Cyberscope

Audit Report
Ethereal

January 2025

Network	ETH
Address	0xe137407Dbf9a768f342DE7b88607e186fC9c09fB
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
●	CR	Code Repetition	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	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**
Risk Classification **4**
Review **5**
Audit Updates 5
Source Files 5
Findings Breakdown **7**
CR - Code Repetition 8
Description 8
Recommendation 10
MEE - Missing Events Emission 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 13
L09 - Dead Code Elimination 14
Description 14
Recommendation 15
L15 - Local Scope Variable Shadowing 16
Description 16
Recommendation 16
L17 - Usage of Solidity Assembly 17
Description 17
Recommendation 17
L18 - Multiple Pragma Directives 18
Description 18
Recommendation 18
L19 - Stable Compiler Version 19
Description 19
Recommendation 19
Functions Analysis **20**
Inheritance Graph **22**
Flow Graph **23**
Summary **24**
Initial Audit, 6 Jan 2025 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	TaxableToken
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://etherscan.io/address/0xe137407dbf9a768f342de7b88607e186fc9c09fb
Address	0xe137407dbf9a768f342de7b88607e186fc9c09fb
Network	ETH
Symbol	ETHR
Decimals	18
Total Supply	100,000,000
Badge Eligibility	Yes

Audit Updates

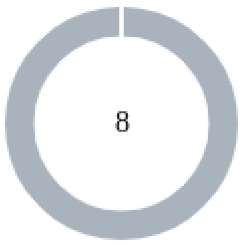
Initial Audit	06 Jan 2025

Source Files

Filename	SHA256
contracts/TaxableToken.sol	7f3fa68a0373b6c6fa4838ddf159af9a4bb80d2c89687a955caa26f71725a8b9
contracts/core/ERC20Taxable.sol	f063a00d4cb8a89429fe4d1537584807f4c5f1af7d60131b831422dca2613414

contracts/core/ERC20.sol	6c11180c5ee7f4c34dec33783f8e225cf6d787e4697cf359b623c429b931619f
contracts/core/utils/BlackList.sol	0fd6a5793c1302245c8a08123b8c56319d44fde95cdab6626c141a085c7d29ef

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	0	0	0

CR - Code Repetition

Criticality	Minor / Informative
Location	core/ERC20Taxable.sol#L42,72
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
    function transfer(address to, uint256 amount) public virtual override
returns (bool) {
        address owner = msg.sender;

        if (_taxRate > 0 && !(_isExcludedFromTaxFee[owner] ||
_isExcludedFromTaxFee[to])) {
            uint256 taxAmount = (amount * _taxRate) / 1000;

            if (taxAmount > 0) {
                _transfer(owner, _taxAddress, taxAmount);
                unchecked {
                    amount -= taxAmount;
                }
            }
        }

        _transfer(owner, to, amount);

        return true;
    }

    function transferFrom(address from, address to, uint256 amount)
public virtual override returns (bool) {
        address spender = msg.sender;
        _spendAllowance(from, spender, amount);

        if (_taxRate > 0 && !(_isExcludedFromTaxFee[from] ||
_isExcludedFromTaxFee[to])) {
            uint256 taxAmount = (amount * _taxRate) / 1000;

            if (taxAmount > 0) {
                _transfer(from, _taxAddress, taxAmount);
                unchecked {
                    amount -= taxAmount;
                }
            }
        }

        _transfer(from, to, amount);

        return true;
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	core/ERC20Taxable.sol#L122,136
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.

```
_taxRate = taxFeePerMille_;
_taxAddress = taxAddress_;
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/TaxableToken.sol#L16,17,18,19,20,21,22,23,48,52,56,60,64 contracts/core/utils/BlackList.sol#L17
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 _owner
string memory _name
string memory _symbol
uint8 _decimals
uint256 _initialSupply
uint256 _maxSupply
uint256 _taxFeePerMille
address _taxAddress
address _account
uint256 _newTaxFee
address _newTaxAddress
bool _status
```

