Audit Report

Lanco - Decentralized Freelancing Platform

July 2023

Network	BSC
Address	0x9d343b89cf72484edd2fc2238a2a0738b5cca805
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	RTC	Redundant Time Condition	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Lanco
Compiler Version	v0.8.20+commit.a1b79de6
Optimization	200 runs
Explorer	https://bscscan.com/address/0x9d343b89cf72484edd2fc2238a2a0738b5cca805
Address	0x9d343b89cf72484edd2fc2238a2a0738b5cca805
Network	BSC
Symbol	$LANC
Decimals	18
Total Supply	1,000,000,000

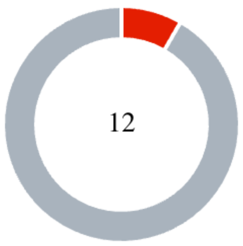
Audit Updates

Initial Audit	31 Jul 2023

Source Files

Filename	SHA256
Vesting.sol	c36af09fa1bc8aaa97b4f32a02983d128b57ee8a0a3ea97f90fb8686e9632c51
LancToken.sol	72c252dba44acceb0c0155669337a92a5562faa26b0bf657da03ee6fcf146ace
IPinkAntiBot.sol	aa707474783d2e21f2949d940cf6a2164ea7c0734f4f85b8bfc547dd2e40da3b
IPancakeFactory.sol	e16138ab10a97414cf6aeaf5e4b5b0a759116b2d4073f4d3ad5aad6c6178507f

Findings Breakdown



🔴 Critical	1
🟠 Medium	0
● Minor / Informative	11

Severity		Unresolved	Acknowledged	Resolved	Other
🔴	Critical	1	0	0	0
🟠	Medium	0	0	0	0
●	Minor / Informative	11	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	LancToken.sol#L230
Status	Unresolved

Description

The owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `antiDumpTime` to a high value. As a result, the contract may operate as a honeypot.

The owner has the authority to pause the transaction for all users excluding the owner. The owner may take advantage of this by calling the `pauseTrading()` or setting the `transactionLimit` to a relatively low value.

```
require(antiDump[from] < block.timestamp, "Err: antiDump active");
...
if (!_isExcluded[from]) {
    require(tradingEnabled == true, "Lanco: Trading not enabled
yet");
}
...
require(
    _isExcluded[from] || _isExcluded[to] || amount <=
transactionLimit,
    "Lanco: Max transaction Limit Exceeds!"
);
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RTC - Redundant Time Condition

Criticality	Minor / Informative
Location	LancToken.sol#L261
Status	Unresolved

Description

tautology The condition enters in the if statement if the expression `launchTime + 3 minutes >= block.timestamp` fulfills, inside the if-statement the contract checks the same condition `if (launchTime + 180 seconds >= block.timestamp)`. Comparing a similar nested if statement is a tautology.

```
if (
    !_isExcluded[from] &&
    !_isExcluded[to] &&
    launchTime + 3 minutes >= block.timestamp
) {
    require(
        automatedMarketMakerPairs[from] ||
            balanceOf(to) + amount <= maxSupply / (10000),
        "AntiBot: Buy Banned!"
    );
    if (launchTime + 180 seconds >= block.timestamp)
        require(automatedMarketMakerPairs[to], "AntiBot: Sell 
Banned!");
}
```

Recommendation

THe team is advices to modify the code so it does not contain tautologies.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	Vesting.sol#L40
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.

Since the `intervel()` does not always return a one-month interval, the `elapsedMonths` naming is misleading.

```
uint256 elapsedMonths = (timestamp - start()) / intervel();
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	LancToken.sol#L68,71,72,73,74,75,76,83,89,95
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.

```
pancakeV2Pair
pinkAntiBot
presalewallet
cexWallet
airdropsWallet
liquidityWallet
devVesting
teamVesting
resvrsVesting
marketingVesting
```

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

Criticality	Minor / Informative
Location	LancToken.sol#L12,19
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.

```
IPancakeFactory public pancakeFactory
uint256 public maxSupply = 10_000_000e18
```

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	LancToken.sol#L116,120,126
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.

```
function setPinkAntiBot_TokenOwner() external onlyOwner {
        pinkAntiBot.setTokenOwner(owner());
    }
bool _enable
```

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	Vesting.sol#L22
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 monthsToSeconds(uint256 months_) internal pure returns
(uint256) {
        return months_ * 10 days;
    }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	LancToken.sol#L242
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(tradingEnabled == true, "Lanco: Trading not enabled yet")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Vesting.sol#L40,43
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 elapsedMonths = (timestamp - start()) / intervel()
return
                (totalAllocation * elapsedMonths) / (duration() /
intervel())
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	LancToken.sol#L72,73,74,75,181
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.

```
presalewallet = _presalewallet
cexWallet = _cexWallet
airdropsWallet = _airdropsWallet
liquidityWallet = _liquidityWallet
feeWallet = feaddress
```

