



BlockApex

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";
    }
}
```



Powered by XORD

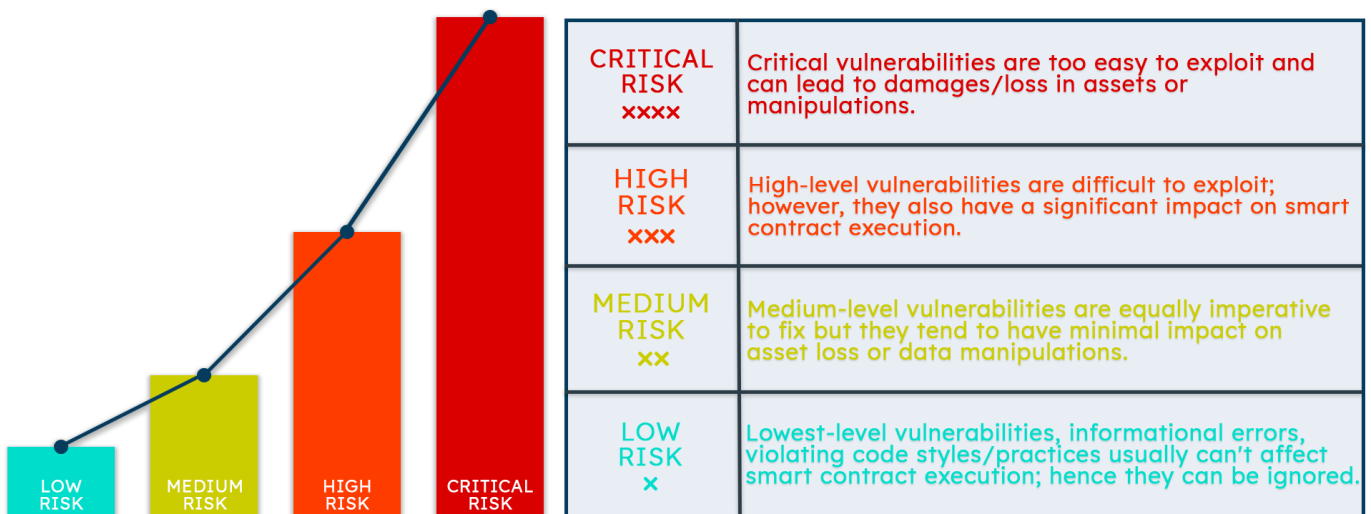
PREFACE

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



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.

[BlockApex | Fortifying The Move Towards Decentralization](#)

Table of Contents

PREFACE	2
Objectives	2
Key understandings	2
INTRODUCTION	4
Scope	5
Project Overview	6
System Design	6
Test Cases	6
AUDIT REPORT	7
Executive Summary	7
Findings	8
Critical-risk issues	8
High-risk issues	8
Medium-risk issues	9
Low-risk issues	10
Informatory issues	10
DISCLAIMER	11

INTRODUCTION

BlockApex (Auditor) was contracted by Sonar (Client) for the purpose of conducting a Smart Contract Audit/Code Review. This document presents the findings of our analysis which took place on 8th September 2021.

Name
Sonar BSC-ETH bridge
Auditor
Moazzam Arif Kaif Ahmed
Platform
Ethereum/Solidity
Type of review
Bridge
Methods
Architecture Review, Functional Testing, Manual Review
Git repository
https://github.com/sonarplatform/eth_bridge/tree/4c086d33fb5cb7a04a2192d6566c7905ea3b1074
White paper/ Documentation
Not provided
Document log
Initial Audit (08-09-2021)
Final Audit (pending)



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



Project Overview

This is a POA bridge application that provides eth to bsc and bsc to eth bridging. It allows its native tokens to be accessed on both ETH and BSC blockchains.

System Design

Steps to burn and mint (BSC => ETH):

1. User calls the burn method on BSC bridge (BSC blockchain)
2. With the burn method, tokens are transferred to the admin account
3. Transfer event is emitted on successful burn
4. Relayer listens the event and call the mint method on ETH bridge
5. With the mint method, the tokens are transferred to the user on the ethereum blockchain

For ETH => BSC similar steps are followed.

Test Cases

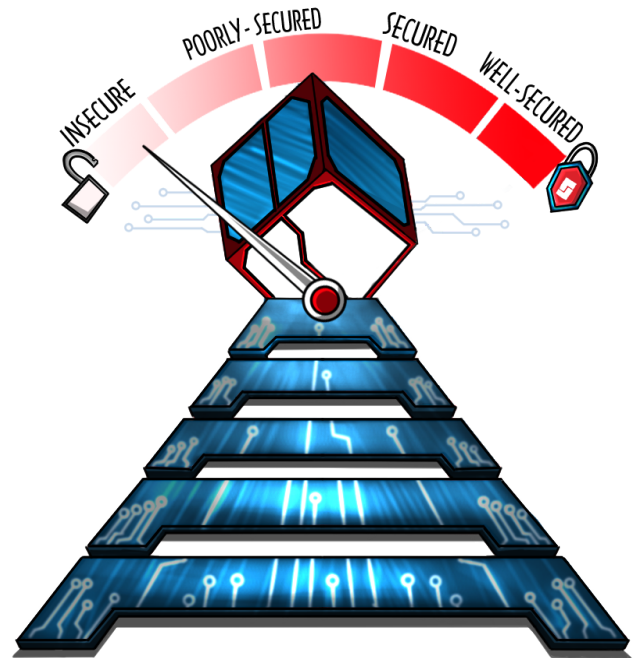
No test cases were provided, we did our own simulations.

AUDIT REPORT

Executive Summary

The analysis indicates that the contracts audited are **insecure**.

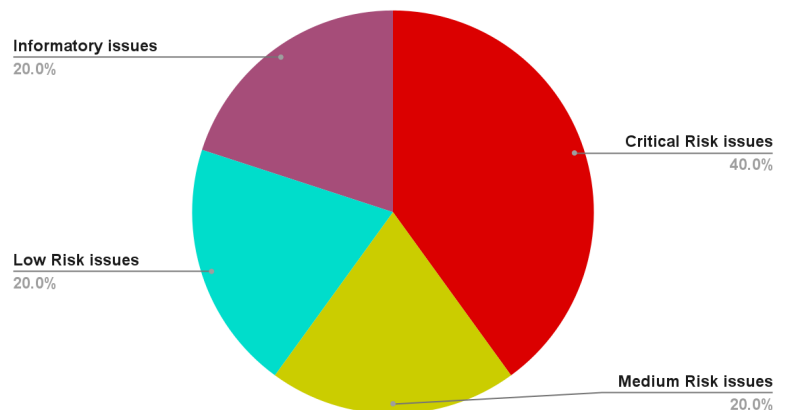
Our team performed a technique called “Filtered Audit”, where the contract was separately audited by two individuals. After a thorough and rigorous process of manual testing, all the flags raised were iteratively reviewed and tested.



Our team found:

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

Proportion of Vulnerabilities



Findings

Critical-risk issues

1. Anyone can lock funds of other users

If user1 wants to transfer tokens from BSC to ETH, user1 will first approve funds to the BSC bridge. Now that the user has approved his funds to the bridge contract, anyone else can call the **burn** method passing the address of user1, the funds of user1 have been locked by someone else.

Remedy:

Burn function contains following logic

