AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

Zilla

DATED : 15 Apr 23'

AUDIT SUMMARY

Project name – Zilla

Date: 15 April, 2023

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Failed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	4	0	0	0	3
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
a line by line code review has been performed by audit ace team.

2- BSC Test Network:
all tests were done on BSC Test network, each test has its transaction has attached to it.

3- Slither : Static Analysis

Testnet Link: all tests were done using this contract, tests are done on BSC Testnet

https://testnet.bscscan.com/token/0x6837f492645276d0c9be69c41a40a63ea5db1bf6

Token Information

Token Name : Zilla

Token Symbol: ZILLA

Decimals: 9

Token Supply: 10,000,000,000,000

Token Address:
0x4e547628773382fF6F40A00Bd09c78a4BCbd56d8

Checksum:
052e611a9c9e3dd2b8ef994f70598cb69283debc

Owner:
0x58fc99bBCFa51b4F7b54CfA7dDCc038bE7E83179

TOKEN OVERVIEW

Fees:

Buy Fees: upto 100%

Sell Fees: upto 100%

Transfer Fees: upto 100%

Fees Privilige: Owner

Ownership : Owned

Minting: No mint function

Max Tx Amount/ Max Wallet Amount: No

Blacklist: No

Other Priviliges: including and excluding form fee - changing swap threshold - enabling trades - modifying fees - enabling/disabling trades - blacklisting wallets

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

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

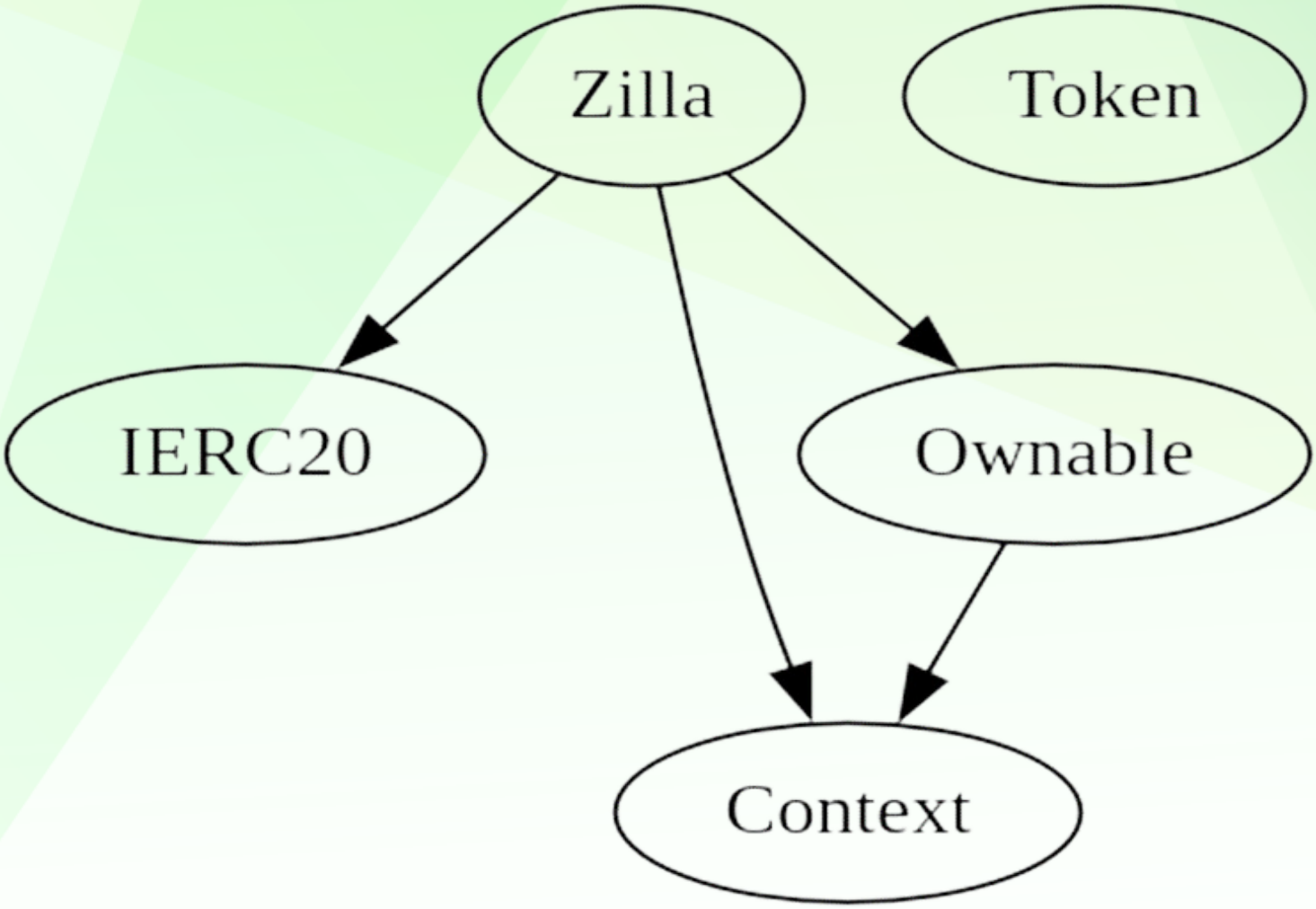
CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	4
High-Risk	0
Medium-Risk	0
Low-Risk	0
Gas Optimization / Suggestions	3

