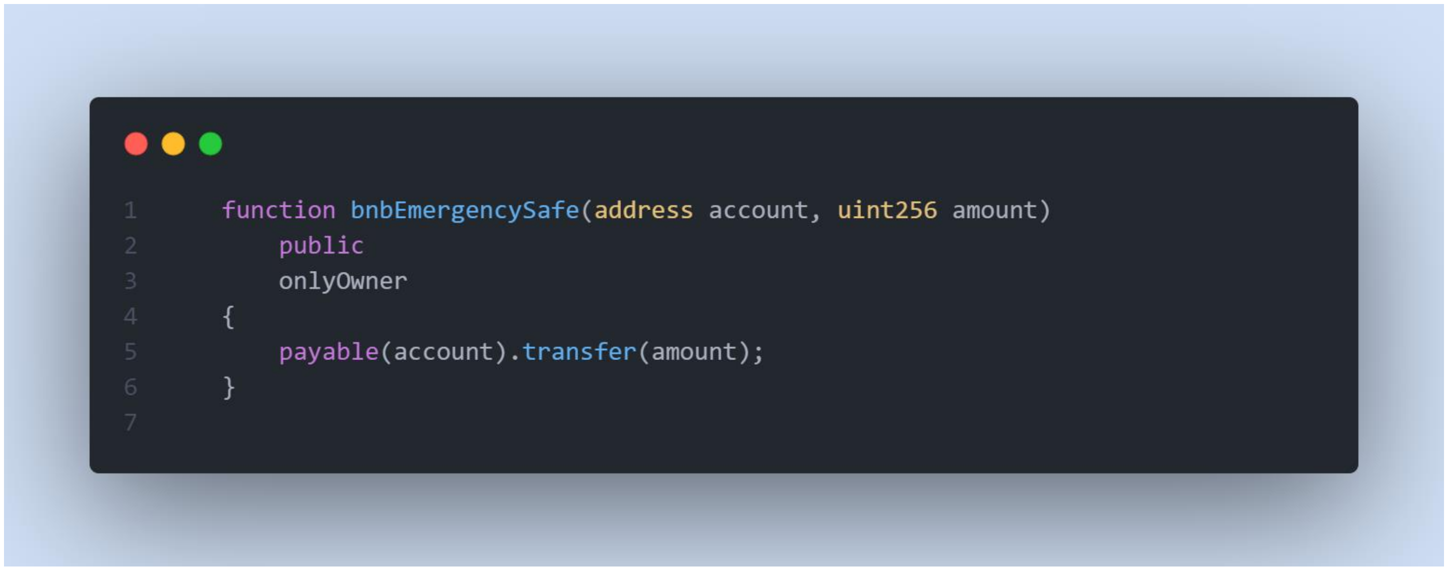
High severity issues

- **MISSING CHECK FOR REENTRANCY ATTACK**

In the function bnbEmergencySafe, we transfer BNB to an external account or contract! but we are not protecting this function from reentrancy attack.

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.



```
    function bnbEmergencySafe(address account, uint256 amount)
        public
        onlyOwner
    {
        payable(account).transfer(amount);
    }

```

Remediation

We recommend applying OpenZeppelin ReentrancyGuard library nonReentrant modifier for the bnbEmergencySafe function to prevent reentrancy attacks.

• Centralization Risks

In the contract, the owner has authority over the functions shown below.

mint(address to, uint256 amount) ; The owner can mint tokens to any address.

burn(address user, uint256 amount) ; The owner can burn tokens from any address.

multiBurn(burnStore[] memory vars) ; The owner can burn tokens from multiple addresses.

taxFeeUpdate(uint256 amount) ; The owner can update tax fee and set it to 99 !

