



Cyberscope

# Audit Report

## **BirdDog**

September 2024

Network    ETH

Address    0xF6Ce4BE313EaD51511215F1874c898239A331E37

Audited by    © cyberscope

# Analysis

● Critical   ● Medium   ● Minor / Informative   ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	MU	Modifiers Usage	Unresolved
●	NWES	Nonconformity with ERC-20 Standard	Unresolved
●	PGA	Potential Griefing Attack	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSRS	Redundant SafeMath Require Statement	Unresolved
●	UAR	Unexcluded Address Restrictions	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

# Table of Contents

<b>Analysis</b>	<b>1</b>
<b>Diagnostics</b>	<b>2</b>
<b>Table of Contents</b>	<b>3</b>
<b>Risk Classification</b>	<b>4</b>
<b>Review</b>	<b>5</b>
Audit Updates	5
Source Files	5
<b>Findings Breakdown</b>	<b>6</b>
AOI - Arithmetic Operations Inconsistency	7
Description	7
Recommendation	7
IDI - Immutable Declaration Improvement	9
Description	9
Recommendation	9
MEM - Missing Error Messages	10
Description	10
Recommendation	10
MEE - Missing Events Emission	11
Description	11
Recommendation	11
MU - Modifiers Usage	12
Description	12
Recommendation	12
NWES - Nonconformity with ERC-20 Standard	13
Description	13
Recommendation	13
PGA - Potential Griefing Attack	14
Description	14
Recommendation	14
PLPI - Potential Liquidity Provision Inadequacy	15
Description	15
Recommendation	16
RRA - Redundant Repeated Approvals	17
Description	17
Recommendation	17
RSML - Redundant SafeMath Library	18
Description	18
Recommendation	18
RSRS - Redundant SafeMath Require Statement	19

Description	19
Recommendation	19
UAR - Unexcluded Address Restrictions	20
Description	20
Recommendation	20
L02 - State Variables could be Declared Constant	21
Description	21
Recommendation	21
L04 - Conformance to Solidity Naming Conventions	22
Description	22
Recommendation	23
<b>Functions Analysis</b>	<b>24</b>
<b>Inheritance Graph</b>	<b>27</b>
<b>Flow Graph</b>	<b>28</b>
<b>Summary</b>	<b>29</b>
<b>Disclaimer</b>	<b>30</b>
<b>About Cyberscope</b>	<b>31</b>

## Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation:** This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation:** This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical:** Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium:** Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor:** Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative:** Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

## Review

Contract Name	BIRDDOG
Compiler Version	v0.8.23+commit.f704f362
Optimization	200 runs
Explorer	<a href="https://etherscan.io/address/0xf6ce4be313ead51511215f1874c898239a331e37">https://etherscan.io/address/0xf6ce4be313ead51511215f1874c898239a331e37</a>
Address	0xf6ce4be313ead51511215f1874c898239a331e37
Network	ETH
Symbol	BIRDDOG
Decimals	9
Total Supply	420,690,000,000

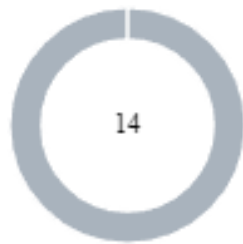
## Audit Updates

Initial Audit	24 Sep 2024
---------------	-------------

## Source Files

Filename	SHA256
BIRDDOG.sol	baec1006fd93fcf9771abd7c4286706f35484242e6eb5ac3e4fc67c4e76fc607

## Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	14

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	14	0	0	0



## AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	BIRDDOG.sol#L243,269,272
Status	Unresolved

### Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, \*, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
if (taxAmount > 0) {  
  
    _balances[address(this)] = _balances[address(this)].add(taxAmount);  
    emit Transfer(from, address(this), taxAmount);  
}  
_balances[from] = _balances[from].sub(amount);  
_balances[to] = _balances[to].add(amount.sub(taxAmount));  
  
require(balanceOf(to) + amount <= _maxWalletSize, "Exceeds the maxWalletSize.");
```

### Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design.

considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

## IDI - Immutable Declaration Improvement

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BIRDDOG.sol#L166
<b>Status</b>	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.

