BLOCKHAT
SECURITY

Green Rabbit

Smart Contract Security Audit

Prepared by BlockHat

January 30th, 2023 - January 31st, 2023

BlockHat.io

contact@blockhat.io

Document Properties

Client	greenrabbit
Version	1.0
Classification	Public

Scope

The Green Rabbit Contract in the Green Rabbit Repository

Repo	Owner
https://alveyscan.com/address/0xbF025d699ff5687E31B0dEC75AF00C8F1d16A05F/contracts#address-tabs	0xA3d5070A9eD99bA9Ec8D4372fC9DC923C45F03Fd

Files	MD5 Hash
GreenRabbit.sol	b430bbc334537cc88b9b066b9c44c852

Contacts

COMPANY	CONTACT
BlockHat	contact@blockhat.io

Contents

1 Introduction

Green Rabbit engaged BlockHat to conduct a security assessment on the Green Rabbit beginning on January 30th, 2023 and ending January 31st, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Green Rabbit

-

Issuer	greenrabbit
Website	–
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

BlockHat used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

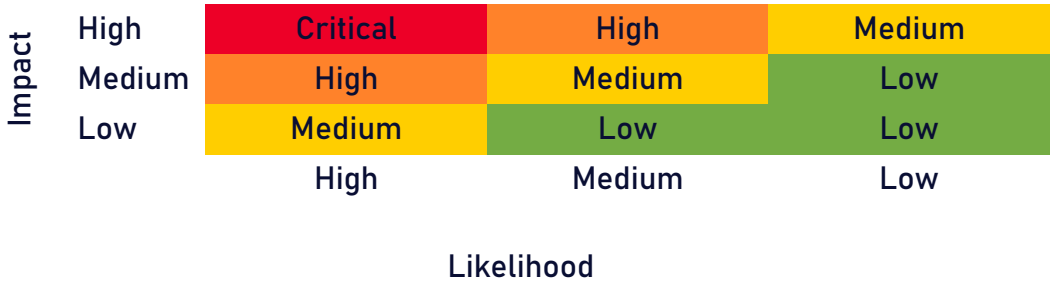
1.2.1 Risk Methodology

Vulnerabilities or bugs identified by BlockHat are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.



2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Green Rabbit implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 2 critical-severity, 2 high-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Auth can control Swap settings	CRITICAL	Not Fixed
Auth can control fees	CRITICAL	Not Fixed
Wrong syntax	HIGH	Not Fixed
Bad Implementation	HIGH	Not Fixed
Missing address verification	LOW	Not Fixed
Avoid using .transfer() to transfer Ether	LOW	Not Fixed
Floating Pragma	LOW	Not Fixed

3 Finding Details

A GreenRabbit.sol

A.1 Auth can control Swap settings [CRITICAL]

Description:

The setSwapBackSettings function grants the owner the ability to disable swapping, leading to centralization risk. Additionally, setting a high swapThreshold value is equivalent to disabling the swap function.

Code:

Listing 1: GreenRabbit.sol

```
418     function setSwapBackSettings(bool _enabled, uint256 _amount)
            ↪ external authorized {
419         swapEnabled = _enabled;
420         swapThreshold = _amount;
421     }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

We recommend implementing a one-time enable for the swapEnabled variable and restricting the range of the swapThreshold variable.

Status - Not Fixed

A.2 Auth can control fees [CRITICAL]

Description:

The setFees setter gives the authorized person control over both fees and feeDominator The owner can set any fee value and even alter the feeDominator For example, if the feeDominator is set to 1 and the total fee is 100, users will end up paying ten times their original amount in fees.

Code:

Listing 2: GreenRabbit.sol

```
405     function setFees(uint256 _liquidityFee, uint256 _teamFee, uint256
            ↪ _marketingFee, uint256 _feeDenominator) external authorized {
406         liquidityFee = _liquidityFee;
407         teamFee = _teamFee;
408         marketingFee = _marketingFee;
409         totalFee = _liquidityFee.add(_teamFee).add(_marketingFee);
410         feeDenominator = _feeDenominator;
411     }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

We recommend fixing the 'feeDominator' variable to a constant value and limiting the total fees.