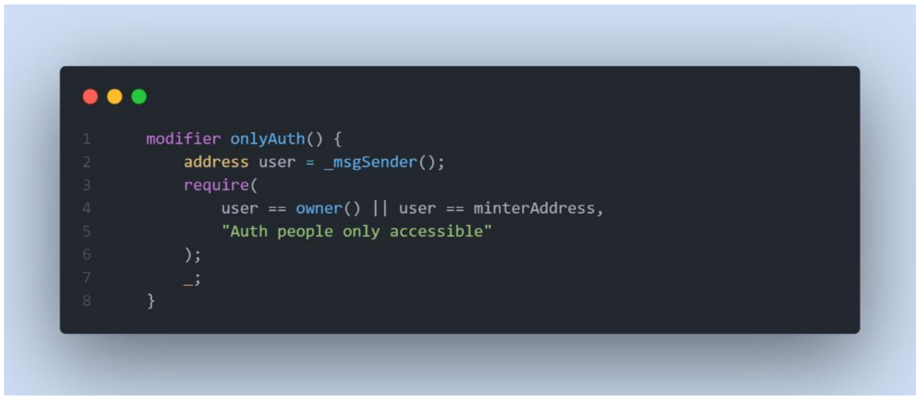
burnFeeUpdate(uint256 amount) ; The owner can update the burn fee and set it to 99!

feeStatusUpdate(bool status) ; The owner can set fee status to false and do his own transactions without paying any fee.

excludeFromFee(address account, bool status) ; The owner can exclude any address from the fee.

taxFeeAddressUpdate(address account) ; The owner can set the tax fee address to any address, even his personal wallet.

minterAddressUpdate(address account); The owner can set token minter to any address.



```
modifier onlyAuth() {
    address user = _msgSender();
    require(
        user == owner() || user == minterAddress,
        "Auth people only accessible"
    );
    _;
}
```

Remediation

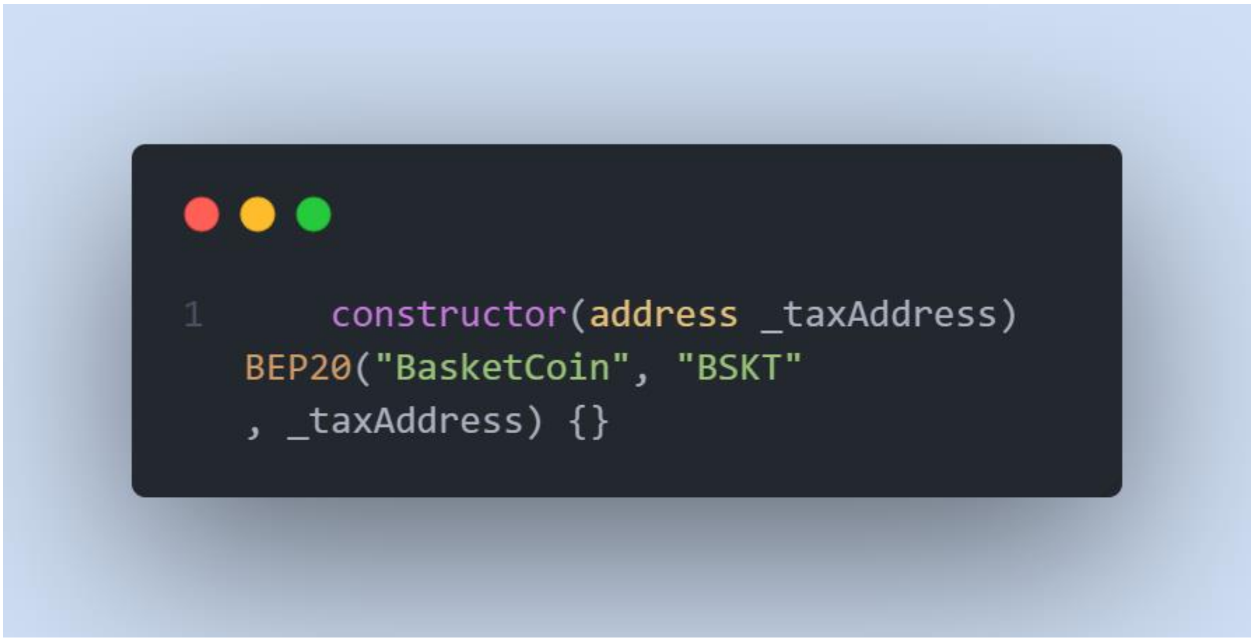
We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallet.

• Lack of declaring the important variable as state in the contract

In the contract, there are some important variables such as _taxFee, _burnFee, feeDeductStatus, excludeFromFee, and _taxFeeAddress. these variables are related to user transaction and contract behavior, we need to declare these variables as the storage state in the contract.

