Audit Report

Immortl

January 2023

Type	ERC20
Network	MATIC
Address	0x217f7dead406c360b2ff676bacd2c73db5a24089
Audited by	© cyberscope

Table of Contents

Review

Contract Name	OneImmortl_ERC20_IMRTL
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://polygonscan.com/address/0x217f7dead406c360b2ff676bacd2c73db5a24089
Address	0x217f7dead406c360b2ff676bacd2c73db5a24089
Network	MATIC
Symbol	IMRTL
Decimals	18
Total Supply	400.000.000

Audit Updates

Initial Audit	25 Jan 2023

Source Files

Filename	SHA256
OneImmortl_ERC20_IMRTL.sol	2df5ae54aae6ded39bde7b84842e75ee7f741b308326a7eb58b9540218a01b12

Roles

The contract consist of three roles.

`SuperAdmin` is rensponsible for configuring the admin roles, taxes, liquidity pool pair and taxes timeline.

`Admin` is rensponsible for configuring taxes, liquidity pool pair and taxes timeline.

`Minter` is rensponsible for minting new tokens

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Medium
Location	OneImmortl_ERC20_IMRTL.sol#L1426
Status	Unresolved

Description

The contract owner has the authority to pause the sales for all users excluding the owner. The owner may take advantage of it by calling the `setEnableTransfer` method.

```
function setEnableTransfer(bool _enable) public
{
   //check
   requireAdmin(true);

   //set
   enableTransfer = _enable;
}
```

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. That risk can be prevented by temporarily locking the contract or renouncing ownership.

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	RSK	Redundant Storage Keyword	Unresolved
●	CO	Code Optimization	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	OneImmortl_ERC20_IMRTL.sol#L1684
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
function getTaxPercent(address _from, address _to) public virtual view
returns (uint32)
{
    //check tax wallet / excluded
    if (taxWallet == address(0)
        || excludedFromTax[_from]
        || excludedFromTax[_to])
    {
        return 0;
    }

    //make decision based on type
    int8 taxType = getTaxType(_from, _to);
    TaxInfo storage taxTier = taxes[getTaxTier(_from, _to)];
}
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions, because it is unnecessary and can make the code less readable.

CO - Code Optimization

Criticality	Minor / Informative
Location	OneImmortl_ERC20_IMRTL.sol#L1417,1427,1526,1537,1594,1739
Status	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 argument passed to the function `requireAdmin` is redundant because it always accepts a true value.

```
requireAdmin(true);
```

The method `requireAdmin` , the statements `require(taxTimeline.length == 0, "Timeline already set");` and `require(lastTimestamp < _timeline[n].beforeTimestamp, "Invalid order");` in the setTaxTimeline function are redundant. Because the function is only called by the constructor and is an internal function, meaning that it is not accessible or callable from external sources. Additionaly the condition `lastTimestamp < _timeline[n].beforeTimestamp` is always true. Hence, the variable lastTimestamp is redundant.

```
function setTaxTimeline(TaxTimeline[] memory _timeline) internal
{
    //check
    requireAdmin(true);
    require(taxTimeline.length == 0, "Timeline already set");

    //check order & push
    uint256 lastTimestamp = 0;
    for (uint256 n = 0; n < _timeline.length; n++)
    {
        require(lastTimestamp < _timeline[n].beforeTimestamp, "Invalid
order");
        taxTimeline.push(_timeline[n]);
    }
}
```

Recommendation

The team is advised to take into consideration these segments 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. The boolean argument and redundant statements could be removed.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	OneImmortl_ERC20_IMRTL.sol#L762,811,825,833,839,1302,1341,1350,1364,1370,1381,1410,1420,1448,1519,1558,1570,1577,1587,1626,1646,1668,1692,1709,1732,1770
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.

```
contract ML_RoleManager is
    AccessControlEnumerable
{
    //========================
    // CONSTANTS
    //========================
...
    function requireAdmin(bool _allowTxOrigin) internal view
    {
        require(
            isAdmin(_allowTxOrigin),
            "User is not Admin");
    }
}

...
```

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	OneImmortl_ERC20_IMRTL.sol#L156,163,170,184,217,264,271,278,292,356,381,1217,1660
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 remove(Bytes32Set storage set, bytes32 value) internal returns
(bool) {
        return _remove(set._inner, value);
    }

function contains(Bytes32Set storage set, bytes32 value) internal view
returns (bool) {
        return _contains(set._inner, value);
...
function length(Bytes32Set storage set) internal view returns (uint256)
{
        return _length(set._inner);
    }

function at(Bytes32Set storage set, uint256 index) internal view
returns (bytes32) {
        return _at(set._inner, index);
    }

...
```

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	OneImmortl_ERC20_IMRTL.sol#L1325,1326,1495,1496
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.

