



Cyberscope

Audit Report

LadyBot

April 2024

Network BASE

Address 0x249247ddf61afC83f2Fd0968d4E7b0424598eeAB

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Acknowledged
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	US	Untrusted Source	Acknowledged
●	EPC	Existing Pair Creation	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis	1
Diagnostics	2
Table of Contents	3
Review	5
Audit Updates	5
Source Files	5
Findings Breakdown	6
ST - Stops Transactions	7
Description	7
Recommendation	7
US - Untrusted Source	8
Description	8
Recommendation	8
Team Update	8
EPC - Existing Pair Creation	9
Description	9
Recommendation	9
IDI - Immutable Declaration Improvement	10
Description	10
Recommendation	10
PLPI - Potential Liquidity Provision Inadequacy	11
Description	11
Recommendation	12
PMRM - Potential Mocked Router Manipulation	14
Description	14
Recommendation	15
PVC - Price Volatility Concern	17
Description	17
Recommendation	17
RSD - Redundant Swap Duplication	19
Description	19
Recommendation	19
L04 - Conformance to Solidity Naming Conventions	20
Description	20
Recommendation	21
L09 - Dead Code Elimination	22
Description	22
Recommendation	23
L14 - Uninitialized Variables in Local Scope	24

Description	24
Recommendation	24
L16 - Validate Variable Setters	25
Description	25
Recommendation	25
L17 - Usage of Solidity Assembly	26
Description	26
Recommendation	26
L18 - Multiple Pragma Directives	28
Description	28
Recommendation	28
L19 - Stable Compiler Version	29
Description	29
Recommendation	29
Functions Analysis	30
Inheritance Graph	41
Flow Graph	42
Summary	43
Disclaimer	44
About Cyberscope	45

Review

Contract Name	DividendTokenWithAntibot
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://basescan.org/address/0x249247ddf61afc83f2fd0968d4e7b0424598eeab
Address	0x249247ddf61afc83f2fd0968d4e7b0424598eeab
Network	BASE
Symbol	\$Lady
Decimals	9
Total Supply	2,000,000
Badge Eligibility	Yes

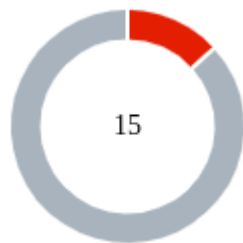
Audit Updates

Initial Audit	24 Apr 2024
---------------	-------------

Source Files

Filename	SHA256
DividendTokenWithAntibot.sol	4b57a4b1183c557abaed59369aac357a7b4de467b024873a11233f2c2f947404

Findings Breakdown



Critical	2
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	2	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Status	Acknowledged

Description

The contract owner has the authority to stop transactions, as described in detail in sections `US` and `PMRM`. As a result, the contract might operate as a honeypot.

Recommendation

The team is advised to consider revising the `US` and `PMRM` sections to limit the contract owner's authority over transaction control. Implementing additional checks can prevent any single point of failure and reduce the risk of misuse.

US - Untrusted Source

Criticality	Critical
Location	DividendTokenWithAntibot.sol#L1444
Status	Acknowledged

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
function updateDividendTracker(address newAddress) public  
onlyOwner {  
    ...  
    dividendTracker = newDividendTracker;  
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

Team Update

The team has acknowledged that this is not a security issue and states:

We're aware of the antibot and the dividends contract it's part of our tokenomics

