QuillAudits

Audit Report
August, 2021

For

Magic
spellbinding crypto

Contents

Overview

DeFiMagic
DeFi Magic is a cross-chain DeFi burning utility token ecosystem

Scope of the Audit

The scope of this audit was to analyse DeFiMagic smart contract's codebase for quality, security, and correctness.

DeFiMagic Contracts
Commit: 3e7b04353d9f43e1c62d7f766a60b3f00b382192
Fixed In: 21b12edb7e743c53f826de9952c7b8a90c388927

Magic.sol	Preminted ERC20 contract with transfer gate to apply fees, burn rates and allow approved pools and routers.
FloorCalculator.sol	To calculate floor calculations checking for locked LP tokens to one in the market.
MagicEventGate.sol	Event gate that zaps locked Magic to Wizard token. Owner sets the time and locking parameters over the time.
MagicTransferGate.sol	For any transfer of Magic token, it should pass through these checks and apply rates and checks.
StakingRewards.sol	Wizard being the staking token, Magic being the multiplier token rewards Magic token for all who staked. One who withdraws before locktime passes pays a fee set dynamically by the owner.
WizardEventGate.sol	Event gate that locks Wizard token. The owner sets the time and locking parameters over the time so that holder receives their Wizard holding from buy gradually back.

axBNB.sol	Wrapped ERC20 with ERC31337 for BNB with dynamic fees.
axBNBDirect.sol	To swap BNB/axBNB token to Magic token and vice versa via PanCakeSwap.
axBNBLiquidity.sol	axBNB<->Magic LP token when deposited mints the Wizard token to sender with dynamic fees.
wizardDirect.sol	To swap BNB/axBNB/Magic token to Wizard token and vice versa via PanCakeSwap.

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
- Exception Disorder
- Gasless Send
- Use of tx.origin
- Malicious libraries
- Compiler version not fixed
- Address hardcoded
- Divide before multiply
- Integer overflow/underflow
- ERC20 transfer() does not return Boolean
- ERC20 approve() race
- 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
Mythril, Slither, SmartCheck, Surya, Solhint.

