SMART CONTRACT AUDIT REPORT

For

Moon Money (MM)

Prepared By: SFI Team

Prepared for: Moon Money Team

Prepared on: 17/11/2021

Table of Content

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the

report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (SaferICO) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

- # Overview of the audit

The project has 16 file. The main file contains approx 772 lines of Solidity code. Most of the functions and state variables are well commented on using the Nat spec documentation, but that does not create any vulnerability.

- # Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices automatically.

1. Unit tests passing.

2. Compilator warnings;

3. Race Conditions. Reentrancy. Cross-function Race Conditions. Pitfalls in Race Condition solutions;

4. Possible delays in data delivery;

5. Transaction-Ordering Dependence (front running);

6. Timestamp Dependence;

7. Integer Overflow and Underflow;

8. DoS with (unexpected) Revert;

9. DoS with Block Gas Limit;

10. Call Depth Attack.

11. Methods execution permissions;

12. Oracles calls;

13. Economy model. It’s important to forecast scenarios when a user is provided with additional economic motivation or faced with limitations. If application logic is based on incorrect economy model, the application will not function correctly and participants will incur financial losses. This type of issue is most often found in bonus rewards systems.
14. The impact of the exchange rate on the logic;
15. Private user data leaks.

- # Good things in smart contract

Compiler version is static: -
=> In this file, you have put “pragma solidity 0.6.12;” which is a good way to define the compiler version.

```
pragma solidity 0.6.12;
```

- **SafeMath library: -**

Moon Money (MM) is using full SafeMath file it is a good thing. This protects you from underflow and overflow attacks.

```
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256)
{
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
```

- **Ownable library : -**
  - Here you Moon Money (MM) token using ownable library,
    Initializes the contract setting the deployer as the initial owner

```
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed
previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the
deployer as the initial owner.
     */
    constructor () public {
```

```
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0),
    msgSender);
    }
```

- Here you Moon Money (MM) token using Dividend Paying Token library,

A mintable ERC20 token that allows anyone to pay and distribute ether to token holders as dividends and allows token holders to withdraw their dividends With `magnitude`, we can properly distribute dividends even if the amount received ether is small.

For more discussion about choosing the value of `magnitude

```
contract DividendPayingToken is ERC20,
DividendPayingTokenInterface,
DividendPayingTokenOptionalInterface {
  using SafeMath for uint256;
  using SafeMathUint for uint256;
  using SafeMathInt for int256;
  uint256 constant internal magnitude = 2**128;

  uint256 internal magnifiedDividendPerShare;
```

- Here you Moon Money (MM) using Uniswap libraries (IUniswapV2Factory interface, IUniswapV2Pair interface, and IUniswapV2Router01 interface)

```
interface IUniswapV2Factory {
        event PairCreated(address indexed token0,
    address indexed token1, address pair, uint);
```

```
interface IUniswapV2Pair {
     event Approval(address indexed owner, address
    indexed spender, uint value);
        event Transfer(address indexed from, address
    indexed to, uint value);
```

```
interface IUniswapV2Router01 {
     function factory() external pure returns
    (address);
        function WETH() external pure returns
    (address);
```

• Critical vulnerabilities found in the contract

There not Critical severity vulnerabilities found

• High vulnerabilities found in the contract

There not High severity vulnerabilities found

• Medium vulnerabilities found in the contract

There not Medium severity vulnerabilities found

• Low severity vulnerabilities found

#Check-effects-interaction:

```
function _withdrawDividendOfUser(address payable user) internal returns (u
  uint256 _withdrawableDividend = withdrawableDividendOf(user);
  if (_withdrawableDividend > 0) {
    withdrawnDividends[user] = withdrawnDividends[user].add(_withdrawable
    emit DividendWithdrawn(user, _withdrawableDividend);
    (bool success,) = user.call{value: _withdrawableDividend, gas: 3000}(

    if(!success) {
      withdrawnDividends[user] = withdrawnDividends[user].sub(_withdrawab
      return 0;
    }
    return _withdrawableDividend;}
  return 0;}
```

In detail

Potential violation of Checks-Effects-Interaction pattern in DividendPayingToken._withdrawDividendOfUser(address payable): Could potentially lead to re-entrancy vulnerability.