EPC - Existing Pair Creation

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1428
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert.

```
mainPair =  
    IUniswapV2Factory(IUniswapV2Router02(mainRouter)).factory()  
        .createPair(address(this), baseTokenForMarket);
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before proceeding to create a new pair. This can be achieved by utilizing the `getPair` function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the `getPair` function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the `createPair` function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the `createPair` function will revert.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L952,955,956
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
gemAntiBot
_decimals
rewardToken
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1964,1987
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
IUniswapV2Router02 (mainRouter)
    .swapExactTokensForETHSupportingFeeOnTransferTokens (
        tokenAmount,
        0, // accept any amount of BaseToken
        path,
        address(this),
        block.timestamp
    );
} else {
    uniswapV2Caller
        .swapExactTokensForTokensSupportingFeeOnTransferTokens (
            mainRouter,
            tokenAmount,
            0, // accept any amount of BaseToken
            path,
            block.timestamp
        );
    ...
    path[0] = address(this);
    path[1] = baseTokenForMarket;
    path[2] = rewardToken;
    IUniswapV2Router02 (mainRouter)
        .swapExactTokensForTokensSupportingFeeOnTransferTokens (
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
} else {
    address[] memory path = new address[] (2);
    path[0] = address(this);
    path[1] = rewardToken;
    uniswapV2Caller
        .swapExactTokensForTokensSupportingFeeOnTransferTokens (
            mainRouter,
            tokenAmount,
            0, // accept any amount of BaseToken
            path,
            block.timestamp
        );
}
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1413
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
function updateMainPair(
    address _mainRouter,
    address _baseTokenForMarket
) external onlyOwner {
    baseTokenForMarket = _baseTokenForMarket;
    if (mainRouter != _mainRouter) {
        _approve(address(this), _mainRouter, MAX);
        if (

!DividendTokenDividendTrackerInterface(dividendTracker)
        .isExcludedFromDividends(_mainRouter)

        )

DividendTokenDividendTrackerInterface(dividendTracker)
        .excludeFromDividends(_mainRouter);
        mainRouter = _mainRouter;
    }
    mainPair =
    IUniswapV2Factory(IUniswapV2Router02(mainRouter).factory())
        .createPair(address(this), baseTokenForMarket);
    if (baseTokenForMarket !=
    IUniswapV2Router02(mainRouter).WETH()) {
        IERC20(baseTokenForMarket).safeApprove(mainRouter,
MAX);
    }

    emit MainRouterUpdated(mainRouter, mainPair,
baseTokenForMarket);
    _setAutomatedMarketMakerPair(mainPair, true);
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1805
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
bool canSwap = contractTokenBalance >= swapTokensAtAmount;

if (
    canSwap &&
    !swapping &&
    !automatedMarketMakerPairs[from] &&
    from != owner() &&
    to != owner()
) {
    swapping = true;
    if (_marketingFeeTokens > 0)
        swapAndSendToFee(_marketingFeeTokens);
    if (_liquidityFeeTokens > 0)
        swapAndLiquify(_liquidityFeeTokens);
    uint256 sellTokens = balanceOf(address(this));
    if (sellTokens > 0) swapAndSendDividends(sellTokens);
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount

should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1815
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
if (_marketingFeeTokens > 0)
    swapAndSendToFee(_marketingFeeTokens);
if (_liquidityFeeTokens > 0)
    swapAndLiquify(_liquidityFeeTokens);

...
if (sellTokens > 0) swapAndSendDividends(sellTokens);
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L313,484,485,502,525,1050,1051,1104,1110,1136,1146,1147,1168,1169,1190,1191
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
address _mainRouter
address _baseTokenForMarket
uint256 _maxWallet
uint256 _maxTransactionAmount
address _tokenForMarketingFee
uint24 _sellLiquidityFee
uint24 _buyLiquidityFee
uint24 _sellMarketingFee
uint24 _buyMarketingFee
uint24 _sellRewardFee

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention>.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L16,22,32,45,48,56,59,84,234,321,334,338,345,353,377,401,413,430
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount)
internal {
    require(address(this).balance >= amount, "Address:
insufficient balance");

    (bool success, ) = recipient.call{value: amount}("");
    require(success, "Address: unable to send value,
recipient may have reverted");
