SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Gamico
Date: September 09th, 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 Gamico.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	Token, Liquidity Provider
Platform	Ethereum / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/gamicoprotocol/smart-contract
Commit	847651D4CA73789A01CC32C7BAA8F4FD25B29694
Technical Documentation	NO
JS tests	NO
Timeline	23 AUG 2021 - 09 SEP 2021
Changelog	26 AUG 2021 - INITIAL AUDIT 09 SEP 2021 - REMEDIATION

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Gamico (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 August 23rd, 2021 - August 26th, 2021.The second code review conducted on September 09th, 2021.

Scope

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

Repository:
https://github.com/gamicoprotocol/smart-contract
Commit:
847651D4CA73789A01CC32C7BAA8F4FD25B29694
Technical Documentation: No
JS tests: No
Contracts:
GMC.sol
LiquidityProvider.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

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, the security engineers found **3** critical, **1** medium and **2** low severity issues.

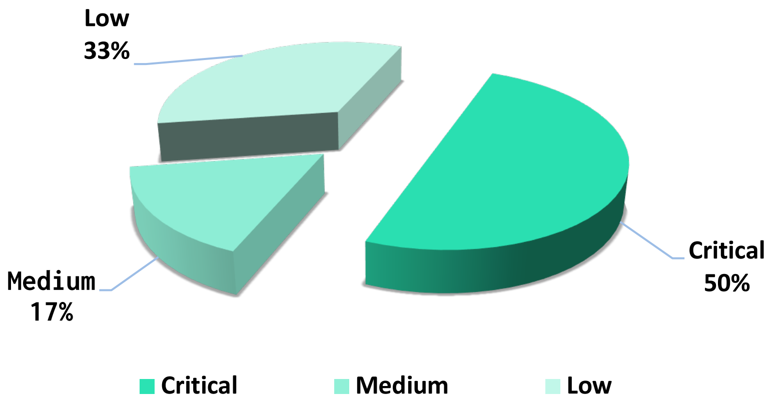
As a result of the second review, the security engineers found **no issues**.

Notice:

The Owner of the LiquidityProvider.sol contract has large capabilities to manipulate users' funds. Due to this audit scope, we cannot be sure that the Owner can be trusted.

Customer’s comment: In production, the owner will be a multisig wallet co-owned by the devs and a community member.

Graph 1. The distribution of vulnerabilities after the audit.



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

1. The transferFrom function does not reduce the amount of the allowance, and after one approval(even 1 token), the spender can transfer all the funds

   Contracts: GMC.sol

   Function: transferFrom

   Recommendation: reduce the amount of the allowance

   Status: Fixed.

2. All users' funds can be transferred to the "Game" address approved by the Owner

   Contracts: LiquidityProvider.sol

   Function: transferToGame

   Recommendation: prove that we can trust the Owner or limit his capabilities.

   Status: Fixed.

   Customer comment: *We added timelock for "approving" a game so users can exit within a day if a malicious game is approved by the owner. In production, the owner will be a multisig wallet co-owned by the devs and a community member.*

3. The setToken function should have a check that all deposits have been withdrawn to prevent the possibility of losing funds

   Contracts: LiquidityProvider.sol

   Function: setToken

   Recommendation: add check.

   Status: Fixed.

■ ■ ■ High

No high severity issues were found.

■ ■ Medium

Zero address checks are recommended

Contracts: GMC.sol

Function: transferFrom, transfer, approve

Recommendation: add checks

Status: Fixed.

■ Low

1. totalSupply, name, symbol, and decimal fractions do not change after deployment and must be defined as constants for gas savings

   Contracts: GMC.sol

   Recommendation: define as constants.

   Status: Fixed.

2. unitPrice code is duplicated. Can be replaced with a function

   Contracts: LiquidityProvider.sol

   Function: deposit, withdraw

   Recommendation: replace with a function.

   Status: Fixed.

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, the security engineers found **3** critical, **1** medium and **2** low severity issues.

As a result of the second review, the security engineers found **no issues**.

Notice:

The Owner of the LiquidityProvider.sol contract has large capabilities to manipulate users' funds. Due to this audit scope, we cannot be sure that the Owner can be trusted.

Customer’s comment: In production, the owner will be a multisig wallet co-owned by the devs and a community member.

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, 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 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.