```
string memory _name
string memory _symbol
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	OneImmortl_ERC20_IMRTL.sol#L1525
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.

```
taxWallet = _wallet
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	OneImmortl_ERC20_IMRTL.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 ^0.8.0;
```

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**
EnumerableSet	Library			
	_add	Private	✓	
	_remove	Private	✓	
	_contains	Private		
	_length	Private		
	_at	Private		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
IERC165	Interface			

	supportsInterface	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
Strings	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IAccessControl	Interface			
	hasRole	External		-
	getRoleAdmin	External		-
	grantRole	External	✓	-
	revokeRole	External	✓	-
	renounceRole	External	✓	-
AccessControl	Implementation	Context, IAccessControl, ERC165		
	supportsInterface	Public		-
	hasRole	Public		-
	_checkRole	Internal		
	getRoleAdmin	Public		-
	grantRole	Public	✓	onlyRole
	revokeRole	Public	✓	onlyRole

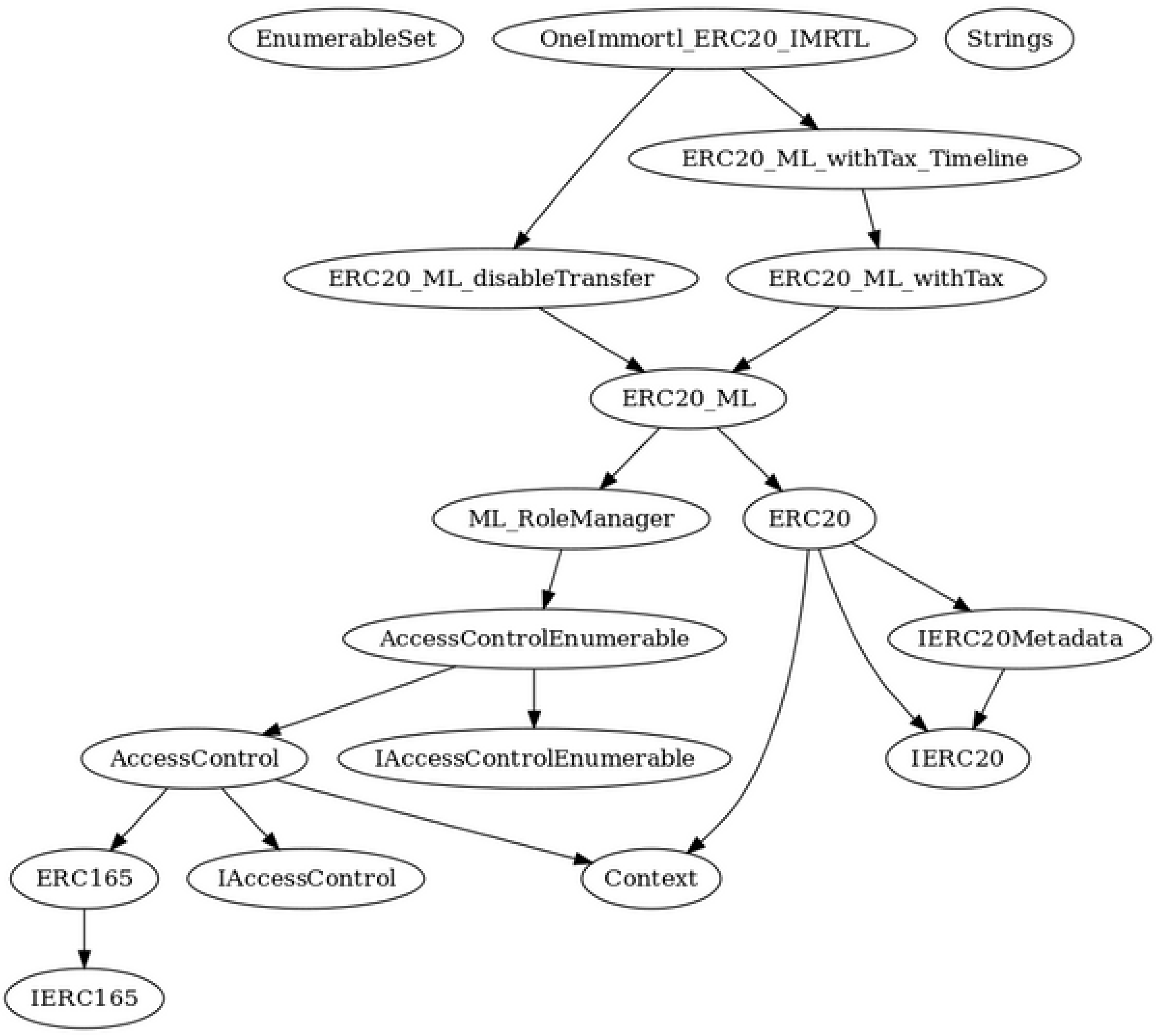
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
	_setRoleAdmin	Internal	✓	
	_grantRole	Private	✓	
	_revokeRole	Private	✓	
IAccessControlEnumerable	Interface			
	getRoleMember	External		-
	getRoleMemberCount	External		-
AccessControlEnumerable	Implementation	IAccessControlEnumerable, AccessControl		
	supportsInterface	Public		-
	getRoleMember	Public		-
	getRoleMemberCount	Public		-
	grantRole	Public	✓	-
	revokeRole	Public	✓	-
	renounceRole	Public	✓	-
	_setupRole	Internal	✓	
ML_RoleManager	Implementation	AccessControlEnumerable		
		Public	✓	-
	getSuperAdmin	Public		-
	transferSuperAdmin	Public	✓	-
	requireRole	Internal		
	isAdmin	Internal		
	requireAdmin	Internal		

