


High severity issues

• 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. And in the constructor, the deployer wallet is excluded from paying any fee.

- ✓ `TaxLiquifyManual`
- ✓ `setTaxLiquifyEnabled`
- ✓ `runDropAuthority`
- ✓ `setNumTokensBeforeSwap`
- ✓ `setWalletLimit`
- ✓ `setIsWalletLimitExempt`
- ✓ `enableDisableWalletLimit`
- ✓ `setTaxWorked`
- ✓ `setMaxTxAmount`
- ✓ `setTaxes`
- ✓ `setAllowThis`
- ✓ `setIsExcludedFromFee`
- ✓ `setIsTxLimitExempt`
- ✓ `setMarketPairStatus`
- ✓ `SetPinkAntiBot`



```
1      isExcludedFromFee[msg.sender] = true;
2      isExcludedFromFee[address(this)] = true;
3
4      isWalletLimitExempt[msg.sender] = true;
5      isWalletLimitExempt[address(this)] =
true;
6
7      isTxLimitExempt[msg.sender] = true;
8      isTxLimitExempt[address(this)] = true;
```

✓ 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

• Missing Emit Events

There should always be events emitted in the sensitive functions.

- ✓ TaxLiquifyManual
- ✓ setTaxLiquifyEnabled
- ✓ runDropAuthority
- ✓ setNumTokensBeforeSwap
- ✓ setWalletLimit
- ✓ setIsWalletLimitExempt
- ✓ enableDisableWalletLimit
- ✓ setTaxWorked
- ✓ setMaxTxAmount
- ✓ setTaxes
- ✓ setAllowThis
- ✓ setIsExcludedFromFee
- ✓ setIsTxLimitExempt
- ✓ setMarketPairStatus
- ✓ SetPinkAntiBot

✓ Recommendation

It is recommended emitting events for sensitive functions.

• Unlocked Pragma Used

Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

```
// bad
pragma solidity ^0.4.4;

// good
pragma solidity 0.4.4;
```

✓ **Recommendation**

Should lock pragmas to a specific compiler version.