QuillAudits

Audit Report
July, 2022

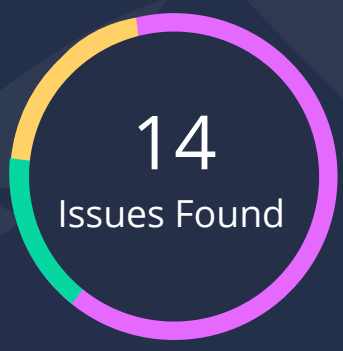
For

Magic
spellbinding crypto

Table of Content

Executive Summary

Project Name	Defi Magic
Overview	Voodoo is a unique token and concept that aims to be an upgrade to the fundamental principles behind defi by utilizing the values of three different token concepts, combining them, and removing their flaws. voodoo is designed in such a way that it will only rebase upwards and will not rebase downwards.
Timeline	23 May, 2022 - 29 June, 2022
Method	Manual Review, Functional Testing, Automated Testing etc.
Scope of Audit	The scope of this audit was to analyze VooDoo.sol, NewRebase.sol, rbDirect.sol codebase for quality, security, and correctness.
GitHub	*https://github.com/DeFi-Magic/VooDoo*
Branch	Main
Commit	66021077a5953ecd80f05e59241d105a3347cdb1
Fixed In	64cb985ac3c60c20f3464290de252b125cf5386d



High
Medium
Low
Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	2	2	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	1	6

Types of Severities

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 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.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

- Re-entrancy
- Timestamp Dependence
- Gas Limit and Loops
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- Dangerous strict equalities
- Tautology or contradiction
- Return values of low-level calls
- Missing Zero Address Validation
- Private modifier
- Revert/require functions
- Using block.timestamp
- Multiple Sends
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

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

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

A. Common Issues

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

A1. Ownership Transfer must be a two-step process

Contracts Affected - newRebase.sol, VooDoo.sol, rbDirect.sol

Description

The contract uses openzeppelin's ownable contract to manage ownership. The transferOwnership() function in ownable contract allows the current owner to transfer his privileges to another address. However, inside transferOwnership() , the newOwner is directly stored in the storage, owner, after validating the newOwner is a non-zero address, which may not be enough.
The newOwner is only validated against the zero address. However, if the current admin enters a wrong address by mistake, he would never be able to take the management permissions back. Besides, if the newOwner is the same as the current admin address stored in _owner , it's a waste of gas.

Remediation

It would be much safer if the transition is managed by implementing a two-step approach: _transferOwnership() and _updateOwnership() . Specifically, the _transferOwnership () function keeps the new address in the storage, _newOwner , instead of modifying the _owner() directly. The updateOwnership() function checks whether _newOwner is msg.sender, which means _newOwner signs the transaction and verifies himself as the new owner. After that, _newOwner could be set into _owner.

Status

Fixed

A2. Missing events for significant actions

Contracts Affected - newRebase.sol, VooDoo.sol, rbDirect.sol

Description

Whenever certain significant privileged actions are performed within the contract, it is recommended to emit an event about it. Changing forceRebase address, FixedRate, DeviationThreshold, FeeAddress, etc are some important actions hence it is recommended to emit an event.

In newRebase.sol:

- setForceRebaseAddress
- setRebaseFunctionGrowth
- setRebaseFunctionLowerPercentage
- setRebaseFunctionUpperPercentage
- setDeviationThreshold
- setRebaseTimingParameters
- setPair
- changeFixedRate

In VooDoo.sol:

- setFeeThreshold
- setZapperContract
- setIgnoredAddresses
- setIgnoredAddressBulk

In rbDirect.sol:

- setFeeAddress

Recommendation

Consider emitting an event whenever certain significant changes are made in the contracts.

Status

Fixed

Informational Issues

A3. Public functions that could be declared external in order to save gas

Contracts Affected - newRebase.sol, VooDoo.sol, rbDirect.sol

Description

Whenever a function is not called internally, it is recommended to define them as external instead of the public, in order to save gas. For all the public functions, the input parameters are copied to memory automatically, and it costs gas. If the function is only called externally, then you should explicitly mark it as external. External function parameters are not copied into memory but are read from calldata directly. This small optimization in your solidity code can save you a lot of gas when the function input parameters are huge.

Here is a list of functions that could be declared external:

In newRebase.sol:
- changeFixedRate

In VooDoo.sol:
- totalSellFeePercent

In rbDirect.sol:
- easySell

Recommendation

Consider declaring the above functions as external in order to save some gas.

