Smart contract security audit report

Audit Number：202105261646

Report query name：Dao swap

Smart Contract Info：

Smart Contract Name	Smart Contract Address	Smart Contract Address Link
BSCSwapAgentImpl	Fill in after deployment	Fill in after deployment
ETHSwapAgentImpl	Fill in after deployment	Fill in after deployment

Sha256：b580787ec15f1653e609d0ecc967245860f4b5f472d2b5457bc9dff91c6a3c92

Start Date：2021.05.25

Completion Date：2021.05.26

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass

		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Disclaimer:This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project.This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of project Dao swap, including Coding Standards, Security, and Business Logic. **The Dao swap project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing Eth.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level function like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

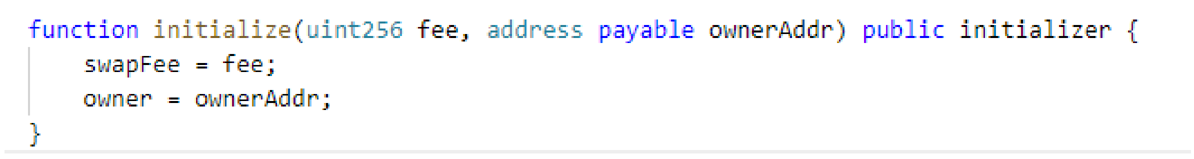
- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

3.1 Business analysis of Contract ETHSwapAgentImpl

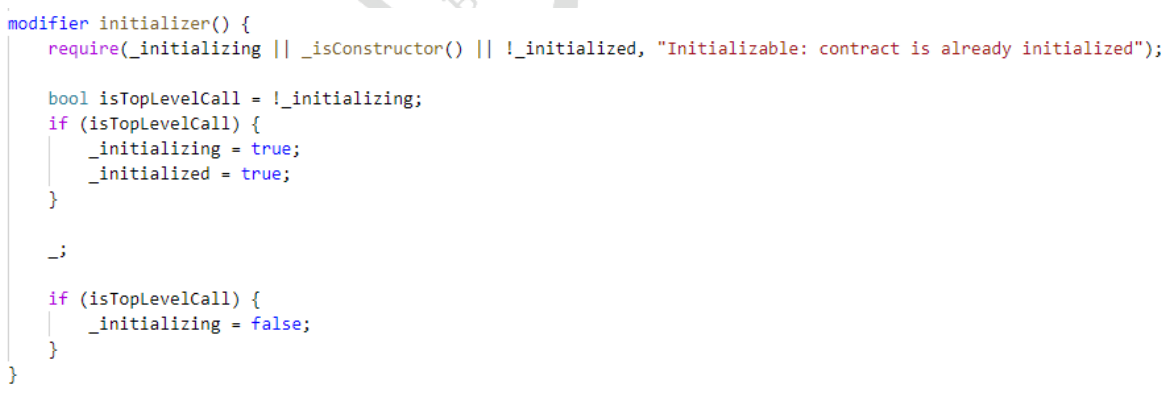
(1) initialize function

- Description: When the contract is deployed, the *initialize* function needs to be called to initialize the swapFee and owner. The function is modified by initializer and the contract can only be called once.



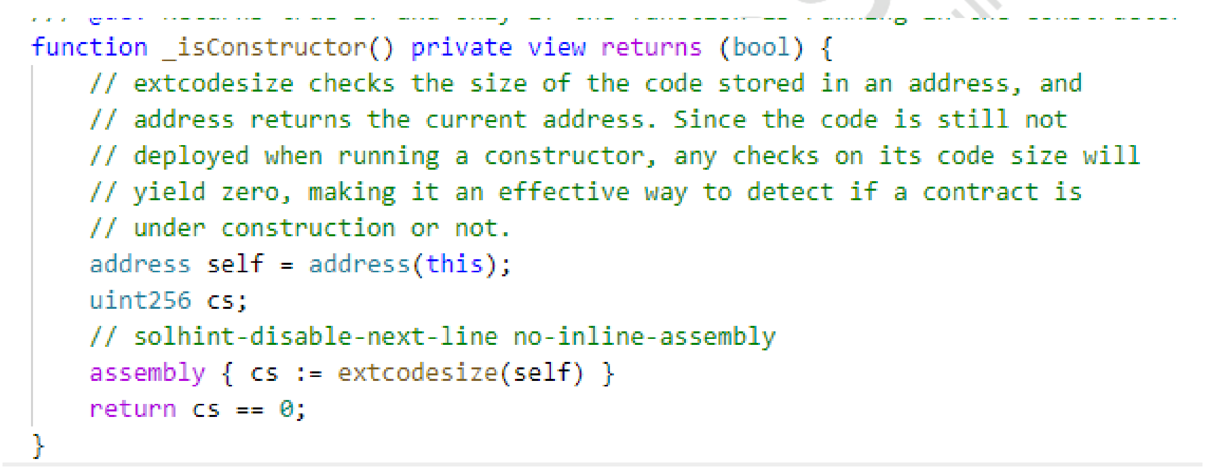
```
function initialize(uint256 fee, address payable ownerAddr) public initializer {
    swapFee = fee;
    owner = ownerAddr;
}
```

