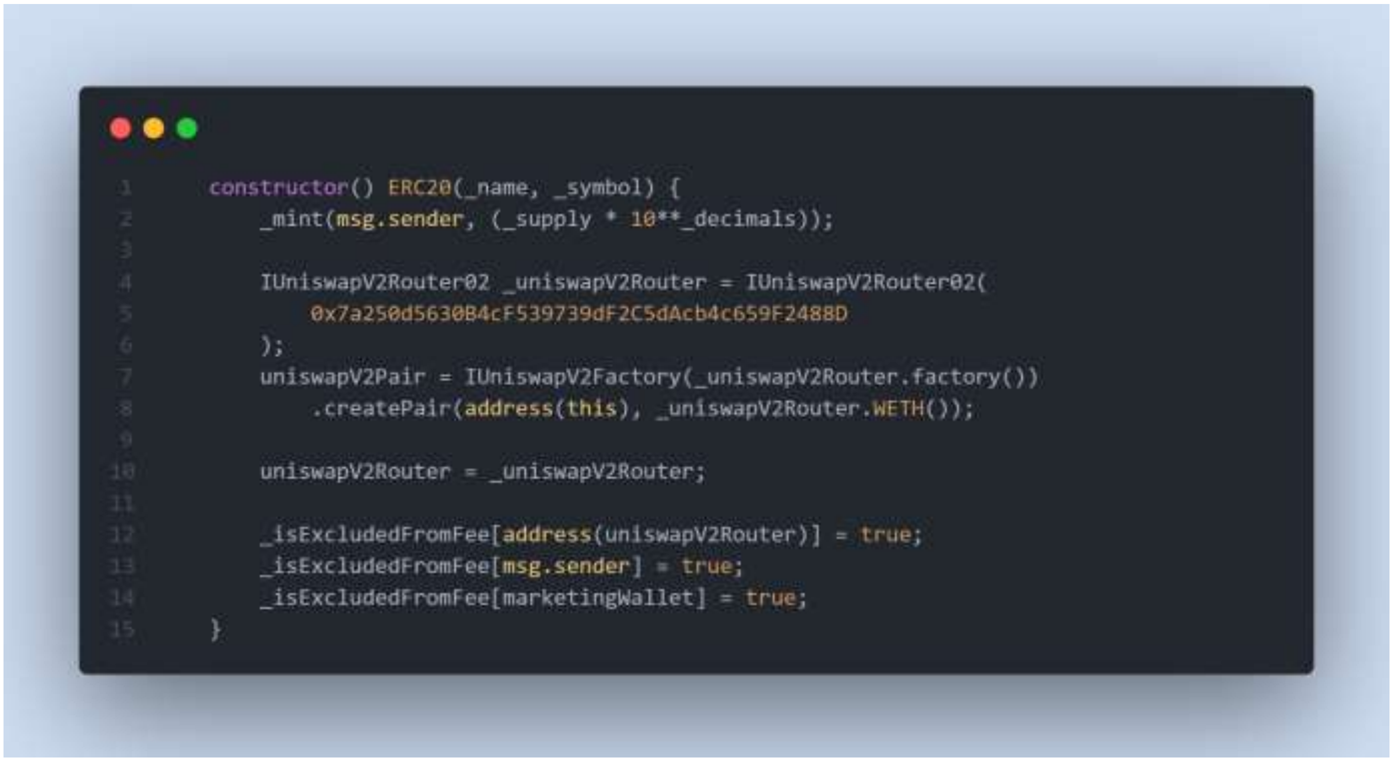
etherscan.io/token/0xf9a7e47f6d00a75be08bb23a804f4153c01a1636#code

Medium severity issues

• Initial Token Distribution / Centralization

In the constructor function, the deployer is minting tokens for his wallet address. All the initial supply are sent to the contract deployer when deploying the contract. This could be a centralization risk, the deployer can distribute those tokens without obtaining the consensus of the community.



```
    constructor() ERC20(_name, _symbol) {
        _mint(msg.sender, (_supply * 10**_decimals));

        IUniswapV2Router02 _uniswapV2Router = IUniswapV2Router02(
            0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D
        );
        uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;

        _isExcludedFromFee[address(uniswapV2Router)] = true;
        _isExcludedFromFee[msg.sender] = true;
        _isExcludedFromFee[marketingWallet] = true;
    }
```

✓ Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key, We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage the specific account in this case.

