HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Xpocket
Date: August 10th, 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 Xpocket.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 Token
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/xpocket/PocketSwap
Commit	df519c25e9330da8eebe3a4c5c446b5d3711ead4 f5229cca6540da423ec4f3e571d2121e802035cd
Timeline	29 JULY 2021 - 10 AUGUST 2021
Changelog	30 JULY 2021 - INITIAL AUDIT 10 AUGUST 2021 - SECOND REVIEW

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 Xpocket (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 between July 29th, 2021 - July 30th, 2021. The second review conducted on August 10th, 2021.

Scope

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

```
Repository: https://github.com/xpocket/PocketSwap
Commit: f5229cca6540da423ec4f3e571d2121e802035cd

Files:
     contracts/Pocket.sol
```

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

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 ▪ 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.

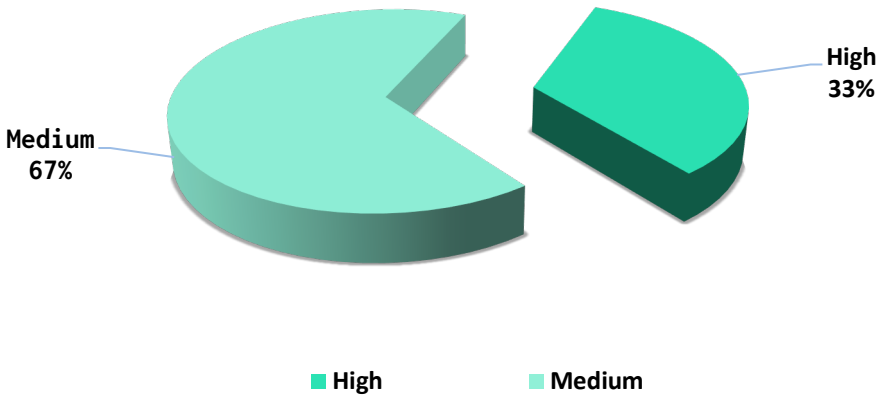
Insecure	Poor secured	Secured	Well-secured
		You are here	

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 **2** medium severity issues.

As a result of the second review, the contract contains **no** issues.

Graph 1. The distribution of vulnerabilities after the audit.



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

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

Audit overview

Critical

No critical issues were found.

High

A reward should be calculated as following `(storedBalance * (dividends - lastPaidDividends)) / rewardsIncludedSupply`. But the lastPaidDividends is always equal to a user current balance not to the last paid dividends amount. A single account can drain all rewards balances by sending multiple transactions.

Contracts: Pocket.sol

Function: _distribute

Recommendation: change rewards distribution logic to more fair one.

Status: fixed.

Medium

1. Calling of the `_distribute` modifier is redundant because a caller is excluded at this point and his reward is 0.

   Contracts: Pocket.sol

   Function: _includeInRewards

   Recommendation: remove redundant calls.

   Status: fixed.

2. The function calculates an account balance by summing real balance and rewards. In a case, when an account send all its funds, the transaction may fail because the account reward can be decreased. As a result, users will lose their gas fees.

   Contracts: Pocket.sol

   Function: _balanceOf

   Recommendation: do include dynamic values into user balance.

Status: fixed.

Low

No low severity issues were found.

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.

As a result of the audit, security engineers found **1** high and **2** medium severity issues.

As a result of the second review, the contract contains **no** issues.

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 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.