GGMV Security Analysis

by Pessimistic

This report is public

August 16, 2023

Abstract

In this report, we consider the security of smart contracts of GGMV project. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. A single audit cannot be considered enough. We always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, a security audit is not investment advice.

Summary

In this report, we considered the security of GGMV smart contracts. We described the audit process in the section below.

The audit showed only several issues of low severity. They do not endanger project security.

The overall code quality is average. The codebase is roughly covered with tests and comments describing the code logic present. However, the developers did not use a formatter/linter to prettify the code, and no NatSpec comments are present.

General recommendations

We recommend fixing the mentioned issues, adding NatSpec comments, and running the formatter/linter on a codebase to improve the code readability. Moreover, we recommend removing the irrelevant `TODO` comments, which may lead to confusion that the codebase is not finished yet.

Project overview

Project description

For the audit, we were provided with GGMV project on a public GitHub repository, commit d7e65523bfe5d757d39e50611a50a258404c451b.

The scope of the audit included the **GGMVIssuer**, **GGMVToken** contracts.

The documentation for the project included https://docs.metamarket.tech.

All 43 tests pass successfully. The code coverage is 100%.

The total LOC of audited sources is 153.

Audit process

We started the audit on August 1, 2023, and finished on August 3, 2023.

We inspected the materials provided for the audit. Then, we contacted the developers for an introduction to the project.

We manually analyzed all the contracts within the scope of the audit and checked their logic. Among other, we verified the following properties of the contracts:

- The users/project will not lose any money;
- That all calculations are correct;
- Possible gas optimizations.

We scanned the project with the static analyzer Slither and our plugin Slitherin with an extended set of rules, Semgrep rules for smart contracts, and manually verified the output.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

Manual analysis

The contracts were completely manually analyzed, their logic was checked. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

The audit showed no critical issues.

Medium severity issues

Medium severity issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

The audit showed no issues of medium severity.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Gas consumption

Consider marking the `minter` variable as immutable at line 18 in the **GGMVToken** contract. This will decrease the gas usage of the contract.

L02. Gas consumption

Consider creating a new variable to store `pools[i]` in `memory`, to reduce the storage reads (`pools[i].pool`, `pools[i].percent`) and improve the gas usage.

Moreover, consider making a new variable for the expression `_GGMTAmount * pools[i].percent / PERCENT_DENOMINATOR`, which is used twice. This will improve the code readability and reduce the gas usage in `_distributeGGMT` function of **GGMVIssuer** contract.

L03. Gas consumption

The `_isBlackListed` function of the **GGMVIssuer** contract iterates through all pools in a for-loop in order to check if the `msg.sender` is a pool. Consider creating a new mapping or marking pools as blacklisted in `blacklist` mapping to remove the for-loop and reduce gas usage.

L04. ERC20 standard violation

The return value of the `transfer` call at line 161 in the `withdrawERC20` function of the **GGMVIssuer** contract is not checked. Even though this function will most likely be used only with the `GGMT` token since the contract does not intend to hold any other tokens, we still recommend using the `safeTransfer` function, which checks the return value.

Notes

N01. Invariant

It is required that the sum of all pool percentages is equal to `PERCENT_DENOMINATOR` in order to **GGMVIssuer** contract work properly. However, this invariant could be broken in `removePoolByIndex()` and `editPoolByIndex()` functions, making the sum greater or less than `PERCENT_DENOMINATOR`. Nevertheless, this does not lead to any security problems:

- If the sum is less than the `PERCENT_DENOMINATOR`, excessive `GMT` tokens will remain in the contract, and the owner has the ability to withdraw them.
- If the sum is greater than the `PERCENT_DENOMINATOR`, the transaction will revert.

This analysis was performed by Pessimistic:

Yhtyyar Sahatov, Security Engineer
Daria Korepanova, Senior Security Engineer
Irina Vikhareva, Project Manager

August 16, 2023