Figure 1 Source code of *initialize*



```
modifier initializer() {
    require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is already initialized");

    bool isTopLevelCall = !_initializing;
    if (isTopLevelCall) {
        _initializing = true;
        _initialized = true;
    }

    _;

    if (isTopLevelCall) {
        _initializing = false;
    }
}
```

Figure 2 Modifier initializer source code



```
function _isConstructor() private view returns (bool) {
    // extcodesize checks the size of the code stored in an address, and
    // address returns the current address. Since the code is still not
    // deployed when running a constructor, any checks on its code size will
    // yield zero, making it an effective way to detect if a contract is
    // under construction or not.
    address self = address(this);
    uint256 cs;
    // solhint-disable-next-line no-inline-assembly
    assembly { cs := extcodesize(self) }
    return cs == 0;
}
```

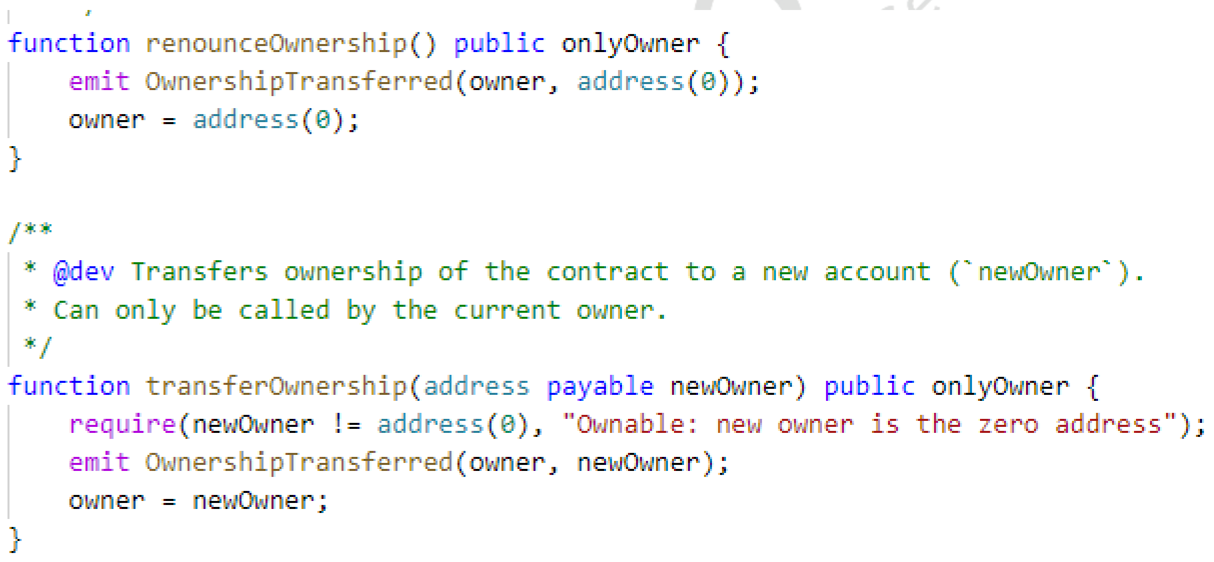
Figure 3 Source code of *_isConstructor*

- Related function: *initialize*, *_isConstructor*

- Result: Pass

(2) Permission transfer function

- Description: The *renounceOwnership* function is called by the owner to renounce the ownership, The *transferOwnership* function is called by the owner to transfer the ownership.



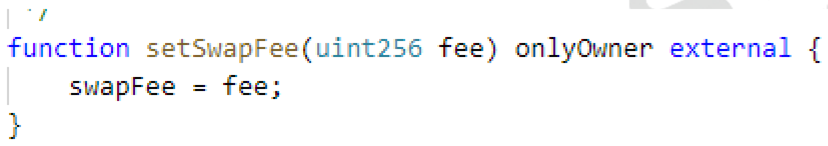
```
function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(owner, address(0));
    owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address payable newOwner) public onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
```

Figure 4 Source code of *renounceOwnership& transferOwnership*

- Related function: *renounceOwnership*, *transferOwnership*
- Result: Pass

(3) setSwapFee function

- Description: The *setSwapFee* function is called by the owner to set the swap fee.



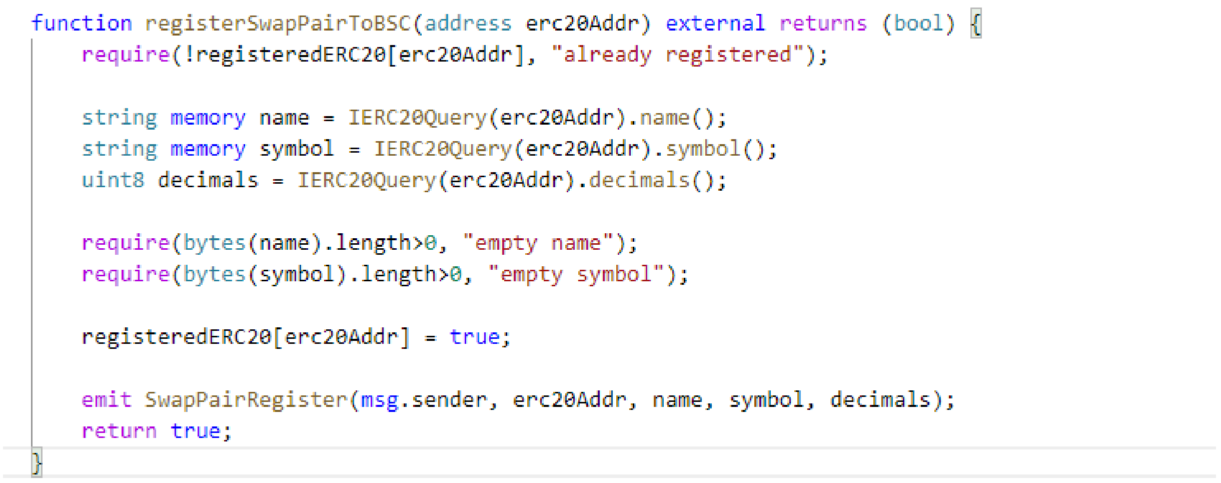
```
function setSwapFee(uint256 fee) onlyOwner external {
    swapFee = fee;
}
```

