SMART CONTRACT AUDIT REPORT

For
xWIN Finance - Fixed Rate Staking Protocol

Prepared By: Kishan Patel **Prepared For**: xWIN Finance

Prepared on: 16/08/2021

Table of Content

• Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

• Overview of the audit

The project has 1 file. It contains approx 1071 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

• Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

• Over and under flows

An overflow happens when the limit of the type variable uint256, 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1. This is quite dangerous.

This contract **does** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, but all the functions have strong validations, which prevented this attack.

• Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the ethereum's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

• Visibility & Delegate call

It is also known as, The Parity Hack, which occurs while misuse of
Delegate call.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume "Public" visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

• Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of ethereum hands over control to that contract (B).

This makes it possible for B to call back into A before this interaction is completed.

Use of “require” function in this smart contract mitigated this vulnerability.

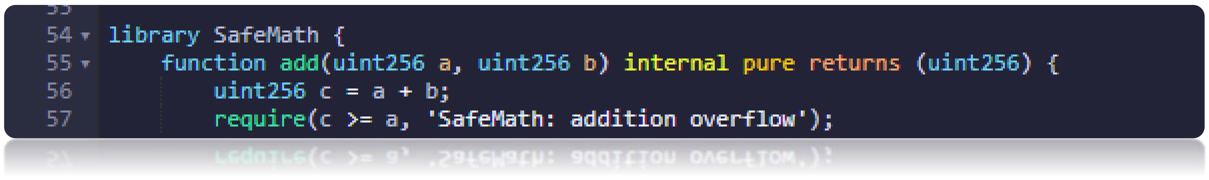
- **Forcing Ethereum to a contract**

While implementing “selfdestruct” in smart contract, it sends all the ethereum to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the “Required” conditions. Here, the Smart Contract’s balance has never been used as guard, which mitigated this vulnerability.

Good things in smart contract

SafeMath library:-

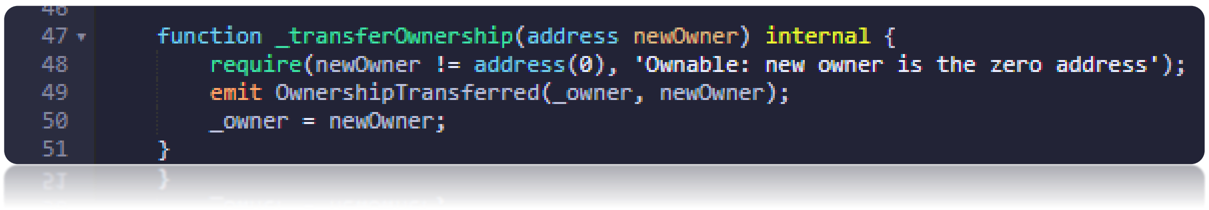
- You are using SafeMath library it is a good thing. This protects you from underflow and overflow attacks.



```
library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, 'SafeMath: addition overflow');
```

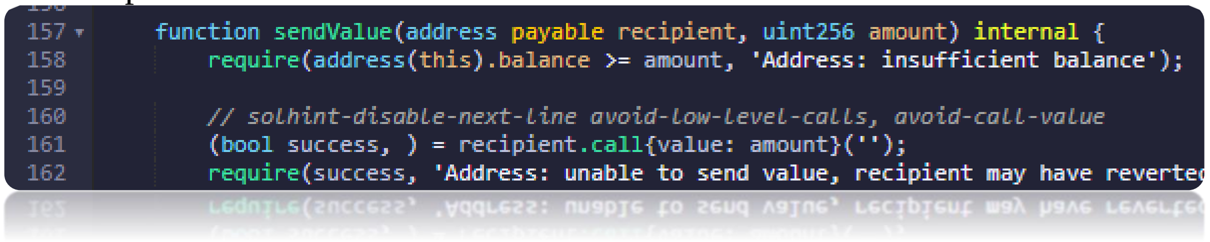
Good required condition in functions:-

- Here you are checking that the newOwner address value is a proper valid address.



