Cyberscope

# Audit Report
# Platy

February 2025

| | |
|---|---|
| Network | BSC |
| Address | 0xec74fb3f080fdacd95401e1b5b4275f06f50aff3 |
| 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 |
|---|---|---|---|
| ● | IDI | Immutable Declaration Improvement | Unresolved |
| ● | MVN | Misleading Variables Naming | Unresolved |
| ● | RF | Redundant Functionality | Unresolved |
| ● | ROF | Redundant Ownable Functionality | Unresolved |
| ● | RSV | Redundant State Variables | Unresolved |
| ● | UL | Unused Library | Unresolved |
| ● | L02 | State Variables could be Declared Constant | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L09 | Dead Code Elimination | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |
| ● | L17 | Usage of Solidity Assembly | Unresolved |
| ● | L18 | Multiple Pragma Directives | Unresolved |

# Table of Contents

**Analysis** **1**
**Diagnostics** **2**
**Table of Contents** **3**
**Risk Classification** **5**
**Review** **6**
Audit Updates 6
Source Files 6
**Findings Breakdown** **7**
IDI - Immutable Declaration Improvement 8
Description 8
Recommendation 8
MVN - Misleading Variables Naming 9
Description 9
Recommendation 9
RF - Redundant Functionality 10
Description 10
Recommendation 10
ROF - Redundant Ownable Functionality 11
Description 11
Recommendation 11
RSV - Redundant State Variables 12
Description 12
Recommendation 12
UL - Unused Library 13
Description 13
Recommendation 13
L02 - State Variables could be Declared Constant 14
Description 14
Recommendation 14
L04 - Conformance to Solidity Naming Conventions 15
Description 15
Recommendation 15
L09 - Dead Code Elimination 16
Description 16
Recommendation 17
L16 - Validate Variable Setters 18
Description 18
Recommendation 18
L17 - Usage of Solidity Assembly 19

Description 19
Recommendation 19
L18 - Multiple Pragma Directives 20
Description 20
Recommendation 20
**Functions Analysis 21**
**Inheritance Graph 22**
**Flow Graph 23**
**Summary 24**
**Disclaimer 25**
**About Cyberscope 26**

# 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** | DxStandardToken |
| **Compiler Version** | v0.8.7+commit.e28d00a7 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0xec74fb3f080fdacd95401e1b5b4275f06f50aff3 |
| **Address** | 0xec74fb3f080fdacd95401e1b5b4275f06f50aff3 |
| **Network** | BSC |
| **Symbol** | PLATY |
| **Decimals** | 18 |
| **Total Supply** | 500.000.000.000 |
| **Badge Eligibility** | Yes |

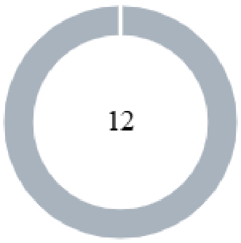
## Audit Updates

| | |
|---|---|
| **Initial Audit** | 23 Feb 2025 |

## Source Files

| **Filename** | SHA256 |
|---|---|
| **DxStandardToken.sol** | ab38df0127d3f27a90e6a7c34b48cc9f9e9b4af9ae1f00873975b2a277af0cae |

# Findings Breakdown



| | | |
|---|---|---|
| ● | Critical | 0 |
| ● | Medium | 0 |
| ● | Minor / Informative | 12 |

| | Severity | Unresolved | Acknowledged | Resolved | Other |
|---|---|---|---|---|---|
| ● | Critical | 0 | 0 | 0 | 0 |
| ● | Medium | 0 | 0 | 0 | 0 |
| ● | Minor / Informative | 12 | 0 | 0 | 0 |

# IDI - Immutable Declaration Improvement

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L475,476,479 |
| **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.

```
_decimals
_creator
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.

# MVN - Misleading Variables Naming

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L464 |
| **Status** | Unresolved |

## Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

The `_creator` variable indicates the creator of the contract however it is passed as a parameter in the `constructor` meaning that it can be a different address than the deployer of the contract.

```
address public _creator;
constructor (address creator_,**args**) {
    //...
    _creator = creator_;
    //...
}
```

## Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

## RF - Redundant Functionality

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L681,729 |
| **Status** | Unresolved |

### Description

The contract uses the `beforeTokenTransfer` in the `_burn` , `_mint` , `_transfer` function. However, beforeTokenTransfer is an empty function so it is redundant.