Medium severity issues

- **Zero Address Validation check**



```
constructor(address _taxAddress)
BEP20("BasketCoin", "BSKT"
, _taxAddress) {}
```

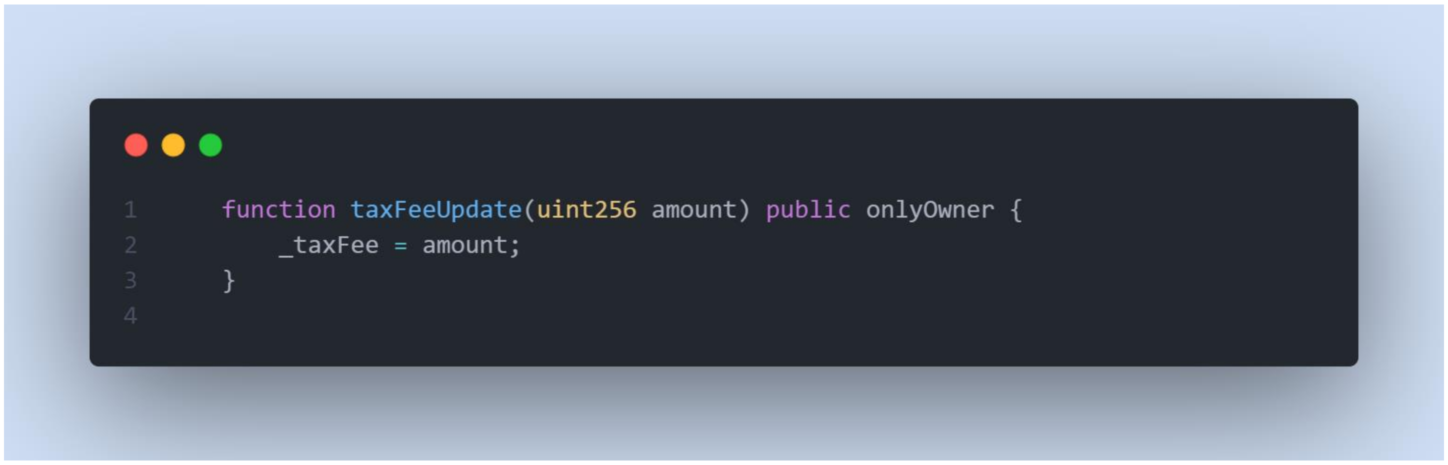
In the constructor function, we accept _ taxAddress as an argument and use it as a Tax Address. Based on the name of the argument, we send tax fees to this address so it’s important to check the value of this address before setting it.

Remediation

We need to check _ taxAddress , it should not be zero address so we advise to add this code in constructor function.

require(_ taxAddress != address(0), “message”);

- **Value Validation check**



```
function taxFeeUpdate(uint256 amount) public onlyOwner {
    _taxFee = amount;
}
```

In the functions shown below, the user is updating variables that are related to user transactions. We need to first check the argument in the function and then update the variable.

taxFeeUpdate(uint256 amount);

burnFeeUpdate(uint256 amount);

Remediation

We need to check the argument in the function before making any changes to the contract. The argument should not be 0 or it should not be more than the maximum.

require(_ amount >= maximumAmountAllowed , “message”);

require(_ amount <= minimumAmountAllowed , “message”);

- **UNCHECKED RETURN VALUE FROM IBEP20 TOKEN TRANSFER**

In the function tokenEmergencySafe, we used the IBEP20 transfer function to transfer the token to the user, but we don’t check the returned value. This value is bool and should be true for successful transfer.

Remediation

We recommend using require(IBEP20 transfer function) to check the return value to be true or using safeERC20 library from openzeppelin contracts.

Low severity issues

- ## Unused imported contract

In the contract and line 7, we imported SignatureChecker.sol, but we don’t use it on the contract.

Remediation

We recommend removing it and saving gas in the deployment process.

- ## Missing Emit Events

There should always be events emitted in the sensitive functions that are controlled by centralization roles.

Remediation

It is recommended emitting events for the sensitive functions that are controlled by centralization roles.

- ## Missing Error Messages

The require can be used to check for conditions and throw an exception if the condition is not met.

Remediation

We advise adding error messages to the linked require statements.

- ## 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.

Remediation

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