SecureWise

AUDIT REPORT

Secure Wise

GEOLEAF (GLT)

GeoLeaf

SecureWise

Quick Result

Quick Result	Status
Owner can mint ?	**Not Detected**
Owner can update tax over 25% ?	**Not Detected**
Owner can pause trade ?	**Not Detected**
Owner can enable trading ?	**Not Detected**
Owner can add Blacklist ?	**Not Detected**
Owner can set Max Tx ?	**Not Detected**
Owner can set Max Wallet Amount ?	**Not Detected**
Ownership Status ?	**Not Renounced**
KYC ?	**Not Done**

Page 10 for more details

GeoLeaf (GLT) as PASSED the smart contract audit with LOW severity issue

SecureWise

Findings

Risk Classification	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, of the contract and its functions. Must be fixed as soon as possible.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.
Low	Effects are minimal in isolation and do not pose a signifcant danger to the project or its users. Issues under this classifcation are recommended to be fixed nonetheless.
Informational	A vulnerability that have informational character but is not effecting any of the code

Severity	Found	Pending	Resolved
High	0	0	0
Medium	0	0	0
Low	5	0	0
Informational	3	0	0
Total	**8**	**0**	**0**

Contents

Overview

Token Name: GeoLeaf **(GLT)**

Language: Solidity

Contract Address: 0x8E258fC9F8201C25ECbD3352b61456E6Db0987A4

Network: Grv Chain

Supply: 1000000000000000

KYC: Not done

Website: https://geoleaf.io/

Twitter: https://twitter.com/GeoLeafToken

Telegram: https://t.me/GeoLeaf

Report Date: June 15, 2023

Testnet:
https://testnet.bscscan.com/address/0x2D8130e0fD5Ad8B02FE29dFd4a7941fF3F3653F2

Auditing Approach and Methodologies

SecureWise has performed starting with analyzing the code, issues, code quality, and libraries. Reviewed line-by-line by our team. Finding any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

Methodology

- Understanding the size, scope and functionality of your project's source code
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Testing and automated analysis of the Smart Contract to determine proper logic has been followed throughout the whole process
- Deploying the code on testnet using multiple live test
- Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.
- Checking whether all the libraries used in the code are on the latest version.

Goals

Smart Contract System is secure, resilient and working according to the specifications and without any vulnerabilities.

Risk Classification

High: Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, of the contract and its functions. Must be fixed as soon as possible.

Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.

Low: Effects are minimal in isolation and do not pose a signifcant danger to the project or its users. Issues under this classifcation are recommended to be fixed nonetheless.

Informational: A vulnerability that have informational character but is not effecting any of the code

Findings Summary

SecureWise has applied the automated and manual analysis of Smart Contract and were reviewed for common contract vulnerabilities and centralized exploits

⚠️	Accounts can be excluded by the owner from receiving rewards.
⚠️	Owner can establish fees with a maximum limit of 10%.
⚠️	Buyback BNB is going to an externally owned account
⚠️	Swap settings can be modified by the owner.
⚠️	Owner is capable of withdrawing tokens excluding the native token from the contract.

