DESSERT FINANCE

BRIDGE.SOL

ERC-20 Audit

PERFORMED BY DESSERT FINANCE
FOR PROVIDED REPOSITORY

PRIVATE REPORT FOR ZCHAIN

PROJECT AUDIT

INITIAL DISCLAIMER

Dessert Finance provides due-diligence project entry level audits for various projects. Dessert Finance in no way guarantees that a project will not remove liquidity, sell off team supply, or otherwise exit scam

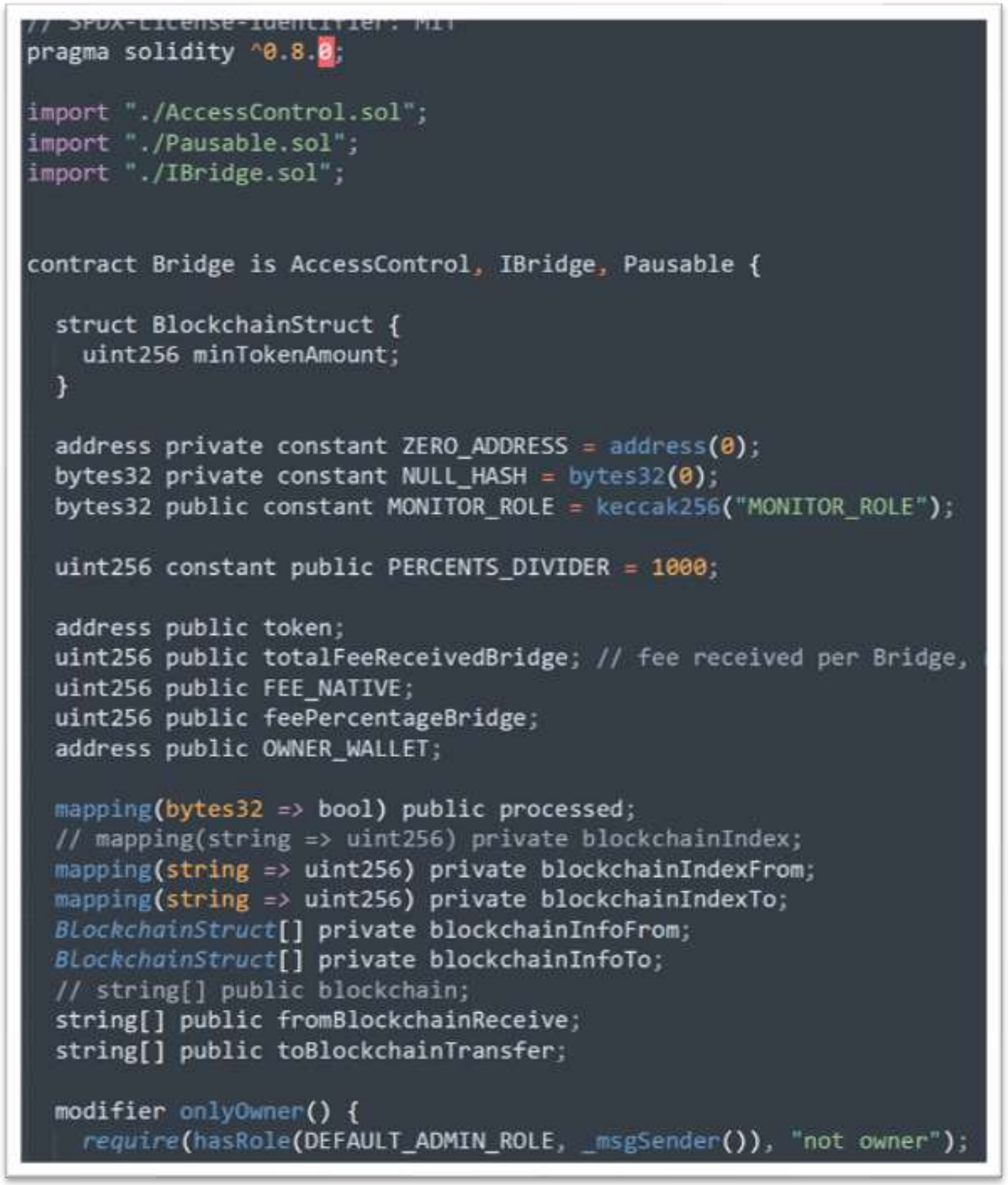
Dessert Finance does the legwork and provides public information about the project in an easy-to-understand format for the common person. In code audits dessert finance makes no guarantees or final claims. Please note that any recommendations for code improvements and additions are provided for informational purposes only. It is advisable to conduct a thorough review and testing before implementing any changes to ensure compatibility and security.

