zero Amount"); if (locked[_to][_reason].amount == 0) lockReason[_to].push(_reason); transferFrom(_from, address(this), _amount); locked[_to][_reason] = LockToken(_amount, validUntil, false); _lockedAmount.add(_amount); emit Locked(_to, _reason, _amount, validUntil); return true; }

Description
The token locks which were created with transferFromWithLock(), can neither be extended nor be increased in amount. This is because the tokens are locked for the ‘to’ address and it is not checked for isWhitelistAdmin. But to access the extendLock() or increaseLockAmount() function ‘to’ address should be isWhitelistAdmin == true.

Remediation
Check the ‘to’ address for isWhitelistAdmin

Status: Acknowledged by the Auditee
Comments from Auditee: “For transferFromWithLock(), _to can be not whitelisted as admin. Moreover, we don't need to extend lock time or increase the amount after transferFromWithLock is executed.”

5. Critical operation lacks event log

Line	Code
1532	function setTaxFeePercent(uint256 taxFee) external onlyOwner() { _taxFee = taxFee; }
1536	function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() { _liquidityFee = liquidityFee; }
1540	function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() { _maxTxAmount = _tTotal.mul(maxTxPercent).div(10**2); }

Description
The role can set the following state variables arbitrary large or small causing potential risks in fees and anti whale :
- _taxFee
- _liquidityFee
- _maxTxAmount

Remediation
We recommend setting ranges and check the following input variables:
- taxFee
- liquidityFee
- maxTxPercent

Status: Fixed
In the Version 2, the input variables were checked before updating the state.

Informational

6. Variable Typo

Line	Code
1293	uint256 tokensIntoLiqudity

Description
There is a typo in tokensIntoLiqudity.

Remediation
We recommend correcting and changing tokensIntoLiqudity to tokensIntoLiquidity.

Status: Fixed
This issue was found fixed in Version 2.

7. Use the latest solidity version

Line	Code
1279	bool inSwapAndLiquify;

Description
The Visibility of the inSwapAndLiquify variable is not defined. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

The default is internal for state variables, but it should be made explicit.

Remediation
We recommend adding the visibility for the state variable of inSwapAndLiquify.

Variables can be specified as being public, internal or private. Explicitly define visibility for all state variables.

Status: Acknowledged by the Auditee

8. **Make variables constant**

pragma solidity ^0.8.0;

Description
Contracts should be deployed using the same compiler version/flags with which they have been tested. Locking the pragma (for e.g., by not using ^ in pragma solidity 0.8.0) ensures that contracts do not accidentally get deployed using an older compiler version with unfixed bugs.

Remediation
Lock the pragma version.

Status: Fixed
This issue was found fixed in Version 2.

9. Other code specification issues

Description

The missing event makes it difficult to track off-chain liquidity fee changes. An event should be emitted for significant transactions changing values of important state variables like fees, etc..

Remediation

We recommend emitting events for such transactions.

Status: Acknowledged by the Auditee

10. Variable Typo

Line	Code
2124	payable(backbuy).transfer(address(this).balance);

Description

Although transfer() and send() have been recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts. For reference, read more.

Remediation

Use .call{ value: ... }("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status: Acknowledged by the Auditee

11. Redundant Code

Line	Code
1815	else if (!_isExcluded[sender] && !_isExcluded[recipient])

Description
When the contract enters the branch else if (!_isExcluded[sender] && !_isExcluded[recipient]) or else, the contract will execute the same piece of code _transferStandard(sender, recipient, amount);

Remediation
We recommend removing the above-mentioned code.

Status: Fixed
In Version 2, it was removed.

12. Public function that could be declared external

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

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

Status: Fixed
In Version 2, functions were declared external.

