# QuillAudit - Task

> Content

this this

## Source Contract

## High Severity Findings

### Access-Control Bug

```
    function isBlacklisted(address _account) internal view returns (bool) {
        if (blacklistEnabled) {
            {
                _isBlackListed[_account] == true;
            }
            return true;
        } else {
            return false;
        }
    }
```

If blacklist is enabled, the function always returns true, no matter the provided account is blacklisted or not.

This will halt the _transfer() function, resulting into a denial of service situation.

POC of this attack can be found here.

> Mitigation: Put 'isBlackListed[_account] == true' under an if statement to return true as shown below

```

    function isBlacklisted(address _account) internal view returns (bool) {
        if(!blacklistEnabled) return false;
        if(!_isBlackListed[_account]) return false;
        return true;
    }
```

---

## Wrong Reset

```
    function withdraw(IERC20 token, uint256 _amt) external {
        UserInfo storage user = userInfo[msg.sender];
        user.stakedAmount = user.stakedAmount - _amt;
        calculateFee(msg.sender, _amt);
        uint256 rewards = user.pendingRewardAmount;
        user.rewardAmount = rewards;

        uint256 fee = _amt.mul(100).div(FEE_RATE);

        user.pendingRewardAmount = 0;
        user.stakedAt = 0;

        IERC20(rewardToken).transfer(msg.sender, rewards);
        IERC20(token).transfer(msg.sender, _amt - fee);
    }
```

In the above function, after the withdrawal, 'stakedAt' is set to 0. This may result in a loophole to draw infinite StakingReward using the following rinse repeat cycle: Withdraw small amount -> claimReward

POC for this attack can be found here:

> Mitigation: Do not alter "user.stakedAt" in withdraw function

---

## No Check For Token Type

```
    function withdraw(IERC20 token, uint256 _amt) external {
        UserInfo storage user = userInfo[msg.sender];
        user.stakedAmount = user.stakedAmount - _amt;
        calculateFee(msg.sender, _amt);
        uint256 rewards = user.pendingRewardAmount;
        user.rewardAmount = rewards;

        uint256 fee = _amt.mul(100).div(FEE_RATE);

        user.pendingRewardAmount = 0;
        user.stakedAt = 0;

        IERC20(rewardToken).transfer(msg.sender, rewards);
        IERC20(token).transfer(msg.sender, _amt - fee);
    }
```

This function does not check if the address of the token to be withdrawn is same that which was deposited. Can lead to depositing shit-coins, and withdrawing valuable tokens.

POC for this attack can be found here

> Mitigation: Save the token address at the time of staking and use the saved address at withdrawal.

---

### Use of wrong variable

```
    function withdraw(IERC20 token, uint256 _amt) external {
        UserInfo storage user = userInfo[msg.sender];
        user.stakedAmount = user.stakedAmount - _amt;
        calculateFee(msg.sender, _amt);
        uint256 rewards = user.pendingRewardAmount;
        user.rewardAmount = rewards;

        uint256 fee = _amt.mul(100).div(FEE_RATE);

        user.pendingRewardAmount = 0;
        user.stakedAt = 0;

        IERC20(rewardToken).transfer(msg.sender, rewards);
        IERC20(token).transfer(msg.sender, _amt - fee);
    }
```

'fee' will always be larger than "_amt" resulting in revert. Also, the output of "calculateFee" is nether stored nor used.

> Mitigation: Correct the "fee" calculation formula

---

### Staking rewrites previous stakes

```
    function stake(uint256 amount, IERC20 token) external {
        if (amount == 0) revert zeroAmount();
        UserInfo memory user = UserInfo(amount, 0, 0, block.number, 0);
        userInfo[msg.sender] = user;
        IERC20(token).transferFrom(msg.sender, address(this), amount);
    }
```

Calling stake more than once by the same address resets the information of the previous stake.

POC for this vulnerability can be found here.

> Mitigation: Either implement a mechanism to sum multiple staking instances or put a require statement to restrict staking twice.

