# DexanceMigrator

## Smart Contract Review

### Deliverable: Smart Contract Audit Report

**Security Report Aug 2022**

# Disclaimer

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the Company. The content, conclusions and recommendations set out in this publication are elaborated in the specific for only project.

HireCA does not guarantee the accuracy of the data included in this study. All representations, warranties, undertakings and guarantees relating to the report are excluded, particularly concerning – but not limited to – the qualities of the assessed projects and products. Neither the Company nor any personating on the Company's behalf may be held responsible for the use that may be made of the information contained herein. HireCA retains the right to display audit reports and other content elements as examples of their work in their portfolio and as content features in other projects with protecting all security purpose of customer. The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities fixed - upon a decision of the Customer.

# Overview

## Background

DexanceMigrator requested that HireCA perform an Extensive Smart Contract audit of their Smart Contract.

## Project Dates

The following is the project schedule for this review and report:

**Aug 17** : Smart Contract Review Completed *(Completed)*
**Aug 17** : Delivery of Smart Contract Audit Report *(Completed)*

## Review Team

The following HireCA team member participated in this review:

- Abhishek Mishra, Security Researcher and Engineer

## Coverage

**Target Specification and Revision**

For this audit, we performed research, investigation, and review of the smart contract of DexanceMigrator.
The following documentation repositories were considered in-scope for the review:
DexanceMigrator Project:

**EXPLORER LINK**

https://bscscan.com/address/0xAcE93A6633fb65d6bAE8B5b1D03d77356b81fA67

SMART CONTRACT AUDIT

0xAcE93A6633fb65d6bAE8B5b1D03d77356b81fA67

## [Smart Contract Breakdown]

```
1  // SPDX-License-Identifier: MIT
2  pragma solidity 0.8.6;
```

Line 1: OBJ

Massachusetts Institute of Technology (MIT) License is free to license software. The MIT License grants the software end-user rights such as copying, modifying, merging, distributing, etc.

Line 2:

The solidity Compiler version used for compiling this Smart Contract is v0.8.6

```
4  import "./interfaces/IBEP20.sol";
5  import "./interfaces/IDexanceERC20.sol";
6  import "./libs/Pausable.sol";
7  import "./libs/Ownable.sol";
8  import "./libs/ReentrancyGuard.sol";
```

Line 4 to 8:

Different Libraries and Interfaces used in the Smart Contract, here

IBEP20, IDexanceERC20, Pausable, Ownable, ReentrancyGuard.

[All these are Solidity files and bundled with the Smart Contract]

```
10 contract DexanceMigrator is Ownable,Pausable,ReentrancyGuard {
11
12     IBEP20 public payb;
13
14     IDexanceERC20 public dexance;
15
16     address constant public DEAD = address(0x000000000000000000000000000000000000dEaD);
```

OBJOBJ

Line 10:

The contract is created and named `DexanceMigrator` and inherits properties of Ownable, Pausable, and ReentrancyGuard [imported above]

Line 12:

A Variable is created of IBEP20 type named `payb` with public scope, which means it is visible to everyone on the network.

Line 14:
A Variable is created of IDexanceERC20 type named `dexance` with public scope, which means it is visible to everyone on the network.
Line 16:
A Constant is created of type address named `DEAD` with public scope, which means it is visible to everyone on the network.
And initialized with a Dead wallet Address.

```
17
18      event Migrate(address indexed who, uint256 amount);
19
20      event Burn(address indexed who, uint256 amount);
21
```

Line 18 and 20:
Events are created named `Migrate` and `Burn` with different parameters and generally fired or emitted on Successful Function execution and used in the Front end part of the Application.

```
22      constructor(address _payb,address _dexance) {
23          require(_payb != address(0),"Bad address");
24          require(_dexance != address(0),"Bad address");
25          require(_payb != _dexance,"Bad address");
26          payb = IBEP20(_payb);
27          dexance = IDexanceERC20(_dexance);
28      }
```

Line 22 to 28:
Constructor, A type of special function that is executed on Smart Contract deployment. Here it is a type or parameterized constructor which holds two parameters that are `_payb` and `_dexance`.
[This constructor checks for ]
*_payb must not equal to a null address*
*_dexance must not be equal to a null address*
*_payb and _dexance must not be the same address*