Figure 5 Source code of *setSwapFee*

- Related function: *setSwapFee*
- Result: Pass

(4) registerSwapPairToBSC function

- Description: *registerSwapPairToBSC* function is used to register the ERC20 token contract address, When calling this function, the token name and symbol length must be greater than 0.



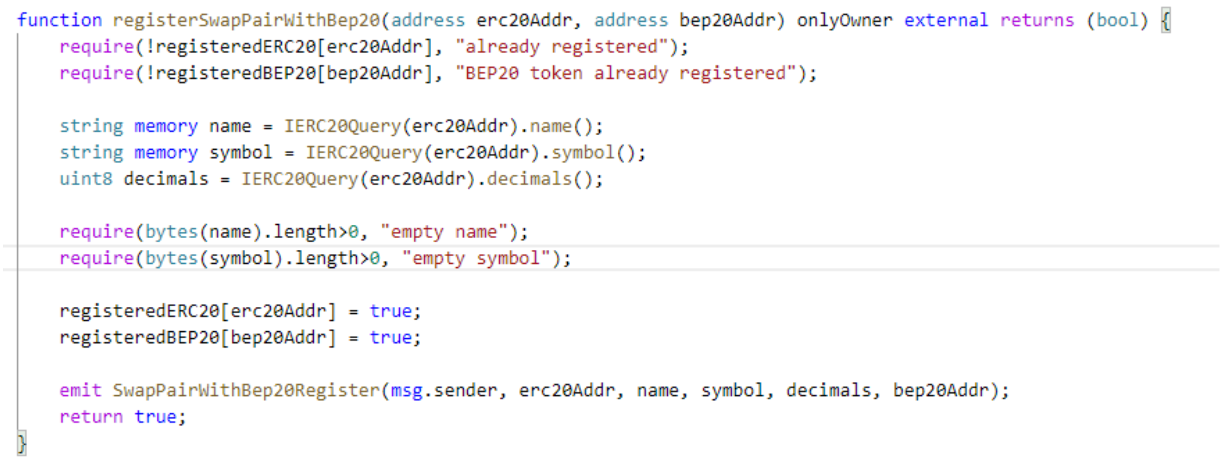
```
function registerSwapPairToBSC(address erc20Addr) external returns (bool) {
    require(!registeredERC20[erc20Addr], "already registered");

    string memory name = IERC20Query(erc20Addr).name();
    string memory symbol = IERC20Query(erc20Addr).symbol();
    uint8 decimals = IERC20Query(erc20Addr).decimals();

    require(bytes(name).length>0, "empty name");
    require(bytes(symbol).length>0, "empty symbol");

    registeredERC20[erc20Addr] = true;

    emit SwapPairRegister(msg.sender, erc20Addr, name, symbol, decimals);
    return true;
}
```

Figure 6 Source code for *registerSwapPairToBSC*

- Related function: *registerSwapPairToBSC, name, symbol, decimals*
- Result: Pass

(5) registerSwapPairWithBep20 function

- Description: The *registerSwapPairWithBep20* function is used to register ERC20 tokens and DEP20 tokens at the same time, and can only be called by the owner. When calling this function, the name and symbol length of ERC20 must be greater than zero.



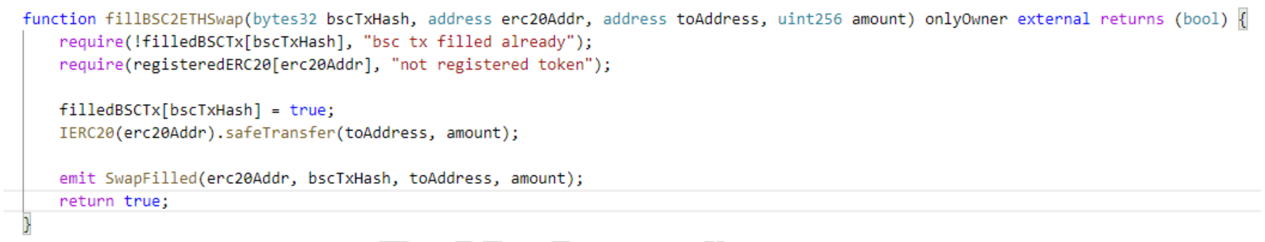
```
function registerSwapPairWithBep20(address erc20Addr, address bep20Addr) onlyOwner external returns (bool) {
    require(!registeredERC20[erc20Addr], "already registered");
    require(!registeredBEP20[bep20Addr], "BEP20 token already registered");

    string memory name = IERC20Query(erc20Addr).name();
    string memory symbol = IERC20Query(erc20Addr).symbol();
    uint8 decimals = IERC20Query(erc20Addr).decimals();

    require(bytes(name).length>0, "empty name");
    require(bytes(symbol).length>0, "empty symbol");

    registeredERC20[erc20Addr] = true;
    registeredBEP20[bep20Addr] = true;

    emit SwapPairWithBep20Register(msg.sender, erc20Addr, name, symbol, decimals, bep20Addr);
    return true;
}
```

