Cyberscope

Audit Report

Paco De Llama

October 2023

Network	BSC
Address	0xDe5438D66CeBF2311A16565E44d5f0bD7981aE4D
Audited by	© cyberscope

Table of Contents

Review

Explorer	https://bscscan.com/address/0xde5438d66cebf2311a16565e44d5f0bd7981ae4d

Audit Updates

Initial Audit	16 Oct 2023

Source Files

Filename	SHA256
pacoStaking.sol	37ec9047873499529b2dcf617446d5b167dd791bf75dbc331f2659689855075e

Overview

The PacoStaking smart contract is designed to facilitate a staking mechanism for a specific token. Users can stake their tokens for a certain period and earn daily rewards based on the daily rate of interest (dailyROI). Staking comes with a staking tax rate, and there is also a tax for unstaking (unstakingTaxRate). Users must stake for a minimum amount of time (stakeTime) to qualify for rewards. The contract tracks staked amounts, rewards, and referral bonuses. Users can withdraw their earnings, which include staking rewards, once they meet the stake duration requirement. The contract's owner can adjust various parameters, such as tax rates and minimum stake value, and also control the contract's activity status. This smart contract primarily serves as a staking and reward distribution mechanism for a specific token, with flexibility for configuration by the contract owner.

Roles

Owner

The owner has authority over the following functions:

- `function rewardPool()`
- `function changeActiveStatus()`
- `function setStakingTaxRate(uint256 _stakingTaxRate)`
- `function setUnstakingTaxRate(uint256 _unstakingTaxRate)`
- `function setDailyROI(uint256 _dailyROI)`
- `function setMinimumStakeValue(uint256 _minimumStakeValue)`
- `function setStakeTime(uint256 _newStakeTime)`

User

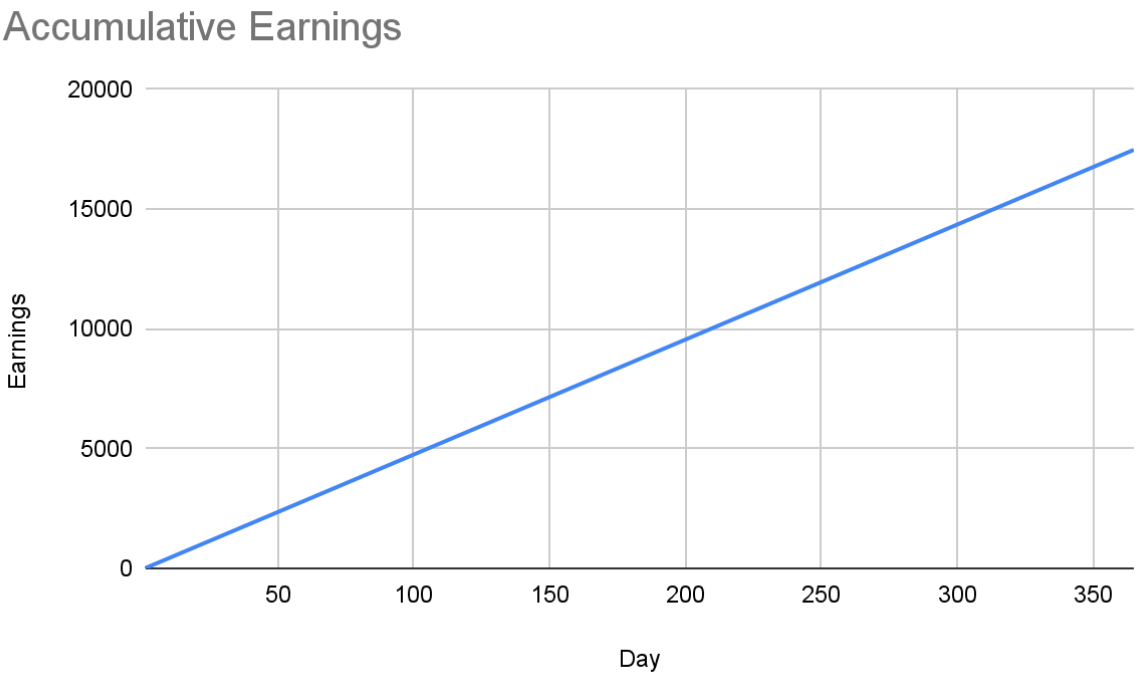
The user can interact with the following functions:

- `function calculateEarnings(address _stakeholder)`
- `function stake(uint256 _amount)`
- `function unstake(uint256 _amount)`
- `function withdrawEarnings()`
- `function checkUnstakeStatus(address _unstaker)`

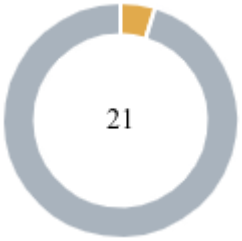
Reward Distribution

The contract's reward distribution mechanism operates on a linear distribution model, commencing from the initiation of staking and continuing throughout the staking period. The rewards multiplier is detailed in the rewards sheet and is graphically represented on the Y-Axis.

- Time: 365 days
- Participation Amount: 10000 tokens



Findings Breakdown



Critical	0
Medium	1
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	20	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	IRC	Inaccurate Reward Calculation	Unresolved
Minor / Informative	PUA	Possible Unstakeable Amount	Unresolved
Minor / Informative	NR	Non-Guaranteed Rewards	Unresolved
Minor / Informative	CR	Code Repetition	Unresolved
Minor / Informative	MSC	Missing Sanity Check	Unresolved
Minor / Informative	MEE	Missing Events Emission	Unresolved
Minor / Informative	RRRF	Redundant Referral Reward Functionality	Unresolved
Minor / Informative	PTAI	Potential Transfer Amount Inconsistency	Unresolved
Minor / Informative	TUU	Time Units Usage	Unresolved
Minor / Informative	RED	Redundant Event Declaration	Unresolved
Minor / Informative	RVD	Redundant Variable Declaration	Unresolved
Minor / Informative	RRS	Redundant Require Statement	Unresolved
Minor / Informative	RSML	Redundant SafeMath Library	Unresolved
Minor / Informative	IDI	Immutable Declaration Improvement	Unresolved

●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

IRC - Inaccurate Reward Calculation

Criticality	Medium
Location	pacoStaking.sol#L114,127
Status	Unresolved

Description

The contract has an issue in the `calculateEarnings` function where it determines the number of staking days for a user based on the timestamp. When a user stakes, the `lastClock` variable is set to the beginning of the day (00:00). This setup means that a user who stakes near the end of the day can withdraw a full day's worth of rewards after just a few minutes, as the contract calculates an active day. This behavior can lead to unintended gains for users who stake at the end of the day.

```
function calculateEarnings(address _stakeholder) public view returns(uint256) {
    uint256 activeDays = (block.timestamp - lastClock[_stakeholder]) / 86400;
    return (stakes[_stakeholder] * dailyROI * activeDays) / 10000;
}

uint256 remainder = (block.timestamp - lastClock[msg.sender]) % 86400;
lastClock[msg.sender] = block.timestamp - remainder;
```

Recommendation

To address this issue, the team is advised to adjust the calculation of staking days in the `calculateEarnings` function. Instead of using the timestamp of when the day started, the team should consider using the timestamp of when user actually made the staking or another method that accurately reflects the user's staking duration. This adjustment will ensure that users are rewarded proportionally to the actual time their funds are staked, avoiding the issue of earning a full day's reward after just a few minutes.

PUA - Possible Unstakeable Amount

Criticality	Minor / Informative
Location	pacoStaking.sol#L142
Status	Unresolved

Description

The owner has the authority to stop users from unstaking. The owner may take advantage of this by calling the `setStakeTime` function with a large value.

```
require(block.timestamp - timeOfStake[msg.sender] > stakeTime, "You need
to stake for the minimum amount of days");
```

Recommendation

The contract could embody a check for not allowing setting the `stakeTime` more than a reasonable amount. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

NR - Non-Guaranteed Rewards

Criticality	Minor / Informative
Location	pacoStaking.sol#L114,179
Status	Unresolved

Description

The contract's staking reward calculation formula is heavily dependent on the `dailyROI` variable. The owner has authority to manipulate this variable without restrictions. Hence, if the owner assigns the value of zero to the variable, the users will not receive any rewards.

```
function calculateEarnings(address _stakeholder) public view
returns(uint256) {
    uint256 activeDays = (block.timestamp - lastClock[_stakeholder]) /
86400;
    return (stakes[_stakeholder] * dailyROI * activeDays) / 10000;
}

function setDailyROI(uint256 _dailyROI) external onlyOwner() {
    dailyROI = _dailyROI;
}
```

Recommendation

The contract could embody a check for not allowing setting the `dailyROI` less than a reasonable amount. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

CR - Code Repetition

Criticality	Minor / Informative
Location	pacoStaking.sol#L127,140,155
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.

```
uint256 remainder = (block.timestamp - lastClock[msg.sender]) % 86400;
lastClock[msg.sender] = block.timestamp - remainder;
```

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.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	pacoStaking.sol#L171,175
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `stakingTaxRate` and `unstakingTaxRate` should be less than 1000.

```
stakingTaxRate = _stakingTaxRate;
unstakingTaxRate = _unstakingTaxRate;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	pacoStaking.sol#L167,171,175,179,183,187
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.

```
active = !active;
stakingTaxRate = _stakingTaxRate;
unstakingTaxRate = _unstakingTaxRate;
dailyROI = _dailyROI;
minimumStakeValue = _minimumStakeValue;
stakeTime = _newStakeTime;
```

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.

RRRF - Redundant Referral Reward Functionality

Criticality	Minor / Informative
Location	pacoStaking.sol#L149,153,154
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.

In the `withdrawEarnings` function, the contract calculates the `totalReward` by adding the referralRewards, stakeRewards, and the earnings of a user. However, the referralRewards variable is not initialized or updated anywhere in the contract, resulting in it always holding its default value, which is zero. This variable serves no purpose and is redundant.