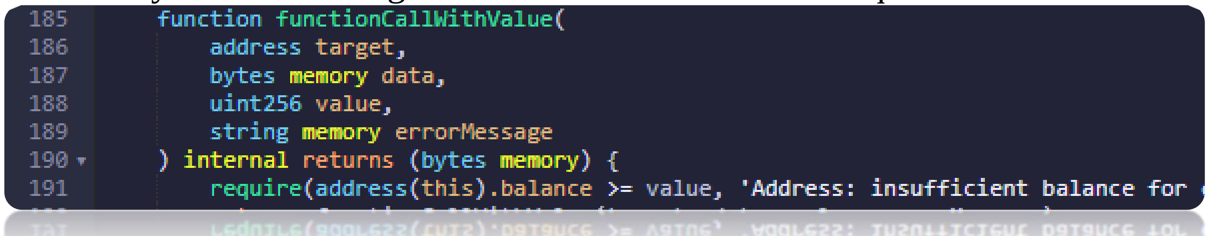
```
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), 'Ownable: new owner is the zero address');
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

- Here you are checking that balance of the contract is bigger or equal to the amount value and checking that token is successfully transferred to the recipient's address.



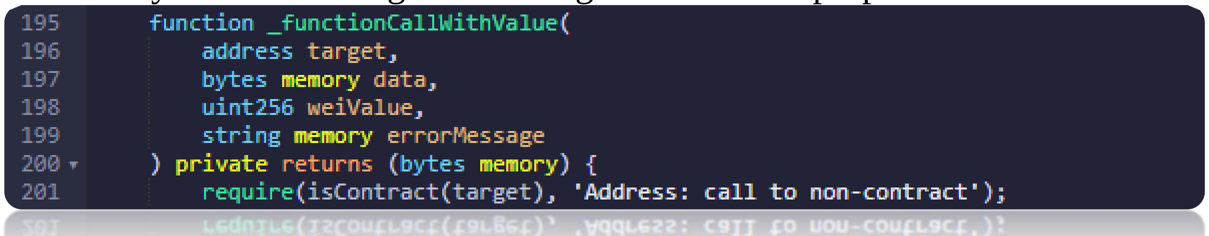
```
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, 'Address: insufficient balance');

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{value: amount}('');
        require(success, 'Address: unable to send value, recipient may have reverte
```

- Here you are checking that the contract has more or equal balance then value.



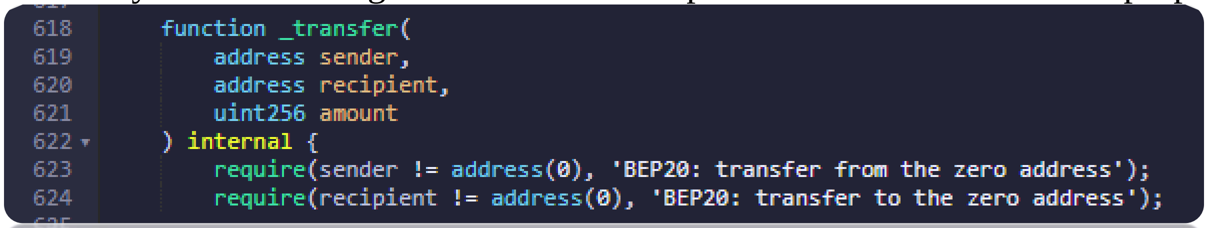
```
    function functionCallWithValue(
        address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, 'Address: insufficient balance for
```

- Here you are checking that the target address is a proper contract address or not.



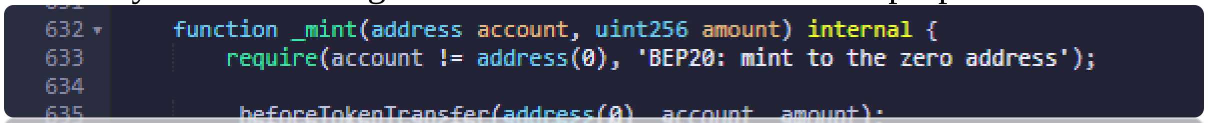
```
    function _functionCallWithValue(
        address target,
        bytes memory data,
        uint256 weiValue,
        string memory errorMessage
    ) private returns (bytes memory) {
        require(isContract(target), 'Address: call to non-contract');
```

- Here you are checking that sender and recipient addresses are valid and proper.

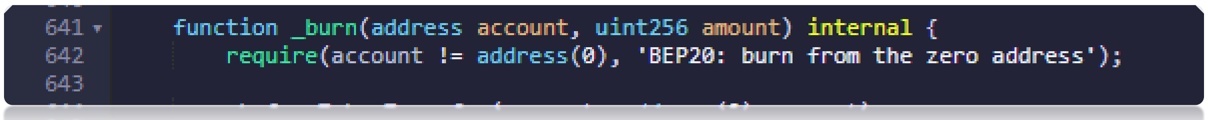


```
    function _transfer(
        address sender,
        address recipient,
        uint256 amount
    ) internal {
        require(sender != address(0), 'BEP20: transfer from the zero address');
        require(recipient != address(0), 'BEP20: transfer to the zero address');
```

- Here you are checking that account address is valid and proper.

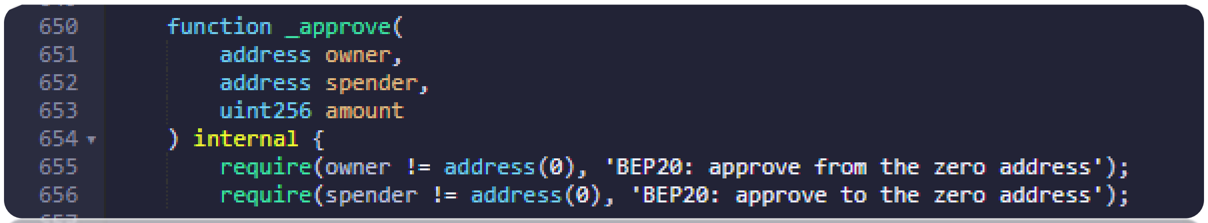


```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), 'BEP20: mint to the zero address');

        _beforeTokenTransfer(address(0), account, amount);
```



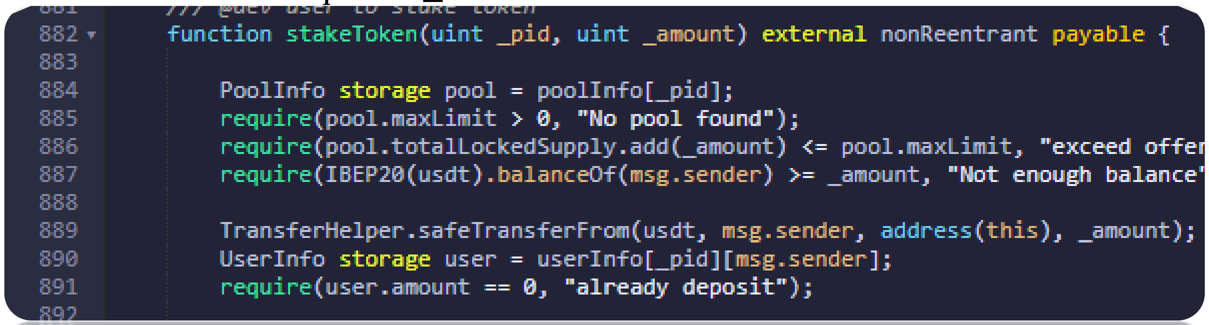
```
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), 'BEP20: burn from the zero address');

```

- Here you are checking that owner and spender addresses value are proper addresses.



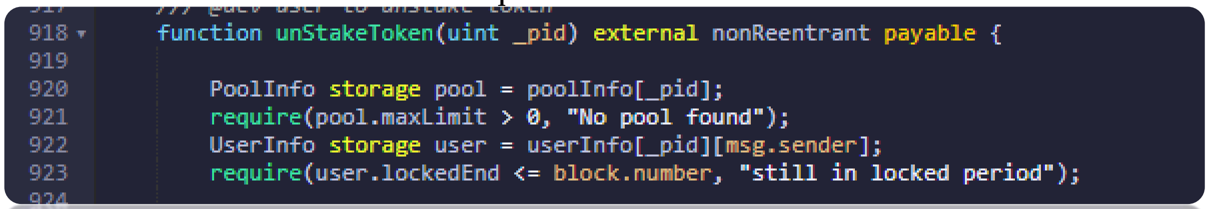
```
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal {
        require(owner != address(0), 'BEP20: approve from the zero address');
        require(spender != address(0), 'BEP20: approve to the zero address');
```

- Here you are checking that max limit of pool should be more than 0, _amount + totalLockedSupply should not be bigger than maxlimit of pool, msg.sender has balance more or equal to _amount and user.amount should be not 0.