Functional test

Function Names	Testing results
name()	Passed
symbol()	Passed
decimals()	Passed
totalSupply()	Passed
balanceOf()	Passed
transfer()	Passed
approve()	Passed
allowance()	Passed
transferFrom()	Passed
decreaseAllowance()	Passed
totalFees()	Passed
deliver()	Passed
tokenFromReflection()	Passed
reflectionFromToken()	Passed
excludeFromReward()	Passed
includeInReward()	Passed
isExcludedFromReward()	Passed
setTaxFeePercent()	Passed
setLiquidityFeePercent()	Passed
setSwapAndLiquifyEnabled()	Passed

isExcludedFromFee()	Passed
setMaxTxPercent()	Passed
setExcludeFromMaxTx()	Passed
lock()	Passed
transferFromWithLock	Passed
tokensLocked()	Passed
tokensLockedAtTime	Passed
totalBalanceOf	Passed
extendLock	Passed
increaseLockAmount	Passed
tokensUnlockable	Passed
unlock	Passed
getUnlockableTokens	Passed
withdrawAll	Passed

Automated Testing

Slither

```
INFO:Detectors:
Plenty.addLiquidity(uint256,uint256) (PlentyCoin.sol#1787-1800) sends eth to arbitrary user
	Dangerous calls:
	- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.sol#1792-1799)
Plenty.withdrawAll(address) (PlentyCoin.sol#2117-2125) sends eth to arbitrary user
	Dangerous calls:
	- address(backbuy).transfer(address(this).balance) (PlentyCoin.sol#2124)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
Reentrancy in Plenty._transfer(address,address,uint256) (PlentyCoin.sol#1690-1742):
	External calls:
	- swapAndLiquify(contractTokenBalance) (PlentyCoin.sol#1729)
		- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.sol#1792-1799)
		- uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (PlentyCoin.sol#1778-1784)
	External calls sending eth:
	- swapAndLiquify(contractTokenBalance) (PlentyCoin.sol#1729)
		- uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.sol#1792-1799)
	State variables written after the call(s):
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _liquidityFee = _previousLiquidityFee (PlentyCoin.sol#1671)
		- _liquidityFee = 0 (PlentyCoin.sol#1666)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _previousLiquidityFee = _liquidityFee (PlentyCoin.sol#1663)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _previousTaxFee = _taxFee (PlentyCoin.sol#1662)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _rOwned[address(this)] = _rOwned[address(this)].add(rLiquidity) (PlentyCoin.sol#1642)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (PlentyCoin.sol#1839)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (PlentyCoin.sol#1859)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (PlentyCoin.sol#1840)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (PlentyCoin.sol#1881)
		- _rOwned[sender] = _rOwned[sender].sub(rAmount) (PlentyCoin.sol#1516)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (PlentyCoin.sol#1882)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (PlentyCoin.sol#1861)
		- _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount) (PlentyCoin.sol#1518)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _rTotal = _rTotal.sub(rFee) (PlentyCoin.sol#1557)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _tFeeTotal = _tFeeTotal.add(tFee) (PlentyCoin.sol#1558)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _tOwned[address(this)] = _tOwned[address(this)].add(tLiquidity) (PlentyCoin.sol#1644)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (PlentyCoin.sol#1880)
		- _tOwned[sender] = _tOwned[sender].sub(tAmount) (PlentyCoin.sol#1515)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (PlentyCoin.sol#1860)
		- _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount) (PlentyCoin.sol#1517)
	- _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
		- _taxFee = _previousTaxFee (PlentyCoin.sol#1670)
		- _taxFee = 0 (PlentyCoin.sol#1665)
Reentrancy in Plenty.increaseLockAmount(bytes32,uint256) (PlentyCoin.sol#2031-2051):
```

```
Reentrancy in Plenty.increaseLockAmount(bytes32,uint256) (PlentyCoin.sol#2031-2051):
        External calls:
        - transfer(address(this),_amount) (PlentyCoin.sol#2038)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transfer(address(this),_amount) (PlentyCoin.sol#2038)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        State variables written after the call(s):
        - locked[msg.sender][_reason].amount = locked[msg.sender][_reason].amount.add(_amount) (PlentyCoin.sol#2040-2042)
Reentrancy in Plenty.lock(bytes32,uint256,uint256) (PlentyCoin.sol#1895-1916):
        External calls:
        - transfer(address(this),_amount) (PlentyCoin.sol#1910)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transfer(address(this),_amount) (PlentyCoin.sol#1910)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        State variables written after the call(s):
        - locked[msg.sender][_reason] = LockToken(_amount,validUntil,false) (PlentyCoin.sol#1912)