• Owner Role / Centralized Risk

Only the owner role has authority over the functions shown below. Any compromise to the _owner account may allow the hacker to take advantage of this authority.

- ✓ changeMaxTxAmount
- ✓ changeMaxWalletAmount
- ✓ changeMarketingWallet
- ✓ changeTaxForLiquidityAndMarketing

✓ Recommendation

The risk describes the current project design and potentially makes iterations to improve the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Low severity issues

• Third Party Dependency

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

- ✓ uniswapV2Router

✓ Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

• Missing Zero Address Validation

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

- ✓ changeMarketingWallet

✓ Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ changeMaxTxAmount
- ✓ changeMaxWalletAmount
- ✓ changeMarketingWallet
- ✓ changeTaxForLiquidityAndMarketing

✓ Recommendation

It is recommended emitting events for sensitive functions .

• Improper Usage of public and external Type

public functions that are never called by the contract could be declared as external. external functions are more efficient than public functions.

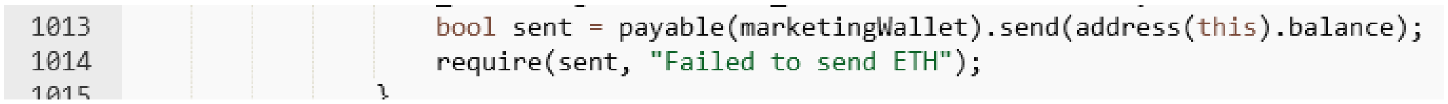
- ✓ changeMaxTxAmount
- ✓ changeMaxWalletAmount
- ✓ changeMarketingWallet
- ✓ changeTaxForLiquidityAndMarketing

✓ Recommendation

Consider using the external attribute for public functions that are never called within the contract.

• Usage of transfer/send for sending Ether

It is not recommended to use Solidity's transfer() and send() functions for transferring Ether, since some contracts may not be able to receive the funds. Those functions forward only a fixed amount of gas (2300 specifically) and the receiving contracts may run out of gas before finishing the transfer. Also, EVM instructions' gas costs may increase in the future. Thus, some contracts that can receive now may stop working in the future due to the gas limitation.



```
                bool sent = payable(marketingWallet).send(address(this).balance);
                require(sent, "Failed to send ETH");
            }
```