Dessert Finance in no way takes responsibility for any losses, nor does Dessert Finance encourage any speculative investments. The information provided in this audit is for information purposes only and should not be considered investment advice. Dessert Finance does not endorse, recommend, support, or suggest any projects that have been audited. An audit is an informational report based on our findings, We recommend you do your own research, we will never endorse any project to invest in.

Table of Contents

Solidity Contract Code Audit – Overview

Dessert Finance was commissioned to perform an audit on Bridge.sol



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./AccessControl.sol";
import "./Pausable.sol";
import "./IBridge.sol";


contract Bridge is AccessControl, IBridge, Pausable {

  struct BlockchainStruct {
    uint256 minTokenAmount;
  }

  address private constant ZERO_ADDRESS = address(0);
  bytes32 private constant NULL_HASH = bytes32(0);
  bytes32 public constant MONITOR_ROLE = keccak256("MONITOR_ROLE");

  uint256 constant public PERCENTS_DIVIDER = 1000;

  address public token;
  uint256 public totalFeeReceivedBridge; // fee received per Bridge,
  uint256 public FEE_NATIVE;
  uint256 public feePercentageBridge;
  address public OWNER_WALLET;

  mapping(bytes32 => bool) public processed;
  // mapping(string => uint256) private blockchainIndex;
  mapping(string => uint256) private blockchainIndexFrom;
  mapping(string => uint256) private blockchainIndexTo;
  BlockchainStruct[] private blockchainInfoFrom;
  BlockchainStruct[] private blockchainInfoTo;
  // string[] public blockchain;
  string[] public fromBlockchainReceive;
  string[] public toBlockchainTransfer;