Figure 7 Source code of *registerSwapPairWithBep20*

- Related function: *registerSwapPairWithBep20, name, symbol, decimals*
- Result: Pass

(6) *fillBSC2ETHSwap* function

- Description: The *fillBSC2ETHSwap* function is used to call the specified ERC20 token contract to transfer tokens to the specified address, this function can only be called by the owner.(The project owner declares that the owner privilege will be given to the contract control, it should be noted that the ISwap contract is not in the scope of the audit.)



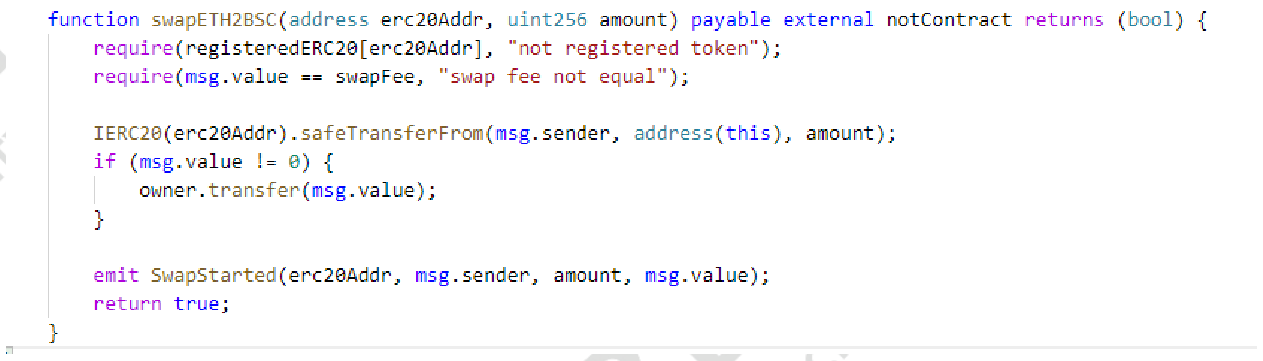
```
function fillBSC2ETHSwap(bytes32 bscTxHash, address erc20Addr, address toAddress, uint256 amount) onlyOwner external returns (bool) {
    require(!filledBSCTx[bscTxHash], "bsc tx filled already");
    require(registeredERC20[erc20Addr], "not registered token");

    filledBSCTx[bscTxHash] = true;
    IERC20(erc20Addr).safeTransfer(toAddress, amount);

    emit SwapFilled(erc20Addr, bscTxHash, toAddress, amount);
    return true;
}
```

Figure 8 Source code of *fillBSC2ETHSwap*

- Related function: *fillBSC2ETHSwap, safeTransfer*
- Result: Pass

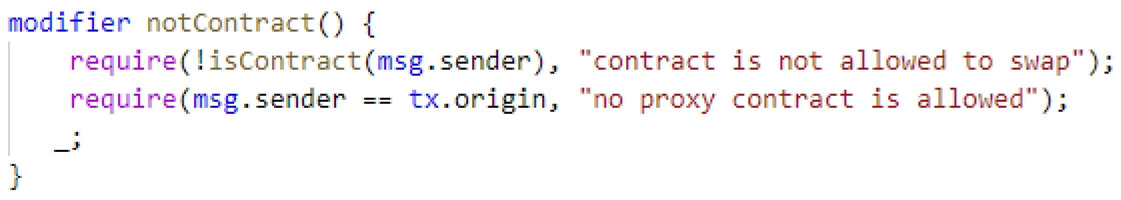
(7) swapETH2BSC function

- Description: The *swapETH2BSC* function is used to transfer a specified number of ERC20 tokens to the contract. The call requires that the ERC20 token address is registered and a certain amount of swap fee is paid, if the swap fee is not zero, then it will be sent to the owner address. The function is modified by notContract to restrict the contract call.



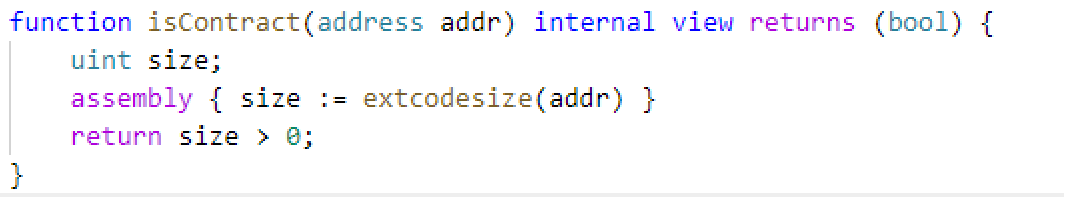
```
function swapETH2BSC(address erc20Addr, uint256 amount) payable external notContract returns (bool) {
    require(registeredERC20[erc20Addr], "not registered token");
    require(msg.value == swapFee, "swap fee not equal");

    IERC20(erc20Addr).safeTransferFrom(msg.sender, address(this), amount);
    if (msg.value != 0) {
        owner.transfer(msg.value);
    }

    emit SwapStarted(erc20Addr, msg.sender, amount, msg.value);
    return true;
}
```

