



Cyberscope

# Audit Report

## **Booskie**

July 2023

Network    ETH

Address    0xe1C076D22524Ed30829A45546C3A29f69o2712F4

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
●	MEE	Missing Events Emission	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	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>Review</b>	<b>5</b>
Audit Updates	5
Source Files	5
<b>Findings Breakdown</b>	<b>6</b>
MEE - Missing Events Emission	7
Description	7
Recommendation	7
RVD - Redundant Variable Declaration	8
Description	8
Recommendation	8
IDI - Immutable Declaration Improvement	9
Description	9
Recommendation	9
L02 - State Variables could be Declared Constant	10
Description	10
Recommendation	10
L04 - Conformance to Solidity Naming Conventions	11
Description	11
Recommendation	12
L07 - Missing Events Arithmetic	13
Description	13
Recommendation	13
L08 - Tautology or Contradiction	14
Description	14
Recommendation	14
L09 - Dead Code Elimination	15
Description	15
Recommendation	16
L16 - Validate Variable Setters	17
Description	17
Recommendation	17
L17 - Usage of Solidity Assembly	18
Description	18
Recommendation	18
<b>Functions Analysis</b>	<b>19</b>
<b>Inheritance Graph</b>	<b>23</b>

<b>Flow Graph</b>	<b>24</b>
<b>Summary</b>	<b>25</b>
<b>Disclaimer</b>	<b>26</b>
<b>About Cyberscope</b>	<b>27</b>

## Review

Contract Name	DxBurnToken
Compiler Version	v0.8.7+commit.e28d00a7
Optimization	200 runs
Explorer	<a href="https://etherscan.io/address/0xe1c076d22524ed30829a45546c3a29f69a2712f4">https://etherscan.io/address/0xe1c076d22524ed30829a45546c3a29f69a2712f4</a>
Address	0xe1c076d22524ed30829a45546c3a29f69a2712f4
Network	ETH
Symbol	BOOSKIE
Decimals	18
Total Supply	1,000,000,000,000

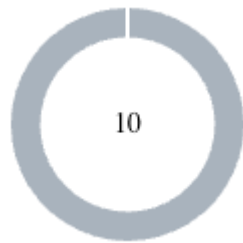
## Audit Updates

Initial Audit	15 Jul 2023
---------------	-------------

## Source Files

Filename	SHA256
DxBurnToken.sol	1c23fbc3265121cf68c9388ac66a50c54a9b5d00acad6828d07e22ab0452c90f

## Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

## MEE - Missing Events Emission

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L642,647,655
<b>Status</b>	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.

```
_isExcludedFromFee[account] = true  
_isExcludedFromFee[account] = false  
_devWalletAddress = _addr
```

### 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.



## RVD - Redundant Variable Declaration

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L454
<b>Status</b>	Unresolved

### Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
bool public mintedByDxsale = true
```

### Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

## IDI - Immutable Declaration Improvement

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L473,474,475,476,479,480,493
<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.

```
_name  
_symbol  
_decimals  
_creator  
_maxBurnFee  
_maxDevFee  
mintingFinishedPermanent
```

### 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.

## L02 - State Variables could be Declared Constant

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L454
<b>Status</b>	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.

```
bool public mintedByDxsale = true
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L460,461,462,463,464,465,650,658,678
<b>Status</b>	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.

```
address public _creator
uint8 public _burnFee
uint8 public _devFee
uint256 public _maxBurnFee
uint256 public _maxDevFee
address public _devWalletAddress
address _addr
address _newAddr
uint256 _value
```

## 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>.

## L07 - Missing Events Arithmetic

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L670,675
<b>Status</b>	Unresolved

### Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
_devFee = devFee  
_burnFee = burnFee
```

### Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

## L08 - Tautology or Contradiction

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L669,674
<b>Status</b>	Unresolved

### Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(devFee >= 0 && devFee <= _maxDevFee, "teamFee out of range")
require(burnFee >= 0 && burnFee <= _maxBurnFee, "teamFee out of range")
```

### Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

## L09 - Dead Code Elimination

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L246,270,295,305,324,334,351,361,376,386,401,425,437
<b>Status</b>	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 isContract(address account) internal view returns (bool) {
    // This method relies on extcodesize/address.code.length, which
    returns 0
    // for contracts in construction, since the code is only stored at
    the end
    // of the constructor execution.

    return account.code.length > 0;
}

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");
}

...
```

## 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.

## L16 - Validate Variable Setters

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L476,481
<b>Status</b>	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.

```
_creator = creator_  
_devWalletAddress = devWalletAddress_
```

### 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

<b>Criticality</b>	Minor / Informative
<b>Location</b>	DxBurnToken.sol#L442
<b>Status</b>	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)  
}
```

