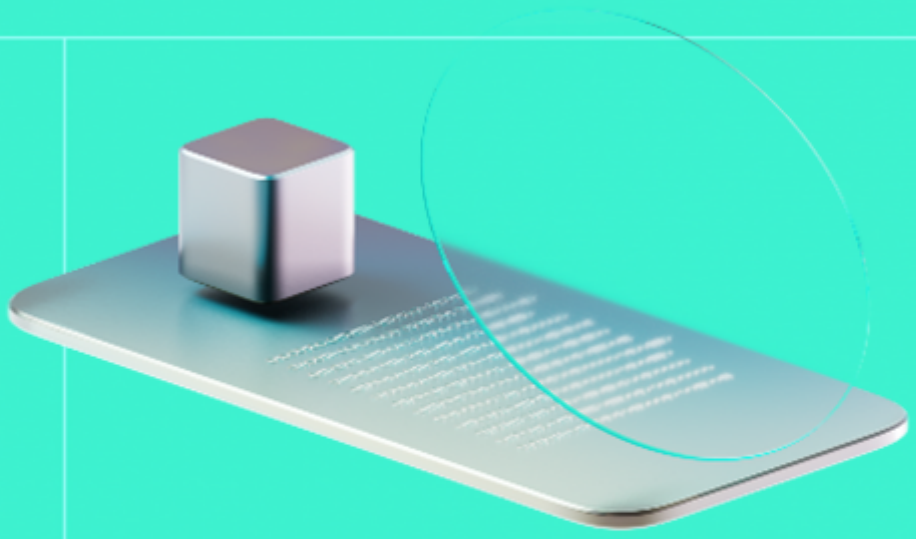
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Gaimin.io

Date: 3/1/2024

We express our gratitude to the Gaimin.io team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Gaimin is a token that will be used during the vesting process.

Platform: EVM

Language: Solidity

Tags: BEP-20

Timeline: 26.12.23 - 03.01.24

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/Gaimin-io-Limited/gmrx-vesting/tree/main
Commit	6429225

Audit Summary

10/10	10/10	0%	8/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.8/10

The system users should acknowledge all the risks summed up in the risks section of the report

0	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	0

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Gaimin.io
Audited By	Viktor Lavrenenko
Approved By	Przemyslaw Swiatowiec, Ataberk Yavuzer
Website	http://gaimin.io/
Changelog	03/01/2024 - Final Report

Table to Contents

System Overview

Gaimin Token is a vesting token with the following contracts:

GMRX — simple BEP-20 token that mints all initial supply to a deployer. Additional minting is not allowed.

It has the following attributes:

- Name: Gaimin Token.
- Symbol: GMRX.
- Decimals: 18.
- Total supply: 100 billion tokens.

Privileged roles

- The contract does not have any roles.

Executive Summary

This report presents an in-depth analysis and scoring of the Customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **8** out of **10**.

- Functional requirements are partially provided:
  - Business logic is partially provided: the total supply of GMRX, which is 100.000.000.000 is not documented.
- The technical description is partially provided:
  - NatSpec is sufficient, since it is inherited from OpenZeppelin.
  - A description of the development environment is missing.
  - Technical specification is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- All best practices are followed.
- The development environment is configured.

Test coverage

Code coverage of the project is **0%** (branch coverage).

- For projects with less than 250 LOC (Lines of Code) the test coverage is not mandatory, and it is not accounted for in the final score.

Security score

Upon auditing, the code was found to contain **0** issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the Customer's smart contract yields an overall score of **9.8**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- All tokens are minted to a single address. The secureness of the supply depends on the secureness of key storage.

Findings

Vulnerability Details

Observation Details

F-2023-0310 - Floating Pragma - Info

Description:

A **floating pragma** in Solidity refers to the practice of using a pragma statement that does not specify a fixed compiler version but instead allows the contract to be compiled with any compatible compiler version. This issue arises when pragma statements like pragma solidity ^0.8.20 are used without a specific version number, allowing the contract to be compiled with the latest available compiler version. This can lead to various compatibility and stability issues.

Version Compatibility: Using a floating pragma makes the contract susceptible to potential breaking changes or unexpected behavior introduced in newer compiler versions. Contracts that rely on specific compiler features or behaviors may break when compiled with a different version.

Interoperability Issues: Contracts compiled with different compiler versions may have compatibility issues when interacting with each other or with external services. This can hinder the interoperability of the contract within the Ethereum ecosystem.

The project uses floating pragma `^0.8.20`.

Assets:

- GMRX.sol [https://github.com/Gaimin-io-Limited/gmrx-vesting/tree/main]

Status: Fixed

Recommendations

Recommendation:

Consider locking the pragma version and avoid using a floating pragma in the final deployment. Furthermore, use the up-to-date Solidity version such as 0.8.23.

Remediation (Revised commit: 6429225): The floating pragma issue was addressed by locking the pragma to a Solidity version 0.8.23.

Disclaimers

This audit report focuses exclusively on the security assessment of the contracts within the specified review scope. Interactions with out-of-scope contracts are presumed to be correct and are not examined in this audit.

While we have diligently identified and mitigated potential security risks within the defined scope, it is important to note that our assessment is confined to the isolated contracts within this scope. The overall security of the entire system, including external contracts and integrations beyond our audit scope, cannot be guaranteed.

Users and stakeholders are urged to exercise caution when assessing the security of the broader ecosystem and interactions with out-of-scope functionality. For a comprehensive evaluation of the entire system, additional audits and assessments outside the scope of this report are necessary.

This report serves as a snapshot of the security status of the audited contracts within the specified scope at the time of the audit. We strongly recommend ongoing security evaluations and continuous monitoring to maintain and enhance the overall system's security.

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

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 Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/Gaimin-io-Limited/gmrx-vesting/tree/main
Commit	6429225
Whitepaper	Whitepaper
Requirements	Whitepaper
Technical Requirements	NatSpec

Contracts in Scope

./contracts/GMRX.sol

Deployed contract [code]