Figure 9 Source code of *swapETH2BSC*



```
modifier notContract() {
    require(!isContract(msg.sender), "contract is not allowed to swap");
    require(msg.sender == tx.origin, "no proxy contract is allowed");
   _;
}
```

Figure 10 Modifier notContract source code



```
function isContract(address addr) internal view returns (bool) {
    uint size;
    assembly { size := extcodesize(addr) }
    return size > 0;
}
```

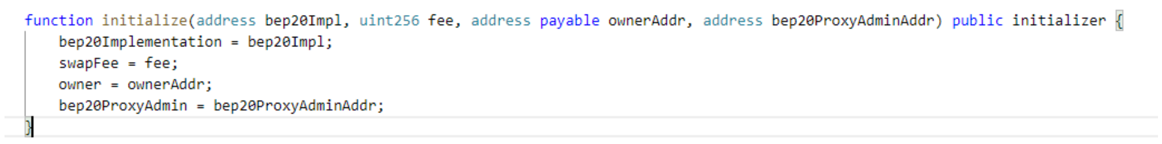
Figure 11 Source code of isContract

- Related function: *swapETH2BSC, safeTransferFrom, transfer*
- Result: Pass

3.2 Business analysis of Contract BSCSwapAgentImpl

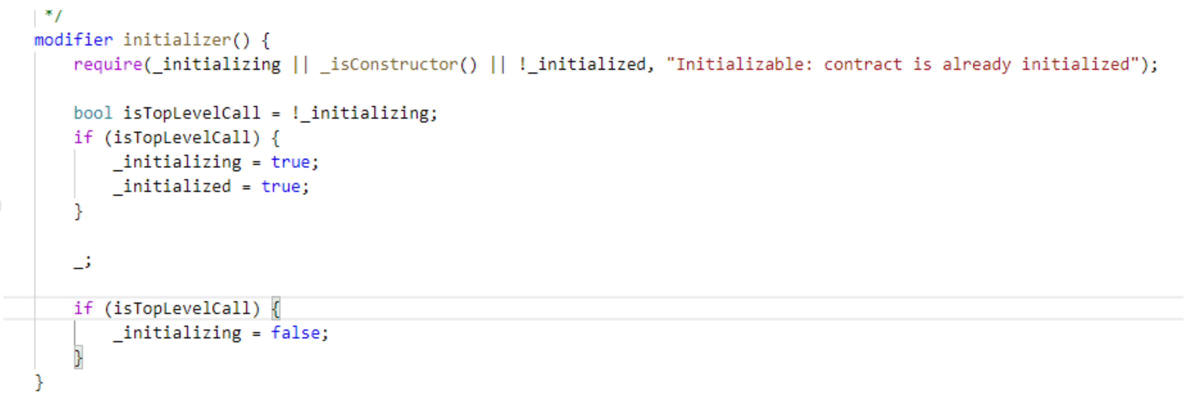
(1) initialize function

- Description: The *initialize* function is used to initialize the bep20Implementation address, swapFee, owner address and bep20ProxyAdmin address. The function is modified by initializer and the contract can only be called once.



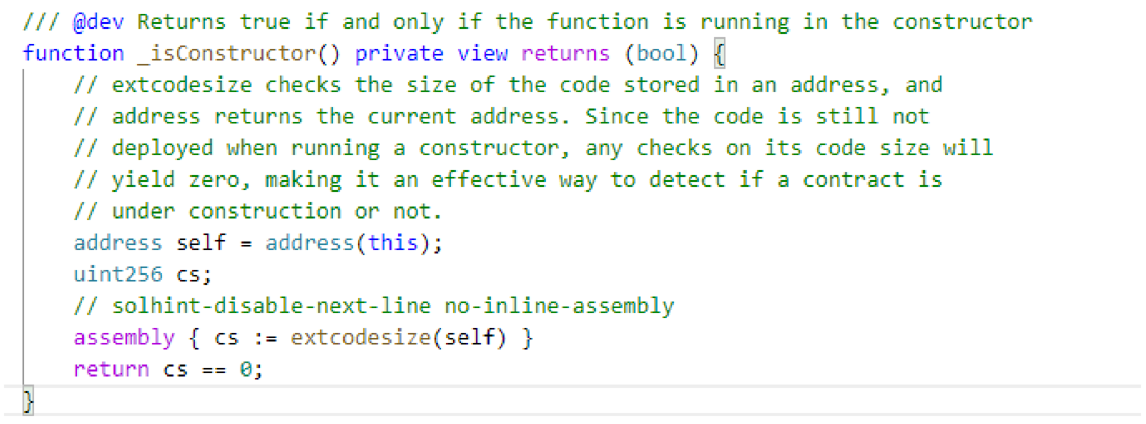
```
function initialize(address bep20Impl, uint256 fee, address payable ownerAddr, address bep20ProxyAdminAddr) public initializer {
    bep20Implementation = bep20Impl;
    swapFee = fee;
    owner = ownerAddr;
    bep20ProxyAdmin = bep20ProxyAdminAddr;
}
```