Status - Not Fixed

A.3 Wrong syntax [HIGH]

Description:

The function 'transferForeignToken' is intended to transfer other tokens within the contract to the msg.sender, however, it is transferring Ether instead.

Code:

Listing 3: GreenRabbit.sol

```
428     function transferForeignToken(address _token) public authorized {
429         require(_token != address(this), "Can't let you take all native
                ↪ token");
430         uint256 _contractBalance = IBEP20(_token).balanceOf(address(this)
                ↪ );
431         payable(marketingFeeReceiver).transfer(_contractBalance);
432     }
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

We recommend removing the last line in the function and using safe transfer library to transfer tokens instead.

Status - Not Fixed

A.4 Bad Implementation [HIGH]

Description:

The 'getTotalFee' function checks if the value of 'launchedAt + 1' is greater than or equal to the current block number. If this is true, the function returns 'feeDenominator - 1,' which means the first person to call 'transferFrom' will have 99

Code:

Listing 4: GreenRabbit.sol

```
302     function getTotalFee(bool selling) public view returns (uint256) {
303         if(launchedAt + 1 >= block.number){ return feeDenominator.sub(1);
                ↪ }
304         if(selling) { return totalFee.add(1); }
305         return totalFee;
306     }
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

We recommend removing this line of code

Status - Not Fixed

A.5 Missing address verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible.

Code:

Listing 5: GreenRabbit.sol

```
243     function approve(address spender, uint256 amount) public override
            ↪ returns (bool) {
244         _allowances[msg.sender][spender] = amount;
245         emit Approval(msg.sender, spender, amount);
246         return true;
247     }
```

Listing 6: GreenRabbit.sol

```
257     function transferFrom(address sender, address recipient, uint256
            ↪ amount) external override returns (bool) {
258         if(_allowances[sender][msg.sender] != type(uint256).max){
259             _allowances[sender][msg.sender] = _allowances[sender][msg.
                    ↪ sender].sub(amount, "Insufficient Allowance");
260         }

262         return _transferFrom(sender, recipient, amount);
263     }
```

Listing 7: GreenRabbit.sol

```
265     function _transferFrom(address sender, address recipient, uint256
            ↪ amount) internal returns (bool) { address verification
266         if(inSwap){ return _basicTransfer(sender, recipient, amount); }
```

```
268         checkTxLimit(sender, amount);

270         if (recipient != pair && recipient != DEAD) {
271             require(isTxLimitExempt[recipient]  _balances[recipient] +
                ↪ amount <= _maxWalletSize, "Transfer amount exceeds the
                ↪ bag size.");
272         }

274         if(shouldSwapBack()){ swapBack(); }

276         if(!launched() && recipient == pair){ require(_balances[sender] >
                ↪  0); launch(); }

278         _balances[sender] = _balances[sender].sub(amount, "Insufficient
                ↪ Balance");

280         uint256 amountReceived = shouldTakeFee(sender) ? takeFee(sender,
                ↪ recipient, amount) : amount;
281         _balances[recipient] = _balances[recipient].add(amountReceived);

283         emit Transfer(sender, recipient, amountReceived);
284         return true;
285     }
```

Listing 8: GreenRabbit.sol

```
287     function _basicTransfer(address sender, address recipient, uint256
            ↪ amount) internal returns (bool) {
288         _balances[sender] = _balances[sender].sub(amount, "Insufficient
                ↪ Balance");
289         _balances[recipient] = _balances[recipient].add(amount);
290         emit Transfer(sender, recipient, amount);
291         return true;
292     }
```

Listing 9: GreenRabbit.sol

```
398     function setIsFeeExempt(address holder, bool exempt) external
            ↪ authorized {
399         isFeeExempt[holder] = exempt;
400     }

402     function setIsTxLimitExempt(address holder, bool exempt) external
            ↪ authorized {
403         isTxLimitExempt[holder] = exempt;
404     }
```

Listing 10: GreenRabbit.sol

```
414     function setFeeReceiver(address _marketingFeeReceiver, address
            ↪ _teamFeeReceiver) external authorized {
415         marketingFeeReceiver = _marketingFeeReceiver;
416         teamFeeReceiver = _teamFeeReceiver;
417     }
```

Risk Level:

Likelihood – 1

Impact - 2

Recommendation:

It is recommended to verify that the addresses provided in the arguments are different from the address(0) .