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	@openzeppelin/contracts/utils/Address.sol#L64,89,99,118,128,145,155,170,180,195,219,231 @openzeppelin/contracts/proxy/utils/Initializable.sol#L156,163
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");
    }

...
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/TaxableToken.sol#L23
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _taxAddress
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	@openzeppelin/contracts/utils/Address.sol#L236
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	contracts/TaxableToken.sol#L2 contracts/core/utils/BlackList.sol#L2 contracts/core/ERC20Taxable.sol#L2 contracts/core/ERC20.sol#L2 @openzeppelin/contracts/utils/Context.sol#L4 @openzeppelin/contracts/utils/Address.sol#L4 @openzeppelin/contracts/token/ERC20/IERC20.sol#L4 @openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#L4 @openzeppelin/contracts/security/Pausable.sol#L4 @openzeppelin/contracts/proxy/utils/Initializable.sol#L4 @openzeppelin/contracts/access/Ownable.sol#L4
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 ^0.8.0;
pragma solidity ^0.8.1;
pragma solidity ^0.8.19;
pragma solidity ^0.8.2;
```

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	contracts/TaxableToken.sol#L2 contracts/core/utils/BlackList.sol#L2 contracts/core/ERC20Taxable.sol#L2
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 ^0.8.19;
pragma solidity ^0.8.0;
```

Recommendation

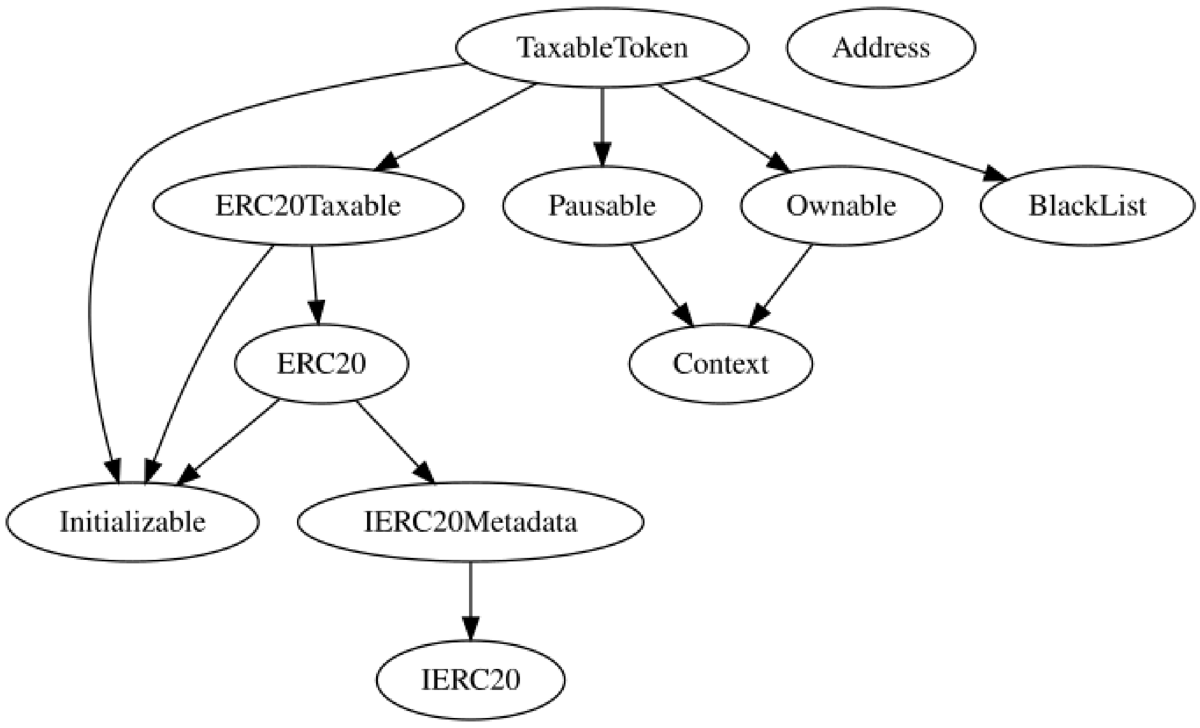
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