Figure 12 Source code for *initialize*



```
 */
modifier initializer() {
    require(_initializing || _isConstructor() || !_initialized, "Initializable: contract is already initialized");

    bool isTopLevelCall = !_initializing;
    if (isTopLevelCall) {
        _initializing = true;
        _initialized = true;
    }

    _;

    if (isTopLevelCall) {
        _initializing = false;
    }
}
```

Figure 13 Modifier initializer source code



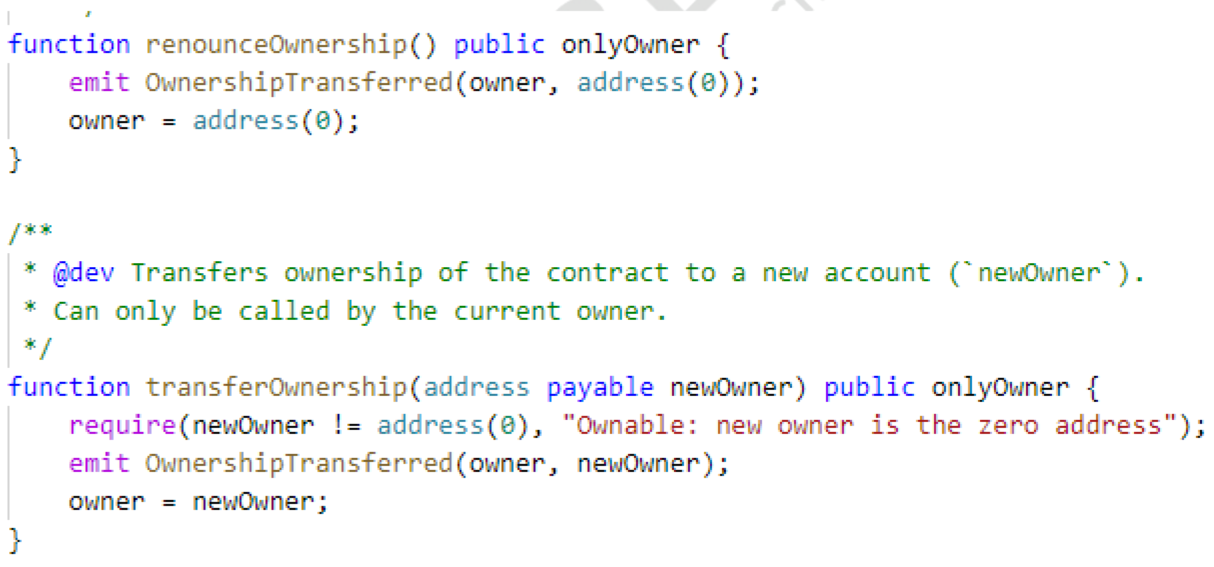
```
/// @dev Returns true if and only if the function is running in the constructor
function _isConstructor() private view returns (bool) {
    // extcodesize checks the size of the code stored in an address, and
    // address returns the current address. Since the code is still not
    // deployed when running a constructor, any checks on its code size will
    // yield zero, making it an effective way to detect if a contract is
    // under construction or not.
    address self = address(this);
    uint256 cs;
    // solhint-disable-next-line no-inline-assembly
    assembly { cs := extcodesize(self) }
    return cs == 0;
}
```

Figure 14 Source code for *_isConstructor*

- Related function: *initialize, _isConstructor*
- Result: Pass

(2) Permission transfer function

- Description: The *renounceOwnership* function is called by the owner to renounce the ownership, The transferOwnership function is called by the owner to transfer the ownership.



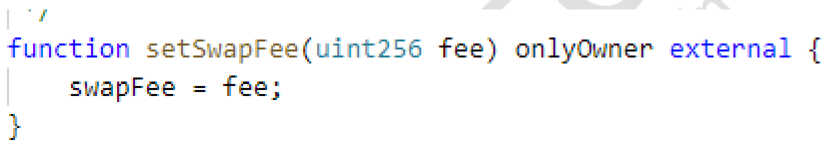
```
function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(owner, address(0));
    owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
function transferOwnership(address payable newOwner) public onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    emit OwnershipTransferred(owner, newOwner);
    owner = newOwner;
}
```

Figure 15 Source code of *renounceOwnership& transferOwnership*

- Related function: *renounceOwnership, transferOwnership*
- Result: Pass

(3) setSwapFee function

- Description: The *setSwapFee* function is called by the owner to set the swap fee.



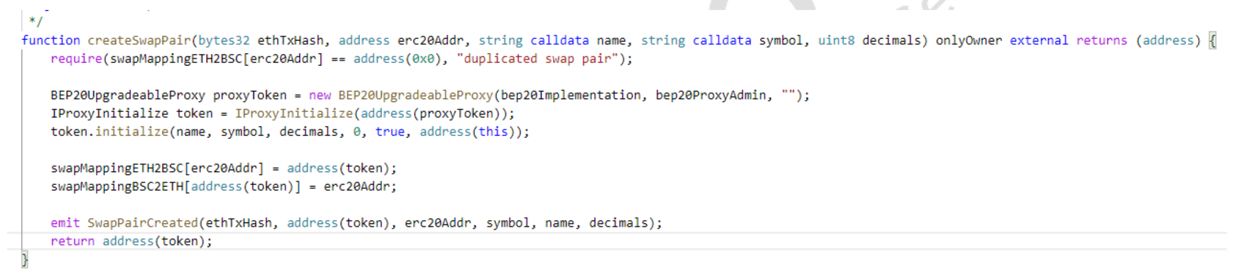
```
function setSwapFee(uint256 fee) onlyOwner external {
    swapFee = fee;
}
```

Figure 16 Source code of *setSwapFee*

- Related function: *setSwapFee*
- Result: Pass

(4) createSwapPair function

- Description: The *createSwapPair* function is used to create a swap pair. When calling the function, it is necessary to satisfy that ERC20 has not been bound before, generate a proxytoken to get the token address, then initialize it by calling the *initialize* function in the token contract, and finally bind the swap pair, which can only be called by the owner.