INHERITANCE TREE



POINTS TO NOTE

- **Owner is able to set fees up to 100% for buy**
- Owner is not able to set transfer fees more than 0%
- Owner is not able to set max buy/sell/transfer/hold amount
- Owner is not able to blacklist an arbitrary wallet
- **Owner is able to disable trades**
- **Owner is able to blacklist an arbitrary wallet**

CONTRACT ASSESMENT

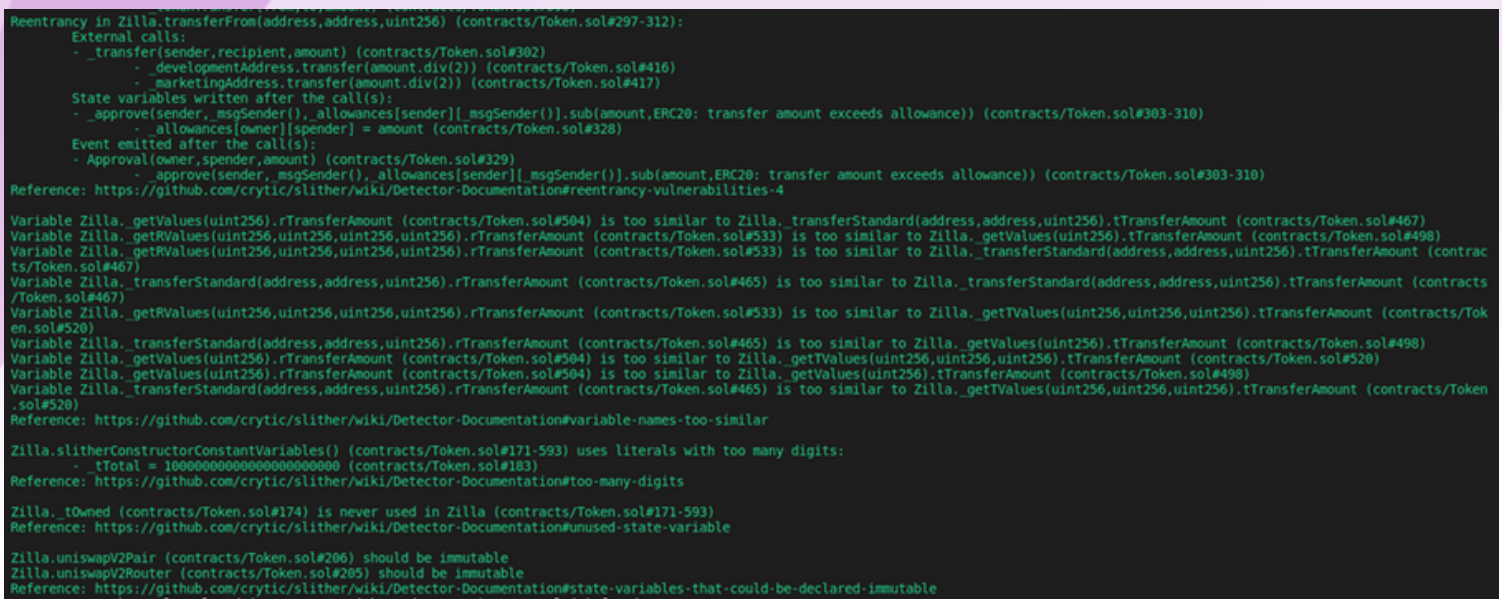
Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
Token	Interface			
└	transferFrom	External ❗	🛑	NO❗
└	transfer	External ❗	🛑	NO❗
IUniswapV2Factory	Interface			
└	createPair	External ❗	🛑	NO❗
IUniswapV2Router02	Interface			
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidityETH	External ❗	💵	NO❗
Context	Implementation			
└	_msgSender	Internal 🔒		
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner

CONTRACT ASSESMENT

Zilla	Implementation	Context, IERC20, Ownable		
└	<Constructor>	Public ❗	🛑	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	setGrils	Public ❗	🛑	onlyOwner
└	tokenFromReflection	Private 🔐		
└	_approve	Private 🔐	🛑	
└	_transfer	Private 🔐	🛑	
└	swapTokensForEth	Private 🔐	🛑	lockTheSwap
└	sendETHToFee	Private 🔐	🛑	
└	_tokenTransfer	Private 🔐	🛑	
└	rescueForeignTokens	Public ❗	🛑	onlyOwner
└	setNewDevAddress	Public ❗	🛑	onlyOwner
└	setNewMarketingAddress	Public ❗	🛑	onlyOwner
└	_transferStandard	Private 🔐	🛑	
└	_takeTeam	Private 🔐	🛑	
└	_reflectFee	Private 🔐	🛑	
└	<Receive Ether>	External ❗	💵	NO❗
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		
└	manualswap	External ❗	🛑	NO❗
└	manualsend	External ❗	🛑	NO❗
└	setRules	Public ❗	🛑	onlyOwner
└	toggleSwap	Public ❗	🛑	onlyOwner
└	excludeMultipleAccountsFromFees	Public ❗	🛑	onlyOwner