```
    /// @dev user to stake token
    function stakeToken(uint _pid, uint _amount) external nonReentrant payable {

        PoolInfo storage pool = poolInfo[_pid];
        require(pool.maxLimit > 0, "No pool found");
        require(pool.totalLockedSupply.add(_amount) <= pool.maxLimit, "exceed offer
        require(IBEP20(usdt).balanceOf(msg.sender) >= _amount, "Not enough balance'

        TransferHelper.safeTransferFrom(usdt, msg.sender, address(this), _amount);
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount == 0, "already deposit");
```

- Here you are checking that max limit of pool should be more than 0, lockedEnd of user should be smaller or equal to current time.



```
    /// @dev user to unstake token
    function unStakeToken(uint _pid) external nonReentrant payable {

        PoolInfo storage pool = poolInfo[_pid];
        require(pool.maxLimit > 0, "No pool found");
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.lockedEnd <= block.number, "still in locked period");
```

• Critical vulnerabilities found in the contract

=> No Critial vulnerabilities found

• Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

- **Low severity vulnerabilities found**

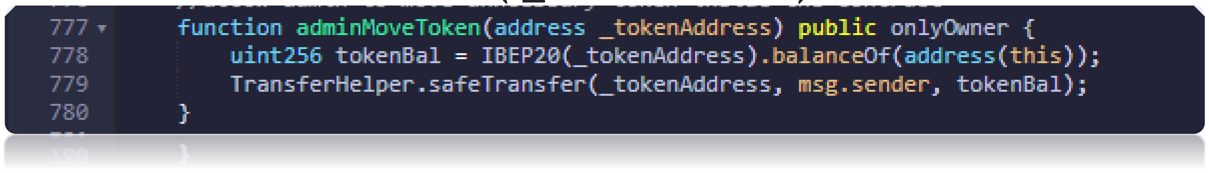
7.1: Short address attack:-

=> This is not a big issue in solidity, because of a new release of the solidity version. But it is good practice to check for the short address.

=> After updating the version of solidity it's not mandatory.

=> In some functions you are not checking the value of Address parameter here I am showing only necessary functions.

Function: -adminMoveToken ('_tokenAddress')



```
    function adminMoveToken(address _tokenAddress) public onlyOwner {
        uint256 tokenBal = IBEP20(_tokenAddress).balanceOf(address(this));
        TransferHelper.safeTransfer(_tokenAddress, msg.sender, tokenBal);
    }
```

- It's necessary to check the address value of "_tokenAddress". Because here you are passing whatever variable comes in "_tokenAddress" address from outside.

7.2: Compiler version is not fixed:-

=> In this file you have put "pragma solidity ^0.6.0;" which is not a good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with. Pragma solidity >=0.6.0; // bad: compiles 0.6.0 and above pragma solidity 0.6.0; //good: compiles 0.6.0 only

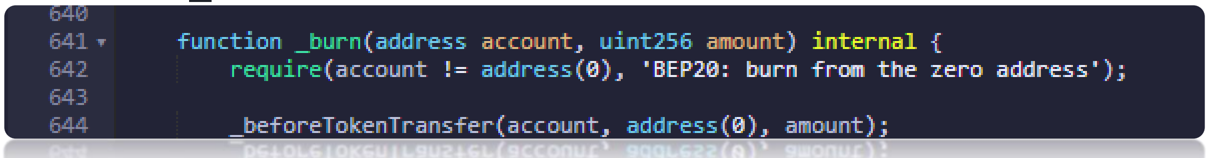
=> If you put(>=) symbol then you are able to get compiler version 0.6.0 and above. But if you don't use(^/>=) symbol then you are able to use only 0.6.0 version. And if there are some changes come in the compiler and you use the old version then some issues may come at deploy time.

=> Use latest version of solidity.

7.3: Check user balance in _burn and _approve:-

=> I have found that in _burn and _approve function user can give more value than their balance.
=> It is necessary to check that user can give allowance less or equal to their amount.
=> There is no validation about user balance. So it is good to check that a user not set approval wrongly.

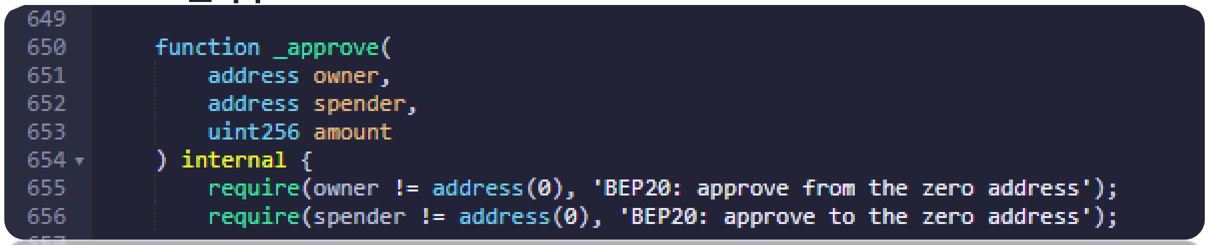
Function: - _burn



