HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: DeRace
Date: May 17th, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for DeRace
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Bridge
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://derace.com
Timeline	02.05.2022 - 17.05.2022
Changelog	06.05.2022 - Initial Review 12.05.2022 - Second Review 21.05.2022 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by DeRace (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/Derace/bridge-contract
Commit:
6a26d259645ff4572b83e688ebad9b194ed22fd8
Technical Documentation:
Type: Technical description
Link: DeRaceBridge.md

JS tests: No
Contracts Addresses: None
Contracts:
File: ./contract.sol
SHA3: 9aaa4236d912c5e0adbc4f0b088b05b849dce08b71c8bbedb66f4ec598432ad0

Second review scope

Repository:
https://github.com/Derace/bridge-contract
Commit:
0e029d0a90efd1937aef8e03ccadd97a3472e27a
Technical Documentation:
Type: Technical description
Link: DeRaceBridge.md

JS tests: Yes
Contracts Addresses: None
Contracts:
File: ./contract.sol
SHA3: 440ca9a30f4f1ea5c9774c6720b7be130e1e4bf76ae6a6c1cc142ed90f78586b

Third review scope

Repository:
https://github.com/Derace/bridge-contract
Commit:
ad4f3bd813346b34504c9744997d137c8bb904e1
Technical Documentation:
Type: Technical description
Link: DeRaceBridge.md

JS tests: Yes
Contracts Addresses: None
Contracts:
File: ./contracts/DeRaceBridge.sol
SHA3: 2251ebecdc92147c121acda245c6011e3e948c5c10de1ff44b125573b076ba2e

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 Customer provided a technical description of the smart contract and functional details. The total Documentation Quality score is **7** out of **10**. The whitepaper was not provided.

Code quality

The total Code Quality score is **10** out of **10**.

Architecture quality

The architecture quality score is **9** out of **10**. The test environment is missing *typechain* module required for launching the tests.

Security score

As a result of the audit, security engineers found no issues. The security score is **10** out of **10**.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**

1
2
3
4
5
6
7
8
9
10
The final score

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.	Not Relevant
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 destroyed until it has funds belonging to users.	Passed
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
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.	Not Relevant
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.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed

Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Passed
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.	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.	Passed
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 fails due to the block Gas limit.	Passed
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
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed

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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Failed

System Overview

DeRaceBridge is the bridge that holds all the assets that are transferred off-chain. When assets are transferred back on the chain, they are redeemed from the bridge with a specific validator-signed message.

Privileged roles

- The creator of the contract is given *DEFAULT_ADMIN_ROLE.* This role can grant *SUPER_VALIDATOR_ROLE* and *VALIDATOR_ROLE* to the other addresses.
- *SUPER_VALIDATOR_ROLE* is essential for managing the contract. As such, it is able to pause or unpause the contract, skip a nonce for an address, and migrate different types of tokens to a certain address. This role should be used by multisig wallets only.
- *VALIDATOR_ROLE* is an off-chain entity that confirms off-chain transfers and transfers funds to a certain address.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

Highly permissive role access

The *SUPER_VALIDATOR_ROLE* has a right to migrate funds from the smart contract to any address without any restrictions.

Contract: DeRaceBridge.sol

Function: migrateErc20, migrateErc721, migrateErc721Any

Recommendation: set up certain prerequisites about when SUPER_VALIDATOR_ROLE is able to migrate funds or add a multi-signature request for a withdraw.

Status: Mitigated. The customer stated that wallets with such a role are multisig.

🟩🟩 Medium

No medium severity issues were found.

🟦 Low

1. **Interface casting Gas cost**

Converting *address* to be an interface is gas inefficient compared to converting *address* to an interface.

This increases transaction Gas.

Contract: DeRaceBridge.sol

Function: transferErc20

Recommendation: replace *address* with *IERC20* in the function signature.

Status: Fixed (c0345b2)

2. **Function visibility Gas cost**

Public visibility is used for functions that are not called internally.

This increases transaction Gas.

Contract: DeRaceBridge.sol

Function: transferErc20

Recommendation: replace function visibility from *public* to *external*.

Status: Fixed (c0345b2)

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.