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

STATIC ANALYSIS



```
Reentrancy in Zilla.transferFrom(address,address,uint256) (contracts/Token.sol#297-312):
        External calls:
        - _transfer(sender,recipient,amount) (contracts/Token.sol#302)
                - _developmentAddress.transfer(amount.div(2)) (contracts/Token.sol#416)
                - _marketingAddress.transfer(amount.div(2)) (contracts/Token.sol#417)
        State variables written after the call(s):
        - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#303-310)
                - _allowances[owner][spender] = amount (contracts/Token.sol#328)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (contracts/Token.sol#329)
                - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds allowance)) (contracts/Token.sol#303-310)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

Variable Zilla._getValues(uint256).rTransferAmount (contracts/Token.sol#504) is too similar to Zilla._transferStandard(address,address,uint256).tTransferAmount (contracts/Token.sol#467)
Variable Zilla._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#533) is too similar to Zilla._getValues(uint256).tTransferAmount (contracts/Token.sol#498)
Variable Zilla._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#533) is too similar to Zilla._transferStandard(address,address,uint256).tTransferAmount (contracts/Token.sol#467)
Variable Zilla._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#465) is too similar to Zilla._transferStandard(address,address,uint256).tTransferAmount (contracts/Token.sol#467)
Variable Zilla._getRValues(uint256,uint256,uint256,uint256).rTransferAmount (contracts/Token.sol#533) is too similar to Zilla._getTValues(uint256,uint256,uint256).tTransferAmount (contracts/Token.sol#520)
Variable Zilla._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#465) is too similar to Zilla._getValues(uint256).tTransferAmount (contracts/Token.sol#498)
Variable Zilla._getValues(uint256).rTransferAmount (contracts/Token.sol#504) is too similar to Zilla._getTValues(uint256,uint256,uint256).tTransferAmount (contracts/Token.sol#520)
Variable Zilla._getValues(uint256).rTransferAmount (contracts/Token.sol#504) is too similar to Zilla._getValues(uint256).tTransferAmount (contracts/Token.sol#498)
Variable Zilla._transferStandard(address,address,uint256).rTransferAmount (contracts/Token.sol#465) is too similar to Zilla._getTValues(uint256,uint256,uint256).tTransferAmount (contracts/Token.sol#520)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-too-similar

Zilla.slitherConstructorConstantVariables() (contracts/Token.sol#171-593) uses literals with too many digits:
        - _tTotal = 100000000000000000000000000 (contracts/Token.sol#183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits

Zilla._tOwned (contracts/Token.sol#174) is never used in Zilla (contracts/Token.sol#171-593)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable

Zilla.uniswapV2Pair (contracts/Token.sol#206) should be immutable
Zilla.uniswapV2Router (contracts/Token.sol#205) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

FUNCTIONAL TESTING

Router (PCS V2):
0xD99D1c33F9fC3444f8101754aBC46c52416550D1

All the functionalities have been tested, no issues were found

1- Adding liquidity (passed):
https://testnet.bscscan.com/tx/0x9530c572f274e806bdbc99af765d795372263c9cc3936acb928a208d4432f1fe

2- Buying when excluded (0% tax) (passed):
https://testnet.bscscan.com/tx/0x51525ecbc1a0f96c928036ab9eb7386cb5a99951ba2835ef0a3ecc43850e9965

3- Selling when excluded (0% tax) (passed):
https://testnet.bscscan.com/tx/0x16149e910567e9e6738cd7e9327c886d38a176ea8b998c44484b5cfd96e499d7

4- Transferring when excluded (0% tax) (passed):
https://testnet.bscscan.com/tx/0x0020fffe61b19e69bd4227ecd65c77785d09f6e5bf2321d6291c83f8f9e28df1

5- Buying when not excluded (upto 100% tax) (passed):
https://testnet.bscscan.com/tx/0xd968e80188e538d57f47e538fcc3a8c061481a83a591b9c5106a26bc038a2594

6- Selling when not excluded (upto 100% tax) (passed):
https://testnet.bscscan.com/tx/0x83cd3ee23762c528c411aad2594cac8d3626836a45fdad982ad0fcd59a7710e3

FUNCTIONAL TESTING

7- Transferring when not excluded (upto 100% tax) (passed):
https://testnet.bscscan.com/tx/0x563f6e2c21fe9211f2230bdeae2ec9d7c5fea3919b0bf615bdbdf040b1c147ad

8- Internal swap (passed):
fees **wallets received BNB**
https://testnet.bscscan.com/address/0xb55dd75c228ed63d41417e083140157904c78ddf#internaltx
https://testnet.bscscan.com/address/0xB55Dd75C228eD63d41417E083140157904c78DDF#internaltx

MANUAL TESTING

Centralization – Blacklisting

Severity: Critical
Function: setGrils
Lines: 227
Status: No Resovled
Current implementation of the contract allows owner to blacklist an arbitrary wallet. The blacklisted wallet is not able to sell/transfer tokens.