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		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	✓	

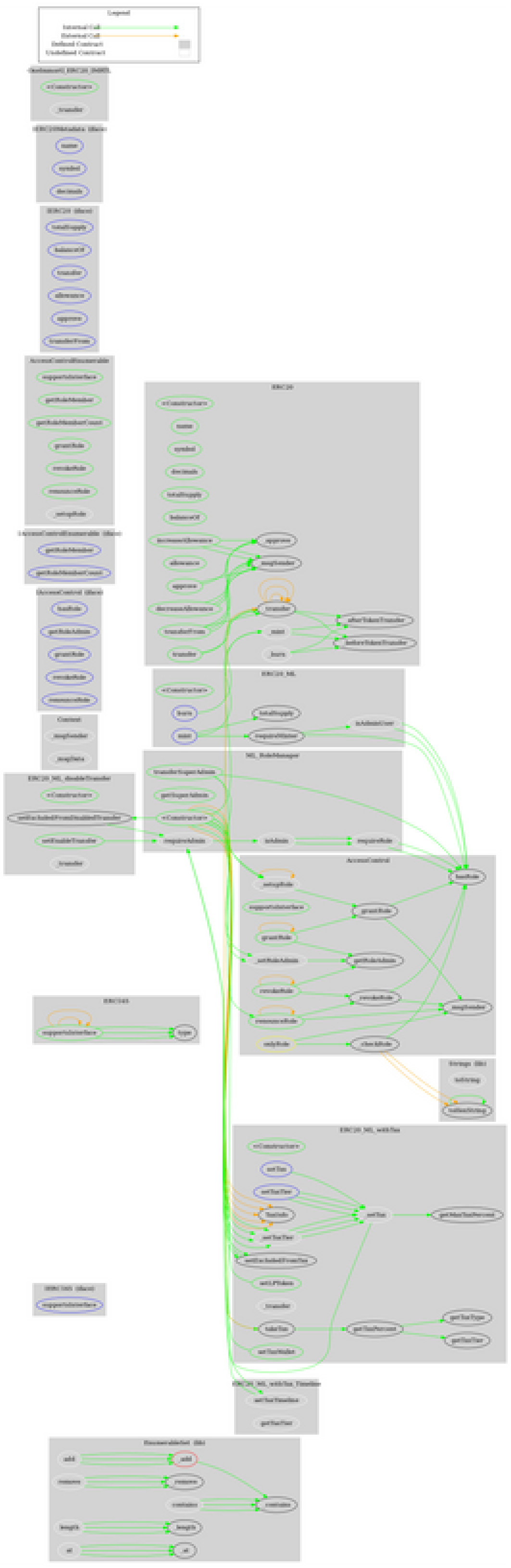
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC20_ML	Implementation	ERC20, ML_RoleManager		
		Public	✓	ERC20
	burn	External	✓	-
	mint	External	✓	-
	isAdminUser	Internal		
	requireMinter	Internal		
ERC20_ML_disableTransfer	Implementation	ERC20_ML		
		Public	✓	-
	setExcludedFromDisabledTransfer	Public	✓	-
	setEnableTransfer	Public	✓	-
	_transfer	Internal	✓	
ERC20_ML_withTax	Implementation	ERC20_ML		
		Public	✓	ERC20_ML
	setTaxWallet	Public	✓	-
	_setTax	Internal	✓	
	setTax	External	✓	-
	_setTaxTier	Internal	✓	
	setTaxTier	External	✓	-
	setExcludedFromTax	Public	✓	-
	setLPToken	Public	✓	-

	_transfer	Internal	✓	
	takeTax	Internal	✓	
	getTaxType	Internal		
	getTaxTier	Internal		
	getTaxPercent	Public		-
	getMaxTaxPercent	Internal		
ERC20_ML_withTax_Timeline	Implementation	ERC20_ML_withTax		
	setTaxTimeline	Internal	✓	
	getTaxTier	Internal		
OneImmortl_ERC20_IMRTL	Implementation	ERC20_ML_withTax_Timeline, ERC20_ML_disableTransfer		
		Public	✓	ERC20_ML_withTax
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph



Summary

The Smart Contract analysis reported no critical compiler error or issues. The contract owner has the ability to stop the transactions. Other than that, The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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

https://www.cyberscope.io