BlockÂpex

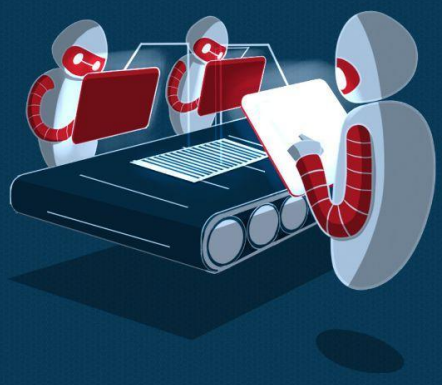
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



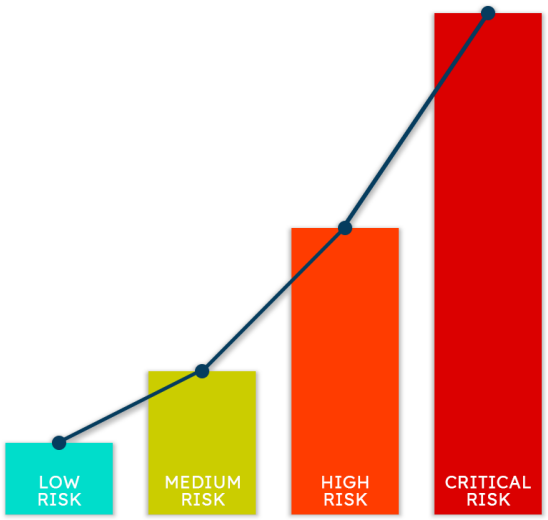
PREFACE

Objectives

The purpose of this document is to highlight the identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and intellectual property of the client. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below.

The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly with the intention to aid our growing blockchain community; under the discretion of the client.

Key Understandings



CRITICAL RISK ××××	Critical vulnerabilities are too easy to exploit and can lead to damages/loss in assets or manipulations.
HIGH RISK ×××	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution.
MEDIUM RISK ××	Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on asset loss or data manipulations.
LOW RISK ×	Lowest-level vulnerabilities, informational errors, violating code styles/practices usually can't affect smart contract execution; hence they can be ignored.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

TABLE OF CONTENTS

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

INTRODUCTION

BlockApex (Auditor) was contracted by __Dafi__ (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place from ___27th November 2021____.

Name
Dafi bridge
Auditor
Moazzam Arif | Muhammad Jarir
Platform
Ethereum/Solidity
Type of review
ETH - BSC Bridge
Methods
Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Git repository
https://github.com/DAFIProtocol/dBridge/tree/dafi-bridge-contracts
White paper/ Documentation
Dafi Bridge Flow Document (1).pdf
Document log
Initial Audit: 30th November 2021
Final Audit: 2nd December 2021

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect **major issues/vulnerabilities**. Some specific checks are as follows:

Code review		**Functional review**
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	ERC20 API violation	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset's integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop		Operation Trails & Event Generation

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

Project Overview

A detailed overview of the project can be found here:
https://blog.dafiprotocol.io/introducing-the-dbridge-testnet-c564f5b4eea2

Dafi's "dbridge" enables users to bring their ERC-20 $DAFI tokens across from the Ethereum network to Binance Smart Chain, and vice versa, with aims of making $DAFI available on multiple high-speed and low-cost networks. As Dafi's goal is to support every token on most chains, to launch their own dToken, it's important that their protocol is cross-chain.

Dafi Bridge is an implementation of a generic POA Bridge. Authority is distributed among validators. Validators sign the proof-of-burn message and the user submits (to avoid *gas griefing* attacks) the signature on the alternate chain to claim tokens.

System Architecture

A basic user flow described by team Dafi is as follows:

Alice wishes to bridge her DAFI tokens from Ethereum to Binance, she first approves the bridge request of the said tokens a nd then confirms locking those tokens in the dBridge on Ethereum. Once Alice's transaction gets confirmed, an event is emitted containing the *Receiving Chain Address* and *Amount*. Using these details a merkle tree is generated for all locking transactions that happened within the same block as Alice. The root of this specific Merkle Tree is stored in the opposing chain's *rootHeader* mapping.

When a user comes to claim/mint their tokens on the receiving chain, the smart contract expects:

1. a proof of the user's claim
2. the root of the merkle tree the users transaction was included in
3. the encoded transaction data which was emitted in the event,

It checks the provided root against its root storage to see if the provided root is among the roots of the merkle trees generated by locking tokens on the preceding chain.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

The dBridge then checks that the encoded leaf which we class a TX is not already claimed. It then verifies that the provided tx (leaf) & proof match the root provided. If all of these checks are passed, The user can mint or claim their tokens.

If the user wishes to bridge their assets back to Ethereum, he has to once again approve the dBridge on the Binance side of the said tokens, after his tokens are burned, an event is generated containing his tx details. Again a merkle tree is generated, this time the root is stored on the Ethereum side. To unlock his tokens the user provides a proof of his transaction, merkle root of the three his transaction was included in and his encoded tx (leaf).

If all of this checks out and merkle proof is verified he is able to claim/unlock his tokens back on the Ethereum side.

The codebase:

The system consists of 3 smart contracts (i.e ETH Bridge, BSC Bridge & a burnable/mintable ERC20 token representing Dafi on alternate chains)

Bridge contracts have ***onlyOwner*** modifier to set configurations (i.e adding/removing validators, minimum signers required(threshold)).

