
# Project: Nexus

**0xc01154b4ccb518232d6bbfc9b9e6c5068b766f82**

**11/03/2024**

# AUDIT REPORT

**SAFETY SCORE: 84**

## 1 - Arbitrary Jump/Storage Write

**Result: Pass**

## 2 - Centralization of Control

**Result: Medium**

Details: The contract has an Ownable pattern, which gives the owner special privileges that can affect the contract's behavior. The

owner can exclude addresses from fees, set fees, and enable or disable trading. This centralization could be a risk if the owner's

account is compromised or if the owner acts maliciously.

Code:

```
function excludeFromFees(address account, bool excluded)
public onlyOwner {

_isExcludedFromFees[account] = excluded;

emit ExcludeFromFees(account, excluded);

}
```

Correction:

```
// No direct correction but decentralizing control through a
DAO or multi-sig can mitigate risks.
```

## 3 - Compiler Issues

**Result: Pass**

## 4 - Delegate Call to Untrusted Contract

**Result: Pass**

## 5 - Dependence on Predictable Variables

**Result: Pass**

## 6 - Ether/Token Theft

**Result: Medium**

Details: The transferForeignToken and withdrawStuckETH functions allow the TreasuryAddress to withdraw any ERC20 tokens or

ETH sent to the contract. This could be a risk if the TreasuryAddress is compromised.

Code:

```
function transferForeignToken(address _token, address _to)
public returns (bool _sent) {

require(msg.sender==TreasuryAddress,"only TreasuryAddress can
withdraw");

...

}


function withdrawStuckETH() public {

bool success;

require(msg.sender==TreasuryAddress,"only TreasuryAddress can
withdraw");

...

}
```

Correction:

```
// Implement additional checks or a multi-sig requirement for
withdrawals.
```

## 7 - Flash Loans

**Result: Pass**

## 8 - Front Running

Result: Low

Details: The contract is susceptible to front-running attacks because it interacts with DEXes without any specific anti-front-running

measures like using tx.origin for comparisons or applying a commit-reveal scheme.

Code:

```
function _transfer(address from, address to, uint256 amount) internal override {

...

if(canSwap && swapEnabled && !swapping && !automatedMarketMakerPairs[from] &&

!_isExcludedFromFees[from] && !_isExcludedFromFees[to]) {

swapping = true;

swapBack();

swapping = false;

}

...

}
```

Correction:

```
// Implement commit-reveal scheme or other anti-front-running measures.
```

## 9 - Improper Events

**Result: Pass**

10 - Improper Authorization Scheme

**Result: Medium**

Details: The contract uses a simple onlyOwner modifier for critical functions, which could lead to a single point of failure. A more

robust authorization scheme like multi-sig could improve security.

Code:

```
modifier onlyOwner() {

require(_owner == _msgSender(), "Ownable: caller is not the owner");

_;

}
```

Correction:

```
// Implement a multi-sig mechanism for critical function calls.
```

11 - Integer Over/Underflow

**Result: Pass**

12 - Logical Issues

**Result: Pass**

13 - Oracle Issues

**Result: Pass**

14 - Outdated Compiler Version

**Result: Pass**

15 - Race Conditions

**Result: Pass**

16 - Reentrancy

**Result: Pass**

17 - Signature Issues

**Result: Pass**

18 - Sybil Attack

**Result: Pass**

19 - Unbounded Loops

**Result: Pass**

20 - Unused Code

Result: Informational

Details: The _msgData function in the Context contract is never used, which leads to dead code and can be removed for gas

optimization.

Code:

```
function _msgData() internal view virtual returns (bytes calldata) {

this; // Dead code as 'this' is only used to suppress a warning.

return msg.data;

}
```

Correction:

```
// Simply remove the unused function.
```