Cyberscope

Audit Report

Elancer

October 2023

Network BSC

Address 0xa7b94a42479190662f78c51938e2da8bd623857e

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
	MOEE	Misleading Ownership Event Emission	Unresolved
	MEE	Missing Events Emission	Unresolved
	CR	Code Repetition	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis 1
Diagnostics 2
Table of Contents 3
Review 4
Audit Updates 4
Source Files 4
Findings Breakdown 5
MOEE - Misleading Ownership Event Emission 6
Description 6
Recommendation 7
MEE - Missing Events Emission 8
Description 8
Recommendation 8
CR - Code Repetition 9
Description 9
Recommendation 10
IDI - Immutable Declaration Improvement 11
Description 11
Recommendation 11
L04 - Conformance to Solidity Naming Conventions 12
Description 12
Recommendation 12
L19 - Stable Compiler Version 14
Description 14
Recommendation 14
Functions Analysis 15
Flow Graph 17
Summary 18
Disclaimer 19
About Cyberscope 20

Review

Contract Name	Token
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xa7b94a42479190662f78c51938e2da8bd623857e
Address	0xa7b94a42479190662f78c51938e2da8bd623857e
Network	BSC
Symbol	ELCR
Decimals	9
Total Supply	1,000,000,000

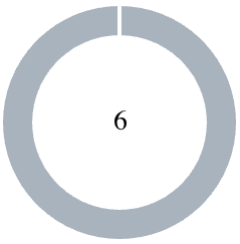
Audit Updates

Initial Audit	27 Oct 2023

Source Files

Filename	SHA256
Token.sol	1701a9cbf286d97d89bfd8af77780f2e5f60dd75cc118d59100250d7ff3e65c2

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

MOEE - Misleading Ownership Event Emission

Criticality	Minor / Informative
Location	Token.sol#L295
Status	Unresolved

Description

The contract contains a potential issue with the emission of the OwnershipTransferred event. The issue arises when the methods transferOwnership, renounceOwnership, and lockOwnership are called. In all cases, the contract emits an OwnershipTransferred event, but the emitted event's parameters may lead to a misleading interpretation.

In particular, when the lockOwnership method is invoked, it emits an OwnershipTransferred event with the zero address as the new owner. However, in this context, the ownership is not permanently renounced but rather temporarily moved to the zero address and is expected to revert back once the ownership time locker expires. This could potentially lead users and decentralized applications (Dapps) to misinterpret the event as the ownership being permanently renounced, causing confusion and misalignment with the actual contract behavior.

```
function renounceOwnership() public onlyOwner {
    owner = address(0);
    _previousOwner = address(0);
    ownershipRenounced = true;
    emit OwnershipTransferred(owner, address(0));
}

function transferOwnership(address _newOwner) public onlyOwner {
    require(_newOwner != address(0), "New owner can't be zero
address. If you still want to do this use renounceOwnership
function.");
    owner = _newOwner;
    _previousOwner = _newOwner;
    emit OwnershipTransferred(owner, _newOwner);
}

function lockOwnership(uint _duration) public onlyOwner {
    _previousOwner = owner;
    owner = address(0);
    _unlockTime = block.timestamp + _duration;
    emit OwnershipTransferred(_previousOwner, address(0));
}
```

Recommendation

To improve transparency and prevent misleading interpretations, we recommend differentiating between the temporary transfer of ownership in the `lockOwnership` method and the permanent renouncement of ownership in the `renounceOwnership` method. The team could modify the `lockOwnership` method to emit a new event, such as `OwnershipLocked`, to clearly indicate the temporary transfer of ownership to the zero address. Ensure that the event description includes relevant information, such as the time when the ownership is expected to revert.