## Mid Severity Findings

### Unsafe ERC20 Transfer

ERC20 operations can be unsafe due to different implementations and vulnerabilities in the standard.

It is therefore recommended to always either use OpenZeppelin's SafeERC20 library or at least to wrap each operation in a require statement for external ERC20 token interactions.

**Instances**

```
QuillTest.sol::949 => ERC20token.transfer(msg.sender, balance);
QuillTest.sol::1300 => IERC20(token).transferFrom(msg.sender,
address(this), amount);
QuillTest.sol::1339 => IERC20(rewardToken).transfer(msg.sender, rewards);
QuillTest.sol::1340 => IERC20(token).transfer(msg.sender, _amt - fee);
```

## Gas Optimizations Suggestions

### Use != 0 instead of > 0 for Unsigned Integer Comparison

**Instances**

```
QuillTest.sol::374 => return account.code.length > 0;
QuillTest.sol::536 => if (returndata.length > 0) {
QuillTest.sol::647 => require(b > 0, errorMessage);
QuillTest.sol::1079 => _totalFeesOnBuy + _totalFeesOnSell > 0 &&
QuillTest.sol::1088 => if (liquidityShare > 0) {
QuillTest.sol::1094 => if (marketingShare > 0) {
QuillTest.sol::1114 => if (_totalFees > 0) {
```

### Long Revert Strings, consider switching to custom errors

**Instances**

```
QuillTest.sol::386 => "Address: unable to send value, recipient may have
reverted"
QuillTest.sol::421 => "Address: low-level call with value failed"
QuillTest.sol::433 => "Address: insufficient balance for call"
QuillTest.sol::455 => "Address: low-level static call failed"
QuillTest.sol::482 => "Address: low-level delegate call failed"
QuillTest.sol::588 => "Ownable: new owner is the zero address"
QuillTest.sol::633 => require(c / a == b, "SafeMath: multiplication
overflow");
QuillTest.sol::739 => "ERC20: transfer amount exceeds allowance"
QuillTest.sol::766 => "ERC20: decreased allowance below zero"
QuillTest.sol::777 => require(sender != address(0), "ERC20: transfer from the zero address");
QuillTest.sol::778 => require(recipient != address(0), "ERC20: transfer to the zero address");
QuillTest.sol::782 => "ERC20: transfer amount exceeds balance"
QuillTest.sol::797 => require(account != address(0), "ERC20: burn from the zero address");
QuillTest.sol::801 => "ERC20: burn amount exceeds balance"
QuillTest.sol::812 => require(owner != address(0), "ERC20: approve from the zero address");
QuillTest.sol::813 => require(spender != address(0), "ERC20: approve to the zero address");
QuillTest.sol::940 => "Owner cannot claim contract's balance of its own tokens"
QuillTest.sol::962 => "Account is already the value of 'excluded'"
QuillTest.sol::1034 => require(from != address(0), "ERC20: transfer from the zero address");
QuillTest.sol::1035 => require(to != address(0), "ERC20: transfer to the zero address");
QuillTest.sol::1061 => "AntiWhale: Transfer amount exceeds the maxTransactionAmount"
QuillTest.sol::1066 => "AntiWhale: Transfer amount exceeds the maxTransactionAmount"
QuillTest.sol::1129 => "MaxWallet: Recipient exceeds the maxWalletAmount"
QuillTest.sol::1138 => require(swapEnabled != _enabled, "swapEnabled already at this state.");
QuillTest.sol::1145 => "SwapTokensAtAmount must be greater than 0.0001% of total supply"
QuillTest.sol::1207 => "Max wallet limit is already set to that state"
QuillTest.sol::1215 => "Max wallet percentage cannot be lower than 1%"
QuillTest.sol::1226 => "Account is already set to that state"
QuillTest.sol::1255 => "Max transaction limit is already set to that state"
QuillTest.sol::1268 => "Max Transaction limis cannot be lower than 0.1% of total supply"
QuillTest.sol::1280 => "Account is already set to that state"
```