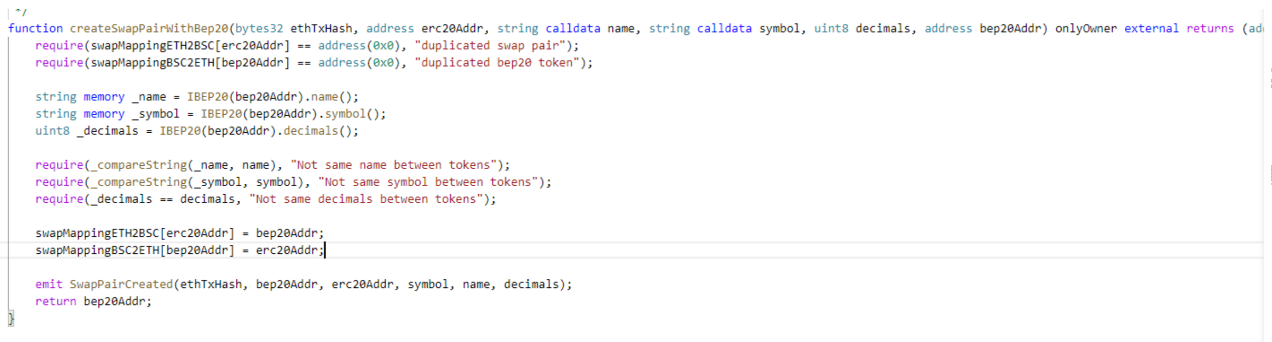
```
*/
function createSwapPair(bytes32 ethTxHash, address erc20Addr, string calldata name, string calldata symbol, uint8 decimals) onlyOwner external returns (address) {
    require(swapMappingETH2BSC[erc20Addr] == address(0x0), "duplicated swap pair");

    BEP20UpgradeableProxy proxyToken = new BEP20UpgradeableProxy(bep20Implementation, bep20ProxyAdmin, "");
    IProxyInitialize token = IProxyInitialize(address(proxyToken));
    token.initialize(name, symbol, decimals, 0, true, address(this));

    swapMappingETH2BSC[erc20Addr] = address(token);
    swapMappingBSC2ETH[address(token)] = erc20Addr;

    emit SwapPairCreated(ethTxHash, address(token), erc20Addr, symbol, name, decimals);
    return address(token);
}
```

Figure 17 Source code of *createSwapPair*

- Related function: *createSwapPair*
- Result: Pass

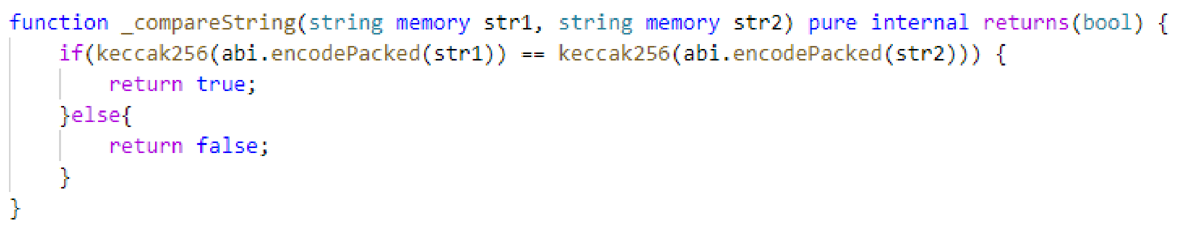
(5) createSwapPairWithBep20 function

- Description: This function is used to add swap pairs, which can only be called by the owner. The call requires that ERC20 tokens and DEP20 tokens have not been bound before.



```
*/
function createSwapPairWithBep20(bytes32 ethTxHash, address erc20Addr, string calldata name, string calldata symbol, uint8 decimals, address bep20Addr) onlyOwner external returns (ad
    require(swapMappingETH2BSC[erc20Addr] == address(0x0), "duplicated swap pair");
    require(swapMappingBSC2ETH[bep20Addr] == address(0x0), "duplicated bep20 token");

    string memory _name = IBEP20(bep20Addr).name();
    string memory _symbol = IBEP20(bep20Addr).symbol();
    uint8 _decimals = IBEP20(bep20Addr).decimals();

    require(_compareString(_name, name), "Not same name between tokens");
    require(_compareString(_symbol, symbol), "Not same symbol between tokens");
    require(_decimals == decimals, "Not same decimals between tokens");

    swapMappingETH2BSC[erc20Addr] = bep20Addr;
    swapMappingBSC2ETH[bep20Addr] = erc20Addr;

    emit SwapPairCreated(ethTxHash, bep20Addr, erc20Addr, symbol, name, decimals);
    return bep20Addr;
}
```

