HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Galeon
Date: Feb 11th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Client.
Approved by	Andrew Matiukhin	CTO Hacken OU
Type	ERC20 token
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Deployed contract	https://bscscan.com/address/0x20613fD4251a1B81fda2872706a4ffbb4369481E#code https://bscscan.com/address/0x1d0Ac23F03870f768ca005c84cBb6FB82aa884fD#code
Technical Documentation	YES
JS tests	NO
Website	https://galeon.care
Timeline	07 FEB 2022
Changelog	07 FEB 2022 - Initial Audit 10 FEB 2022 - Second review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Client (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 Feb 06th, 2022 - Feb 07th, 2022.

Second review conducted on Feb 10th, 2022.

Scope

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

Repository:
  N/A
Commit:
  NONE
Technical Documentation: Yes (https://docsend.com/view/waueb7v3yjc2phus)
JS tests: No
Contracts:
  Galeon.sol
  IERC20Metadata.sol, Context.sol, ERC20.sol, ERC20Capped.sol,
IERC20.sol, Ownable.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.


Our team performed an analysis of code functionality, manual audit, and automated checks with Mythril, SmartCheck, Solgraph, 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 **3** low severity issues.

As a result of the second review, the code contains **2** low severity issues.

Graph 1. The distribution of vulnerabilities after the audit.

Low
100,0%
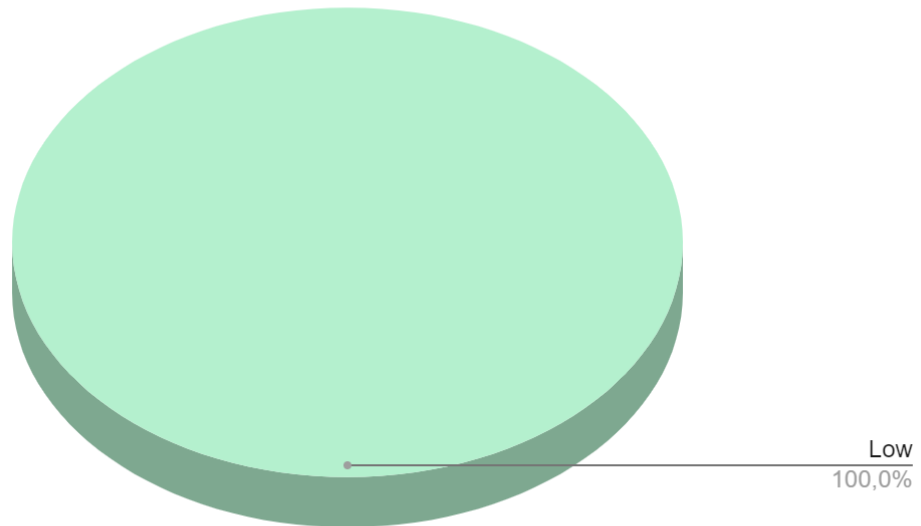
Severity Definitions

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

T

■■ Medium

No high severity issues were found.

■ Low

1. Empty functions invocation

   _beforeTokenTransfer and **_afterTokenTransfer** functions are empty
   therefore when they are executed - nothing happens.
   Those functions are invoked in: _transfer(), _mint(), _burn().

   Recommendation:

   To save gas on contract deployment and execution we recommend to
   remove those functions and all their invocations.

   Status: fixed

2. Unused function
   Contract *Context.sol*. Function **_msgData** is unused.
   Recommendation: Delete **_msgData** function from the contract.
3. Some functions are declared as **public** instead of being declared
   external.

     - ERC20.name()

     - ERC20.symbol()

     - ERC20.allowance(address,address)

     - ERC20.approve(address,uint256)

     - ERC20.transferFrom(address,address,uint256)

     - ERC20.increaseAllowance(address,uint256)

     - ERC20.decreaseAllowance(address,uint256)

     - Ownable.renounceOwnership()

- Ownable.transferOwnership(address)

- Galeon.isTransferLimitEnabled()

- Galeon.mint(address,uint256)

- Galeon.burn(address,uint256)

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

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 **3** low severity issues.

As a result of the second review, the code contains **2** low severity issues.

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.