```
function _beforeTokenTransfer(address from, address to, uint256
amount) internal virtual { }
```

### Recommendation

It is recommended to remove empty functions to enhance code optimization and readability as well as reduce deployment costs.

# ROF - Redundant Ownable Functionality

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L453 |
| **Status** | Unresolved |

## Description

The contract inherits from the `Ownable` smart contract. `Ownable` allows the owner of the contract control over key functions of the contract that have the `onlyOwner` modifier. However, no function has this modifier and furthermore there is no need for it on any of the implemented functions. Therefore `Ownable` is redundant.

```
contract DxStandardToken is Context, IERC20,
IERC20Metadata,Ownable {/*...*/}
```

## Recommendation

It is recommended to remove the `Ownable` contract to enhance code optimization and readability as well as reduce deployment costs.

## RSV - Redundant State Variables

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L458,460,464 |
| **Status** | Unresolved |

### Description

The contract has multiple state variables that are set once and are not used in the rest of the functionality.

`mintedByDxsale` is set to true by default and is not used in any function. `mintingFinishedPermanent` is set as true in the `constructor` and is only used in the `_mint` function to determine if the minting of tokens should happen. However the `_mint` function is internal and is only called in the `constructor` so there is no need to make additional checks as it is only used once. The case is similar for `_creator` address, as it is set in the constructor, and then receives the `totalSupply` of tokens but it is not used again.

```
bool public mintedByDxsale = true;
bool public mintingFinishedPermanent = false;
address public _creator;

constructor (**args**) {
    //...
    _creator = creator_;
    _mint(_creator,tokenSupply_);
    mintingFinishedPermanent = true;
}
```

### Recommendation

It is recommended to remove unused state variables to enhance code optimization and readability as well as reduce deployment costs.

## UL - Unused Library

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L215 |
| **Status** | Unresolved |

### Description

In the contract the `Address` library is declared that provides additional functionality like checking if an address belongs to a contract and optimization of transfers and function calls. However it is not used anywhere in the main or parent contracts, therefore, it is redundant

```
library Address {/*...*/}
```

### Recommendation

It is recommended to remove unused libraries to enhance code optimization and readability as well as reduce deployment costs.

# L02 - State Variables could be Declared Constant

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L458 |
| **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.

```
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

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L464 |
| **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.

```
address public _creator
```

### 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.

## L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L681 |
| **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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
        require(accountBalance >= amount, "ERC20: burn amount
exceeds balance");
        _balances[account] = accountBalance - amount;
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);
    }
```

## 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

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | DxStandardToken.sol#L476 |
| **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.

```
_creator = creator_
```

## 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** | DxStandardToken.sol#L442 |
| **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)
}
```