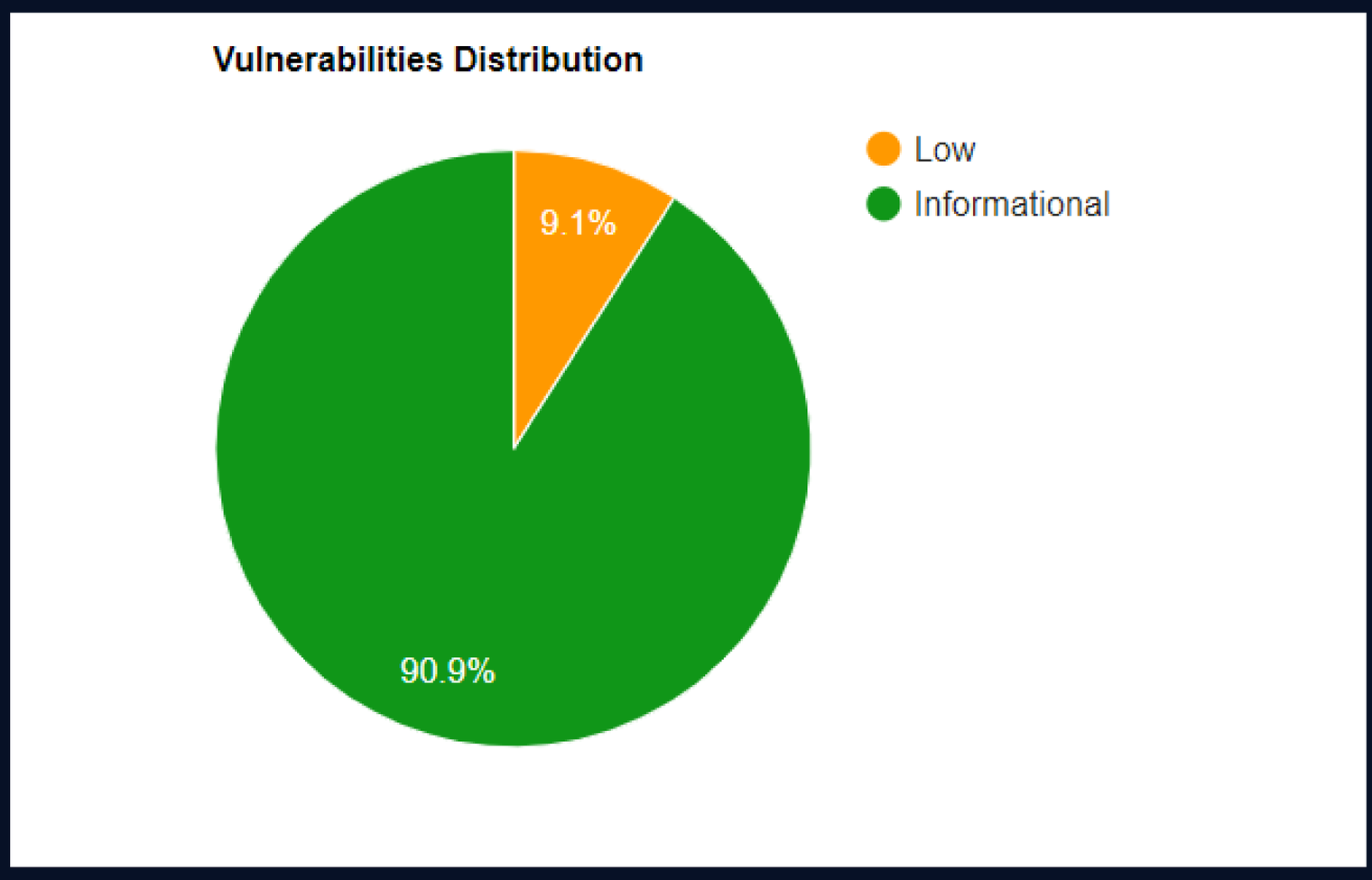
Issue Categories

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.

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 to 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 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	1	10
Acknowledged	0	0	0	5
Closed	7	2	16	6



Functional Testing Results

Function Names	Testing results
Magic.sol	Passed
FloorCalculator.sol	Passed
MagicEventGate.sol	Passed
MagicTransferGate.sol	Passed
StakingRewards.sol	Passed
WizardEventGate.sol	Passed
axBNB.sol	Passed
axBNBDirect.sol	Passed
axBNBLiquidity.sol	Passed
wizardDirect.sol	Passed

Issues Found

High severity issues

axBNB

- [FIXED] No Fee Deduction from the user's balance while withdrawing amount

[120-145] function withdraw() doesn't deduct the feeAmount from the user's balance and burns the amountAfterFee, meaning the feeAmount worth of axBNB is still available which the user can withdraw later

```
130         else{
131             uint256 feeAmount = _amount.mul(FEE).div(100000);
132             if(feeAmount>0){
133                 _balanceOf[FEE_ADDRESS] = _balanceOf[FEE_ADDRESS].add(feeAmount);
134                 emit Transfer(msg.sender, FEE_ADDRESS, feeAmount);
135             }
136             uint256 amountAfterFee = _amount.sub(feeAmount);
137             _burn(msg.sender, amountAfterFee);
138             IWETH(address(wrappedToken)).withdraw(amountAfterFee);
139             (bool success,) = msg.sender.call{ value: amountAfterFee }("");
140             require (success, "Transfer failed");
141 
142             emit Withdrawal(msg.sender, amountAfterFee);
```

Case
Considering, FEE = 1000 => 1%
_amount = 100
feeAmount = _amount * FEE / 100000 => 1
amountAfterFee = _amount - feeAmount => 100 - 1 => 99
_burn(amountAfterFee)

Hence, 1 axBNB, is still available for the user to withdraw, the feeAmount of which will be 0
So, this way, a user has accumulated all it's 100axBNB bypassing the fee Collection

Recommendation
Deduct feeAmount from msg.sender

axBNBDirect

- [FIXED] 100% or More than 100% Fee deduction

```
150             if(axBNB.isIgnored(msg.sender)==false){
151
152                 uint feeAXBNB = axBNB.FEE();
153                 address feeAddress = axBNB.FEE_ADDRESS();
154
155                 uint feeAmount= axBNBAmount.mul(feeAXBNB).div(1000);
156                 uint remAmount = axBNBAmount.sub(feeAmount);
157                 axBNB.transfer(feeAddress, feeAmount);
```

[#L155] calculates feeAmount from axBNBAmount and subtracts from it. The Project considers 100,000 as 100%, so the feeAXBNB will also be having 1000 as a 1% factor. But here, the divisor is 1000 instead of 100,000. As a result of which, it can deduct 100% or more than 100% fees from the amount.