#Delete from dynamic array:

```
delete map.inserted[key];
        delete map.values[key];

        uint index = map.indexOf[key];
        uint lastIndex = map.keys.length - 1;
        address lastKey = map.keys[lastIndex];

        map.indexOf[lastKey] = index;
        delete map.indexOf[key];
```

In detail

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

#Transaction origin:

```
 emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex,
false, gas, tx.origin);
emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex,
true, gas, tx.origin);
```

In detail

Use of tx.origin: "tx.origin" is useful only in very exceptional cases.

If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.

For loop over dynamic array:

```
for(uint256 i = 0; i < accounts.length; i++) {
            _isExcludedFromFees[accounts[i]] = excluded;
        }
```

```
if(_lastProcessedIndex >= tokenHoldersMap.keys.length) {
                _lastProcessedIndex = 0;
            }
```

In detail

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

- # Notes

#Call

```
   (bool success,) = user.call{value: _withdrawableDividend, gas:
3000}("");
```

In detail
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if the return value is not handled properly.
Please use Direct Calls via specifying the called contract's interface.

#ERC20:

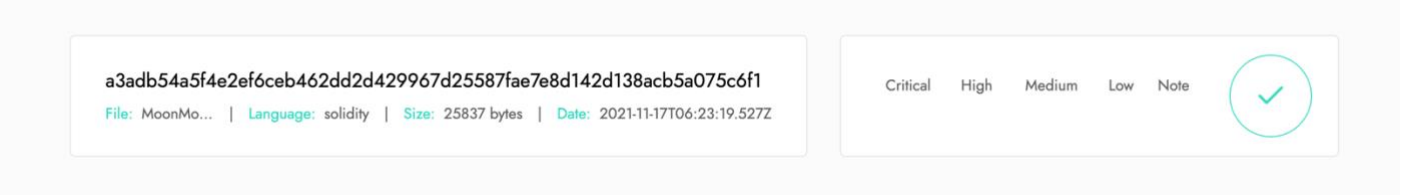
```
event Approval(address indexed owner,
address indexed spender, uint value);

 event Transfer(address indexed from,
address indexed to, uint value);
```

In detail
ERC20 contract's "decimals" function should have "uint8" as return type.

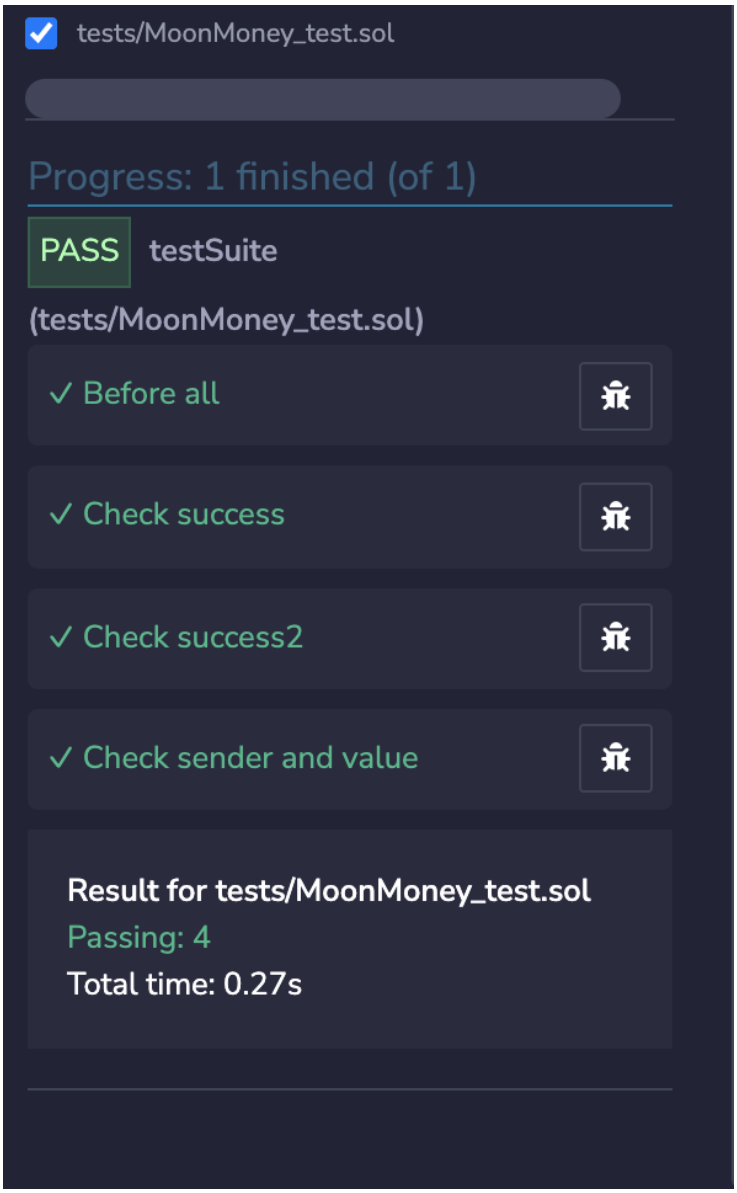
Testing proves:

1- Check for security

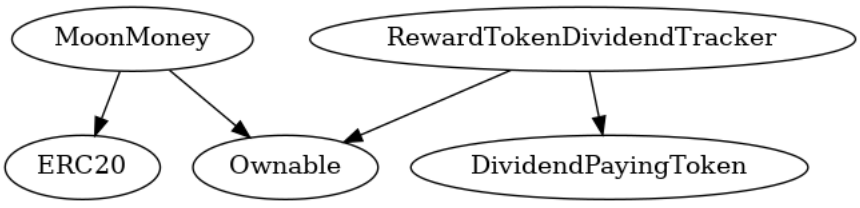


2- SOLIDITY STATIC ANALYSIS

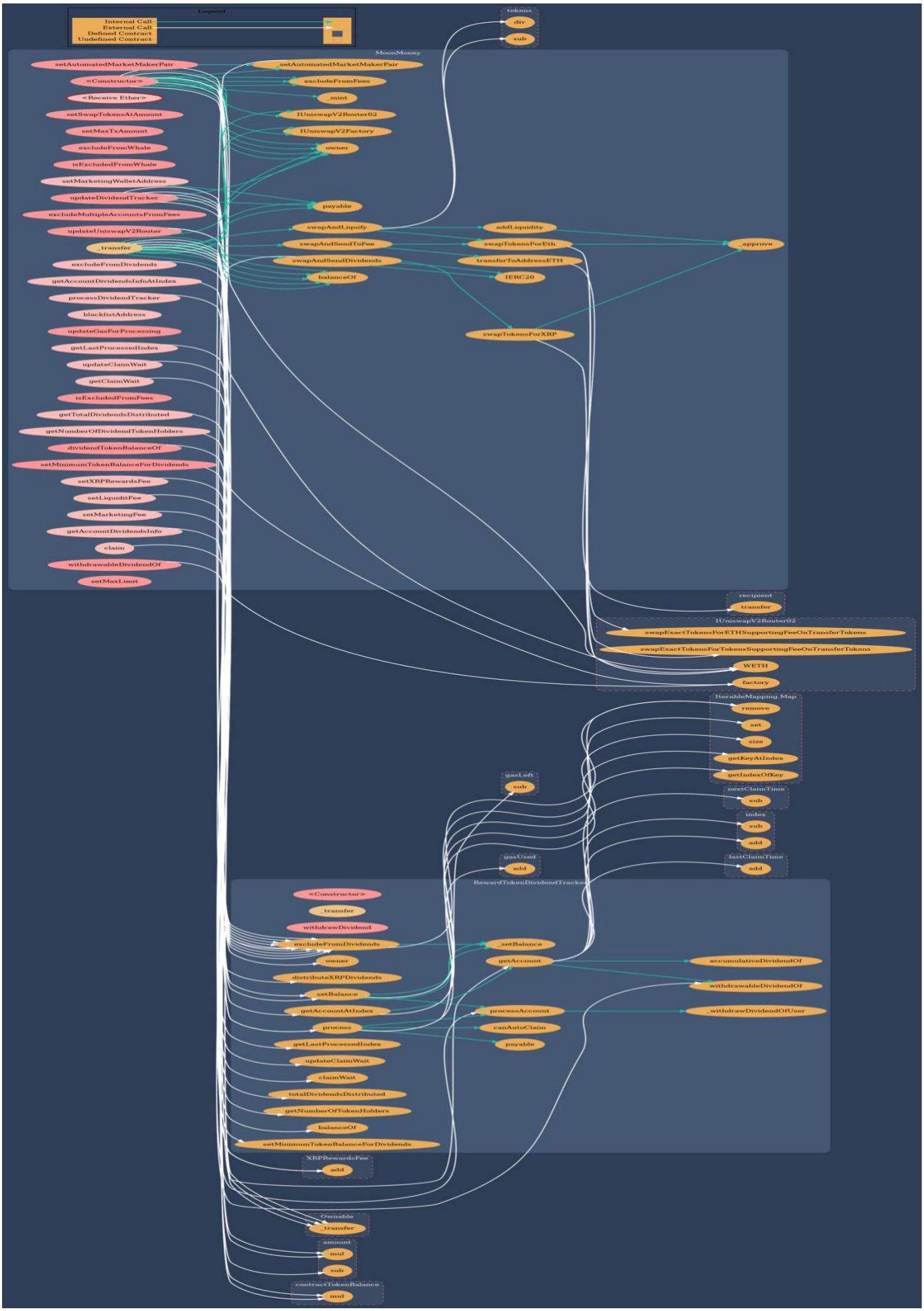
3- SOLIDITY UNIT TESTING



4- Inheritance graph



5- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
tokens
div
sub
MoonMoney
setAutomatedMarketMakerPair
_setAutomatedMarketMakerPair
<Constructor>
excludeFromFees
<Receive Ether>
_mint
setSwapTokensAtAmount
IUniswapV2Router02
setMaxTxAmount
IUniswapV2Factory
excludeFromWhale
owner
isExcludedFromWhale
setMarketingWalletAddress
updateDividendTracker
payable
excludeMultipleAccountsFromFees
updateUniswapV2Router
swapAndLiquify
addLiquidity
_transfer
swapAndSendToFee
swapTokensForEth
_approve
excludeFromDividends
swapAndSendDividends
transferToAddressETH
getAccountDividendsInfoAtIndex
balanceOf
IERC20
processDividendTracker
blacklistAddress
swapTokensForXRP
updateGasForProcessing
getLastProcessedIndex
updateClaimWait
getClaimWait
isExcludedFromFees
getTotalDividendsDistributed
getNumberOfDividendTokenHolders
dividendTokenBalanceOf
setMinimumTokenBalanceForDividends
setXRPRewardsFee
setLiquiditFee
setMarketingFee
getAccountDividendsInfo
claim
withdrawableDividendOf
setMaxLimit
recipient
transfer
IUniswapV2Router02
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
WETH
factory
IterableMapping.Map
remove
set
size
getKeyAtIndex
getIndexOfKey
gasLeft
sub
nextClaimTime
sub
index
sub
add
gasUsed
add
lastClaimTime
add
RewardTokenDividendTracker
<Constructor>
_transfer
withdrawDividend
excludeFromDividends
_setBalance
owner
getAccount
accumulativeDividendOf
distributeXRPDividends
withdrawableDividendOf
setBalance
getAccountAtIndex
processAccount
_withdrawDividendOfUser
process
canAutoClaim
getLastProcessedIndex
payable
updateClaimWait
claimWait
totalDividendsDistributed
getNumberOfTokenHolders
balanceOf
setMinimumTokenBalanceForDividends
XRPRewardsFee
add
Ownable
_transfer
amount
mul
sub
contractTokenBalance
mul