Status

Fixed

A4. Missing zero address validation

Contracts Affected - newRebase.sol, VooDoo.sol, rbDirect.sol

Description

There are multiple functions in contracts that are missing zero address validation. Adding a zero address check is necessary because, in Ethereum, a zero address is something to which if any funds or tokens are transferred, it can not be retrieved back. In this case, there won't be any loss of token but if the amount or address is zero it would be a wastage of gas and might cause some other issues. Hence, it is recommended to add a check for zero addresses.

In newRebase.sol:

- initialize
- setForceRebaseAddress
- setPair

In VooDoo.sol:

- setTransferParameters
- setMonetaryPolicy
- Permit

In rbDirect.sol:

- constructor
- easySellForRecipient
- setFeeAddress

Recommendation

Consider adding a require statement that validates input against zero address to mitigate the same.

Status

Fixed

Auditors' comment: Defi Magic team decided to fix this issue by adding zero address validation at all the instance except setForceRebaseAddress. The check wasn't added to setForceRebaseAddress for decentralization purpose.

A5. Missing Explicit Visibility

Contracts Affected - newRebase.sol, VooDoo.sol, rbDirect.sol

Description

Explicitly declaring the state variables' default visibility makes it easier to catch incorrect assumptions about who can access the variable. It also makes the code less dependent on compiler defaults that may be subject to change increasing the maintainability of the code.

In newRebase.sol:

- forceRebase
- USDC_ADDRESS
- WETH_ADDRESS

In VooDoo.sol:

- feeThreshold
- ZapperContract
- USDC_ADDRESS
- WETH_ADDRESS
- IGNORED_ADDRESSES
- unrestricted

In rbDirect.sol:

- SLIPPAGE
- WETH_ADDRESS

Recommendation

Consider declaring appropriate visibility to avoid any kind of confusion.

Status

Fixed

A6. Incorrect Naming Convention

Contracts Affected - newRebase.sol, rbDirect.sol

Description

While developing contracts, a naming convention is usually followed by developers across most of the smart contracts. An underscore is added before the name of any private and internal functions. While public and external functions do not have an underscore. This is to avoid any kind of confusion and follow the same pattern across all the contracts. Here this pattern is not followed. It is recommended to follow this pattern and avoid any kind of confusion.

In newRebase.sol:

- computeSupplyDelta
- withinDeviationThreshold
- rebaseLagDeprecated

In rbDirect.sol:

- zapToETH
- zapFromETH

Recommendation

Consider adding an underscore (_) prior to the name of any private or internal function name or variable.

Status

Fixed

A7. Unused code

Contracts Affected - newRebase.sol, VooDoo.sol

Description

Description - While auditing, it has been found that the contracts have multiple lines of code which are not being used. It is recommended to remove the unused code in order to save gas and improve the overall code quality

In newRebase.sol:

- The code is a fork of Ampleforth which has a function named rebaseLag. However, Ampleforth added this function only to maintain backward compatibility. In this case the function rebaseLag serves no purpose and must be removed. Moreover, rebaseLagDeprecated is also not used and must be removed.

In VooDoo.sol:

- WETH_ADDRESS
- USDC_ADDRESS
- PeggedValueChange

Recommendation

Consider Removing the unused code.

Status

Fixed

A8. Require statements without reason string

Contracts Affected - newRebase.sol, VooDoo.sol

Description

The contracts have multiple require statements. However, the reason string (error message) is missing. It is a best practice to have a human-readable revert reason, unique across the contract. Hence, whenever a transaction fails, the proper human-readable error is displayed.

In newRebase.sol: (#L-106, 109, 140, 145, 150, 181, 182, 106, 109)

In VooDoo.sol: (#L 55, 63, 64, 606, 616)

Remediation

Consider adding a reason string to error message

Status

Fixed

A9. Third-Party Dependencies

Contracts Affected - newRebase.sol, rbDirect.sol

Description

The logic of the contract requires it to interact with third-party protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts, such as increasing fees of 3rd parties, migrating to new LP pools, etc.

