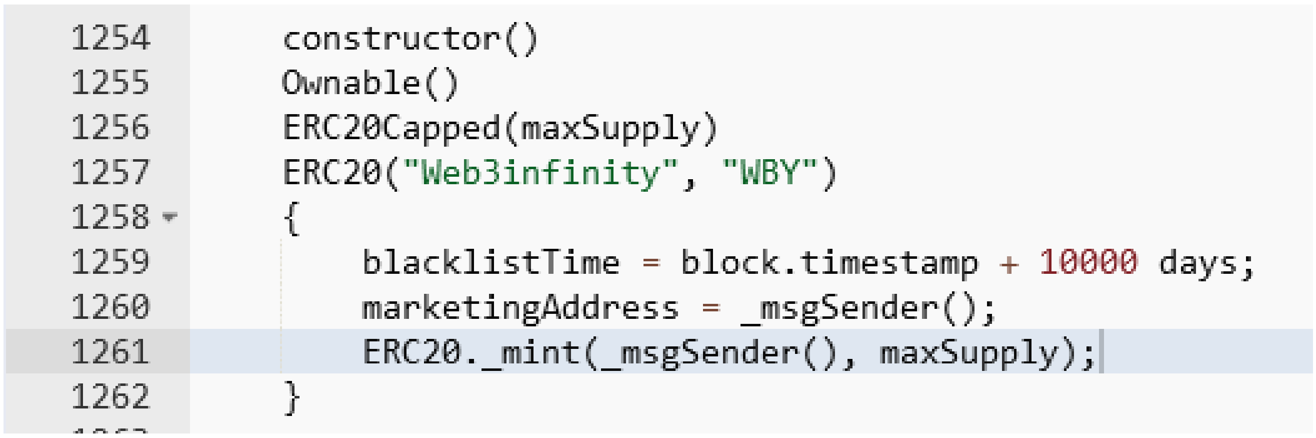
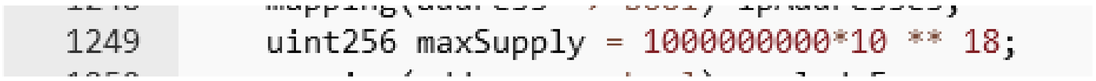
Medium severity issues

• Initial Token Distribution / Centralization

In the constructor function, the deployer is minting / transferring tokens/tokens for marketing for his wallet address. All the initial supplies and tokens for marketing 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()
    Ownable()
    ERC20Capped(maxSupply)
    ERC20("Web3infinity", "WBY")
    {
        blacklistTime = block.timestamp + 10000 days;
        marketingAddress = _msgSender();
        ERC20._mint(_msgSender(), maxSupply);
    }
```



```
    uint256 maxSupply = 1000000000*10 ** 18;
```

✓ 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.

- ✓ setFee
- ✓ setExcludeFee
- ✓ setMarketingAddress
- ✓ multiBlacklist
- ✓ multiRemoveFromBlacklist
- ✓ wiToken
- ✓ wiBNB
- ✓ wiAmountBNB
- ✓ wiAmountToken

✓ 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.

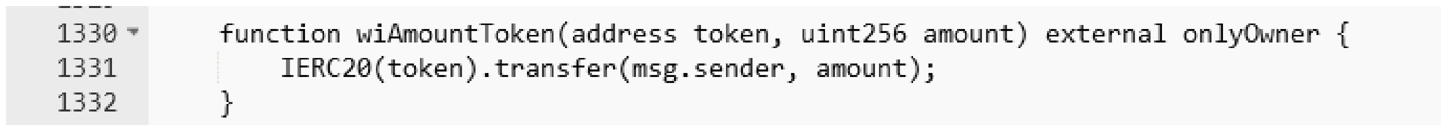
- ✓ setFee
- ✓ setExcludeFee
- ✓ setMarketingAddress
- ✓ multiBlacklist
- ✓ multiRemoveFromBlacklist
- ✓ wiToken
- ✓ wiBNB
- ✓ wiAmountBNB
- ✓ wiAmountToken

✓ Recommendation

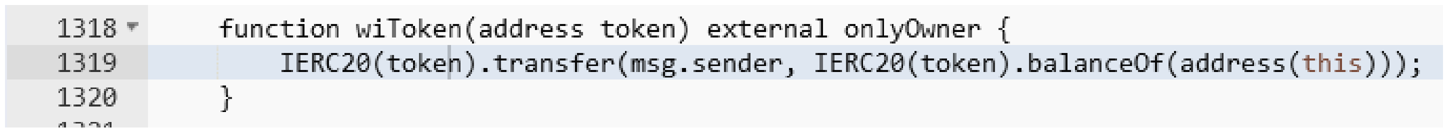
It is recommended emitting events for the sensitive functions.

• Unchecked Value of ERC-20 transfer()

the external call to transfer of ERC20 contracts and the return value is not checked in either case.



```
function wiAmountToken(address token, uint256 amount) external onlyOwner {
    IERC20(token).transfer(msg.sender, amount);
}
```



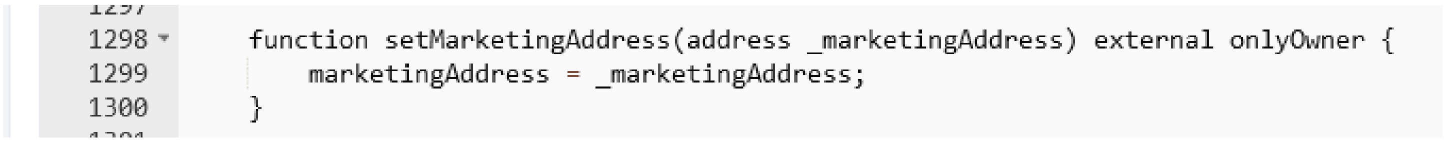
```
function wiToken(address token) external onlyOwner {
    IERC20(token).transfer(msg.sender, IERC20(token).balanceOf(address(this)));
}
```

✓ Recommendation

We advise the team to use SafeERC20 or make sure that the value returned from transfer()' is checked.

• Missing Zero Address Validation

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



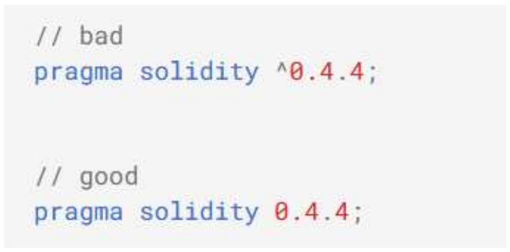
```
    function setMarketingAddress(address _marketingAddress) external onlyOwner {
        marketingAddress = _marketingAddress;
    }
```

✓ Recommendation

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

• 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.