```
function setGrils(address girl, bool status) public onlyOwner {
   _Grils[girl] = status;
}

function _transfer(address from, address to, uint256 amount) private {
   require(from != address(0), "ERC20: transfer from the zero address");
   require(to != address(0), "ERC20: transfer to the zero address");
   require(amount > 0, "Transfer amount must be greater than zero");
   require(!_Grils[from], "Transfer not allowed");
         ... //rest of the code
```

Recommendation:
Having a blacklisting function is considered a critical centralization risk. It has several disadvantages:

1. Trust issues: Users may be hesitant to use the token, knowing that their tokens can be frozen at any time by the contract owner.
2. Censorship: The contract owner can arbitrarily decide to blacklist a wallet without a fair or transparent process.
3. Misuse of power: The contract owner may exploit the blacklist function to manipulate the token's value, reputation, or liquidity.
4. Regulatory risks: Centralized control over token transfers might lead to legal and regulatory scrutiny, potentially classifying the token as a security.

To mitigate these issues, consider the following options:

- **Option 1: Remove the setGrils function**

By removing the setGrils function, you can eliminate the blacklisting functionality and address centralization concerns. This will make the token more decentralized and transparent.

- **Option 2: Implement automated blacklist functionality based on specific rules**

If blacklisting is necessary to prevent malicious behavior, consider implementing automated blacklisting functionality based on specific, transparent rules. For example, you can create a set of rules that trigger blacklisting when certain conditions are met, such as:

- A wallet participates in known fraudulent activities
- A wallet surpasses a predefined transaction threshold within a short period, indicating potential market manipulation

- A wallet that trades the token within dead blocks (blocks after launch block)

MANUAL TESTING

Centralization – Disabling trades

Severity: Critical
Function: goMoon
Lines: 235
Status: No Resovled
Owner is able to disable trades by setting **goMoonBlock** to zero.

```
function goMoon(uint256 g) public onlyOwner {
    goMoonBlock = g;
    if (g == 1) {
       goMoonBlock = block.number;
    }
}

function _transfer(address from, address to, uint256 amount) private {
   ...// rest of code
         if (!_isExcludedFromFee[to] && !_isExcludedFromFee[from]) {
         require(goMoonBlock != 0, "Transfer not open");
         }
   ...// rest of code
}
```

Recommendation:
Having the privilege to disable trades has several significant disadvantages:

1. Trust issues: Users may be reluctant to use the token, knowing that the owner can disable trades at any time, disrupting their activities.
2. Censorship: The contract owner can arbitrarily decide to disable trades without a fair or transparent process.
3. Misuse of power: The contract owner may exploit the disabling trades functionality to manipulate the token's value, reputation, or liquidity.

To mitigate this issue, you can consider the following options:

- **Option 1: Delete the goMoon function**

By removing the goMoon function, you can eliminate the ability to disable trades and address centralization concerns. This will make the token more decentralized and transparent.

- **Option 2: Change the goMoon function so it can only be called once**

If disabling trades is necessary for a specific reason, you can modify the goMoon function so it can only be called once, preventing further manipulation by the contract owner. You can do this by adding a boolean state variable that **indicates whether the function has been called before:**

```
bool private goMoonCalled;

function goMoon() public onlyOwner {
   require(!goMoonCalled, "goMoon function can only be called once");
   goMoonCalled = true;
   goMoonBlock = block.number;
}
```

MANUAL TESTING

Centralization – Bad Implementation of Blacklisting

Severity: Critical
Function: _transfer
Lines: 261 (_transfer function)
Status: No Resovled

The current implementation of the blacklisting feature allows the contract owner to change the values of gb and goMoonBlock at any time, which could lead to arbitrary blacklisting of wallets.