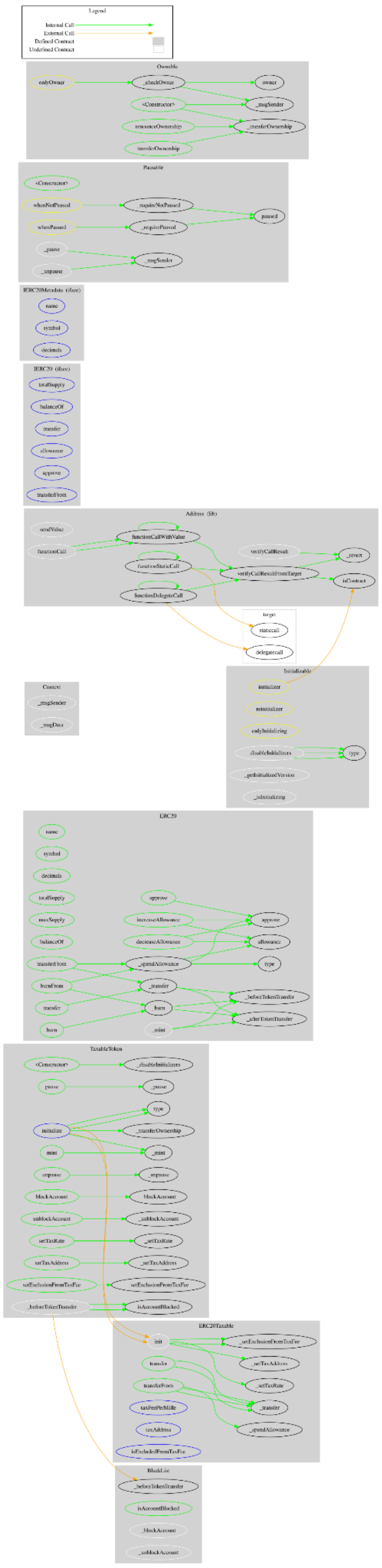
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TaxableToken	Implementation	Initializable, ERC20Taxable, Pausable, Ownable, BlackList		
		Public	✓	-
	initialize	External	✓	initializer
	pause	Public	✓	onlyOwner
	unpause	Public	✓	onlyOwner
	mint	Public	✓	onlyOwner
	blockAccount	Public	✓	onlyOwner
	unblockAccount	Public	✓	onlyOwner
	setTaxRate	Public	✓	onlyOwner
	setTaxAddress	Public	✓	onlyOwner
	setExclusionFromTaxFee	Public	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	whenNotPaused
ERC20Taxable	Implementation	Initializable, ERC20		
	init	Internal	✓	onlyInitializing
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	taxFeePerMille	External		-

	taxAddress	External		-
	isExcludedFromTaxFee	External		-
	_setTaxRate	Internal	✓	
	_setTaxAddress	Internal	✓	
	_setExclusionFromTaxFee	Internal	✓	
BlackList	Implementation			
	isAccountBlocked	Public		-
	_blockAccount	Internal	✓	
	_unblockAccount	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
Pausable
IERC20Metadata (iface)
IERC20 (iface)
Address (lib)
target
Context
Initializable
ERC20
TaxableToken
ERC20Taxable
BlockList

Summary

Ethereal contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Ethereal is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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 fees are locked at 0.3%.

Initial Audit, 6 Jan 2025

At the time of the audit report, the contract with address 0xe137407Dbf9a768f342DE7b88607e186fC9c09fB is pointed out by the following proxy address: 0x052c252cF4909f084Bd4f95f355C192ffb87bbAa.

The ownership of the proxy contract at address 0x052c252cF4909f084Bd4f95f355C192ffb87bbAa has been renounced. The information regarding the transaction can be accessed through the following link: https://etherscan.io/tx/0x00695628c76b353ee431d71557bec37effdbc69c8273b9b7f2d4fb5d64d95b37

Disclaimer

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