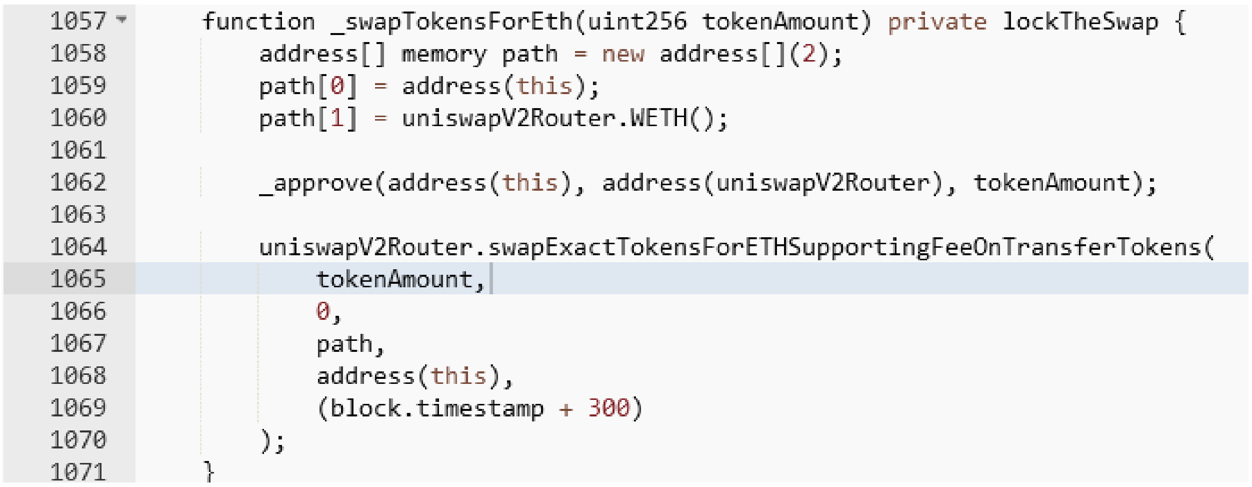
✓ Recommendation

We recommend using the Address.sendValue() function from OpenZeppelin. Since Address.sendValue() may allow reentrancy, we also recommend guarding against reentrancy attacks by utilizing the Checks-Effects-Interactions Pattern or applying OpenZeppelin ReentrancyGuard.

• Potential Sandwich Attacks

A sandwich attack might happen when an attacker observes a transaction swapping tokens or adding liquidity without setting restrictions on slippage or minimum output amount. The attacker can manipulate the exchange rate by frontrunning (before the transaction is attacked) a transaction to purchase one of the assets and make profits by back running (after the transaction is attacked) a transaction to sell the asset after the transaction is attacked.

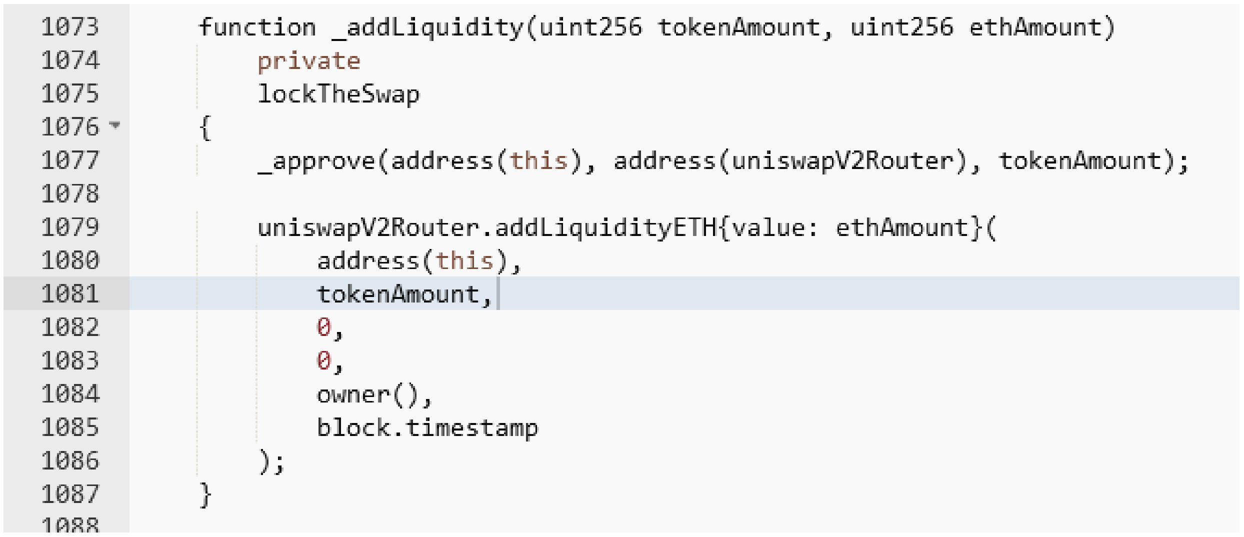
The following functions are called without setting restrictions on slippage or minimum output amount, so transactions triggering these functions are vulnerable to sandwich attacks, especially when the input amount is large:



```
    function _swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router), tokenAmount);

        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0,
            path,
            address(this),
            (block.timestamp + 300)
        );
    }
```



```
    function _addLiquidity(uint256 tokenAmount, uint256 ethAmount)
        private
        lockTheSwap
    {
        _approve(address(this), address(uniswapV2Router), tokenAmount);

        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0,
            0,
            owner(),
            block.timestamp
        );
    }
```

✓ Recommendation

We recommend setting reasonable minimum output amounts, instead of 0, based on token prices when calling the mentioned functions.