By implementing these changes, the contract will provide more accurate and user-friendly event emissions, reducing the risk of misinterpretation and confusion among users and Dapps. This approach will also align the events with the actual behavior of the contract, enhancing its overall usability and transparency.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Token.sol#L256,260,264,268,272,276,287,291
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 deflationOff() public onlyOwner {
    deflationStatus = false;
}

function setTax1Status(bool _on) public onlyOwner {
    tax1Status = _on;
}
...
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	Token.sol#L102,163
Status	Unresolved

Description

The contract contains redundancy in the implementation of the transfer and transferFrom methods. These methods, while conceptually different in their purpose, both contain identical code. The only distinction between them is the source of the sender's address. In the transfer method, it relies on msg.sender as the sender's address, while in the transferFrom method, the sender's address is determined from the _from parameter.

Here is a simplified example of the redundancy:

```
function transfer(address _to, uint256 _value) public returns
(bool) {
    require(balances[msg.sender] >= _value, "Insufficient
balance");
    balances[msg.sender] -= _value;
    balances[_to] += _value;
    emit Transfer(msg.sender, _to, _value);
    return true;
}

function transferFrom(address _from, address _to, uint256 _value)
public returns (bool) {
    require(_allowed[_from][msg.sender] >= _value, "Allowance is
too low");
    _allowed[_from][msg.sender] -= _value;

    require(balances[_from] >= _value, "Insufficient balance");
    balances[_from] -= _value;
    balances[_to] += _value;
    emit Transfer(_from, _to, _value);
    return true;
}
```

Both methods essentially perform the same operations, which include checking the balance of the sender, updating the balances, and emitting a Transfer event. This repetition of code

not only increases the codebase's size but also introduces the potential for inconsistencies or errors in the future if the logic in one method is modified and not reflected in the other.

Recommendation

To enhance code maintainability and reduce redundancy, we recommend refactoring the code to create a common private function that encapsulates the shared logic for transferring tokens. 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Token.sol#L57,58,60,62,64,65,66,69,70,71,74,75,76
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
totalSupply
deflationPercent
deflationTotal
tax1Percent
tax1Address
tax1Total
tax2Percent
tax2Address
tax2Total
tax3Percent
tax3Address

...
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Token.sol#L94,98,159,222,228,232,241,260,264,268,272,279,283,287,298,309
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
uint _value
address _to
address _from
address _spender
uint _amount
bool _on
address _account
address _newOwner
uint _duration
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Token.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;
```

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**
Token	Implementation			
		Public	✓	-
	balanceOf	Public		-
	transfer	Public	✓	-
	transferFrom	Public	✓	-
	approve	Public	✓	-
	allowance	Public		-
	burn	Public	✓	-
	burnFrom	Public	✓	-
	deflationOn	Public	✓	onlyOwner
	deflationOff	Public	✓	onlyOwner
	setTax1Status	Public	✓	onlyOwner
	setTax2Status	Public	✓	onlyOwner
	setTax3Status	Public	✓	onlyOwner
	setAllTaxesStatus	Public	✓	onlyOwner
	isExcludedFromFee	Public		-
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner

	transferOwnership	Public	✓	onlyOwner
	getUnlockTime	Public		-
	lockOwnership	Public	✓	onlyOwner
	unlockOwnership	Public	✓	-

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Token
onlyOwner
<Constructor>
balanceOf
transfer
transferFrom
approve
allowance
burn
burnFrom
deflationOn
deflationOff
setTax1Status
setTax2Status
setTax3Status
setAllTaxesStatus
isExcludedFromFee
excludeFromFee
includeInFee
renounceOwnership
transferOwnership
getUnlockTime
lockOwnership
unlockOwnership

Summary

The Elancer contract incorporates a token mechanism, and this audit is conducted to examine security vulnerabilities, evaluate the integrity of the business logic, and identify opportunities for enhancements. Elancer is a notable project with an amiable and expanding user community. Our analysis of the smart contract revealed no compiler errors or critical concerns. While the contract owner holds access to specific administrative functions, it is important to note that these functions do not have the potential for malicious use to disrupt user transactions. Additionally, the contract imposes a maximum fee limit of 5%, which can be toggled on or off by the contract owner as needed.

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