Reentrancy in Plenty.transferFromWithLock(address,address,bytes32,uint256,uint256) (PlentyCoin.sol#1926-1948):
```

```
Reentrancy in Plenty.transferFromWithLock(address,address,bytes32,uint256,uint256) (PlentyCoin.sol#1926-1948):
        External calls:
        - transferFrom(_from,address(this),_amount) (PlentyCoin.sol#1941)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transferFrom(_from,address(this),_amount) (PlentyCoin.sol#1941)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        State variables written after the call(s):
        - locked[_to][_reason] = LockToken(_amount,validUntil,false) (PlentyCoin.sol#1943)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities
INFO:Detectors:
OwnableUpgradeable.__gap (PlentyCoin.sol#229) shadows:
        - ContextUpgradeable.__gap (PlentyCoin.sol#155)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variable-shadowing
INFO:Detectors:
Plenty.unlock(address) (PlentyCoin.sol#2074-2098) ignores return value by this.transfer(_of,unlockableTokens) (PlentyCoin.sol#2095)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
Plenty._lockedAmount (PlentyCoin.sol#1272) is never initialized. It is used in:
        - Plenty._transfer(address,address,uint256) (PlentyCoin.sol#1690-1742)
        - Plenty.transferFromWithLock(address,address,bytes32,uint256,uint256) (PlentyCoin.sol#1926-1948)
        - Plenty.unlock(address) (PlentyCoin.sol#2074-2098)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables
```

```
INFO:Detectors:
Plenty.addLiquidity(uint256,uint256) (PlentyCoin.sol#1787-1800) ignores return value by uniswapV2Router.addLiquidityETH{value: ethAmoun
t}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.sol#1792-1799)
Plenty.transferFromWithLock(address,address,bytes32,uint256,uint256) (PlentyCoin.sol#1926-1948) ignores return value by _lockedAmount.a
dd(_amount) (PlentyCoin.sol#1944)
Plenty.unlock(address) (PlentyCoin.sol#2074-2098) ignores return value by _lockedAmount.sub((balanceOf(_of).sub(beforeBalance))) (Plent
yCoin.sol#2096)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
INFO:Detectors:
Plenty.allowance(address,address).owner (PlentyCoin.sol#1374) shadows:
        - OwnableUpgradeable.owner() (PlentyCoin.sol#196-198) (function)
Plenty._approve(address,address,uint256).owner (PlentyCoin.sol#1679) shadows:
        - OwnableUpgradeable.owner() (PlentyCoin.sol#196-198) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Plenty.setBuyBack(address).buyBack (PlentyCoin.sol#1544) lacks a zero-check on :
                - _buyBackAddress = buyBack (PlentyCoin.sol#1545)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in Plenty.initialize(address,address) (PlentyCoin.sol#1302-1342):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (PlentyCoin.s
ol#1329-1332)
        State variables written after the call(s):
        - _isExcludedFromFee[owner()] = true (PlentyCoin.sol#1338)
        - _isExcludedFromFee[address(this)] = true (PlentyCoin.sol#1339)
        - uniswapV2Router = _uniswapV2Router (PlentyCoin.sol#1335)
Reentrancy in Plenty.swapAndLiquify(uint256) (PlentyCoin.sol#1744-1767):
        External calls:
        - transfer(_buyBackAddress,rest) (PlentyCoin.sol#1749)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        - swapTokensForEth(half) (PlentyCoin.sol#1758)
```

```
        - swapTokensForEth(half) (PlentyCoin.sol#1758)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
 (PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transfer(_buyBackAddress,rest) (PlentyCoin.sol#1749)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
 sol#1792-1799)
        State variables written after the call(s):
        - swapTokensForEth(half) (PlentyCoin.sol#1758)
                - _allowances[owner][spender] = amount (PlentyCoin.sol#1686)
Reentrancy in Plenty.swapAndLiquify(uint256) (PlentyCoin.sol#1744-1767):
        External calls:
        - transfer(_buyBackAddress,rest) (PlentyCoin.sol#1749)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        - swapTokensForEth(half) (PlentyCoin.sol#1758)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        - addLiquidity(half,newBalance) (PlentyCoin.sol#1764)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        External calls sending eth:
        - transfer(_buyBackAddress,rest) (PlentyCoin.sol#1749)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
```

```
sol#1792-1799)
        - addLiquidity(half,newBalance) (PlentyCoin.sol#1764)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        State variables written after the call(s):
        - addLiquidity(half,newBalance) (PlentyCoin.sol#1764)
                - _allowances[owner][spender] = amount (PlentyCoin.sol#1686)
Reentrancy in Plenty.transferFrom(address,address,uint256) (PlentyCoin.sol#1392-1407):