Status - Not Fixed

A.6 Avoid using .transfer() to transfer Ether [LOW]

Description:

Although transfer() and send() are 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.

Code:

Listing 11: GreenRabbit.sol

```
423     function manualSend() external authorized {
424         uint256 contractETHBalance = address(this).balance;
425         payable(marketingFeeReceiver).transfer(contractETHBalance);
426     }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

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

Status - Not Fixed

A.7 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.5 . Contracts should be deployed using the same compiler version and flags that were used during the testing process.Lock-

ing the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 12: GreenRabbit.sol

```
243 pragma solidity ^0.8.5;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Not Fixed

4 Best Practices

BP.1 Division Before Multiplication

Description:

The result of integer division in solidity is an integer value. As a result, dividing before multiplying will result in inaccurate results, which may result in certain anomalies in the contract's logic. the dev team should put the multiplication operations before the division operations

Code:

Listing 13: GreenRabbit.sol

```
216         uint256 public swapThreshold = _totalSupply / 1000 * 3;
```

BP.2 Public functions can be external

Description:

Functions with a public scope that are not called inside the contract should be declared external to reduce the gas fees

Code:

Listing 14: GreenRabbit.sol

```
91      function authorize(address adr) public onlyOwner {
92          authorizations[adr] = true;
93      }
```

Listing 15: GreenRabbit.sol

```
98      function unauthorize(address adr) public onlyOwner {
99          authorizations[adr] = false;
100     }
```

Listing 16: GreenRabbit.sol

```
118     function transferOwnership(address payable adr) public onlyOwner {
119         owner = adr;
120         authorizations[adr] = true;
121         emit OwnershipTransferred(adr);
122     }
```

5 Static Analysis (Slither)

Description:

Block Hat expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
GreenRabbit.swapBack() (GreenRabbit.sol#324-365) sends eth to arbitrary
    ↪ user
        Dangerous calls:
        - (MarketingSuccess) = address(marketingFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBMarketing}() (GreenRabbit.sol#349)
        - (developmentSuccess) = address(teamFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBdevelopment}() (GreenRabbit.sol#351)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #functions-that-send-ether-to-arbitrary-destinations

Reentrancy in GreenRabbit._transferFrom(address,address,uint256) (
    ↪ GreenRabbit.sol#265-285):
        External calls:
        - swapBack() (GreenRabbit.sol#274)
                - router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ amountToSwap,0,path,address(this),block.timestamp)
                    ↪ (GreenRabbit.sol#335-341)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (
                    ↪ GreenRabbit.sol#349)
```

```
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (GreenRabbit
                    ↪ .sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,
                    ↪ marketingFeeReceiver,block.timestamp) (GreenRabbit.
                    ↪ sol#355-362)
        External calls sending eth:
        - swapBack() (GreenRabbit.sol#274)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (
                    ↪ GreenRabbit.sol#349)
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (GreenRabbit
                    ↪ .sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,
                    ↪ marketingFeeReceiver,block.timestamp) (GreenRabbit.
                    ↪ sol#355-362)
        State variables written after the call(s):
        - _balances[sender] = _balances[sender].sub(amount,Insufficient
            ↪ Balance) (GreenRabbit.sol#278)
        - _balances[recipient] = _balances[recipient].add(amountReceived)
            ↪  (GreenRabbit.sol#281)
        - amountReceived = takeFee(sender,recipient,amount) (GreenRabbit.
            ↪ sol#280)
                - _balances[address(this)] = _balances[address(this)].add(
                    ↪ feeAmount) (GreenRabbit.sol#311)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities

GreenRabbit.slitherConstructorVariables() (GreenRabbit.sol#180-449)
    ↪ performs a multiplication on the result of a divi sion:
        -swapThreshold = _totalSupply / 1000 * 3 (GreenRabbit.sol#216)
```

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

GreenRabbit.swapBack() (GreenRabbit.sol#324-365) ignores return value by
    ↪  router.addLiquidityETH{value: amountBNBLiqui dity}(address(this)
    ↪ ,amountToLiquify,0,0,marketingFeeReceiver,block.timestamp) (
    ↪ GreenRabbit.sol#355-362)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unused-return

GreenRabbit.setTxLimit(uint256) (GreenRabbit.sol#388-391) should emit an
    ↪  event for:
        - _maxTxAmount = amount (GreenRabbit.sol#390)
GreenRabbit.setFees(uint256,uint256,uint256,uint256) (GreenRabbit.sol
    ↪ #406-412) should emit an event for:
        - liquidityFee = _liquidityFee (GreenRabbit.sol#407)
        - teamFee = _teamFee (GreenRabbit.sol#408)
        - marketingFee = _marketingFee (GreenRabbit.sol#409)
        - totalFee = _liquidityFee.add(_teamFee).add(_marketingFee) (
            ↪ GreenRabbit.sol#410)
        - feeDenominator = _feeDenominator (GreenRabbit.sol#411)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #missing-events-arithmetic

Auth.transferOwnership(address).adr (GreenRabbit.sol#119) lacks a zero-
    ↪ check on :
                - owner = adr (GreenRabbit.sol#120)
GreenRabbit.setFeeReceiver(address,address)._marketingFeeReceiver (
    ↪ GreenRabbit.sol#414) lacks a zero-check on :
                - marketingFeeReceiver = _marketingFeeReceiver (
                    ↪ GreenRabbit.sol#415)
GreenRabbit.setFeeReceiver(address,address)._teamFeeReceiver (
    ↪ GreenRabbit.sol#414) lacks a zero-check on :
                - teamFeeReceiver = _teamFeeReceiver (GreenRabbit.sol#416)
```

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

Reentrancy in GreenRabbit._transferFrom(address,address,uint256) (
    ↪ GreenRabbit.sol#265-285):
        External calls:
        - swapBack() (GreenRabbit.sol#274)
                - router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ amountToSwap,0,path,address(this),block.t imestamp)
                    ↪  (GreenRabbit.sol#335-341)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (Gr
                    ↪ eenRabbit.sol#349)
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (Gre
                    ↪ enRabbit.sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,marketingFeeRec
                    ↪ eiver,block.timestamp) (GreenRabbit.sol#355-362)
        External calls sending eth:
        - swapBack() (GreenRabbit.sol#274)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (Gr
                    ↪ eenRabbit.sol#349)
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (Gre
                    ↪ enRabbit.sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,marketingFeeRec
                    ↪ eiver,block.timestamp) (GreenRabbit.sol#355-362)
        State variables written after the call(s):
        - launch() (GreenRabbit.sol#276)
                - launchedAt = block.number (GreenRabbit.sol#385)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-2

Reentrancy in GreenRabbit._transferFrom(address,address,uint256) (
    ↪ GreenRabbit.sol#265-285):
        External calls:
        - swapBack() (GreenRabbit.sol#274)
                - router.
                    ↪ swapExactTokensForETHSupportingFeeOnTransferTokens(
                    ↪ amountToSwap,0,path,address(this),block.t imestamp)
                    ↪  (GreenRabbit.sol#335-341)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (Gr
                    ↪ eenRabbit.sol#349)
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (Gre
                    ↪ enRabbit.sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,marketingFeeRec
                    ↪ eiver,block.timestamp) (GreenRabbit.sol#355-362)
        External calls sending eth:
        - swapBack() (GreenRabbit.sol#274)
                - (MarketingSuccess) = address(marketingFeeReceiver).call{
                    ↪ gas: 30000,value: amountBNBMarketing}() (Gr
                    ↪ eenRabbit.sol#349)
                - (developmentSuccess) = address(teamFeeReceiver).call{gas
                    ↪ : 30000,value: amountBNBdevelopment}() (Gre
                    ↪ enRabbit.sol#351)
                - router.addLiquidityETH{value: amountBNBLiquidity}(
                    ↪ address(this),amountToLiquify,0,0,marketingFeeRec
                    ↪ eiver,block.timestamp) (GreenRabbit.sol#355-362)
        Event emitted after the call(s):
        - Transfer(sender,address(this),feeAmount) (GreenRabbit.sol#312)
```