```
token.transferFrom(to, address(this), AmountwithFees);
```

The parameter contains “to” but should contain “msg.sender”

2. Server API to mint the tokens

There is an api written in the server to mint tokens. It does not verify that the burn has been called or not. The users can mint without burning using that api.

Remedy:

User should submit proof of burn before calling the minting API

High-risk issues

No high-risk issues were found.

Medium-risk issues

1. Locking/Unlocking both side

The contract has locking and unlocking functionality on both sides. For minting it is assumed that the admin wallet should have enough token balance.

Attack scenario:

- a. User 1 locks 1000 tokens on BSC bridge
- b. There is not enough token balance equivalent to be given to user1 on ETH bridge

```
require( adminBal > amount, "insuffient funds in admin account.");
```

- c. Now user1 can not unlock on ETH bridge
 - i. He will have to unlock his token again to get his funds back from BSC bridge (extra gas, because the function is not performing the intended behavior)
 - ii. Unlocking of tokens on the same bridge is inconvenient from user's perspective

Remedy:

Usually the lock/unlock strategy is used on one bridge and mint/burn strategy is used on the other bridge.

Low-risk issues

1. Centralization Risk

Admin wallet is a single wallet and there is high risk of centralization if the private key gets compromised.

```
require(admin == msg.sender, "only admin can call this");
```

Remedy:

Assignment of privileged roles to multi-signature wallets to prevent single point of failure due to the private key.

Informatory issues

1. Lock pragma versions

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remedy:

Lock the pragma version and also consider known bugs (<https://github.com/ethereum/solidity/releases>) for the compiler version that is chosen. Please refer to this doc for more details.

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.