Recommendation

Use 100,000 as 100% factor as a divisor

MagicEventGate

- [FIXED] Wrong calculation for totalZapFactor

```
148     function setZapParams(uint256 _startZapTime, uint256 _startingZapFeePercent, uint256 _totalEventGateTime, uint256 _reductionSector,
149                           uint256 _reductionRate) external onlyOwner{
150
151         uint256 powerPercent = _totalEventGateTime.div(_reductionSector);
152         uint256 reductionFeeMultiplier = 100000 - _reductionRate;
153
154         uint256 factor = reductionFeeMultiplier;
155         uint256 prevFactor = reductionFeeMultiplier;
156         for(uint256 i=1;i<=powerPercent;i++){
157             factor = prevFactor.mul(reductionFeeMultiplier).div(100000);
158             prevFactor = factor;
159         }
160
161         uint256 totalZapFactor = _startingZapFeePercent.mul(factor).div(10000);
162
163         require(totalZapFactor>100,"Fees will go less than 0.1% in given _totalEventGateTime");
164
165         zapParams.startZapTime = _startZapTime;
166         zapParams.startingZapFeePercent = _startingZapFeePercent;
167         zapParams.totalEventGateTime = _totalEventGateTime;
168         zapParams.reductionSector = _reductionSector;
169         zapParams.reductionRate = _reductionRate;
170     }
```

[#L161] calculates totalZapFactor as
_startingZapFeePercent.mul(factor).div(10000);
Project considers 100,000 as 100%, but the divisor used here is 10000 as 10% which will calculate a 10x more zap factor large enough to satisfy the require check at #L163 most of the time, which may result into inconsistent values being set for zapParams

Recommendation
Use 100,000 as 100% factor as a divisor

- **[FIXED] Sending 0 while trying to emergencyClaimWizard**

```
376             if(magicLockedOfHolder>totalSwappedMagic){
377                 totalSwappedMagic = 0;
378
379                 if(totalAvailableWizard<wizardToTransfer) {
380                     totalAvailableWizard = 0;
381                     IERC31337(wizardToken).transfer(holder, totalAvailableWizard);
382                 }
```

[#L369-395] function emergencyClaimWizard tries to send the residue to a holder address, but sets the totalAvailableWizard to 0 prior to sending it to the address. Hence the holder's address will be getting nothing.

Recommendation
copy the totalAvailableWizard value to a local variable
Prior setting it to 0

wizardDirect

- **[FIXED] Unused BNB**

[#L91-108] function easyBuyDirect() divides the supplied BNB into half. The first half is used to get axBNB by depositing BNB into the axBNB contract, the second half is used to buy Magic. This Magic is then swapped to wizard and sent to the User. But the first half of the supplied BNB remains unused. As there is a Magic<->Wizard pool. The whole BNB supplied can be used to buy Magic and then can be swapped to Wizard.

```
90      //  BNB => Wizard
91      function easyBuyDirect() external payable nonReentrant
92      {
93
94          uint256 axBNBTotal=SafeMath.div(msg.value,2);
95          axBNB.deposit{ value: axBNBTotal }();
96
97          uint256 magicAmtTotal = axBNBDirect.easyBuy{ value: axBNBTotal }();
98
99          // swap magic to Wizard
100         address[] memory path = new address[](2);
101         path[0] = address(magic);
102         path[1] = address(Wizard);
103         (uint256[] memory amountsMin) = UniswapV2Library.getAmountsOut(address(uniswapV2Factory), magicAmtTotal, path);
104         uint256 WizardMin = amountsMin[1].mul(100000-SLIPPAGE_Wizard).div(100000);
105
106         uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(magicAmtTotal, WizardMin, path, msg.sender, block.timestamp);
107
108     }
109
```