```
                - amountReceived = takeFee(sender,recipient,amount) (
                    ↪ GreenRabbit.sol#280)
        - Transfer(sender,recipient,amountReceived) (GreenRabbit.sol#283)
Reentrancy in GreenRabbit.swapBack() (GreenRabbit.sol#324-365):
        External calls:
        - router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            ↪ amountToSwap,0,path,address(this),block.timestamp ) (
            ↪ GreenRabbit.sol#335-341)
        - (MarketingSuccess) = address(marketingFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBMarketing}() (GreenRabbi t.sol#349)
        - (developmentSuccess) = address(teamFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBdevelopment}() (GreenRabbit .sol
            ↪ #351)
        - router.addLiquidityETH{value: amountBNBLiquidity}(address(this)
            ↪ ,amountToLiquify,0,0,marketingFeeReceiver,bl ock.timestamp
            ↪ ) (GreenRabbit.sol#355-362)
        External calls sending eth:
        - (MarketingSuccess) = address(marketingFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBMarketing}() (GreenRabbi t.sol#349)
        - (developmentSuccess) = address(teamFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBdevelopment}() (GreenRabbit .sol
            ↪ #351)
        - router.addLiquidityETH{value: amountBNBLiquidity}(address(this)
            ↪ ,amountToLiquify,0,0,marketingFeeReceiver,bl ock.timestamp
            ↪ ) (GreenRabbit.sol#355-362)
        Event emitted after the call(s):
        - AutoLiquify(amountBNBLiquidity,amountToLiquify) (GreenRabbit.
            ↪ sol#363)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3

GreenRabbit.buyTokens(uint256,address) (GreenRabbit.sol#367-378) is
    ↪ never used and should be removed
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

GreenRabbit._maxTxAmount (GreenRabbit.sol#192) is set pre-construction
    ↪ with a non-constant function or state variable :
        - (_totalSupply * 2) / 100
GreenRabbit._maxWalletSize (GreenRabbit.sol#193) is set pre-construction
    ↪  with a non-constant function or state variab le:
        - (_totalSupply * 2) / 100
GreenRabbit.swapThreshold (GreenRabbit.sol#216) is set pre-construction
    ↪ with a non-constant function or state variabl e:
        - _totalSupply / 1000 * 3
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #function-initializing-state

solc-0.8.16 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Low level call in GreenRabbit.swapBack() (GreenRabbit.sol#324-365):
        - (MarketingSuccess) = address(marketingFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBMarketing}() (GreenRabbi t.sol#349)
        - (developmentSuccess) = address(teamFeeReceiver).call{gas:
            ↪ 30000,value: amountBNBdevelopment}() (GreenRabbit .sol
            ↪ #351)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #low-level-calls

Function IDEXRouter.WETH() (GreenRabbit.sol#134) is not in mixedCase
Parameter GreenRabbit.setFees(uint256,uint256,uint256,uint256).
    ↪ _liquidityFee (GreenRabbit.sol#406) is not in mixedCas e
Parameter GreenRabbit.setFees(uint256,uint256,uint256,uint256)._teamFee
    ↪ (GreenRabbit.sol#406) is not in mixedCase
```

