HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Centrality
Date: September 29th, 2021

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 Centrality - Audit report
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Bridge validator
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Github	https://github.com/cennznet/bridge-contracts/tree/047cecbfc86f10cdc3310c6ebb399de2e7c737a3
Timeline	24 September 2021 - 7 October 2021
Changelog	24 September 2021 - 29 September 2021 - Initial Audit 29 September 2021 - 7 October 2021 - Remediation check

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Centrality (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on September 29th, 2021.

Remediation check was conducted - October 7th, 2021

Scope

The scope of the project is the smart contracts:

Repository: https://github.com/cennznet/bridge-contracts
Last commit: 047cecbfc86f10cdc3310c6ebb399de2e7c737a3
Contracts:
contracts/CENNZnetBridge.sol
contracts/ERC20Peg.sol

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

Category	Check Item
Code review	▪ Reentrancy ▪ Ownership Takeover ▪ Timestamp Dependence ▪ Gas Limit and Loops ▪ DoS with (Unexpected) Throw ▪ DoS with Block Gas Limit ▪ Transaction-Ordering Dependence ▪ Style guide violation ▪ Costly Loop ▪ ERC20 API violation ▪ Unchecked external call ▪ Unchecked math ▪ Unsafe type inference ▪ Implicit visibility level ▪ Deployment Consistency ▪ Repository Consistency ▪ Data Consistency

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

Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Asset’s integrity ▪ User Balances manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

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

Executive Summary

According to the assessment, the Customer's smart contracts are Well-secured.

Insecure	Poor secured	Secured	Well-secured

You are here

Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril and Slither. All issues found during automated analysis were manually reviewed, and important vulnerabilities are presented in the Audit overview section. All found issues can be found in the Audit overview section.

Security engineers found 6 critical, 2 medium, 3 low and 1 informational issues during the first review.

There are no vulnerabilities found after remediation check.

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

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 can't lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that can't have a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can be ignored.

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

Audit overview

■ ■ ■ ■ Critical

1. RESOLVED Vulnerability: Event can be submitted without any verification.

   CENNZnetBridge.sol, verifyMessage()
   If the message is passed with proof.validatorSetId which is not registered yet or is cleaned (contains empty array of validators) the message is not validated at all. The reason is that the acceptanceTreshold is set to 0 in case of zero length of the validators array. Thus the event can be passed without any proof. Since the method is public, everyone can call it as a workaround.

   Recommendation: Add checks for the existence of the validators for the chosen set of validators.

2. RESOLVED Vulnerability: There is no way to override or exclude compromised validators.

   Both function setValidators() and forceSetValidators() can only add new validators and cannot change existing once - both functions have conditions for validatorSetId to be greater than activeValidatorSetId. Thus in the case of validators set compromised there is no way to change or deactivate it.
   Furthermore - function verifyMessage() does not check or validate the validatorsSetId passed as parameter and there is no connection to activeValidatorSet.
   Thus, the bridge contract will be compromised if validators are compromised.

   Recommendation: Add ability to exclude compromised validators.

3. RESOLVED Vulnerability: Anyone can withdraw funds from the bridge contract.

   Function setValidators() has no restriction to the caller (it is public for anyone) and since empty validators list can be passed (which will pass verifyMessage() function) - anyone can call the function and withdraw all collected currency.

   Recommendation: Provide restrictions for the caller and for the array of validators.

4. RESOLVED Vulnerability: Any token can be freely withdrawn.

ERC20Peg.sol, withdraw()
Based on the other issues for now any user can freely withdraw any amount of the token from the lock contract by manipulating with the array of validators.

Recommendation: The issue depends on other issues with restriction for validator's sets, restrictions for methods calls, etc. The overall recommendation is to rebuild the validation system.

5. RESOLVED Vulnerability: Any validators array can be set.

   Since there are no restrictions for the caller or for the array of validators and since passed validators verify their own messages - anyone can set any validators and use the bridge on its own. Thus regular users can be manipulated in order to send funds for malicious actors.

   Recommendation: Provide restrictions for the caller and for the array of validators.

6. RESOLVED Vulnerability: No restrictions on the message.

   There are no restriction on the message passed to the bridge contract. So, since anyone can call verifyMessage() function with any set of validators and signatures - anyone can manipulate storage in order to prevent correct events to be passed. So, for example, if event N should be passed to the bridge, malicious actor can send transaction with more gas in order to override eventsId[N] before the correct transaction. Thus user's tx will constantly fail

   Recommendation: The issue depends on other issues with restriction for validator's sets, restrictions for methods calls, etc. The overall recommendation is to rebuild the validation system.

High

No High severity issues were found.

Medium

1. RESOLVED Vulnerability: There are no restrictions for zero address of the receiver.

   ERC20Peg.sol, deposit(), withdraw()
   Any token can be maliciously (or mistakenly) sent to zero address.

   Recommendation: Provide restrictions

2. UNRESOLVED Vulnerability: There is no ability to pause CENNZ deposits.

   There is activate function but no pause function.

   Recommendation: Verify the functionality

Low

1. RESOLVED Vulnerability: Set variables as constants.

   CENNZnetBridge.sol, verificationFee, THRESHOLD
   ERC20Peg.sol, ETH_RESERVED_TOKEN_ADDRESS
   Variables are never changed and are set just once. Use public constants or add setters for these variables.

   Recommendation: Use constants.

2. RESOLVED Vulnerability: Use local storage for gas saving.

   CENNZnetBridge.sol, verifyMessage()
   The function contains multiple calls to validators[validatorsSetId] thus it creates a lot of calls to the storage. Use local memory variable to copy validators array just once, thus all other calls to the array will consume less gas. Since no storage change for the array is performed in this function it will work for gas savings.

   Recommendation: Use memory array to decrease gas usage.

3. RESOLVED Vulnerability: Use SafeERC20 library.

   ERC20Peg.sol, deposit(), withdraw()
   Since there are no restrictions for tokens use SafeERC20 library (safeTransfer and safeTransferFrom) for tokens in order to prevent fails for modified ERC20 tokens (like USDT).

   Recommendation: Use SafeERC20 library.

Lowest / Code style / Best Practice

1. RESOLVED Vulnerability: Use public constant for the address.

   ERC20Peg.sol, deposit()
   For better readability and code quality, move token address to the public constant.

   Recommendation: Use public constant.

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

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found 6 critical, 1 medium, 3 low and 1 informational issues during the first review.

There are no vulnerabilities found after remediation check.

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

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to 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 security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree 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.

Technical Disclaimer

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