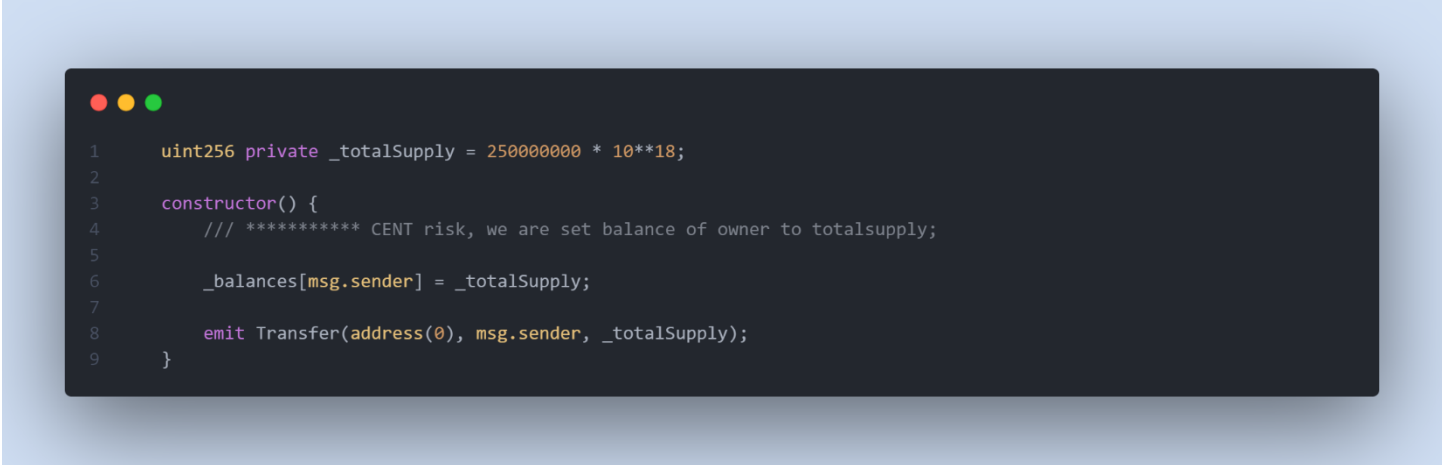

Source : <https://github.com/OrcaniaEdyRepo/Orcania-DEX/>

OCA.sol

Medium severity issues

- **CENTRALIZATION RISKS**

In the constructor function, we set the deployer wallet balance to 250,000,000 tokens. This could be a centralization risk as the deployer can distribute tokens without obtaining the consensus of the community.



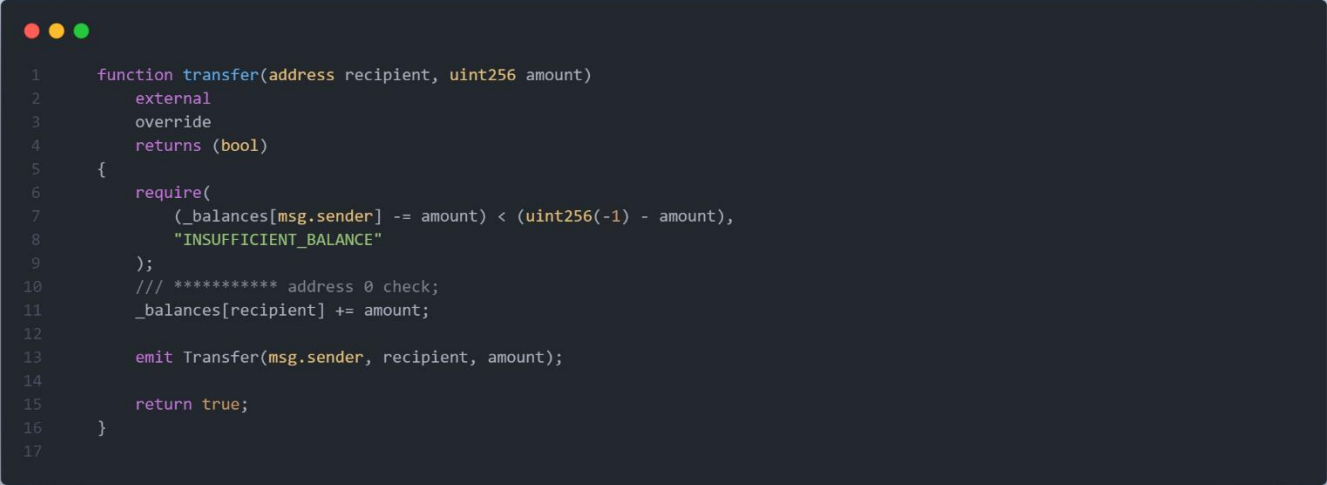
```
1  uint256 private _totalSupply = 250000000 * 10**18;
2
3  constructor() {
4      /// ***** CENT risk, we are set balance of owner to totalsupply;
5
6      _balances[msg.sender] = _totalSupply;
7
8      emit Transfer(address(0), msg.sender, _totalSupply);
9  }
```