Recommendation
Whole BNB can be used to buy Magic

- **[FIXED] 100% or More than 100% Fee deduction**

```
266
267         uint feeAmount= axBNBAmt.mul(axBNB.FEE()).div(1000);
268         uint remAmount = axBNBAmt.sub(feeAmount);
269         axBNB.transfer(axBNB.FEE_ADDRESS(), feeAmount);
270         axBNB.withdraw(remAmount);
271
272         (bool success,) = msg.sender.call{ value: remAmount }("");
273         require (success, "Transfer failed");
274
```

[#L267] calculates feeAmount from axBNBAmt and subtracts from it. The Project considers 100,000 as 100%, so the axBNB.FEE() will also be having 1000 as 1% factor. But here the divisor is 1000 instead of 100,000. As a result of which, it can deduct 100% or more than 100% fees from the amount

Recommendation
Use 100,000 as 100% factor as a divisor

WizardEventGate

- [FIXED] Wrong calculation for totalZapFactor

```
97      function setZapParams(uint256 _startZapTime, uint256 _startingZapFeePercent, uint256 _totalEventGateTime, uint256 _reductionSector,
98                            uint256 _reductionRate) external onlyOwner{
99
100         uint256 powerPercent = _totalEventGateTime.div(_reductionSector);
101         uint256 reductionFeeMultiplier = 100000 - _reductionRate;
102
103         uint256 factor = reductionFeeMultiplier;
104         uint256 prevFactor = reductionFeeMultiplier;
105         for(uint256 i=1;i<=powerPercent;i++){
106             factor = prevFactor.mul(reductionFeeMultiplier).div(100000);
107             prevFactor = factor;
108         }
109
110         uint256 totalZapFactor = _startingZapFeePercent.mul(factor).div(10000);
111
112         require(totalZapFactor>100,"Fees will go less than 0.1% in given _totalEventGateTime");
113
114         zapParams.startZapTime = _startZapTime;
115         zapParams.startingZapFeePercent = _startingZapFeePercent;
116         zapParams.totalEventGateTime = _totalEventGateTime;
117         zapParams.reductionSector = _reductionSector;
118         zapParams.reductionRate = _reductionRate;
119     }
```

[#L110] calculates totalZapFactor as _startingZapFeePercent.mul(factor).div(10000);
Project considers 100,000 as 100%, but the divisor used here is 10000 as 10% which will calculate a 10x more zap factor large enough to satisfy the require check at #L112 most of the time, which may result into inconsistent values being set for zapParams

Recommendation
Use 100,000 as 100% factor as a divisor

Medium severity issues

wizardDirect

- **[FIXED] Fee Deduction for all. No concept of ignoredAddress**

  [#L248-280] function easySellToBNB() deducts fee on the BNB amount while trying to sell Wizard for every user irrespective of the ignoredAddress status. In other words there is no concept of ignoredAddress unlike function sell in axBNBDirect contract, where only non - ignoredAddress are subject to pay fees for selling Magic to BNB

  Recommendation
  Implement fees deduction only for non - ignoredAddress

MagicEventGate

- **[FIXED] Loss of precision while performing division before multiplication**

```
369     function emergencyClaimWizard(address holder) external onlyOwner{
370
371         uint256 magicLockedOfHolder = balanceLocked[holder];
372         uint256 wizardPerMagic = totalAvailableWizard.div(totalSwappedMagic);
373         uint256 wizardToTransfer = magicLockedOfHolder.mul(wizardPerMagic);
374
```

[#L369-395] function emergencyClaimWizard performs division before Multiplication to calculate wizardToTransfer which may result into loss of some funds and precision.

Recommendation
Performing multiplication before division can sometimes avoid loss of precision.

Low level severity issues

axBNB

- **[FIXED] Missing Events for critical parameters**

  [#L75-78] function setFee: No event for FEE_ADDRESS and FEE updates,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

Magic

- **[FIXED] Missing Zero Address Validation**

  [#L57-60] function setLPAddress
  [#L63-66]function setZapper

  Missing Zero Address checks for _LPAddress, _magicZapper and _wizardZapper addresses

  Recommendation
  Add a Zero Address check.