### 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** | DxStandardToken.sol#L11,12 |
| **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 0.8.7;
pragma experimental ABIEncoderV2;
```

### Recommendation

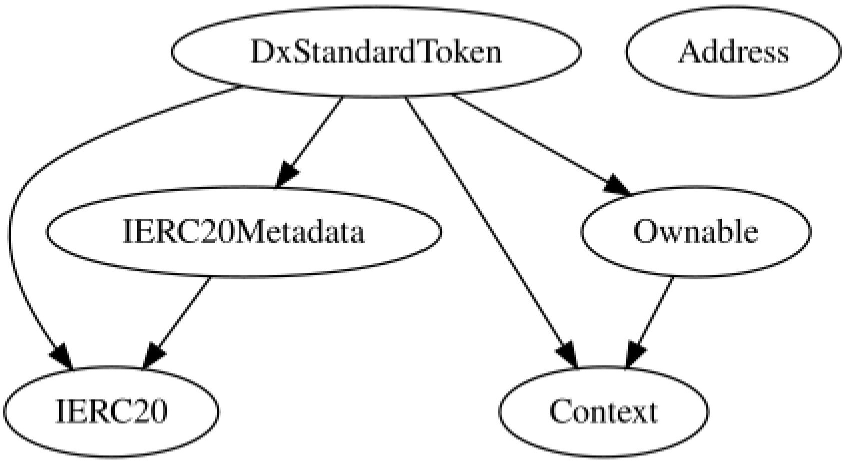
It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

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.

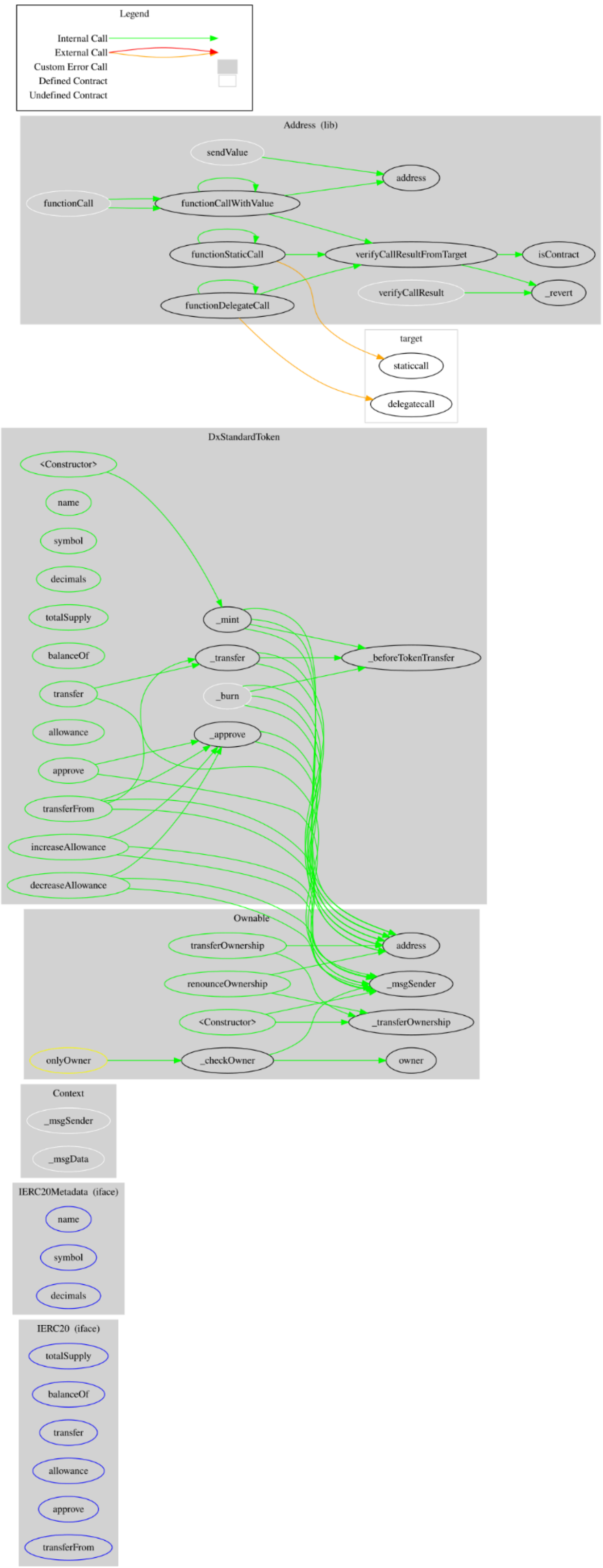
# Functions Analysis

| Contract | Type | Bases | | |
|---|---|---|---|---|
| | **Function Name** | **Visibility** | **Mutability** | **Modifiers** |
| | | | | |
| **DxStandardToken** | Implementation | Context, IERC20, IERC20Metadata, Ownable | | |
| | | 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 | ✓ | |
| | _beforeTokenTransfer | Internal | ✓ | |

# Inheritance Graph



# Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
address
functionCall
functionCallWithValue
functionStaticCall
verifyCallResultFromTarget
isContract
verifyCallResult
_revert
functionDelegateCall
target
staticcall
delegatecall
DxStandardToken
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
increaseAllowance
decreaseAllowance
_mint
_transfer
_burn
_approve
_beforeTokenTransfer
Ownable
transferOwnership
renounceOwnership
<Constructor>
onlyOwner
_checkOwner
address
_msgSender
_transferOwnership
owner
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

# Summary

Platy contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Platy 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. The contract does not implement any 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**

cyberscope.io