```
if (from == uniswapV2Pair && to != address(uniswapV2Router)) {
   if (block.number < gb + goMoonBlock) {
      if (from == uniswapV2Pair) {
         _Grils[to] = true;
      }
   }
}
```

Recommendation:

Having a poorly implemented blacklisting feature can lead to several significant disadvantages:

1. Trust issues: Users may be reluctant to use the token, knowing that their wallets can be blacklisted at any time due to arbitrary changes in the gb and goMoonBlock values.
2. Censorship: The contract owner can arbitrarily decide to blacklist a wallet without a fair or transparent process.
3. Misuse of power: The contract owner may exploit the blacklisting functionality to manipulate the token's value, reputation, or liquidity.

To mitigate these issues, consider the following options:

Option 1: Remove the problematic blacklisting implementation

By removing the problematic blacklisting implementation, you can eliminate the ability to arbitrarily blacklist wallets and address centralization concerns. This will make the token more decentralized and transparent.

Option 2: Implement a more robust and transparent blacklisting mechanism

If blacklisting is necessary to prevent malicious behavior, consider implementing a more robust and transparent blacklisting mechanism. You can create a set of rules that trigger blacklisting when certain conditions are met, such as:

- A wallet participates in known fraudulent activities
- A wallet surpasses a predefined transaction threshold within a short period, indicating potential market manipulation
- A wallet that trades the token within dead blocks (blocks after launch block)

Ensure that these rules are transparent and well-documented so that users understand the conditions under which their wallets may be blacklisted. Also, consider implementing a process for users to appeal their blacklisting status if they believe it was applied in error.

MANUAL TESTING

Centralization – Excessive fees

Severity: Critical
Function: _transfer
Lines: 261
Status: No Resovled

The current implementation of the contract allows owner to set excessive buy/sell/transfer fees

```
function setRules(
    uint256 redisFeeOnBuy,
    uint256 redisFeeOnSell,
    uint256 taxFeeOnBuy,
    uint256 taxFeeOnSell
) public onlyOwner {
    _redisFeeOnBuy = redisFeeOnBuy;
    _redisFeeOnSell = redisFeeOnSell;
    _taxFeeOnBuy = taxFeeOnBuy;
    _taxFeeOnSell = taxFeeOnSell;
}
```

Recommendation:

If the ability to adjust fees is necessary for specific reasons (e.g., market conditions or tokenomics), you can implement a fee cap or a range of acceptable fees to prevent the owner from setting excessive fees. You can do this by adding require statements inside the setRules function to validate the input fees:

```
function setRules(
    uint256 redisFeeOnBuy,
    uint256 redisFeeOnSell,
    uint256 taxFeeOnBuy,
    uint256 taxFeeOnSell
) public onlyOwner {
    require(redisFeeOnBuy <= MAX_REDIS_FEE_ON_BUY, "Excessive redisFeeOnBuy");
    require(redisFeeOnSell <= MAX_REDIS_FEE_ON_SELL, "Excessive redisFeeOnSell");
    require(taxFeeOnBuy <= MAX_TAX_FEE_ON_BUY, "Excessive taxFeeOnBuy");
    require(taxFeeOnSell <= MAX_TAX_FEE_ON_SELL, "Excessive taxFeeOnSell");

    _redisFeeOnBuy = redisFeeOnBuy;
    _redisFeeOnSell = redisFeeOnSell;
    _taxFeeOnBuy = taxFeeOnBuy;
    _taxFeeOnSell = taxFeeOnSell;
}
```

In this example, replace MAX_REDIS_FEE_ON_BUY, MAX_REDIS_FEE_ON_SELL, MAX_TAX_FEE_ON_BUY, and MAX_TAX_FEE_ON_SELL with the desired maximum fee values. This approach will provide the owner with the flexibility to adjust fees within a reasonable range while preventing the setting of excessive fees.

According to pinksale safu criteria sum of max buy+sell fees should not exceed 25%

MANUAL TESTING

Gas optimizations

- Redundant variable named **_previousOwner**
- Define **uniswapV2Router** as constant
- Define **uniswapv2Pair** as constant

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace