Cyberscope

Audit Report

Propchain

June 2023

SHA256 036567f20d6528acbb25cecebe14c50f5ff1a821e1245dd87b5501cffca4d6f7

Audited by © cyberscope

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0xa0d91043cc4a352f8a7968918d1faad72a8325ac

Audit Updates

Initial Audit	27 Jun 2023

Source Files

Filename	SHA256
contracts/PROPCStaking.sol	036567f20d6528acbb25cecebe14c50f5ff1a821e1245dd87b5501cffca4d6f7

Introduction

The PROPCStaking contract is a staking mechanism that allows users to deposit tokens into various pools and earn rewards in the form of another token. Each pool has its own unique parameters, including a start time, claim time limit, minimum stake amount, penalty fee, penalty time limit, rewards token, and Annual Percentage Yield (APY). The contract also supports VIP pools, which are exclusive to certain addresses.

The contract owner has the ability to add and update the pools parameters, as well as manage VIP addresses. The rewards for staking are calculated based on the Annual Percentage Yield (APY) set for each pool, the amount of tokens a user has staked, and the duration of the stake. The APY can be updated over time and the rewards are distributed from a designated rewards wallet.

Users can join a pool by depositing tokens by calling the joinPool function, and their rewards accrue over time and they can claim their rewards after a certain time limi, by calling the redeem function.

Users can leave a pool by calling the leavePool function, which allows them to withdraw their staked tokens and any pending rewards. If the withdrawal occurs before the penalty time limit, a penalty fee is deducted from the withdrawal.

The contract uses the OpenZeppelin library for secure and standard compliant implementations of ERC20 token interactions, safe math operations, and ownership management.

Roles

Owner

The owner has authority over the following functions:

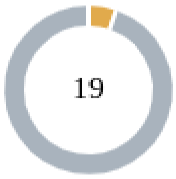
- `function updateRewardsWallet(address _wallet)`
- `function setPool( uint256 _pid, uint256 _startTime, IERC20 _rewardsToken, uint256 _apyPercent, uint256 _claimTimeLimit, uint256 _penaltyFee, uint256 _penaltyTimeLimit, bool _active, address _penaltyWallet, bool _isVIPPool)`
- `function addVIPAddress(uint256 _pid, address _vipAddress)`
- `function addVIPAddresses(uint256 _pid, address[] memory _vipAddresses)`
- `function removeVIPAddress(uint256 _pid, address _vipAddress)`
- `function removeVIPAddresses(uint256 _pid, address[] memory _vipAddresses)`

User

The user can interact with the following functions:

- `function poolLength()`
- `function getMultiplier(uint256 _from, uint256 _to)`
- `function pendingRewardsToken(uint256 _pid, address _user)`
- `function allPendingRewardsToken(address _user)`
- `function joinPool(uint256 _pid, uint256 _amount)`
- `function leavePool(uint256 _pid, uint256 _amount)`
- `function redeem(uint256 _pid)`
- `function redeemAll()`
- `function getUserInfo(uint256 _pid, address _account)`
- `function getPoolInfo(uint256 _pid)`

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	18

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	18	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
🟠	CTI	Claim Time Inconsistency	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	CR	Code Repetition	Unresolved
●	SRAI	Sufficient Reward Amount Issue	Unresolved
●	ADU	Arbitrary Decimals Usage	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	MC	Missing Check	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved

●	L11	Unnecessary Boolean equality	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

CTI - Claim Time Inconsistency

Criticality	Medium
Location	contracts/PROPCStaking.sol#L964,991
Status	Unresolved

Description

The smart contract implements a staking mechanism where users can claim rewards after a certain time limit (`pool.claimTimeLimit`). This time limit is enforced in the `redeem` function with the a require statement. This requirement ensures that users can only claim their rewards after the `claimTimeLimit` has passed since their last claim.

However, the contract does not enforce this time limit when users call the `leavePool` or `joinPool` functions. This allows users to potentially claim their rewards before the `claimTimeLimit` by leaving the pool and then rejoining or just rejoining the pool or by staking a very small amount of tokens, for instance 0,000000001. This effectively allows users to bypass the `claimTimeLimit` enforced in the `redeem` function.

This issue could lead to unexpected behavior and potential exploitation, as users could claim their rewards more frequently than intended.

```
function joinPool(uint256 _pid, uint256 _amount) external {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    require(pool.startTime < block.timestamp, "mining is not
started yet");
    require(pool.active, "pool not active");
    require(!pool.isVIPPool || pool.isVIPAddress[msg.sender] ==
true || user.amount + _amount >= pool.minStakeAmount, "not vip
qualified");

    if (user.amount > 0) {
        uint256 pendingRewards = pendingRewardsToken(_pid,
msg.sender);
    if(pendingRewards > 0) {
        safeRewardTransfer(_pid, msg.sender, pendingRewards);
        user.totalRedeemed =
user.totalRedeemed.ad(pendingRewards);             }
    }

    propcToken.transferFrom(msg.sender, address(this),
_amount);

    user.amount = user.amount.add(_amount);
    user.lastClaimTimestamp = block.timestamp;
    user.depositTimestamp = block.timestamp;

    pool.totalStaked = pool.totalStaked.add(_amount);
    emit Deposit(msg.sender, _pid, _amount);
}



function leavePool(uint256 _pid, uint256 _amount) external {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    uint256 pendingRewards = pendingRewardsToken(_pid,
msg.sender);
    if(pendingRewards > 0) {
        safeRewardTransfer(_pid, msg.sender, pendingRewards);
        user.totalRedeemed =
user.totalRedeemed.add(pendingRewards);
    }
    ...

}
```

Recommendation

The contract could enforce the `claimTimeLimit` in the `leavePool` and `joinPool` functions as well as in the `redeem` function. This could be done by adding a requirement similar to the one in the `redeem` function, which checks whether the `claimTimeLimit` has passed since the user's last claim. This would ensure that users cannot claim their rewards before the `claimTimeLimit` , even if they leave and rejoin the pool.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L980
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

```
propcToken.transferFrom(msg.sender, address(this), _amount);
```

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
```

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L973,995,1024
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 pendingRewards = pendingRewardsToken(_pid, msg.sender);
if(pendingRewards > 0) {
    user.totalRedeemed += pendingRewards;
    safeRewardTransfer(_pid, msg.sender, pendingRewards);
}
```

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.

SRAI - Sufficient Reward Amount Issue

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L975,997,1027,1041
Status	Unresolved

Description

The contract is distributing rewards without checking if the `rewardsToken` balance is sufficient to cover the reward amount. As a result, the expected rewards might not be transferred.

```
safeRewardTransfer(_pid, msg.sender, pendingRewards);

function safeRewardTransfer(uint256 _pid, address _to, uint256 _amount) internal {

IERC20(poolInfo[_pid].rewardsToken).safeTransferFrom(rewardsWallet, _to, _amount);
}
```

Recommendation

The contract could check if the `rewardsToken` balance is sufficient to cover the reward amount. If it is not sufficient then it could return a descriptive message. A possible solution could be to check if there is sufficient balance prior to adding airdrop wallets.

ADU - Arbitrary Decimals Usage

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L919
Status	Unresolved

Description

The contract calculates the rewards assuming the tokens decimals are fixed. The contract owner has the authority to add any token with different amounts of decimals. As a result, the precision will be wrong.

```
function pendingRewardsToken(uint256 _pid, address _user) public
view returns (uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];

    ...

    uint256 multiplier = getMultiplier(_fromTime, _toTime);
    uint256 rewardsPerAPYBlock =
multiplier.mul(pool.apyInfo[apyIndex].apyPercent).mul(user.amount).d
iv(365 days).div(10000);
        pendingRewards = pendingRewards.add(rewardsPerAPYBlock);
    }

    return pendingRewards;
}
```

Recommendation

The contract could calculate the reward ratio with the corresponding token's decimals ERC20.decimals() instead of adding a fixed value.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L919,1045,1053
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 pendingRewardsToken(uint256 _pid, address _user)
public view returns (uint256) {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_user];

    ...
}

function getUserInfo(uint256 _pid, address _account) public
view returns(uint256 amount, uint256 totalRedeemed) {
    UserInfo storage user = userInfo[_pid][_account];
    return (
    user.amount,
    user.totalRedeemed
    );
}

function getPoolInfo(uint256 _pid) public view returns(
    uint256 startTime,
    ...
  ) {
    PoolInfo storage pool = poolInfo[_pid];
    ...
}
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L811
Status	Unresolved

Description

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

```
constructor(
        IPropcToken _propc,
        address _rewardsWallet
    ) {
        propcToken = _propc;
        rewardsWallet = _rewardsWallet;
    }
```

Also the `leavePool` function allows a user to withdraw their staked tokens and any pending rewards from a specified pool. However, the function does not verify if the user is a participant in the specified pool or if they have staked any tokens.

If a user who has not staked any tokens in the pool calls the `leavePool` function with an amount of zero, the function will not revert. Instead, it will update the `lastClaimTimestamp` for the user in the specified pool and emit a `Withdraw` event, even though the user has not actually withdrawn any tokens.

```
function leavePool(uint256 _pid, uint256 _amount) external {
    ...

    user.amount = user.amount.sub(_amount);

    ...

    emit Withdraw(msg.sender, _pid, _amount);
    }
```

This could potentially mislead external observers or off-chain systems monitoring these events for tracking user activity or for other purposes. It could also confuse the user, as they might mistakenly believe they have withdrawn tokens from a pool in which they have not participated.

Recommendation

The contract should properly check the variables according to the required specifications.

- The address _propc should not be set to zero address.

Also it is recommended to add a check at the beginning of the `leavePool` function to verify that the user has staked tokens in the specified pool. This can be done by checking if the user's `_amount` is greater than zero. If it is not, the function should revert with an appropriate error message. This will ensure that only users who have staked tokens in the pool can call the `leavePool` function and that the `Withdraw` event is only emitted when tokens are actually withdrawn.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L819,831,873,880,889,896
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 updateRewardsWallet(address _wallet) external onlyOwner {
        require(_wallet != address(0x0), "invalid rewards wallet
address");
        rewardsWallet = _wallet;

    }

    function setPool( uint256 _pid,
        uint256 _startTime,

        ...
    ) public onlyOwner {
        uint256 pid = _pid == 0 ? ++totalPools : _pid;

        PoolInfo storage pool = poolInfo[pid];

        ...
    }

    function addVIPAddress(uint256 _pid, address _vipAddress) external
onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];

        ...
    }
    function addVIPAddresses(uint256 _pid, address[] memory
_vipAddresses) external onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];

         ...
    }

    function removeVIPAddress(uint256 _pid, address _vipAddress)
external onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];

         ...
    }

    function removeVIPAddresses(uint256 _pid, address[] memory
_vipAddresses) external onlyOwner {
        PoolInfo storage pool = poolInfo[_pid];

        ...
        }
    }
```

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.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L998,1026
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
user.totalRedeemed = user.totalRedeemed.add(pendingRewards);

user.totalRedeemed += pendingRewards;
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/PROPCStaking.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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L920,921,1046,1066
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.

```
PoolInfo storage pool
UserInfo storage user
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L815
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.

```
propcToken
```

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	contracts/PROPCStaking.sol#L819,832,833,834,835,836,837,838,839,840,841,873,880,889,896,906,919,953,964,991,1018,1041,1045,1053
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 _wallet
uint256 _pid
uint256 _startTime
IERC20 _rewardsToken
uint256 _apyPercent
uint256 _claimTimeLimit
uint256 _penaltyFee
uint256 _penaltyTimeLimit
bool _active
address _penaltyWallet
bool _isVIPPool
address _vipAddress
address[] memory _vipAddresses
uint256 _from

...
```

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	contracts/PROPCStaking.sol#L163,188,217,250,260,277,287,663,687,702,711,914
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 functionCall(target, data, "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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L875,882,891,898,970
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(pool.isVIPPool == true, "not vip pool")
require(!pool.isVIPPool || pool.isVIPAddress[msg.sender] ==
true || user.amount + _amount >= pool.minStakeAmount, "not vip
qualified")
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L816
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.

```
rewardsWallet = _rewardsWallet
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L141,316
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 {
            size := extcodesize(account)
        }

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/PROPCStaking.sol#L7,34,112,332,562,647,746
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	contracts/PROPCStaking.sol#L980,1006,1007
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.

```
propcToken.transferFrom(msg.sender, address(this), _amount)
propcToken.transfer(msg.sender, _amount.sub(penaltyAmount))
propcToken.transfer(pool.penaltyWallet, penaltyAmount)
```

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

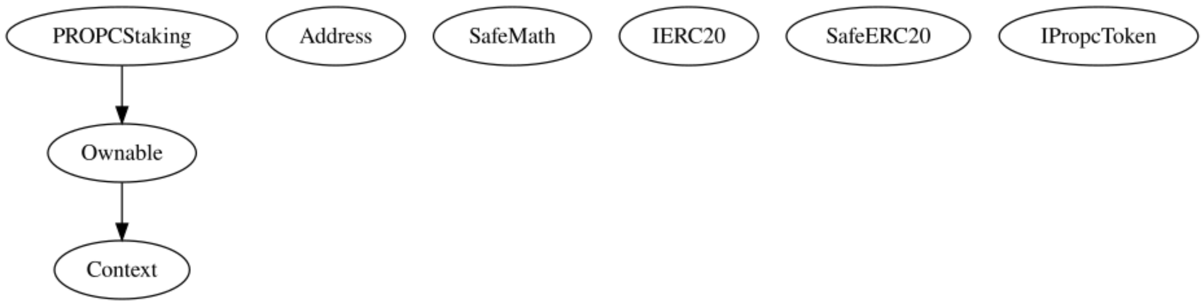
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	

	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	verifyCallResult	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IERC20	Interface			
	totalSupply	External		-

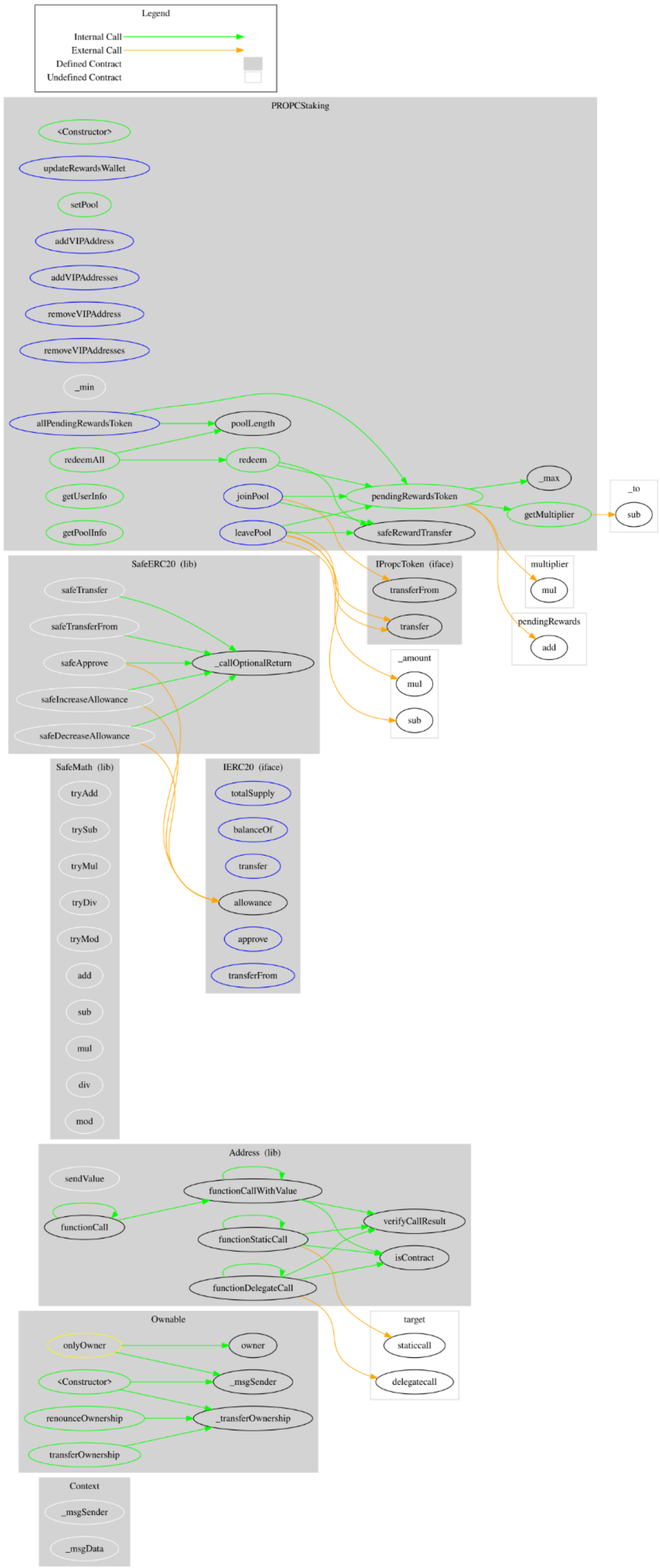
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeERC20	Library			
	safeTransfer	Internal	✓	
	safeTransferFrom	Internal	✓	
	safeApprove	Internal	✓	
	safeIncreaseAllowance	Internal	✓	
	safeDecreaseAllowance	Internal	✓	
	_callOptionalReturn	Private	✓	
IPropcToken	Interface			
	transferFrom	External	✓	-
	transfer	External	✓	-
PROPCStaking	Implementation	Ownable		
		Public	✓	-
	updateRewardsWallet	External	✓	onlyOwner
	poolLength	Public		-
	setPool	Public	✓	onlyOwner

	addVIPAddress	External	✓	onlyOwner
	addVIPAddresses	External	✓	onlyOwner
	removeVIPAddress	External	✓	onlyOwner
	removeVIPAddresses	External	✓	onlyOwner
	getMultiplier	Public		-
	_max	Internal		
	_min	Internal		
	pendingRewardsToken	Public		-
	allPendingRewardsToken	External		-
	joinPool	External	✓	-
	leavePool	External	✓	-
	redeem	Public	✓	-
	redeemAll	Public	✓	-
	safeRewardTransfer	Internal	✓	
	getUserInfo	Public		-
	getPoolInfo	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
PROPCStaking
<Constructor>
updateRewardsWallet
setPool
addVIPAddress
addVIPAddresses
removeVIPAddress
removeVIPAddresses
_min
allPendingRewardsToken
poolLength
redeemAll
redeem
_max
getUserInfo
joinPool
pendingRewardsToken
getMultiplier
_to
sub
getPoolInfo
leavePool
safeRewardTransfer
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
_callOptionalReturn
safeIncreaseAllowance
safeDecreaseAllowance
IPropcToken (iface)
transferFrom
transfer
multiplier
mul
pendingRewards
add
_amount
mul
sub
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
Ownable
onlyOwner
owner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
Context
_msgSender
_msgData

Summary

Propchain contract implements a staking and rewards 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