Remediation

We understand that the business logic of the defi magic contract requires interaction with third-party protocols. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Status

Acknowledged

B. VooDoo.sol

High Severity Issues

No issues found

Medium Severity Issues

B1. Centralization issue

Line #231

```
function setTransferParameters(address _feeAddress, uint256 _transferFeePercent, uint256 _buyFeePercent, ui

    require(_feeAddress != address(0), "Blood: _feeAddress cannot be zero address");
    require(_minTransferPercent<=100000,"_minTransferPercent cannot be more than 100%");
    require(_transferFeePercent.add(_buyFeePercent).add(_fixSellFeePercent)<=100000 - _minTransferPercent,'

    FEE_ADDRESS=_feeAddress;
    transferFeePercent = _transferFeePercent;
    buyFeePercent=_buyFeePercent;
    fixSellFeePercent=_fixSellFeePercent;
    varSellFeePercentAllow = _varSellFeePercentAllow;
    minTransferPercent = _minTransferPercent;
    emit TransferParamsSet(FEE_ADDRESS, transferFeePercent, buyFeePercent, fixSellFeePercent, varSellFeePer
}
```

Description

VooDoo.sol contract has a function named setTransferParameters that allows owner to set values of various crucial parameters like feeAddress, _transferFeePercent, _fixSellFeePercent, _buyFeePercent etc. However, there is no check on the inputs to the function. This basically means owner can set the values of these parameters to whatever value he wishes. In an event of owner account compromise or owner not acting judiciously this can be extremely dangerous since these values directly affect purchase, sell and transfer of tokens. For instance if these values are set too high, no one will be able to buy or sell tokens. Apart from that, there is a parameter named minTransferPercent which basically means this percent to the total traction amount must go to the receiver regardless of the fees. If the value of minTransferPercent is set too high, the fees will be affected which may create further issues in maintaining the price of the token stable. It is recommended to add checks ensure these values are always within the predetermined threshold.

Exploit scenario

The token does not rebase downwards and has Variable sell fees in place to stabilize the price whenever the price goes down. For instance, if the price is $0.94 then 6% variable sell fee will be charged. Now if the minTransferPercent is set to 33%, due to check on L#365 and 434 variable sell fee can never be more than 67%. Another possible issue can be if the fixed sell fee or buy is set to ~100%, it will make it practically impossible to use the contract.

Remediation

This issue can be prevented by adding a small require check that ensures the fees can not be more than specific preset values.

Status

Acknowledged

Low Severity Issues

B2. Out of gas issue

Line #209

```
function setIgnoredAddressBulk(address[] memory _ignoredAddressBulk, bool ignore)external onlyOwner{

    for(uint256 i=0;i<_ignoredAddressBulk.length;i++){
        address _ignoredAddress = _ignoredAddressBulk[i];
        IGNORED_ADDRESSES[_ignoredAddress] = ignore;
    }
}
```

Description

The contract has a function named setIgnoredAddressBulk() which can be used by owner of the contract to set multiple addresses to ignored_addresses. However, there is no check-in place to limit the number of addresses to be ignored. Thus, a large number of addresses at the same time may lead the contract to run out of gas during loop execution.

Remediation

Add a limit to the number of addresses that can be ignored at one time.

Status

Acknowledged

Auditors' Comment: They decided to leave this issue unfixed for a few valid reason: 1) This function is only callable by the owner of the contract. Hence normal user won't be affected. 2) In the worst case scenario even if the length of ignored addresses increases to a point at which the transaction runs out of gas and fails, the worst it can lead to is transaction failure. Since there isn't much impact on normal users, the team decided to not fix this issue at this point.

Informational Issues

No issues found

C. NewRebase.sol

High Severity Issues

No issues found

Medium Severity Issues

C1. Centralization Risk - Rebase percentage can be manipulated by owner

Description

The contract is a fork of Ampleforth which uses a sigmoid curve to calculate the rebase percentage. The results from the sigmoid curve depend on the parameters set by the owner. Here their parameters do not have any validation and can be manipulated by owner. These changes can result in a miscalculation of rebase percentage. Additionally, there are certain parameters such as minRebaseTimeIntervalSec which are under the control of owner and can be manipulated. For instance, if minRebaseTimeIntervalSec is set to 1 year, no one except owner will be able to call rebase function until the time has passed. While we do acknowledge that there needs to be some sort of owner interference in the contract in order to maintain smooth functioning, we would recommend defi magic developers add a few small checks that ensure the parameters can not be manipulated above a certain fixed threshold.

Remediation

This issue can be prevented by adding a small require check that ensures that sensitive parameters are in accordance with preset values.

Status

Acknowledged

Low Severity Issues

No issues found

Informational Issues

C2. Relying on block.timestamp

Description

In newRebase.sol contract, a control flow decision is made based on The ‘block.timestamp’ environment variable. Note that the values of variables like coinbase, gas limit, block number, and timestamp are predictable and can be manipulated by malicious miners. Also, keep in mind that attackers know hashes of earlier blocks. Don’t use any of those environment variables as sources of randomness and be aware that the use of these variables introduces a certain level of trust into miners. This is not a major issue here since these values can not be manipulated too far ahead in the future and the business logic of the contract is in such a way that it doesn’t affect much. However, developers must keep in mind that these values can be manipulated by miners.

