SecureBlock

Security Audit

T-Swaps Bridge Audit

Prepared for:
T-Swaps
www.tswaps.com

Prepared by:
SecureBlock
www.secureblock.io

Table of Contents

1 Introduction

This document includes observations and findings during the audit of the smart contract.

a. About SecureBlock

Founded in 2021 by an association of experts in the field of computer security with many years of experience.

Our researchers are continuously working on the development of internal tools and knowledge sharing, as well as by holding recognized certificates in the industry such as OSCP, OSCE, OSWE, CEH, CISSP.

Our mission is to simplify and provide a quality security testing service for blockchain projects and technologies. Taking an individual approach and manual review of each project allows us to better understand use case of the applications and find vulnerabilities and problems that standard automated tools will not find.

We believe that openness and trust are one of the key aspects of blockchain technology, which is increasingly finding its purpose in more and more industries. For this reason, our clients have an insight into the state of security testing, a preliminary description of vulnerabilities and the public management of the final report through an application we have developed internally.

b. Purpose of the audit

The purpose of the testing was primarily to find security issues, as well as compliance of the code with best practice and, if possible, reduce the gas fee.

2 Executive Summary

a. Results

The conducted testing indicates that the tested smart contract is very safe and follows best practices in decentralized smart contract development.

A total of 1 issue was found during the manual review of the smart contract. Identified issue is related to floating pragma version. More information is given in the vulnerability description.

b. Scope

Contract Name	**T-Swaps Bridge**
Language	Solidity
Network	Telos
File Hash (SHA256)	teleportaudit.zip a4d458fac1c91e817097c56c2c0638433a72820795367b1d7cf020 72ce76a00a
Network Address	0x7a82dE1B2159C33C8d93cd9Dbb06350820458E4f
TeleportTokenFactory Bytecode Hash	18d813eb7175e3f5446b6c68fe500927babe46b5f204c66cc62eb76 0aa857b7f
TeleportToken Bytecode Hash	e7ae8cc63e4d309f01b3cef5d8caa9f58ea120fd63ce594ea3cfdd50f 6077f93
Owner Bytecode Hash	54aa3578bd43b53a1382522795291e32b5e360f3647ab9e15138a 31c90dad25d

3 Attack Narrative

a. Checklist

In order to find vulnerabilities during the test, we go through a checklist that helps us to cover more tests as well as demonstrate to the client which checks were included during testing. In addition to the list below, we check for business logic vulnerabilities that we find on the deployed contract on our local private network so that there are no unexpected consequences for users.

Name	Description
ERC standards	The contract is using ERC standards.
Compiler Version	The compiler version should be specified.
Constructor Mismatch	The constructor syntax is changed with Solidity versions. Need extra attention to make the constructor function right.
Return standard	Following the ERC20 specification, the transfer and approve functions should return a bool value, and a return value code needs to be added.
Address(0) validation	It is recommended to add the verification of require(_to!=address(0)) to effectively avoid unnecessary loss caused by user misuse or unknown errors
Unused Variable	Unused variables should be removed.
Untrusted Libraries	The contract should avoid using untrusted libraries, or the libraries need to be thoroughly audited too.
Event Standard	Define and use Event appropriately

Name	Description
Safe Transfer	Using transfer to send funds instead of send.
Gas consumption	Optimize the code for better gas consumption.
Deprecated uses	Avoid using deprecated functions.
Sanity Checks	Sanity checks when setting key parameters in the system
Integer overflows	Integer overflow or underflow issues.
Reentrancy	Avoid using calls to trade in smart contracts to avoid reentrancy vulnerability.
Transaction Ordering Dependence	Avoid transaction ordering dependence vulnerability.
Tx.origin usage	Avoid using tx.origin for authentication.
Fake recharge	The judgment of the balance and the transfer amount needs to use the “require function”.
Replay	If the contract involves the demands for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks.
External call checks	For external contracts, pull instead of push is preferred.
Weak random	The method of generating random numbers on smart contracts requires more considerations.
Access Control	Well defined access control for functions.

Name	Description
Authentication management	The authentication management is well defined.
Semantic Consistency	Semantics are consistent
Functionality checks	The functionality is well implemented.

4 Identified Vulnerabilities

Issue ID	Severity	Title	Status
APP-01	Info	Floating pragma	Acknowledged

APP-01 - Floating pragma

The floating pragma found indicates that the code can be compiled with any version of Solidity newer than ^0.8.6 because the version starts with ^ (caret sign).

Status: Acknowledged

5 Additional Notes

The following notes refer to potential vulnerabilities that are not exploitable in the current environment, but we would like to draw attention as they could cause unexpected behavior in the future if contract gets updated or deployed in different environment.

- *TeleportToken.sol:445* - Transaction ID should be marked as claimed after making sure it has been signed by the oracle inside *claim()* function in case *verifySigData()* becomes public function, the risk of unauthorized marking IDs as claimed arises.

6 Vulnerability Remediation

Detailed remediation steps for found issues can be found by project owner at **https://secureblock.io/dashboard**