```
_taxWallet
```

### 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.

## MEM - Missing Error Messages

Criticality	Minor / Informative
Location	BIRDDOG.sol#L228,336,337,345
Status	Unresolved

### Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(!bots[from] && !bots[to])
require(_msgSender() == _taxWallet)
require(_newFee <= _finalBuyTax && _newFee <= _finalSellTax)
```

### Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

## MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BIRDDOG.sol#L311,317
Status	Unresolved

### Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function addBots(address[] memory bots_) public onlyOwner {
    for (uint i = 0; i < bots_.length; i++) {
        bots[bots_[i]] = true;
    }
}

function delBots(address[] memory notbot) public onlyOwner {
    for (uint i = 0; i < notbot.length; i++) {
        bots[notbot[i]] = false;
    }
}
```

### Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

## MU - Modifiers Usage

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BIRDDOG.sol#L340,349
<b>Status</b>	Unresolved

### Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_msgSender() == _taxWallet);
```

### Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

## NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	BIRDDOG.sol#L229
Status	Unresolved

### Description

The contract is not fully conforming to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However the contract implements, a conditional check that prohibits transfers of 0 values.

This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
function _transfer(address from,address to,int256 amount)
private {
    ...
    require(amount > 0, "Transfer amount must be greater than
    zero");
    ...
}
```

### Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. <https://eips.ethereum.org/EIPS/eip-20>.

## PGA - Potential Griefing Attack

Criticality	Minor / Informative
Location	BIRDDOG.sol#L257
Status	Unresolved

### Description

The `_transfer` function includes a `require` statement intended to impose a maximum of 3 sales in a single block. This opens up the potential for a griefing attack. During such an incident, a malicious actor could front-run normal sales with spam transactions to prevent legitimate transactions from selling tokens.

```
function _transfer(address from,address to,uint256 amount)
private {
    ...
    if (!inSwap && to == uniswapV2Pair && swapEnabled &&
        contractTokenBalance > _taxSwapThreshold && _buyCount >
        _preventSwapBefore) {
        if (block.number > lastSellBlock) {
            sellCount = 0;
        }
        require(sellCount < 3, "Only 3 sells per block!");
        swapTokensForEth(min(amount, min(contractTokenBalance,
            _maxTaxSwap)));
        ...
    }
}
```

### Recommendation

The team is advised to review the transfer mechanism in regards to sales to ensure all legitimate transactions are processed according to the intended behavior. A lower threshold on the transferred amount could be implemented before the sales counter is incremented.



## PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	BIRDDOG.sol#L282
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.

```
function swapTokensForEth(uint256 tokenAmount) private
lockTheSwap {
    address[] memory path = new address[] (2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    tokenAmount,
    0,
    path,
    address(this),
    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.

## RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	BIRDDOG.sol#L282
Status	Unresolved

### Description

The contract is designed to `approve` token transfers during the contract's operation by calling the `_approve` function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
function swapTokensForEth(uint256 tokenAmount) private  
lockTheSwap {  
    ...  
    _approve(address(this), address(uniswapV2Router),  
tokenAmount);  
    ...  
}
```

### Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

## RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	BIRDDOG.sol
Status	Unresolved

### Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

### Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on

<https://docs.soliditylang.org/en/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes>.

## RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	BIRDDOG.sol#L40
Status	Unresolved

### Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");
    return c;
}
```

### Recommendation

It is recommended to remove the `require` statement from the `add` function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

## UAR - Unexcluded Address Restrictions

<b>Criticality</b>	Minor / Informative
<b>Location</b>	BIRDDOG.sol#L241
<b>Status</b>	Unresolved

### Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract. In this case, it is not possible to exclude certain decentralized applications from the implementation of fees.

```
if (from == uniswapV2Pair && to != address(uniswapV2Router) &&  
    !_isExcludedFromFee[to] ) { ... }
```

### Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

## L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BIRDDOG.sol#L131,132,135,136,137,147,148
Status	Unresolved

### Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
nt256 private _initialBuyTax=23;
nt256 private _initialSellTax=23;
nt256 private _reduceBuyTaxAt=20;
nt256 private _reduceSellTaxAt=20;
nt256 private _preventSwapBefore=27;
nt256 public _taxSwapThreshold= 4200000000 * 10**_decimals;
nt256 public _maxTaxSwap= 4200000000 * 10**_decimals;
```

### Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

## L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	BIRDDOG.sol#L112,141,142,143,144,145,146,147,148,335
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.

```
nction WETH() external pure returns (address);

nt8 private constant _decimals = 9;
nt256 private constant _tTotal = 42069000000 * 10**_decimals;
ring private constant _name = unicode"Bird Dog";
ring private constant _symbol = unicode"BIRDDOG";
nt256 public _maxTxAmount = 8400000000 * 10**_decimals;
nt256 public _maxWalletSize = 8400000000 * 10**_decimals;
nt256 public _taxSwapThreshold= 4200000000 * 10**_decimals;