```
Parameter GreenRabbit.setFees(uint256,uint256,uint256,uint256).
   ↪ _marketingFee (GreenRabbit.sol#406) is not in mixedCas e
Parameter GreenRabbit.setFees(uint256,uint256,uint256,uint256).
   ↪ _feeDenominator (GreenRabbit.sol#406) is not in mixedC ase
Parameter GreenRabbit.setFeeReceiver(address,address).
   ↪ _marketingFeeReceiver (GreenRabbit.sol#414) is not in mixedCase
Parameter GreenRabbit.setFeeReceiver(address,address)._teamFeeReceiver (
   ↪ GreenRabbit.sol#414) is not in mixedCase
Parameter GreenRabbit.setSwapBackSettings(bool,uint256)._enabled (
   ↪ GreenRabbit.sol#419) is not in mixedCase
Parameter GreenRabbit.setSwapBackSettings(bool,uint256)._amount (
   ↪ GreenRabbit.sol#419) is not in mixedCase
Parameter GreenRabbit.transferForeignToken(address)._token (GreenRabbit.
   ↪ sol#429) is not in mixedCase
Variable GreenRabbit.WALV (GreenRabbit.sol#183) is not in mixedCase
Variable GreenRabbit.DEAD (GreenRabbit.sol#184) is not in mixedCase
Variable GreenRabbit.ZERO (GreenRabbit.sol#185) is not in mixedCase
Constant GreenRabbit._name (GreenRabbit.sol#187) is not in
   ↪ UPPER_CASE_WITH_UNDERSCORES
Constant GreenRabbit._symbol (GreenRabbit.sol#188) is not in
   ↪ UPPER_CASE_WITH_UNDERSCORES
Constant GreenRabbit._decimals (GreenRabbit.sol#189) is not in
   ↪ UPPER_CASE_WITH_UNDERSCORES
Variable GreenRabbit._totalSupply (GreenRabbit.sol#191) is not in
   ↪ mixedCase
Variable GreenRabbit._maxTxAmount (GreenRabbit.sol#192) is not in
   ↪ mixedCase
Variable GreenRabbit._maxWalletSize (GreenRabbit.sol#193) is not in
   ↪ mixedCase
Variable GreenRabbit._balances (GreenRabbit.sol#195) is not in mixedCase
Variable GreenRabbit._allowances (GreenRabbit.sol#196) is not in
   ↪ mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #conformance-to-solidity-naming-conventions
```

```
Variable IDEXRouter.addLiquidity(address,address,uint256,uint256,uint256
    ↪ ,uint256,address,uint256).amountADesired (GreenRabbit.sol#139) is
    ↪  too similar to IDEXRouter.addLiquidity(address,address,uint256,
    ↪ uint256,uint256,uint256,address,uint256).amountBDesired (
    ↪ GreenRabbit.sol#140)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #variable-names-are-too-similar

GreenRabbit.slitherConstructorVariables() (GreenRabbit.sol#180-449) uses
    ↪  literals with too many digits:
        - DEAD = 0x000000000000000000000000000000000000dEaD (GreenRabbit.
          ↪ sol#184)
GreenRabbit.slitherConstructorVariables() (GreenRabbit.sol#180-449) uses
    ↪  literals with too many digits:
        - ZERO = 0x0000000000000000000000000000000000000000 (GreenRabbit.
          ↪ sol#185)
GreenRabbit.slitherConstructorVariables() (GreenRabbit.sol#180-449) uses
    ↪  literals with too many digits:
        - _totalSupply = 4720000000 * (10 ** _decimals) (GreenRabbit.sol
          ↪ #191)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #too-many-digits

GreenRabbit.DEAD (GreenRabbit.sol#184) should be constant
GreenRabbit.WALV (GreenRabbit.sol#183) should be constant
GreenRabbit.ZERO (GreenRabbit.sol#185) should be constant
GreenRabbit._totalSupply (GreenRabbit.sol#191) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #state-variables-that-could-be-declared-constant

authorize(address) should be declared external:
        - Auth.authorize(address) (GreenRabbit.sol#91-93)
unauthorize(address) should be declared external:
```

```
        - Auth.unauthorize(address) (GreenRabbit.sol#98-100)
transferOwnership(address) should be declared external:
        - Auth.transferOwnership(address) (GreenRabbit.sol#119-123)
transferForeignToken(address) should be declared external:
        - GreenRabbit.transferForeignToken(address) (GreenRabbit.sol
            ↪ #429-433)
isOverLiquified(uint256,uint256) should be declared external:
        - GreenRabbit.isOverLiquified(uint256,uint256) (GreenRabbit.sol
            ↪ #443-445)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
GreenRabbit.sol analyzed (6 contracts with 78 detectors), 52 result(s)
    ↪ found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

We examined the design and implementation of Green Rabbit in this audit and found several issues of various severities. We advise greenrabbit team to implement the recommendations contained in all 7 of our findings to further enhance the code's security. It is of utmost priority to start by addressing the most severe exploit discovered by the auditors then followed by the remaining exploits, and finally we will be conducting a re-audit following the implementation of the remediation plan contained in this report.

We would much appreciate any constructive feedback or suggestions regarding our methodology, audit findings, or potential scope gaps in this report.

BLOCKHAT
SECURITY

For a Contract Audit, contact us at contact@blockhat.io