Page 10 for more details

GeoLeaf (GLT) as PASSED the smart contract audit with LOW severity issue

Function Privileges

```
| **GLT** | Implementation | ERC20, Ownable |||
| └ | <Constructor> | Public ! |    🔲 | ERC20 |
| └ | <Receive Ether> | External ! |    🔲 |NO! |
| └ | claimStuckTokens | External ! | 🔴 | onlyOwner |
| └ | isContract | Internal 🔒 | | |
| └ | sendBNB | Internal 🔒 | 🔴 | |
| └ | _setAutomatedMarketMakerPair | Private 🔓 | 🔴 | |
| └ | excludeFromFees | External ! | 🔴 | onlyOwner |
| └ | isExcludedFromFees | Public ! | |NO! |
| └ | updateBuyFees | External ! | 🔴 | onlyOwner |
| └ | updateSellFees | External ! | 🔴 | onlyOwner |
| └ | changeTreasuryWallet | External ! | 🔴 | onlyOwner |
| └ | _transfer | Internal 🔒 | 🔴 | |
| └ | swapAndLiquify | Private 🔓 | 🔴 | |
| └ | swapAndSendDividends | Private 🔓 | 🔴 | |
| └ | buyBackAndBurn | Private 🔓 | 🔴 | |
| └ | setSwapTokensAtAmount | External ! | 🔴 | onlyOwner |
| └ | updateGasForProcessing | Public ! | 🔴 | onlyOwner |
| └ | updateMinimumBalanceForDividends | External ! | 🔴 | onlyOwner |
| └ | updateClaimWait | External ! | 🔴 | onlyOwner |
| └ | getClaimWait | External ! | |NO! |
| └ | getTotalDividendsDistributed | External ! | |NO! |
| └ | withdrawableDividendOf | Public ! | |NO! |
| └ | dividendTokenBalanceOf | Public ! | |NO! |
| └ | totalRewardsEarned | Public ! | |NO! |
| └ | excludeFromDividends | External ! | 🔴 | onlyOwner |
| └ | getAccountDividendsInfo | External ! | |NO! |
| └ | getAccountDividendsInfoAtIndex | External ! | |NO! |
| └ | processDividendTracker | External ! | 🔴 |NO! |
| └ | claim | External ! | 🔴 |NO! |
| └ | claimAddress | External ! | 🔴 | onlyOwner |
| └ | getLastProcessedIndex | External ! | |NO! |
| └ | setLastProcessedIndex | External ! | 🔴 | onlyOwner |
| └ | getNumberOfDividendTokenHolders | External ! | |NO! |
```
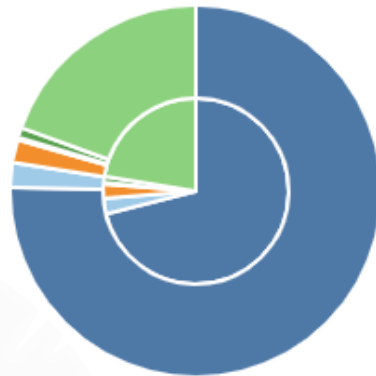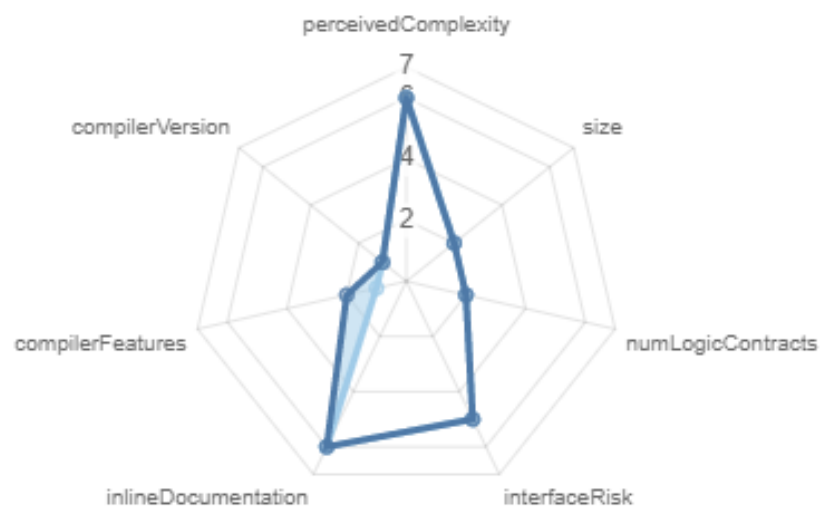
Inheritance Graph
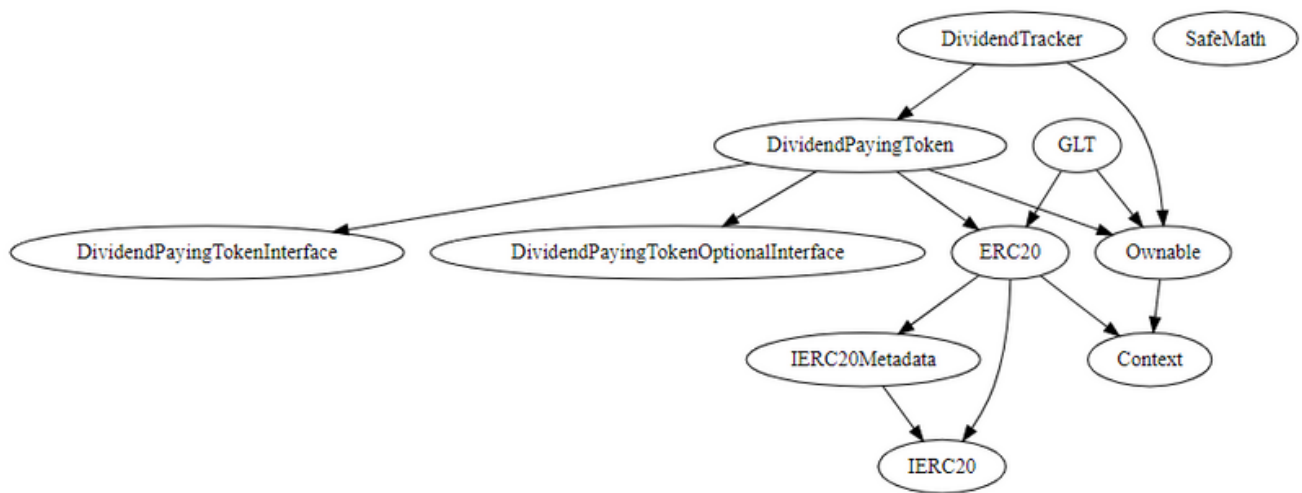
Source Lines