nt256 public _maxTaxSwap= 4200000000 * 10**_decimals;
nt256 _newFee)
```



## 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/stable/style-guide.html#naming-conventions>.

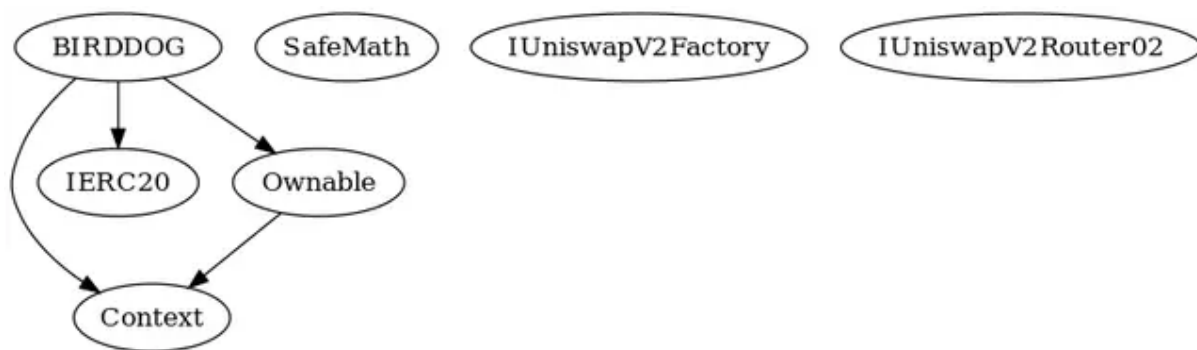
## Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>Context</b>	Implementation			
	_msgSender	Internal		
<b>IERC20</b>	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>SafeMath</b>	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
<b>Ownable</b>	Implementation	Context		

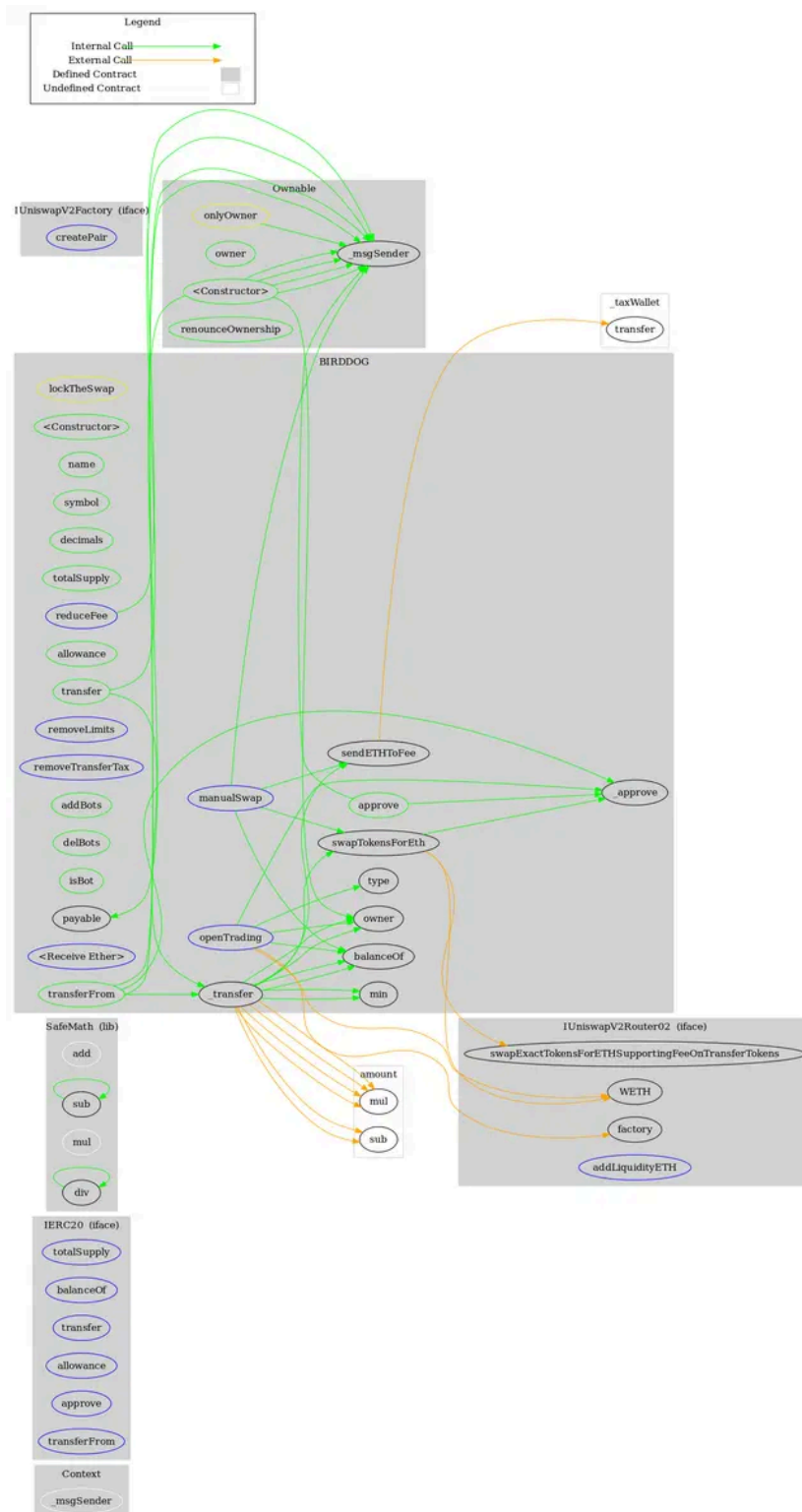
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
<b>IUniswapV2Factory</b>	Interface			
	createPair	External	✓	-
<b>IUniswapV2Router02</b>	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
<b>BIRDDOG</b>	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	min	Private		
	swapTokensForEth	Private	✓	lockTheSwap
	removeLimits	External	✓	onlyOwner
	removeTransferTax	External	✓	onlyOwner
	sendETHToFee	Private	✓	
	addBots	Public	✓	onlyOwner
	delBots	Public	✓	onlyOwner
	isBot	Public		-
	openTrading	External	✓	onlyOwner
	reduceFee	External	✓	-
		External	Payable	-
	manualSwap	External	✓	-

## Inheritance Graph



# Flow Graph



## Summary

BirdDog contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. BirdDog is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract ownership has been renounced.

## 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**

[cyberscope.io](https://cyberscope.io)