Remediation

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.

low severity issues

- ZERO ADDRESS VALIDATION CHECK



```
1  function transfer(address recipient, uint256 amount)
2      external
3      override
4      returns (bool)
5  {
6      require(
7          (_balances[msg.sender] -= amount) < (uint256(-1) - amount),
8          "INSUFFICIENT_BALANCE"
9      );
10     /// ***** address 0 check;
11     _balances[recipient] += amount;
12
13     emit Transfer(msg.sender, recipient, amount);
14
15     return true;
16 }
17
```

In the transfer function, we accept the recipient, as an argument. it's important to check the value of this address before transferring any token to it.

Remediation

We need to check the recipient address, it should not be a zero address.

DEX.sol

High severity issues

- **MISSING CHECK FOR REENTRANCY ATTACK**

In the multiple functions, we used `.call` to send ether .

reentrancy attack happens when a contract send ether to an unknown address. Then, an attacker could write a contract in an external address which includes malicious code in the fallback function. Thus, when a contract send ether to that address, it will invoke the malicious code. The malicious code usually executes a function in the vulnerable contract and performs non-expected operations by developers.

```
1     function sendValue(address payable recipient, uint256 amount) internal {
2         require(address(this).balance >= amount, "INSUFFICIENT_BALANCE");
3
4         (bool success, ) = recipient.call{value: amount}("");
5         require(success, "UNABLE_TO_SEND_VALUE RECIPIENT_MAY_HAVE_REVERTED");
6     }
```

Remediation

We recommend applying OpenZeppelin ReentrancyGuard library `nonReentrant` modifier for the above functions.

Medium severity issues

- **UNCHECKED RETURNED VALUE**

we used OCA.transfer multiple times to transfer the OCA token, but we don't check the return value from this function.



```
1
2     function swapTokenForOCA(
3         address tokenIn,
4         uint256 amountIn,
5         uint256 minAmountOut,
6         uint256 deadline
7     ) external {
8         //require that time now is lower than deadline provided;
9         require(block.timestamp < deadline, "OUT_OF_TIME");
10
11         // now we transfer token amount from user to this address and we check returned value;
12         require(
13             IERC20(tokenIn).transferFrom(msg.sender, address(this), amountIn),
14             "FAILED_TOKEN_TRANSFER"
15         );
16
17         // get amount to get out from internal function;
18         uint256 amountOut = SwapTokenForOCA(tokenIn, amountIn);
19
20         // check amount out is more than min amount out;
21         require(amountOut >= minAmountOut, "INSUFFICIENT_OUTPUT_AMOUNT");
22
23         /// ***** we are use standard transfer method, we should check returned value ***** ///
24         OCA.transfer(msg.sender, amountOut);
25     }
```

Remediation

We recommend checking returned value from this function to be true;