Risk

Manual Review

Low Risk

Accounts can be excluded by the owner from receiving rewards.

```
function excludeFromDividends(address account) external onlyOwner {
    require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

Description

excludeFromDividends function, designed to exclude a specified account from receiving dividends. if sets the **excludedFromDividends** flag to **true**. Additionally, it resets the account's balance to zero and emits an **ExcludeFromDividends** event to notify relevant parties.

Recommendation

While assuming input validation occurs before invoking this function, it is advisable to explicitly validate the account parameter to ensure its conformity as a valid address. Implementing error-handling mechanisms to gracefully manage potential exceptions is also recommended. Ensure that appropriate access control mechanisms are in place to restrict the excludeFromDividends function to only be called by the contract owner

Manual Review

Low Risk

Owner can establish fees with a maximum limit of 10%.

```solidity
function updateBuyFees(uint256 _liquidityFeeOnBuy, uint256 _treasuryFeeOnBuy, uint256 _rewardsFeeOnBuy, uint256 _buybackFeeOnBuy) external onlyOwner {
    liquidityFeeOnBuy  = _liquidityFeeOnBuy;
    treasuryFeeOnBuy   = _treasuryFeeOnBuy;
    rewardsFeeOnBuy    = _rewardsFeeOnBuy;
    buybackFeeOnBuy    = _buybackFeeOnBuy;

    totalBuyFee = _liquidityFeeOnBuy + _treasuryFeeOnBuy + _rewardsFeeOnBuy + _buybackFeeOnBuy;

    require(totalBuyFee <= 10, "Buy fee cannot be more than 10%");

    emit BuyFeesUpdated(totalBuyFee);
}

function updateSellFees(uint256 _liquidityFeeOnSell, uint256 _treasuryFeeOnSell, uint256 _rewardsFeeOnSell, uint256 _buybackFeeOnSell) external onlyOwner {
    liquidityFeeOnSell  = _liquidityFeeOnSell;
    treasuryFeeOnSell   = _treasuryFeeOnSell;
    rewardsFeeOnSell    = _rewardsFeeOnSell;
    buybackFeeOnSell    = _buybackFeeOnSell;

    totalSellFee = _liquidityFeeOnSell + _treasuryFeeOnSell + _rewardsFeeOnSell + _buybackFeeOnSell;

    require(totalSellFee <= 10, "Sell fee cannot be more than 10%");

    emit SellFeesUpdated(totalSellFee);
}
```

Description

updateBuyFees and *updateSellFees*. These functions allow the contract owner to update the fee values associated with buying and selling tokens. Additionally, the functions calculate the total fees by summing up the individual fee components and assign the result to *totalBuyFee* and *totalSellFee*. Finally, the functions check if the total fees are less than or equal to *10%* and emit events to indicate the fee updates.

Recommendation

Validate the inputs provided for the fee values to ensure they are within acceptable ranges and prevent any potential errors or unintended consequences. Ensure that appropriate access control mechanisms are in place to restrict these functions to only be called by the contract owner

Manual Review

Low Risk

Buyback BNB is going to an EOA

```
function buyBackAndBurn(uint256 amount) private{
    address[] memory path = new address[](2);
    path[0] = uniswapV2Router.WETH();
    path[1] = rewardToken;

    uint256 half = amount / 2;
    uint256 otherHalf = amount - half;

    uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: half}(
        0,
        path,
        address(DEAD), //ex DEAD
        block.timestamp
    );

//      path[0] = uniswapV2Router.WETH();
//      path[1] = address(this);
//
//      uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: otherHalf}(
//          0,
//          path,
//          address(DEAD), //ex DEAD
//          block.timestamp
//      );

    payable(otherWallet).transfer(otherHalf); //buybback amount eoa address
}
```

Description

buyBackAndBurn *function facilitates the process of buying back a specified amount of tokens from a decentralized exchange (Pancake,Uniswap) using a portion of the contract's balance in Ether (ETH, BNB), and subsequently burning those tokens. Additionally, the function transfers the remaining Ether balance to another EOA wallet address.*

Recommendation

*The second half of the input amount (**otherHalf**) is transferred to the designated **otherWallet** address. This transfer operation appears to be sending the remaining ETH balance to an external wallet address, possibly an externally controlled account (**EOA**). Ensure that the **otherWallet** address is properly verified and validated before transferring the ETH balance. Confirm that it belongs to the intended recipient and is not a malicious or untrusted address.*

Manual Review

Low Risk

Swap settings can be modified by the owner.