After all the requirements are satisfied, `payb` is initialized with `_payb`
And `dexance` is initialized with `_dexance` by converting into IBEP20 and IDexanceERC20 types respectively.

```
29      modifier onlyEOA(){
30          require(tx.origin == msg.sender,"Only EOA");
31          _;
32      }
33
34      function pause() public onlyOwner whenNotPaused{
35          _pause();
36      }
37      function unpause() public onlyOwner whenPaused{
38          _unpause();
39      }
```

Line 29 to 32:
A modifier is created named `onlyEOA` which requires the function to only be called by an Externally Owned Account.
Line 34 to 36:
A function is created named `pause` which is only called by the Owner of the contract and in the condition of not already paused, which simply calls ***_pause()*** function.
Line 37 to 39:
A function is created named `unpause` which is only called by the Owner of the contract and in the condition of already paused, which simply calls ***_unpause()*** function.

```
41      /// @dev migrate _amount of payb token to dexance
42      function migrate(uint256 _amount) public nonReentrant whenNotPaused onlyEOA{
43          require(_amount > 0,"Bad amount");
44          uint256 paybBefore = payb.balanceOf(address(this));
45          payb.transferFrom(msg.sender, address(this), _amount);
46          uint256 paybAfter = payb.balanceOf(address(this));
47          require(_amount == paybAfter - paybBefore, "Bad amount");
48          _burn();
49          dexance.mint(msg.sender, _amount);
50          emit Migrate(msg.sender,_amount);
51      }
52
```

Line 42 to 51:
A function is created named `migrate` which takes one argument of type uint256 and is referred to as `_amount`. Has public scope and only called by Externally Owned Accounts in the condition of not paused [nonReentrant is used for preventing the smart contract from calling its function by itself, directly or indirectly]
[This function checks for ]
*_amount must be greater than 0*
*_amount must be equal  to the difference between paybAfter and paybBefore*

After all the requirements are satisfied, this function calls the ***_burn()*** function and mint dexance by passing the address by which the function is called and the `_amount`
In the execution process of Function, when it reaches its last line [Line: 50], an event is emitted or fired named `Migrate` which is created is above [Line: 18]

```
53      /// @dev Mannualy burn payb token due to lack of burn function in payb token contract
54      function _burn() internal{
55          uint256 b = payb.balanceOf(address(this));
56          if(b <= 0) return;
57          payb.transfer(address(DEAD),b);
58          emit Burn(msg.sender,b);
59      }
```

Line 54 to 59:
A function is created named `_burn` which is an internal function, which means it can be only called by the smart contract internally.
*This function does nothing in case of the number of assets to be burned is equal to 0 and exits the execution by returning nothing.*
And if about condition is satisfied, it transfers the assets into the `DEAD` wallet declared and initialized above in the Contract [Line: 16] and emits the `Burn` event on successful execution of the function.

## [Libraries/Interfaces Contract]

### IBEP20.sol

An Interface is used for an IBEP20 type token which provides all the necessary functions of a token to the smart contract to be used.

### IDexanceERC20.sol

An Interface contains a function named `_mint` that is used to mint more tokens.

### Pausable.sol

An Interface is used for enabling the feature of Pausing smart contracts, which restricts it from executing anything.

### Ownable.sol

An Interface is used to make a Smart Contract ownable, which means the smart contract has an owner and has some restricted functions that can be only called by the Owner's address.

### ReentrancyGuard.sol

This Interface restricts Smart Contract from calling itself directly or indirectly.

# About HireCA

We believe that people have a fundamental need to security and that the use of secure solutions enables every person to more freely use the Internet and every other connected technology. We aim to provide security consulting service to help others make their solutions more resistant to unauthorized access to data & inadvertent manipulation of the system. We support teams from the design phase through the production to launch and surely after.
The HireCA team has skills for reviewing code in C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, and JavaScript for common security vulnerabilities & specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture, including in crypto currency , block chains, payments, and smart contracts. Additionally, the team can utilize various tools to scan code & networks and build custom tools as necessary.
Although we are a small team, we surely believe that we can have a momentous impact on the world by being translucent and open about the work we do.

**For more information about our security consulting, please mail us at hi@hireca.com**