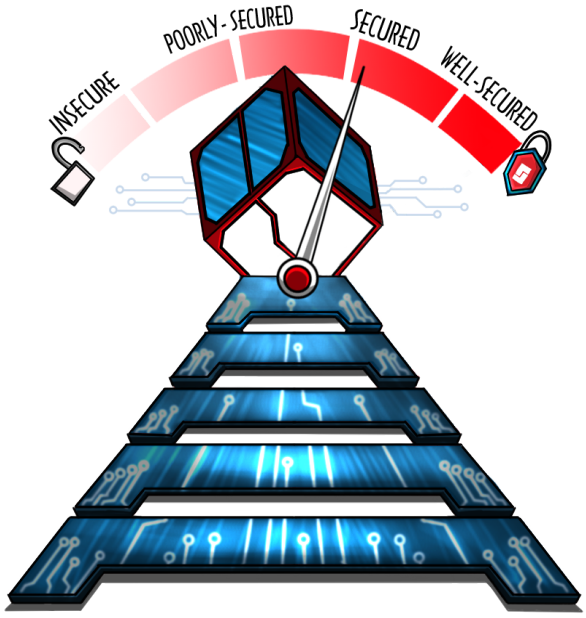
The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

AUDIT REPORT

Executive Summary

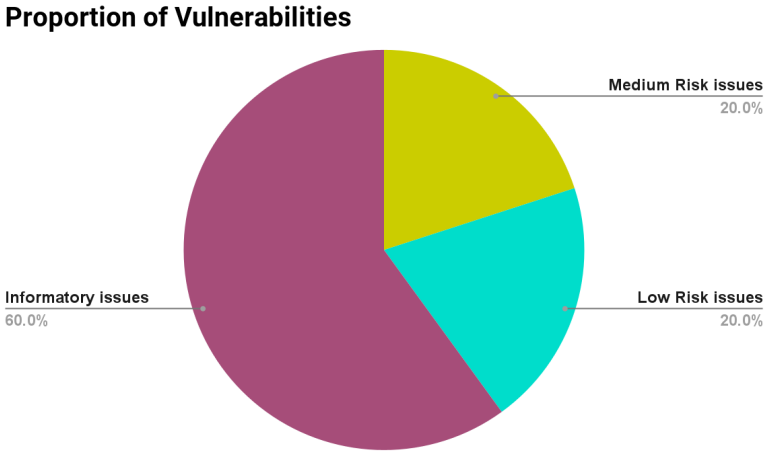
The analysis indicates that the contracts audited are **working properly**.

Our team performed a technique called “Filtered Audit”, where the contract was separately audited by two individuals. After their thorough and rigorous process of manual testing, no potential flags were raised.



Our team found:

# of issues	Severity of the risk
0	Critical Risk issue(s)
0	High Risk issue(s)
1	Medium Risk issue(s)
1	Low Risk issue(s)
3	Informatory issue(s)



The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

Findings

Critical-risk issues

No critical-risk issues were found in the review.

High-risk issues

No high-risk issues were found in the review.

Medium-risk issues

1. **If ETHToken is changed, tokens will be locked forever in the contract**

   File: *ETHBridgeOptimized.sol*

   Description:

   Owner(multisig) can change the underlying *ETHToken* address. If there are tokens locked in the smart contract and *changeToken()* is called, tokens will be locked forever in the contract.

   Remedy:

   Remove *changeToken()* method from the contract. If there is a need to change the token address, a new contract can be deployed.

   Status:

   Fixed.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

Low-risk issues

1. **Unnecessary *allowance* check in burn/lock tokens**

   File: *ETHBridgeOptimized.sol, BinanceBridgeOptimized.sol*

   Description:

```
if (IERC20(BSCTOKEN).allowance(msg.sender, address(this)) < amount)
revert AllowanceInsufficient(
IERC20(BSCTOKEN).allowance(msg.sender, address(this)), amount);
```

   When burning/locking tokens, the contract checks if the *msg.sender* has allowed the smart contract (*address(this)*) to burn/lock. This check will cost extra gas. Normally these checks are used with *burnFrom.* There is already check placed in the *burn* method of ***dafiToken.sol***

   Remedy:

   Remove allowance checks

   Status:

   Fixed.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

Informatory issues and Optimization

1. **Misleading variable/function names**

   File: *ETHBridgeOptimized.sol, BinanceBridgeOptimized.sol, dafiTokenBSC.sol*

```
function burnTokens(uint256 amount, address targetChain) external {

function lockTokens(uint256 amount, address targetChain) external {

function viewOwners(address checkAddress) external view returns
(bool) //viewValidators

function burn(uint256 _value, address _beneficiary) external
onlyBridge { // burnFrom
```

   Remedy:

   Our suggestion is that
   1. *targetChain* should be *recipient*
   2. *viewOwners* should be *ViewValidators*
   2. *burn* should be *burnFrom* in dafiToken

2. **Gas cost optimization while incrementing nonce**

   File: *ETHBridgeOptimized.sol, BinanceBridgeOptimized.sol*

   Description:

   nonceIncrement() method is used to increment a state variable *nonce*. Calling a function instead of directly updating the state variable will save the gas cost. Calling a function introduce ***JUMP*** opcode which has a higher gas cost

   Status:

   Fixed.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

3. Centralization risk on minting/burning on BSC Token

Description:

onlyBridge can burn/mint tokens. Bridge address can be changed by the owner (MultiSig). There are no potential risks as long as the signers of the multisig are honest.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices till date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token, its sale or any other aspect of the project.

Crypto assets/tokens are results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third-party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks.

This audit cannot be considered as a sufficient assessment regarding the utility and safety of the code, bug-free status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

The contents of this document are proprietary and highly confidential. Information from this document should not be extracted/disclosed in any form to a third party without the prior written consent of BlockApex.