  modifier onlyOwner() {
    require(hasRole(DEFAULT_ADMIN_ROLE, _msgSender()), "not owner");
```

Main Contract Name

Bridge

Solidity Version

^0.8.0

License

SPDX-License-Identifier: MIT

Import Files

import "./AccessControl.sol";

import "./Pausable.sol";

import "./IBridge.sol";

Ownability

The contract is owned by an address that has the DEFAULT_ADMIN_ROLE and includes specific functions restricted by the onlyOwner modifier, ensuring only the owner can execute them.

Modifiers

onlyOwner

onlyMonitor

onlyEOAs

The contract code is not yet deployed.

Solidity Contract Code Audit – Bridge.sol (1/2)

Item	Description	Risk Level
Solidity assert violation	Potential for assertion failures, although not present in the code	Low
Integer overflow in arithmetic operation	Properly handled with SafeMath, but no specific library is used	Medium
Integer underflow in arithmetic operation	Properly handled with SafeMath, but no specific library is used	Medium
Caller can redirect execution to arbitrary locations	Not observed in the code	Low
Caller can write to arbitrary storage locations	Not observed in the code	Low
Dangerous use of uninitialized storage variables	Not observed in the code	Low
Any sender can withdraw Coins from the contract account	Only owner can withdraw tokens, protected by modifiers	Low
Any sender can trigger Authorization Control	Authorization control is properly handled using roles	Low
Use of "tx.origin" as a part of authorization control	Present in `onlyEOAs` modifier, considered insecure	Medium
Delegatecall to a user-supplied address	Not observed in the code	Low
Call to a user-supplied address	Present in `_sendToken` and other functions, but safely used with checks	Low
Block timestamp influences a control flow decision	Not observed in the code	Low
Loop over unbounded data structure	Several functions loop over arrays, potential for gas limit issues	Medium

The contract code is not yet deployed.

The vulnerabilities listed above were not found in the token's Smart Contract.

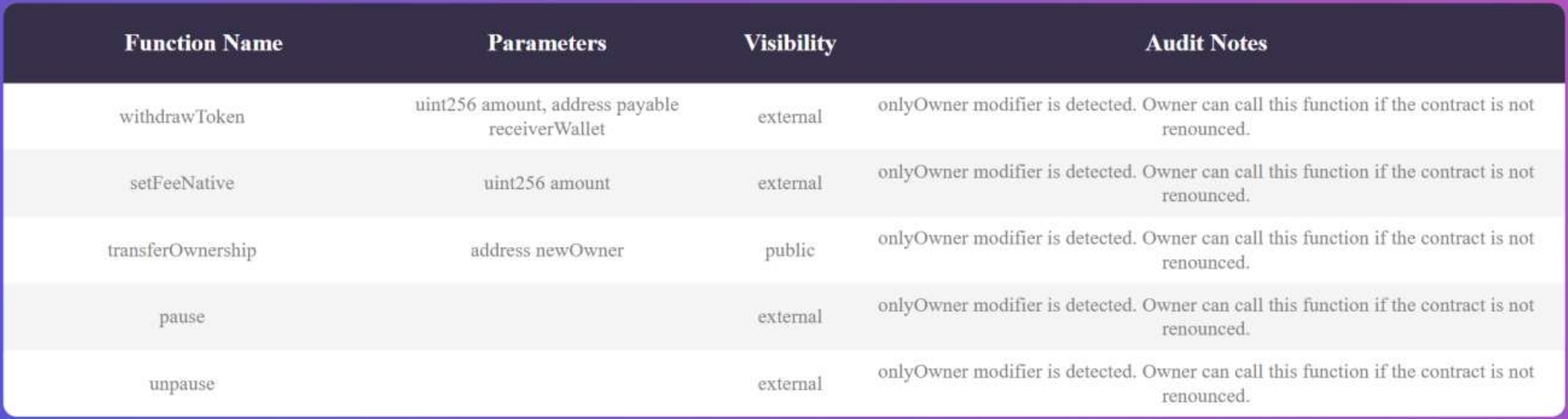
Solidity Contract Code Audit – Bridge.sol (2/2)

Item	Description	Risk Level
Usage of "continue" in "do-while"	Not observed in the code	Low
Persistent state read following external call	Not observed in the code	Low
Persistent state write following external call	Not observed in the code	Low
Account state accessed after call to user-defined address	Not observed in the code	Low
Return value of an external call is not checked	Properly handled in `_sendToken` and other functions	Low
Potential weak source of randomness	Not observed in the code	Low
Requirement violation	Properly handled with `require` statements	Low
Call with hardcoded gas amount	Not observed in the code	Low
Incorrect token implementation	Uses standard ERC20 functions, appears correct	Low
Function parameter shadows a state variable	Not observed in the code	Low
Unary operation directly after assignment	Not observed in the code	Low
Unary operation without effect	Not observed in the code	Low
Unused state variable	None observed	Low
Unused local variable	None observed	Low
Function visibility is not set	Proper visibility set on all functions	Low
Use of deprecated functions: callcode(), sha3(), etc.	Not observed in the code	Low
Use of deprecated global variables (msg.gas, ...)	Not observed in the code	Low
Use of deprecated keywords (throw, var)	Not observed in the code	Low
Incorrect function state mutability	Proper mutability set on all functions	Low

The contract code is not yet deployed.

The vulnerabilities listed above were not found in the token's Smart Contract.

Contract Code Audit – Owner Accessible Functions – Bridge.sol



Function Name	Parameters	Visibility	Audit Notes
withdrawToken	uint256 amount, address payable receiverWallet	external	onlyOwner modifier is detected. Owner can call this function if the contract is not renounced.
setFeeNative	uint256 amount	external	onlyOwner modifier is detected. Owner can call this function if the contract is not renounced.
transferOwnership	address newOwner	public	onlyOwner modifier is detected. Owner can call this function if the contract is not renounced.
pause		external	onlyOwner modifier is detected. Owner can call this function if the contract is not renounced.
unpause		external	onlyOwner modifier is detected. Owner can call this function if the contract is not renounced.

The functions listed above can be called by the contract owner.

If contract ownership has been renounced there is no way for the above listed functions to be called.

Disclaimer

The opinions expressed in this document are for general informational purposes only and are **not intended to provide specific advice or recommendations for any individual or on any specific investment**. It is only intended to provide education and public knowledge regarding projects. This audit is only applied to the type of auditing specified in this report and the scope of given in the results. Other unknown security vulnerabilities are beyond responsibility. Dessert Finance 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, Dessert Finance lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The smart contract analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Dessert Finance or was publicly available before the issuance of this report (issuance of report recorded via block number on cover page), 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, Dessert Finance assumes no responsibility for the resulting loss or adverse effects. Due to the technical limitations of any organization, this report conducted by Dessert Finance still has the possibility that the entire risk cannot be completely detected. Dessert Finance disclaims any liability for the resulting losses.

Dessert Finance provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Even projects with a low risk score have been known to pull liquidity, sell all team tokens, or exit-scam. Please exercise caution when dealing with any cryptocurrency related platforms.

The final interpretation of this statement belongs to Dessert Finance.

Dessert Finance highly advises against using cryptocurrencies as speculative investments and they should be used solely for the utility they aim to provide.

Thank You

DESSERT FINANCE PROJECT AUDIT HAS BEEN COMPLETED FOR ZCHAIN

THIS AUDIT IS ONLY VALID IF VIEWED ON HTTPS://WWW.DESSERTSWAP.FINANCE

www.dessertswap.finance
https://t.me/dessertswap