SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Pocket Arena
Date: November 23rd, 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 Pocket Arena.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 tokens; Cross-chain transfer
Platform	Binance / Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/pocket-arena/POC_ERC20-BEP20
Commit	68c9a327e50c1ae3dad45f95cd104dfd98c78240
Deployed contracts	1. ERC20 Address: https://etherscan.io/token/0x095cf7f3e82a1dcadbf0fbc59023f419883ea296 2. BEP20 Address: https://bscscan.com/token/0x1b6609830c695f1c0692123bd2fd6d01f6794b98
Technical Documentation	YES
JS tests	NO
Website	pocketarena.com
Timeline	27 OCTOBER 2021 – 23 NOVEMBER 2021
Changelog	29 OCTOBER 2021 – INITIAL AUDIT 08 NOVEMBER 2021 - SECOND REVIEW 23 NOVEMBER 2021 - THIRD REVIEW

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Pocket Arena (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 contract and its code review conducted between October 27th, 2021 - October 29th, 2021.

Second code review conducted on November 8th, 2021.

Third code review conducted on November 23rd, 2021.

Scope

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

Repository:
    https://github.com/pocket-arena/POC_ERC20-BEP20
Commit:
    68c9a327e50c1ae3dad45f95cd104dfd98c78240
Technical Documentation: Yes, POC_BEP_Bridge.pdf
(md5: aca347a6ed24b998d37f762cf3833e40)
JS tests: No
Contracts:
    POC_BEP20.sol
    POC_ERC20.sol
Deployed contracts:

1. ERC20 Address:
https://etherscan.io/token/0x095cf7f3e82a1dcadbf0fbc59023f419883ea296

2. BEP20 Address:
https://bscscan.com/token/0x1b6609830c695f1c0692123bd2fd6d01f6794b98

We have scanned this smart contract 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
Functional review	▪ Business Logics Review ▪ Functionality Checks ▪ Access Control & Authorization ▪ Escrow manipulation ▪ Token Supply manipulation ▪ Assets integrity ▪ User Balances manipulation ▪ Data Consistency manipulation ▪ Kill-Switch Mechanism ▪ Operation Trails & Event Generation

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.

As a result of the audit, security engineers found **1** high and **4** low severity issues.

After the second review security engineers found that some contracts were slightly changed. Therefore found **1** medium and **1** low severity issue.

After the third review security engineers found that **all** issues were fixed.

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

Audit overview

🟥 🟥 🟥 🟥 Critical

No critical issues were found.

🟨 🟨 🟨 High

Vulnerability: Contracts are vulnerable to permanent blocking by any token holder.

Contracts:POC_ERC20.sol, POC_BEP20.sol

Functions: pegin_submit, pegout_submit

Arrays ERC20POC.arr_pegin_submit and BEP20POC.pegout_submit could be filled by the malicious token holder using the methods listed above. Every transaction will cost him/her a fixed amount of gas and the minimal amount of tokens. On the other hand, increasing size of that arrays will drastically increase the gas cost of methods pegout_submit_complete, pegout_submit_delete, pegout_submit_cancel, pegin_submit_complete, pegin_submit_delete, pegin_submit_cancel up to the gas limit of the block that resulting in permanent inoperability of these methods.

Recommendation: rewrite contracts to stop using regular arrays of unpredictable size, use mappings instead.

Status: fixed

🟩 🟩 Medium

Potential loss of users' submits and data inconsistency.

When several users call methods within one block, only the last one will create order because the key for storing data in arr_pegout_submit and arr_pegin_submit generate only based on block.timestamp

Contracts: POC_ERC20.sol, POC_BEP20.sol

Functions: pegin_submit, pegout_submit

Recommendation: Also use unique parameters to generate storage index, for example, msg.sender

Status: fixed

🟦 Low

1. Missing event for changing _fee_rate

   Contracts: POC_ERC20.sol, POC_BEP20.sol

Functions: _fee_rate_set

Changing critical values should be followed by the event emitting for better tracking off-chain.

Recommendation: Please emit events on the critical values changing.

Status: fixed

2. A public function that could be declared external.

   public functions that are never called by the contract should be declared **external** to save gas.

   Contracts: POC_ERC20.sol, POC_BEP20.sol

   Functions: pegin_run, remove_arr_pegin_reserve, transferFrom, staff_list, staff_del, staff_quota_add, staff_quota_minus, _fee_rate_get, fee_income, unlocked_POC_total, unlocked_POC_total_add, unlocked_POC_total_minus, pegout_submit, pegout_submit_list, pegout_submit_complete, pegout_submit_delete, pegout_submit_cancel, pegin_reserve, pegin_reserve_cancel, pegin_reserve_list, pegin_reserve_list, pegin_run, pegout_run, remove_arr_pegout_reserve, transferFrom, staff_list, staff_del, staff_quota_add, staff_quota_minus, _fee_rate_get, locked_POC_total, locked_POC_total_add, locked_POC_total_minus, pegin_submit, pegin_submit_list, pegin_submit_complete, pegin_submit_delete, pegin_submit_cancel, pegout_reserve, pegout_reserve_cancel, pegout_reserve_list, pegout_reserve_list, pegout_run

   Recommendation: Use the **external** attribute for functions never called from the contract.

   Status: fixed

3. Boolean equality

   Boolean constants can be used directly and do not need to be compared to true or false.

   Contracts: POC_ERC20.sol, POC_BEP20.sol

   Functions: transfer, transferFrom, staff_add, staff_quota_add, pegin_reserve, pegout_reserve

   Recommendation: remove the equality to the boolean constant.

   Status: fixed

4. Code and documentation inconsistency.

   Contracts: POC_ERC20.sol, POC_BEP20.sol

   Functions: _fee_rate_set

According to documentation maximum fee should be 100%, but contracts allow to set it up to 1000%.

Recommendation: update contracts or documentation

Status: fixed

Conclusion

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

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

As a result of the audit, security engineers found **1** high and **4** low severity issues.

After the second review security engineers found that some contracts were slightly changed. Therefore found **1** medium and **1** low severity issue.

After the third review security engineers found that **all** issues were fixed.

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 the security of the code. It also 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 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 can't guarantee the explicit security of the audited smart contracts.