        External calls:
        - _transfer(sender,recipient,amount) (PlentyCoin.sol#1397)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - _transfer(sender,recipient,amount) (PlentyCoin.sol#1397)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (PlentyC
oin.sol#1398-1405)
                - _allowances[owner][spender] = amount (PlentyCoin.sol#1686)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Plenty._transfer(address,address,uint256) (PlentyCoin.sol#1690-1742):
        External calls:
        - swapAndLiquify(contractTokenBalance) (PlentyCoin.sol#1729)
```

```
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (PlentyCoin.sol#1729)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        Event emitted after the call(s):
        - Transfer(sender,recipient,tTransferAmount) (PlentyCoin.sol#1843)
                - _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
        - Transfer(sender,recipient,tTransferAmount) (PlentyCoin.sol#1885)
                - _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
        - Transfer(sender,recipient,tTransferAmount) (PlentyCoin.sol#1864)
                - _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
        - Transfer(sender,recipient,tTransferAmount) (PlentyCoin.sol#1521)
                - _tokenTransfer(from,to,amount,takeFee) (PlentyCoin.sol#1741)
Reentrancy in Plenty.increaseLockAmount(bytes32,uint256) (PlentyCoin.sol#2031-2051):
        External calls:
        - transfer(address(this),_amount) (PlentyCoin.sol#2038)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transfer(address(this),_amount) (PlentyCoin.sol#2038)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
```

```
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        Event emitted after the call(s):
        - Locked(msg.sender,_reason,locked[msg.sender][_reason].amount,locked[msg.sender][_reason].validity) (PlentyCoin.sol#2044-2049)
Reentrancy in Plenty.initialize(address,address) (PlentyCoin.sol#1302-1342):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (PlentyCoin.s
ol#1329-1332)
        Event emitted after the call(s):
        - Transfer(address(0),_msgSender(),_tTotal) (PlentyCoin.sol#1341)
Reentrancy in Plenty.lock(bytes32,uint256,uint256) (PlentyCoin.sol#1895-1916):
        External calls:
        - transfer(address(this),_amount) (PlentyCoin.sol#1910)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp)
(PlentyCoin.sol#1778-1784)
        External calls sending eth:
        - transfer(address(this),_amount) (PlentyCoin.sol#1910)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
sol#1792-1799)
        Event emitted after the call(s):
        - Locked(msg.sender,_reason,_amount,validUntil) (PlentyCoin.sol#1914)
Reentrancy in Plenty.swapAndLiquify(uint256) (PlentyCoin.sol#1744-1767):
        External calls:
        - transfer(_buyBackAddress,rest) (PlentyCoin.sol#1749)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (PlentyCoin.
```

```
INFO:Detectors:
Pragma version^0.8.0 (PlentyCoin.sol#12) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
INFO:Detectors:
Redundant expression "this (PlentyCoin.sol#152)" inContextUpgradeable (PlentyCoin.sol#140-156)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
Plenty.initialize(address,address) (PlentyCoin.sol#1302-1342) uses literals with too many digits:
        - _tTotal = 1000000000 * 10 ** 6 * 10 ** 18 (PlentyCoin.sol#1308)
Plenty.initialize(address,address) (PlentyCoin.sol#1302-1342) uses literals with too many digits:
        - _maxTxAmount = 5000000 * 10 ** 6 * 10 ** 18 (PlentyCoin.sol#1321)
Plenty.initialize(address,address) (PlentyCoin.sol#1302-1342) uses literals with too many digits:
        - numTokensSellToAddToLiquidity = 500000 * 10 ** 6 * 10 ** 18 (PlentyCoin.sol#1322)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
ContextUpgradeable.__gap (PlentyCoin.sol#155) is never used in Plenty (PlentyCoin.sol#1238-2127)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
Plenty._lockedAmount (PlentyCoin.sol#1272) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

Results

No major issues were found. Some false positive errors were reported by the tool. All the other issues have been categorized above according to their level of severity.

Disclaimer

Quillhash audit is not a security warranty, investment advice, or an endorsement of the PlentyCoin 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 PlentyCoin Team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

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.

Numerous issues were discovered during the initial audit. However, most of them are now either fixed or acknowledged by the PlentyCoin Team.

QuillAudits

Canada, India, Singapore and United Kingdom

audits.quillhash.com

audits@quillhash.com