- **[FIXED] Missing Events for critical parameters**

  [#L51-55] function setTransferGates
  [#L57-60] function setLPAddress
  [#L63-66]function setZapper

  No event for updated values,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

- **[FIXED] Missing Events for critical parameters**

  [#L40-43] function setSlippage: No event for _slippage updates,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

MagicTransferGate

- **[FIXED] Missing Events for critical parameters**

  [#L75-78] function setSlippage
  [#L98-111] function setParameters
  [#L222-225] function setAddressState

  No event for updated values,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

MagicEventGate

- **[FIXED] Missing Events for critical parameters**

  [#L123-125] function setSlippage
  [#L131-133] function setFeesForLock
  [#L136-141]function setTokenAddresses
  [#L148-170]function setZapParams

  No event for updated values,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

- **[FIXED] Missing Zero Address Validation**

  [#L68-115] function initialize
  [#L136-141]function setTokenAddresses

  Missing Zero Address checks for _axBNBToken, _wizardToken and _magicToken addresses

  Recommendation
  Add a Zero Address check.

wizardDirect

- **[FIXED] Missing Events for critical parameters**

  [#L56-59] function setSlippage: No event for _slippage_Wizard updates,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

- **[FIXED] Unnecessary Statements (Route Path Creation)**

  [#L112-150] function easyBuyFromAXBNB

```
124             address[] memory path = new address[](2);
125             path[0] = address(axBNB);
126             path[1] = address(magic);
127
```

Recommendation
Emit an event for critical parameter changes.

WizardEventGate

- **[FIXED] No setters available to update FEE_PERCENT_LOCKED and slippage**

- **[FIXED] Missing Events for critical parameters**

  [#L88-90]function setTokenAddresses
  [#L97-119]function setZapParams

  No event for updated values, which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

- **[FIXED] Missing Zero Address Validation**

  [#L56-81] function initialize
  [#L88-90]function setTokenAddresses

  Missing Zero Address checks for _wizardToken address

  Recommendation
  Add a Zero Address check.

axBNBLiquidity

- **[FIXED] Missing Zero Address Validation**

  [#L147-150] function setLPAddress
  [#L152-154] function setZapper
  [#L172-177] function setTransferParameters

  Missing Zero Address checks for _LPAddress, _wizardZapper, _stakeAddress and _treasuryAddress addresses

  Recommendation
  Add a Zero Address check.

- **[FIXED] Missing Events for critical parameters**

  [#L147-150] function setLPAddress
  [#L152-154] function setZapper
  [#L172-177] function setTransferParameters

  No event for updated values,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

StakingRewards

- **[OPEN] Old compiler has been used. Use the latest compiler to avoid bugs introduced in the older versions**

- **[FIXED] Missing Events for critical parameters**

  [#L18-20] function setRewardsDistribution
  [#L240-249] function setTransferParams
  [#L456-458] function setOnMultiplierAmount

  No event for updated values,which will make it difficult to track down the new updates.

  Recommendation
  Emit an event for critical parameter changes.

- **[FIXED] Missing Zero Address Validation**

  [#L18-20] function setRewardsDistribution
  [#L77-114] function initialize
  [#L240-249] function setTransferParams

  Missing Zero Address checks for _rewardsDistribution, _stakingPoolFeeAdd, _devFundAdd addresses

  Recommendation
  Add a Zero Address check.

Informational

- [FIXED] Public functions that are never called by the contract should be declared external to save gas.

- [OPEN] Use SafeERC20, or ensure that the transfer/transferFrom return value is checked.

- [OPEN] ERC20 approve() race

  The standard ERC20 implementation contains a widely-known racing condition in its approve function, wherein a spender is able to witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender is able to spend their entire approval amount twice.

  Same issues have been found in the function safeApprove() and has been deprecated.

  Reference
  https://eips.ethereum.org/EIPS/eip-20

axBNB

- [OPEN] contract works with BEP20 tokens, so it is better to name contract interfaces appropriately, while the contract use names as ERC31337, WrappedERC20, IWETH

- [Acknowledged by Developer] [#L58] FEE is currently 0. Make sure to set an appropriate value prior to deployment of the contract

Magic

- **[OPEN] Unused Return**

  [#L69-132] function _transfer ignores return value by

  [#L84, #L105]eventGate.handleZap(sender, recipient, remaining);

axBNBDirect

- **[FIXED] [#L150] axBNB.isIgnored(msg.sender)==false**

  in [#L128-167]function sell compares with a boolean constant, whereas Boolean constants can be used directly and do not need to be compared to true or false.