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	Vesting.sol#L3
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	LancToken.sol#L310
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(tokenAddress).transfer(owner(), tokenAmount)
```

Recommendation

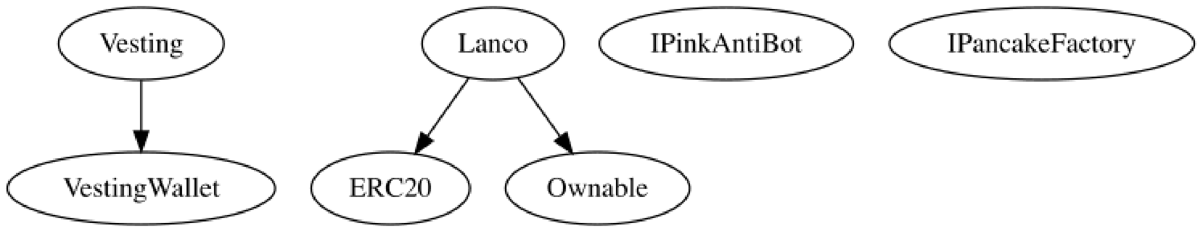
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

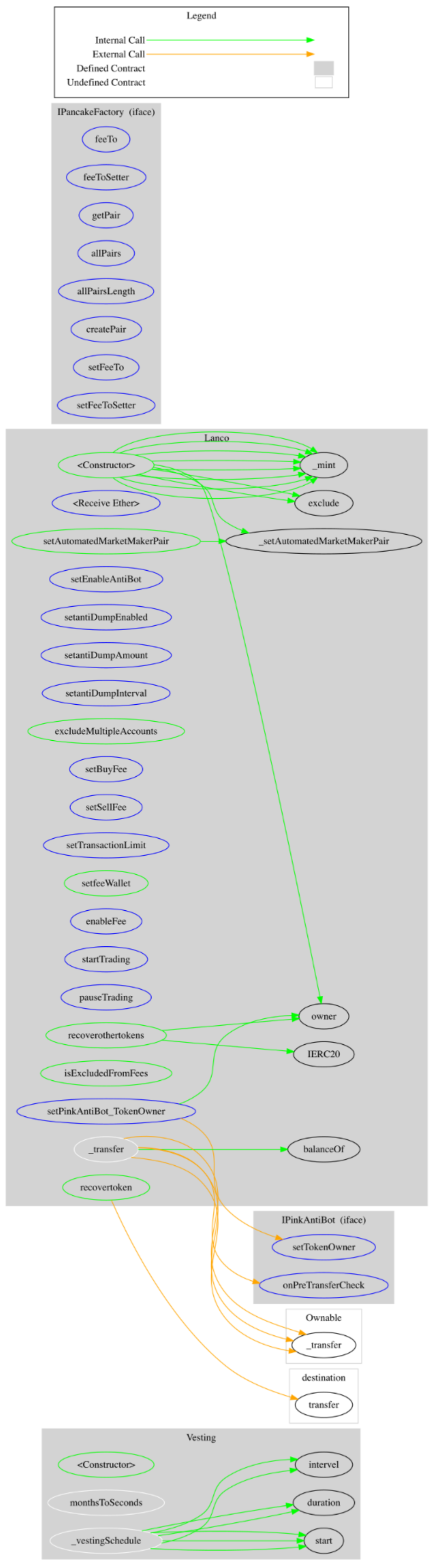
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Vesting	Implementation	VestingWalle t		
		Public	Payable	VestingWallet
	monthsToSeconds	Internal		
	intervel	Public		-
	_vestingSchedule	Internal		
Lanco	Implementation	ERC20, Ownable		
		Public	✓	-
		External	Payable	-
	setPinkAntiBot_TokenOwner	External	✓	onlyOwner
	setEnableAntiBot	External	✓	onlyOwner
	setantiDumpEnabled	External	✓	onlyOwner
	setantiDumpAmount	External	✓	onlyOwner
	setantiDumpInterval	External	✓	onlyOwner
	exclude	Public	✓	onlyOwner
	excludeMultipleAccounts	Public	✓	onlyOwner
	setBuyFee	External	✓	onlyOwner
	setSellFee	External	✓	onlyOwner
	setTransactionLimit	External	✓	onlyOwner

	setfeeWallet	Public	✓	onlyOwner
	enableFee	External	✓	onlyOwner
	startTrading	External	✓	onlyOwner
	pauseTrading	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	recoverothertokens	Public	✓	onlyOwner
	recovertoken	Public	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IPancakeFactory (iface)
feeTo
feeToSetter
getPair
allPairs
allPairsLength
createPair
setFeeTo
setFeeToSetter
Lanco
<Constructor>
_mint
<Receive Ether>
exclude
setAutomatedMarketMakerPair
_setAutomatedMarketMakerPair
setEnableAntiBot
setantiDumpEnabled
setantiDumpAmount
setantiDumpInterval
excludeMultipleAccounts
setBuyFee
setSellFee
setTransactionLimit
setfeeWallet
enableFee
startTrading
pauseTrading
owner
recoverothertokens
IERC20
isExcludedFromFees
setPinkAntiBot_TokenOwner
_transfer
balanceOf
recovertoken
IPinkAntiBot (iface)
setTokenOwner
onPreTransferCheck
Ownable
_transfer
destination
transfer
Vesting
<Constructor>
intervel
monthsToSeconds
duration
_vestingSchedule
start

Summary

Lanco - Decentralized Freelancing Platform. contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 15% 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