```
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), 'BEP20: burn from the zero address');

        _beforeTokenTransfer(account, address(0), amount);
```

- Here you can check that balance of account should be bigger or equal to amount value.

Function: - _approve



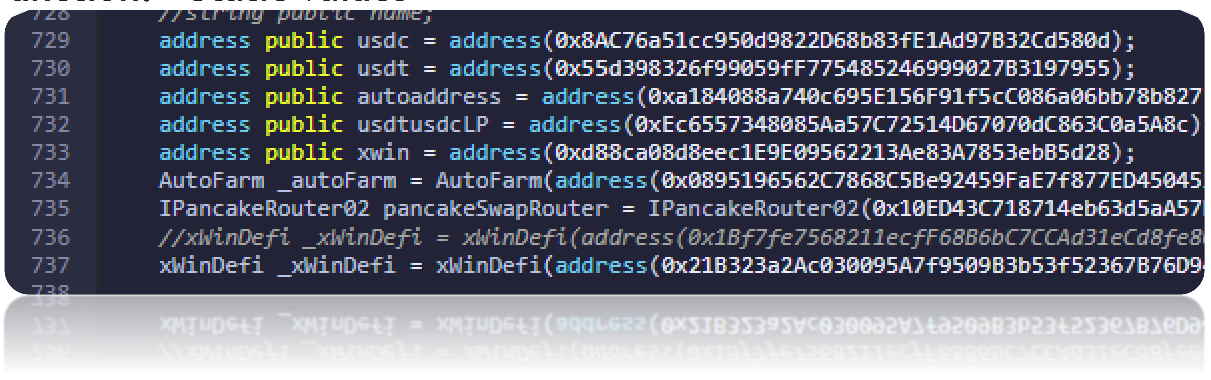
```
    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal {
        require(owner != address(0), 'BEP20: approve from the zero address');
        require(spender != address(0), 'BEP20: approve to the zero address');
```

- Here you can check that balance of owner should be bigger or equal to amount value.

7.4: Suggestions to add validations:-

=> You have implemented required validation in contract.
=> There are some place where you can improve validation and security of your code.
=> These are all just suggestion it is not bug.

Function: - Static values



```
    //string public name;
    address public usdc = address(0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d);
    address public usdt = address(0x55d398326f99059fF775485246999027B3197955);
    address public autoaddress = address(0xa184088a740c695E156F91f5cC086a06bb78b827
    address public usdtusdcLP = address(0xEc6557348085Aa57C72514D67070dC863C0a5A8c)
    address public xwin = address(0xd88ca08d8eec1E9E09562213Ae83A7853ebB5d28);
    AutoFarm _autoFarm = AutoFarm(address(0x0895196562C7868C5Be92459FaE7f877ED45045
    IPancakeRouter02 pancakeSwapRouter = IPancakeRouter02(0x10ED43C718714eb63d5aA57
    //xWinDefi _xWinDefi = xWinDefi(address(0x1Bf7fe7568211ecfF68B6bC7CCAd31eCd8fe8
    xWinDefi _xWinDefi = xWinDefi(address(0x21B323a2Ac030095A7f9509B3b53f52367B76D9
```

- Here you all value are static but admin might need to change these values in future so it is good to have functionality which can change these values.

• Summary of the Audit

Overall the code is well and performs well. There is no back

door to steal fund.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it’s quite important to define who’s supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).

- **Good Point:** Code performance and quality is good.
  Address validation and value validation is done properly.
- **Suggestions:** Please add address validations at some place
  and also try to use the latest and static version of solidity,
  check user balance in _burn and _approve function, try to
  include suggestions in code.