```
function setSwapTokensAtAmount(uint256 newAmount) external onlyOwner{
    require(newAmount > totalSupply() / 100_000, "SwapTokensAtAmount must be greater than 0.001% of total supply");
    swapTokensAtAmount = newAmount;
}
```

Description

setSwapTokensAtAmount function updates the *swapTokensAtAmount* variable to the provided value, which represents the minimum amount of tokens required for a swap to occur.

Recommendation

*Validate the input values provided to ensure they conform to any specific constraints or requirements. For example, ensure that the **newAmount** provided in **setSwapTokensAtAmount** is within acceptable ranges and aligned with the tokenomics of the project. Verify that appropriate access control mechanisms are in place to restrict these functions to only be called by the contract owner*

SecureWise

Manual Review

Low Risk

Owner is capable of withdrawing tokens excluding the native token from the contract.

```
function claimStuckTokens(address token) external onlyOwner {
    require(token != address(this), "Owner cannot claim native tokens");
    if (token == address(0x0)) {
        payable(msg.sender).transfer(address(this).balance);
        return;
    }
    IERC20 ERC20token = IERC20(token);
    uint256 balance = ERC20token.balanceOf(address(this));
    ERC20token.transfer(msg.sender, balance);
}
```

Description

claimStuckTokens *that allows the contract owner to claim tokens that may have become stuck in the contract. For native tokens, if the provided token address is the same as the contract address (address(this)), an error message is returned since the **owner cannot claim native tokens.***

Recommendation

Verify that appropriate access control mechanisms are in place to restrict this function to only be called by the contract owner. Consider adding additional error handling mechanisms to handle exceptional cases, such as when the provided token address is invalid or the transfer of tokens fails. This will provide better feedback and help identify any issues during the token claiming process.

Manual Review

Informational

Owner has the ability to exclude accounts from being charged fees.

```
function excludeFromFees(address account, bool excluded) external onlyOwner {
    require(_isExcludedFromFees[account] != excluded, "Account is already set to that state");
    _isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded);
}
```

Description

excludeFromFees, which allows the contract owner to exclude a specified account from incurring fees. Function checks if the account is not already excluded from fees by verifying the *_isExcludedFromFees* mapping. If the account is not already excluded, the mapping is updated to mark the account as excluded. Lastly, the function emits an ***ExcludeFromFees*** event to indicate the successful exclusion.

Recommendation

Although it is assumed that input validation is performed before this function is called, it is recommended to explicitly validate the account parameter to ensure it is a valid address and to handle potential errors gracefully. Ensure that appropriate access control mechanisms are in place to restrict the excludeFromFees function to only be called by the contract owner

Manual Review

Informational

Missing events arithmetic

claimStuckTokens(...)
setSwapTokensAtAmount(...)
setLastProcessedIndex(...)

Description

Events play a crucial role in providing transparency and facilitating the tracking of important contract actions.

Recommendation

It is highly recommended to include appropriate event declarations for each of the mentioned functions. Events allow external systems and users to monitor and respond to specific contract events. Ensure that the event declarations include relevant information about the action performed, such as addresses involved, token amounts, or status changes. This will improve the overall transparency and usability of the smart contract.

Manual Review

Informational

Old Version of Solidity Compiler and Floating pragma

pragma solidity ^0.8.17;

Description

Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statement. Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Recommendation

Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing. Lock the pragma version and also consider known bugs or the compiler version that is chosen.

Disclaimer

SecureWise provides the smart contract audit of solidity. Audit and report are for informational purposes only and not, nor should be considered, as an endorsement to engage with, invest in, participate, provide an incentive, or disapprove, criticise, discourage, or purport to provide an opinion on any particular project or team.

This audit report doesn't provide any warranty or guarantee regarding the nature of the technology analysed. These reports, in no way, provide investment advice, nor should be used as investment advice of any sort. Investors must always do their own research and manage their risk.

DISCLAIMER: 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 SecureWise and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) SecureWise owe no duty of care towards you or any other person, nor does SecureWise 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 SecureWise 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, SecureWise hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SecureWise, 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.

AUDIT
REPORT

Secure Wise