### 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.

## Functions Analysis

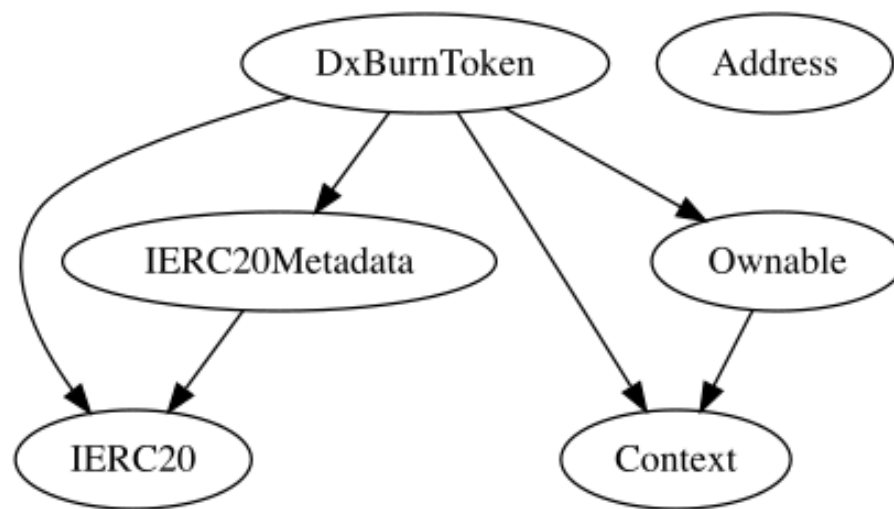
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
<b>IERC20</b>	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
<b>IERC20Metadata</b>	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
<b>Context</b>	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
<b>Ownable</b>	Implementation	Context		

		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
<b>Address</b>	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		
<b>DxBurnToken</b>	Implementation	Context, IERC20, IERC20Meta		

		data, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setDevWalletAddress	Internal	✓	
	replaceDevWalletAddress	Public	✓	onlyOwner
	setDevFeePercent	External	✓	onlyOwner
	setBurnFeePercent	External	✓	onlyOwner
	burn	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	

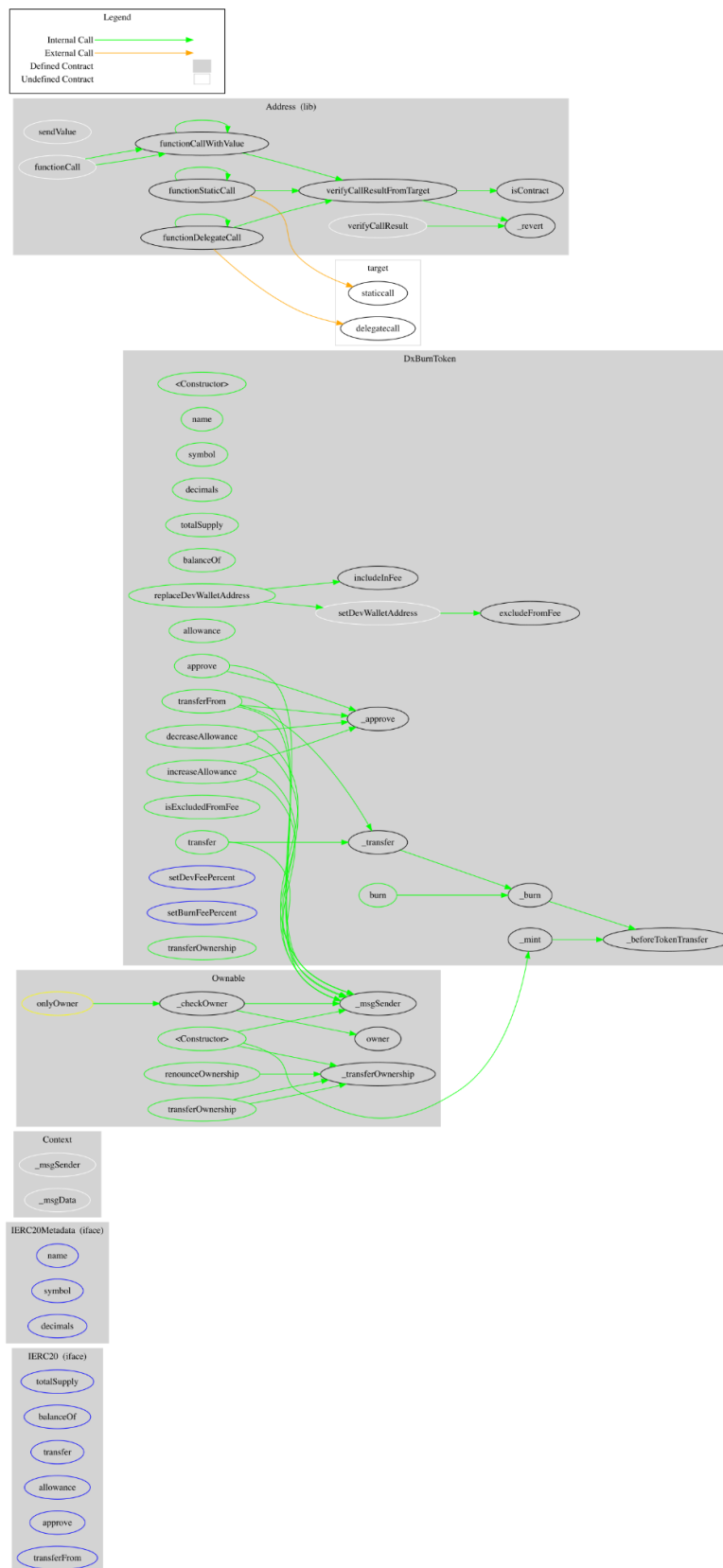
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	transferOwnership	Public	✓	onlyOwner

## Inheritance Graph





# Flow Graph



## Summary

Booskie contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Booskie is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 8% 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>