Remediation

Developers should write smart contracts with the notion that block values are not precise, and their use can lead to unexpected effects. Alternatively, they may make use of oracles.

Status

Acknowledged

D. rbDirect.sol

High Severity Issues

No issues found

Medium Severity Issues

No issues found

Low Severity Issues

No issues found

Informational Issues

D1. Missing non-reentrant modifier

Description

The rbDirect contract has a function named easySellForRecipient which allows user to sell their tokens and transfer received funds to the recipient address. Even though there are necessary checks in place to avoid reentrancy, it is recommended to add a non-reentrant modifier to ensure there is no reentrancy.

Remediation

Consider adding a non-reentrant modifier to the easySellForRecipient function.

Status

Fixed

Auditors' Comments

VooDoo token is a rebasing token that works in the following manner:
The price of the token is fixed and there is a deviation threshold (if the price is within the deviation threshold no changes are made to the supply or fees). Assuming the price of the token is fixed to $1.05 and $0.05 is the deviation threshold, the supply or fees won't be affected until the price of the token is between $1 and $1.10. Now there are two possibilities:

- If the price of the token goes up, in this case, the newrebase contract calculates the rebase percentage and increases the total supply of the token which will eventually bring the price down.

- If the price of the token goes down, the contract has an interesting approach of fees rather than downward rebasing. If the price is $0.94, 6% variable sell fees will be charged whenever someone tries to sell the token. This 6% fee will be transferred to a fee address set by the owner.

The fees will be transferred to the already existing ecosystem growth fund. The fees accumulated in the ecosystem will be used for staking on various platforms. The income generated from all these will be used to add liquidity, buyback tokens, etc. However, users should note that there is no full-proof calculation regarding this (It is extremely difficult to calculate the exact amount since there are lots of variable factors involved). The devs believe that the fees accumulated will be sufficient to stabilize the price and run the project smoothly. It is recommended for the users to do their own research.

The price of the token with respect to USDC is calculated by using uniswap's getAmountsOut function. There a few things that must be noted here. The developers should never assume that 1 USDC is always equal to 1 USD. Additionally, the price of the token is a very sensitive factor in the project which solely depends on the value returned by uniswap. Developers should not blindly trust price from one source and use alternative sources like oracles to fetch the prices of all the tokens involved and then determine the price.

Functional Testing

Voodoo.sol

- ✓ Should test getters
- ✓ Should test all setter functions
- ✓ Should test access controls on setter functions.
- ✓ Should rebase
- ✓ Total Supply after rebase must not be more than MAX_SUPPLY
- ✓ Should increase the supply of token
- ✓ Only MonetaryPolicy should be able to call rebase
- ✓ Should test totalSellFeePercent
- ✓ Should test approve and transfer from
- ✓ Should test increase allowance and decrease allowance
- ✓ Should test transfer and transferfrom
- ✓ Should test fees calculation in _transferGon
- ✓ Should test permit

NewRebase.sol

- ✓ Should test getters
- ✓ Should test all setter functions
- ✓ Should test rebase function
- ✓ Rebase must revert if called minRebaseTimeIntervalSec
- ✓ Should test exchange rate must be always less than or equal to MAX_RATE
- ✓ Total Supply after rebase must not be more than MAX_SUPPLY
- ✓ Should test getTokenUSDPrice
- ✓ Should test computeRebasePercentage
- ✓ If price is within deviation threshold, rebase should not happen
- ✓ Should computeRebasePercentage
- ✓ Should test access controls on setter functions.

rbDirect.sol

- ✓ Should test getters
- ✓ Should test all setter functions
- ✓ Should test zapToETH
- ✓ Should test zapFromETH
- ✓ Should test addLiquidityETH
- ✓ Should test removeLiquidityETH
- ✓ Should test addLiquidityToken
- ✓ Should test removeLiquidityToken
- ✓ No one else other than the person who added the liquidity must be able to remove it.
- ✓ Should test easyBuy
- ✓ Should test easySell
- ✓ Should test easySellForRecipient

Automated Tests

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

Closing Summary

In this report, we have considered the security of Defi Magic. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. At the end, Defi magic Team Resolved and Acknowledged the Issue.

Disclaimer

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

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
July, 2022

For

Magic
spellbinding crypto

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com