- **[FIXED] The contract deals with BNB but refers Matic at different places**

```
87      function buy(uint256 magicOutMin) public payable nonReentrant returns (uint256 darkMagicAmount)
88      {
89          uint256 amount = msg.value;
90          require (amount > 0, "Send MaticIn to buy");
91          uint256 magicPrev=magic.balanceOf(address(this));
92
```

```
128     function sell(uint256 darkMagicAmountIn, uint256 axBNBOutMin) public nonReentrant returns (uint256 maticAmount)
129     {
130         require (darkMagicAmountIn > 0, "Nothing to sell");
131         IMagicTransferGate gate = IMagicTransferGate(address(magic.transferGate()));
```

```
147
148             // will be applied only if MATIC payout is happening
149             //else IGNORED_ADDRESSES in axBNB will handle
150             if(axBNB.isIgnored(msg.sender)==false){
151
```

Recommendation
Replace the appearances of word Matic with BNB so as to avoid confusions

- **[OPEN] The contract works with PancakeSwap: Router and Factory, but contract uses uniswap interfaces**

```
18      IAxBNB immutable axBNB;
19      IGatedERC20 immutable magic;
20      IUniswapV2Router02 private uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);
21      IUniswapV2Factory private uniswapV2Factory = IUniswapV2Factory(0xcA143Ce32Fe78f1f7019d7d551a6402fC5350c73);
22
```

Recommendation
Replace UniSwap interfaces with PancakeSwap to avoid confusions and issues.

axBNBLiquidity

- **[OPEN] Unused Return**

  [#L179-227] function _transfer ignores return value by
  [#L191, #L207] wizardEventGate.lockWizard(sender, recipient, amount);

axBNBLiquidity

- **[Acknowledged by Developer] [#L92] MIN_MAGIC is currently 0. Make sure to set an appropriate value prior to deployment of the contract**
- **[Acknowledged by Developer] No function available to remove Admins**
- **[FIXED] Use safeMath.add instead of + operator to avoid any overflows**
- **[FIXED]** [#L230] isHolderAddress[recipient]==false in [#L205-237] function handleZap
  [#L356] AdminByAddress[msg.sender] == true in [#L353-363] function claimWizardForHolder
  [#L433] AdminByAddress[msg.sender] == true in [#L432-435] modifier onlyAdmin

  compares with a boolean constant, whereas Boolean constants can be used directly and do not need to be compared to true or false.

- **[OPEN] [#L175, #L212] block.timestamp has been used for comparisons. Avoid using block.timestamp, as it can be manipulated by miners**

WizardEventGate

- **[Acknowledged by Developer] [#L72] MIN_WIZARD is currently 0. Make sure to set an appropriate value prior to deployment of the contract**

- **[Acknowledged by Developer] No function available to remove Admins**

- **[FIXED]** [#L175] isHolderAddress[recipient]==false in [#L152-183]function lockWizard

  [#L232] AdminByAddress[msg.sender] == true in [#L229-238] function claimWizardForHolder

  [#L270] AdminByAddress[msg.sender] == true in [#L260-272] modifier onlyAdmin

  compares with a boolean constant, whereas Boolean constants can be used directly and do not need to be compared to true or false.

- **[OPEN] [#L123, #L158] block.timestamp has been used for comparisons. Avoid using block.timestamp, as it can be manipulated by miners**

WizardDirect

- **[OPEN] Unused Return**

  [#L220-244] function easySellToAXBNB ignores return value by

  [#L236]axBNBDirect.easySellToAXBNB(magicAmtAfterSwap.sub(prevmagicAmount));

StakingRewards

- **[OPEN] [#L296, #L317] block.timestamp has been used for comparisons. Avoid using block.timestamp, as it can be manipulated by miners**

Closing Summary

Several issues of high, medium, and low severity have been reported during the audit. Most of them have now been fixed.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of the code. Besides a security audit, please don’t consider this report as investment advice.

Audit Report
August, 2021

For

Magic
spellbinding crypto

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com