}

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
    return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
}

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
    return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
}

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L1416,1417,1418,1461,1462,1463
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 _liquidityFee
uint256 _marketingFee
uint256 _rewardFee
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L952,1054
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
gemAntiBot = _gemAntiBot  
baseTokenForMarket = _baseTokenForMarket
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L100,391,403,419
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
    let returndata_size := mload(returndata)
    revert(add(32, returndata), returndata_size)
}

assembly {
    // Cleans the upper 96 bits of the `implementation`
    word, then packs the first 3 bytes
    // of the `implementation` address with the
    bytecode before the address.
    mstore(0x00, or(shr(0xe8, shl(0x60,
implementation)),
0x3d602d80600a3d3981f3363d3d373d3d3d363d73000000))
    // Packs the remaining 17 bytes of `implementation`
    with the bytecode after the address.
    mstore(0x20, or(shl(0x78, implementation),
0x5af43d82803e903d91602b57fd5bf3))
    instance := create(0, 0x09, 0x37)
}

...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L6,110,438,520,727
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity 0.8.19;  
pragma solidity >=0.5.0;  
pragma solidity >=0.6.2;  
pragma solidity ^0.8.0;  
pragma solidity ^0.8.1;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	DividendTokenWithAntibot.sol#L6,110
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.1;  
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-

	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, IERC20Meta data		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20Permit	Interface			
	permit	External	✓	-
	nonces	External		-

	DOMAIN_SEPARATOR	External		-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	forceApprove	Internal	✓	
	safePermit	Internal	✓	
	_callOptionalReturn	Private	✓	
	_callOptionalReturnBool	Private	✓	
Clones	Library			
	clone	Internal	✓	
	cloneDeterministic	Internal	✓	
	predictDeterministicAddress	Internal		
	predictDeterministicAddress	Internal		
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-

	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-

	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-

	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Caller	Interface			
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
IFee	Interface			

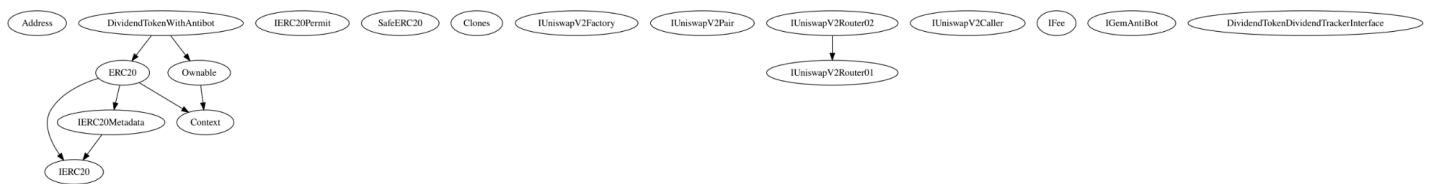
	payFee	External	Payable	-
IGemAntiBot	Interface			
	setTokenOwner	External	✓	-
	onPreTransferCheck	External	✓	-
DividendToken DividendTrackerInterface	Interface			
	initialize	External	✓	-
	excludeFromDividends	External	✓	-
	isExcludedFromDividends	External		-
	owner	External		-
	updateClaimWait	External	✓	-
	claimWait	External		-
	updateMinimumTokenBalanceForDividends	External	✓	-
	minimumTokenBalanceForDividends	External		-
	totalDividendsDistributed	External		-
	withdrawableDividendOf	External		-
	balanceOf	External		-
	getAccount	External		-
	getAccountAtIndex	External		-
	process	External	✓	-
	processAccount	External	✓	-
	getLastProcessedIndex	External		-

	getNumberOfTokenHolders	External		-
	setBalance	External	✓	-
	distributeCAKEDividends	External	✓	-
DividendToken WithAntibot	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
	enableAntibot	External	✓	onlyOwner
		External	Payable	-
	updateMainPair	External	✓	onlyOwner
	setSwapTokensAtAmount	External	✓	onlyOwner
	updateDividendTracker	Public	✓	onlyOwner
	updateMaxWallet	External	✓	onlyOwner
	updateMaxTransactionAmount	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	decimals	Public		-
	setMarketingWallet	External	✓	onlyOwner
	updateTokenForMarketingFee	External	✓	onlyOwner
	updateLiquidityFee	External	✓	onlyOwner
	updateMarketingFee	External	✓	onlyOwner
	updateRewardFee	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromMaxTransactionAmount	External	✓	onlyOwner

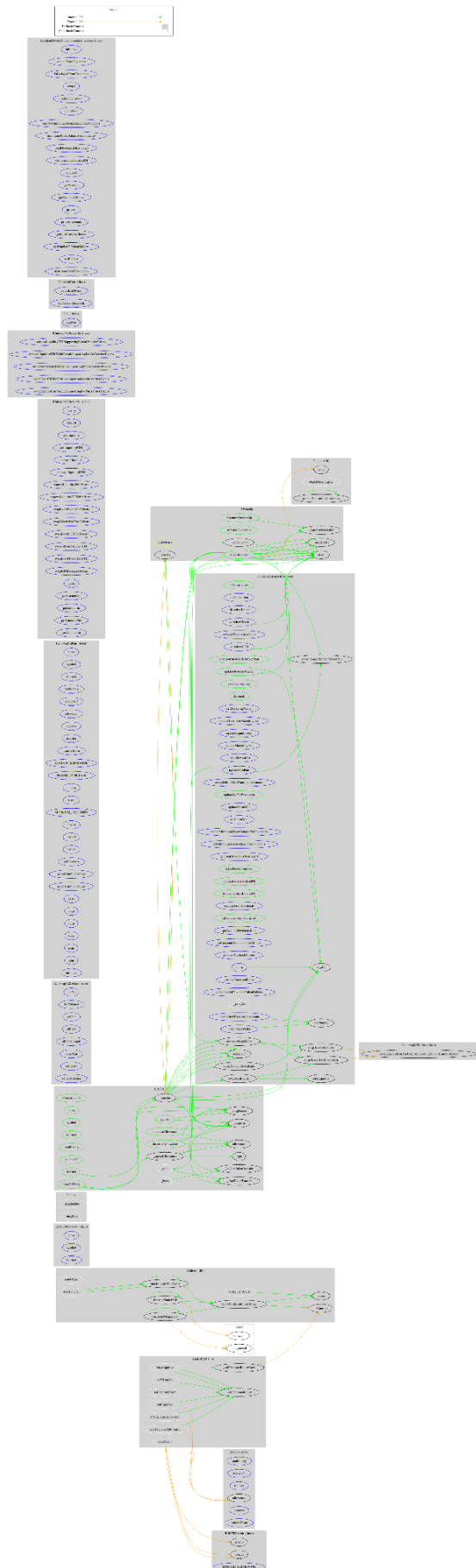
	updateGasForProcessing	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	getMinimumTokenBalanceForDividends	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	isExcludedFromDividends	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
	swapAndSendToFee	Private	✓	
	swapAndLiquify	Private	✓	
	swapTokensForBaseToken	Private	✓	
	swapTokensForCake	Private	✓	
	addLiquidity	Private	✓	

	swapAndSendDividends	Private	✓	
	withdrawETH	External	✓	onlyOwner
	withdrawToken	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

LadyBot contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 20% fees.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

<https://www.cyberscope.io>