Figure 18 Source code of *createSwapPairWithBep20*



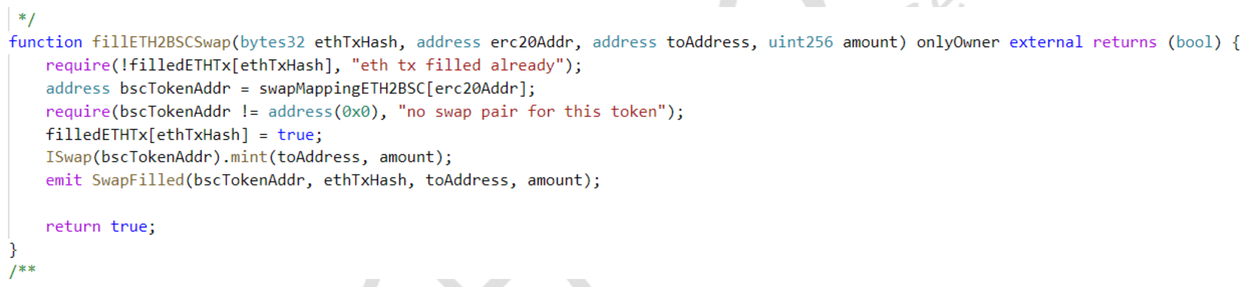
```
function _compareString(string memory str1, string memory str2) pure internal returns(bool) {
    if(keccak256(abi.encodePacked(str1)) == keccak256(abi.encodePacked(str2))) {
        return true;
    }else{
        return false;
    }
}
```

Figure 19 Source code of *_compareString*

- Related function: *createSwapPairWithBep20, name, symbol, decimals, _compareString*
- Result: Pass

(6) fillETH2BSCSwap function

- Description: This function is called by the owner to mint tokens to the specified address via the bscToken contract.(The project owner declares that the owner privilege will be given to the contract control, it should be noted that the ISwap contract is not in the scope of the audit.)



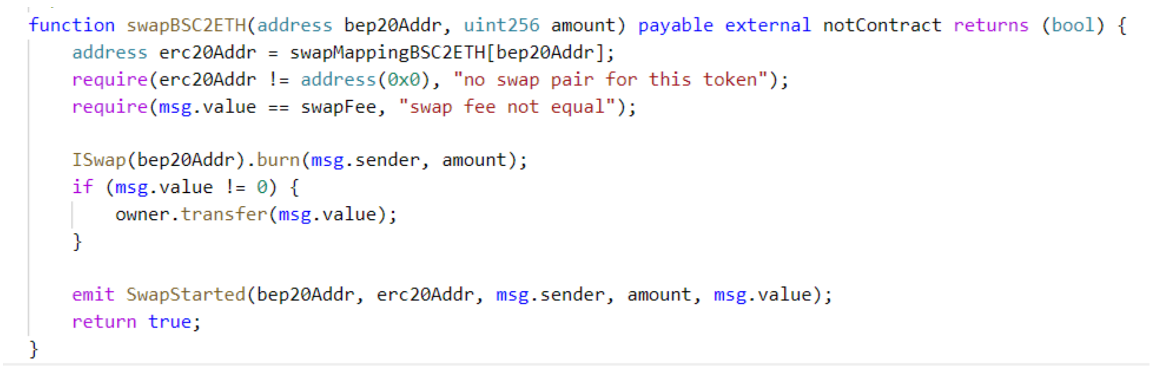
```
*/
function fillETH2BSCSwap(bytes32 ethTxHash, address erc20Addr, address toAddress, uint256 amount) onlyOwner external returns (bool) {
    require(!filledETHTx[ethTxHash], "eth tx filled already");
    address bscTokenAddr = swapMappingETH2BSC[erc20Addr];
    require(bscTokenAddr != address(0x0), "no swap pair for this token");
    filledETHTx[ethTxHash] = true;
    ISwap(bscTokenAddr).mint(toAddress, amount);
    emit SwapFilled(bscTokenAddr, ethTxHash, toAddress, amount);

    return true;
}
/**
```

Figure 20 Source code of *permitExitFund*

- Related function: *fillETH2BSCSwap, mint*
- Result: Pass

(6) swapBSC2ETH function

- Description: This function is modified by notContract and cannot be called with a contract. When it is called, a certain amount of swap fee is paid and the number of tokens specified by the caller is destroyed, if the swap fee is greater than zero, then the swap fee is sent to the owner's address.



```
function swapBSC2ETH(address bep20Addr, uint256 amount) payable external notContract returns (bool) {
    address erc20Addr = swapMappingBSC2ETH[bep20Addr];
    require(erc20Addr != address(0x0), "no swap pair for this token");
    require(msg.value == swapFee, "swap fee not equal");

    ISwap(bep20Addr).burn(msg.sender, amount);
    if (msg.value != 0) {
        owner.transfer(msg.value);
    }

    emit SwapStarted(bep20Addr, erc20Addr, msg.sender, amount, msg.value);
    return true;
}
```

Figure 21 Source code of *swapBSC2ETH*

- Related function: *swapBSC2ETH, burn, transfer*
- Result: Pass

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts. The project Dao swap passed all audit items, The overall audit result is **Pass.**

BEOSIN

Blockchain Security

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com