SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: TeraBlock

Date: July 06th, 2022

This document may contain confidential information about IT systems and the intellectual property of the Customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the Customer, or it can be disclosed publicly after all vulnerabilities are fixed — upon a decision of the Customer.

Document

Name	Smart Contract Code Review and Security Analysis Report for TeraBlock
Approved By	Noah Jelich | Senior Solidity SC Auditor at Hacken OU
Type	ERC-20 token bridge helper
Platform	EVM
Language	Solidity
Methods	Manual Review, Automated Review, Architecture review
Website	https://terablock.com/
Timeline	15.06.2022 - 06.07.2022
Changelog	27.06.2022 - Initial Review 06.07.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by TeraBlock (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
    https://github.com/TeraBlock/swidge-contracts
Commit:
    10ca3c0f9b99f94280cdce7ecf6b6276514cdf49
Technical Documentation:
    Type: Technical description
    https://docs.google.com/document/d/1JcipE9ZGBbOjvjL_PFl87tV6Uq1nxQxoWQr0bSu-Rts/edit?usp=sharing

Integration and Unit Tests: Yes (https://github.com/TeraBlock/swidge-contracts/blob/10ca3c0f9b99f94280cdce7ecf6b6276514cdf49/test/Swidge.ts)
Deployed Contracts Addresses: No
Contracts:
    File: ./contracts/Swidge.sol
    SHA3: 04a2252d0911e7d28c45a5ab85800b26d5d8edc33dd6c370528b28893cb29b90

Second review scope

Repository:
    https://github.com/TeraBlock/swidge-contracts-v1/
Commit:
    2efa2073ab61a81926fafdd0234d76002c96d7e9
Technical Documentation:
    Type: Technical description
    https://docs.google.com/document/d/1JcipE9ZGBbOjvjL_PFl87tV6Uq1nxQxoWQr0bSu-Rts/edit?usp=sharing

Integration and Unit Tests: Yes (https://github.com/TeraBlock/swidge-contracts-v1/blob/main/test/Swidge.ts)
Deployed Contracts Addresses: No
Contracts:
    File: ./contracts/Swidge.sol
    SHA3: 71b579ac66eac5eedd94d16d4e1a6b00cfffe05f9d5ed5170fa89df1690e7675

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurement details can be found in the corresponding section of the methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional and technical requirements are provided.

Code quality

The total CodeQuality score is **8** out of **10**. Code follows the Style guide recommendations. The code is mostly commented. The unit test covers 70% of functions.

Architecture quality

The architecture quality score is **10** out of **10**. The project has clear and clean architecture.

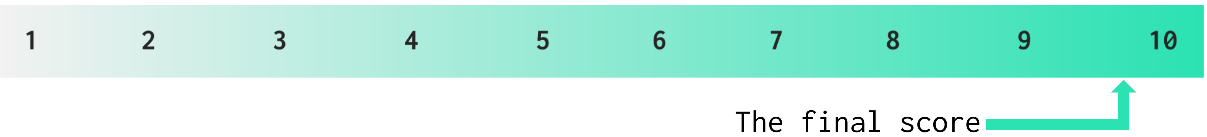
Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.8**.



Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Not relevant
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution	Not Relevant

		fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Swidge contract allows to swap and send tokens to the relevant bridge contract. According to the documentation, the contract uses *1inch* to swap the tokens. The contract takes a fee after each swap. The fee percent is set by the owner of the contract.

Privileged roles

- The owner of the contract may:
  - Pause or unpause the contract;
  - Set bridge address for a specific token;
  - Set fee percent;
  - Withdraw any tokens from the contract.

Risks

- All the bridge contract logic is out of the audit scope. The audited contract only swaps and sends tokens to the bridge contract.
- The contract is upgradable, this allows the admin to change the contract implementation logic.
- Ensure that the contract was deployed with the correct 1inch *exchange* address.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

No checks to prevent percentage overflow.

The contract has the function setFeePercent which allows the admin to update the fee, but the function does not have the value validation, the fee may be greater than 100 percent.

If the fee value is greater than 100 percent, the contract functionality will be blocked.

Contracts: ./contracts/Swidge.sol

Function: initialize, setFeePercent

Recommendation: Add conditional or require statements to validate the input data.

Status: Fixed (8311d5839136f40157b97040c5a67921200a5d29)

🟩🟩 Medium

Redundant allowance value.

The contract has the function _checkAllowance which checks if the allowance value is sufficient. However, if the token allowance value is lower than needed, it calls the approve function and approves the maximum uint value.

This may lead to funds leakage (e.g. accumulated fees) from the Swidge contract if the contract (e.g. exchange, bridge) which requires the allowance is vulnerable.

Contracts: ./contracts/Swidge.sol

Function: _checkAllowance

Recommendation: Approve only the needed amount of tokens for a specific operation.

Status: Fixed (8311d5839136f40157b97040c5a67921200a5d29)

🟦 Low

1. Usage of the low-level calls.

The contract has the function which does call the exchange address.

Users may pass arbitrary data and call any function from the exchange contract.

Contracts: ./contracts/Swidge.sol

Function: _swap

Recommendation: It is recommended to use a predefined interface for interaction with the `exchange` contract.

Status: Reported

2. **Missing zero address validation.**

   Address parameters are being used without checking against the possibility of `0x0`.

   This can lead to unwanted external calls to `0x0`.

   Contracts: ./contracts/Swidge.sol

   Function: initialize, setBridge

   Recommendation: Implement zero address validations.

   Status: Fixed (8311d5839136f40157b97040c5a67921200a5d29)

3. **Missing event emitting.**

   Events for critical state change (e.g. fee percent update) should be emitted for tracking things off-chain.

   Contracts: ./contracts/Swidge.sol

   Function: setFeePercent

   Recommendation: Create and emit a related event.

   Status: Fixed (8311d5839136f40157b97040c5a67921200a5d29)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. 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 the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.