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/MoonMoney.sol	fe80d0029fbca8dfe7032c9f6e9d3d650f49d287

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
MoonMoney	Implementation	ERC20, Ownable		
└	<Constructor>	Public ❗	🛑	ERC20
└	setMinimumTokenBalanceForDividends	Public ❗	🛑	onlyOwner
└	transferToAddressETH	Private 🔐	🛑	
└	<Receive Ether>	External ❗	💵	NO❗
└	setSwapTokensAtAmount	Public ❗	🛑	onlyOwner
└	setMaxTxAmount	Public ❗	🛑	onlyOwner
└	excludeFromWhale	Public ❗	🛑	onlyOwner
└	isExcludedFromWhale	Public ❗		NO❗
└	updateDividendTracker	Public ❗	🛑	onlyOwner
└	updateUniswapV2Router	Public ❗	🛑	onlyOwner
└	excludeFromFees	Public ❗	🛑	onlyOwner
└	excludeMultipleAccountsFromFees	Public ❗	🛑	onlyOwner
└	setMarketingWalletAddress	External ❗	🛑	onlyOwner
└	setXRPRewardsFee	External ❗	🛑	onlyOwner
└	setLiquiditFee	External ❗	🛑	onlyOwner
└	setMarketingFee	External ❗	🛑	onlyOwner
└	setAutomatedMarketMakerPair	Public ❗	🛑	onlyOwner
└	blacklistAddress	External ❗	🛑	onlyOwner
└	_setAutomatedMarketMakerPair	Private 🔐	🛑	
└	updateGasForProcessing	Public ❗	🛑	onlyOwner
└	updateClaimWait	External ❗	🛑	onlyOwner
└	getClaimWait	External ❗		NO❗

└	getTotalDividendsDistributed	External ❗		NO❗
└	isExcludedFromFees	Public ❗		NO❗
└	withdrawableDividendOf	Public ❗		NO❗
└	dividendTokenBalanceOf	Public ❗		NO❗
└	excludeFromDividends	External ❗	🛑	onlyOwner
└	getAccountDividendsInfo	External ❗		NO❗
└	getAccountDividendsInfoAtIndex	External ❗		NO❗
└	processDividendTracker	External ❗	🛑	NO❗
└	claim	External ❗	🛑	NO❗
└	getLastProcessedIndex	External ❗		NO❗
└	getNumberOfDividendTokenHolders	External ❗		NO❗
└	_transfer	Internal 🔒	🛑	
└	setMaxLimit	Public ❗	🛑	onlyOwner
└	swapAndSendToFee	Private 🔐	🛑	
└	swapAndLiquify	Private 🔐	🛑	
└	swapTokensForEth	Private 🔐	🛑	
└	swapTokensForXRP	Private 🔐	🛑	
└	addLiquidity	Private 🔐	🛑	
└	swapAndSendDividends	Private 🔐	🛑	
RewardTokenDividendTracker	Implementation	Ownable, DividendPayingToken		
└	<Constructor>	Public ❗	🛑	DividendPayingToken
└	_transfer	Internal 🔒	🛑	
└	withdrawDividend	Public ❗	🛑	NO❗
└	excludeFromDividends	External ❗	🛑	onlyOwner
└	setMinimumTokenBalanceForDividends	Public ❗	🛑	onlyOwner
└	updateClaimWait	External ❗	🛑	onlyOwner
└	getLastProcessedIndex	External ❗		NO❗
└	getNumberOfTokenHolders	External ❗		NO❗
└	getAccount	Public ❗		NO❗
└	getAccountAtIndex	Public ❗		NO❗
└	canAutoClaim	Private 🔐		
└	setBalance	External ❗	🛑	onlyOwner
└	process	Public ❗	🛑	NO❗
└	processAccount	Public ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🔘	Function can modify state
💸	Function is payable

• Summary of the Audit

According to automatically test, the customer`s solidity smart contract is **Secured**.

The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

The test found 0 critical, 0 high, 0 medium, 4 low issues, and 2 notes.