```
uint256 totalReward = referralRewards[msg.sender] +
stakeRewards[msg.sender] + calculateEarnings(msg.sender);
referralRewards[msg.sender] = 0;
referralCount[msg.sender] = 0;
```

Recommendation

The team is advised to take these segments into consideration 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.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	pacoStaking.sol#L122
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
require(IERC20(token).transferFrom(msg.sender, address(this), _amount),
"Stake failed due to failed amount transfer.");
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
 Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

TUU - Time Units Usage

Criticality	Minor / Informative
Location	pacoStaking.sol#L115
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint256 activeDays = (block.timestamp - lastClock[_stakeholder]) / 86400;
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	pacoStaking.sol#L91
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 contract declares certain events in its code. However, these events are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event OnRegisterAndStake(address stakeholder, uint256 amount, uint256
totalTax, address _referrer);
```

Recommendation

The team is advised to take these segments into consideration 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.

A recommended approach would be to either remove the declared events that are not being emitted or to incorporate the necessary emit statements within the contract's functions to actually emit these events when relevant actions occur.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	pacoStaking.sol#L79
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 contract declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
bool public registered = true;
```

Recommendation

The team is advised to take these segments into consideration 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.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	pacoStaking.sol#L19
Status	Unresolved

Description

The contract utilizes a require statement within the add function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
// SafeMath add overflow redundant require
function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    require(c >= a, "SafeMath: addition overflow");
}
```

Recommendation

It is recommended to remove the require statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	pacoStaking.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	pacoStaking.sol#L101
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.

```
token
```

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	pacoStaking.sol#L79
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 registered = 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	pacoStaking.sol#L61,67,114,119,134,170,174,178,182,186,190
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 _newOwner

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	pacoStaking.sol#L179,187
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
dailyROI = _dailyROI
stakeTime = _newStakeTime
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	pacoStaking.sol#L115,116
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 activeDays = (block.timestamp - lastClock[_stakeholder]) / 86400
return (stakes[_stakeholder] * dailyROI * activeDays) / 10000
```

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	pacoStaking.sol#L62,101
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.

```
owner = _newOwner
token = _token
```

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	pacoStaking.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 ^0.8.16;
```

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	pacoStaking.sol#L144,157
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(token).transfer(msg.sender, afterTax)
IERC20(token).transfer(msg.sender, totalReward)
```

Recommendation

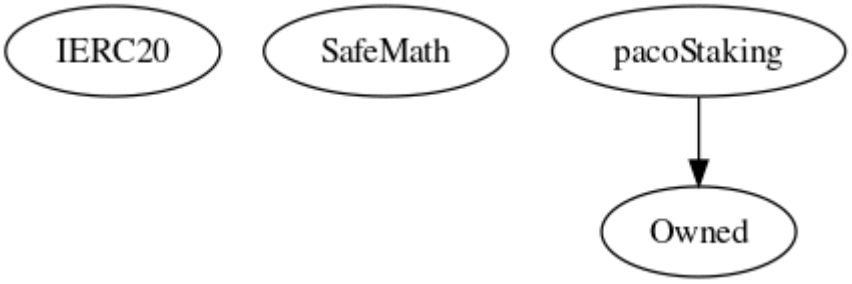
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	transfer	External	✓	-
	transferFrom	External	✓	-
	balanceOf	External		-
	approve	External	✓	-
	allowance	External		-
	totalSupply	External		-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Owned	Implementation			
		Public	✓	-

	transferOwnership	Public	✓	onlyOwner
pacoStaking	Implementation	Owned		
		Public	✓	-
	calculateEarnings	Public		-
	stake	External	✓	whenActive
	unstake	External	✓	-
	withdrawEarnings	External	✓	-
	rewardPool	External		onlyOwner
	changeActiveStatus	External	✓	onlyOwner
	setStakingTaxRate	External	✓	onlyOwner
	setUnstakingTaxRate	External	✓	onlyOwner
	setDailyROI	External	✓	onlyOwner
	setMinimumStakeValue	External	✓	onlyOwner
	setStakeTime	External	✓	onlyOwner
	checkUnstakeStatus	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
pacoStaking
<Constructor>
whenActive
stake
unstake
calculateEarnings
withdrawEarnings
rewardPool
changeActiveStatus
setStakingTaxRate
setUnstakingTaxRate
setDailyROI
setMinimumStakeValue
setStakeTime
checkUnstakeStatus
Owned
<Constructor>
onlyOwner
transferOwnership
SafeMath (lib)
add
sub
mul
div
mod
IERC20 (iface)
transfer
transferFrom
balanceOf
approve
allowance
totalSupply

Summary

Paco De Llama contract implements a staking mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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