HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Metagamz
Date: February 18th, 2022

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 Metagamz.
Approved by	Andrew Matiukhin | CTO Hacken OU
Type	ERC20 token; Vesting
Platform	Avalanche / Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Repository	https://github.com/avnishmishra04/MetagamZ
Commit	bbcc0341d584ddbfe46d0e9e5ae957a100158b6b
Deployed contract	1. https://snowtrace.io/address/0x43d141d7e4e9bd76851ac707b9b55bb9cf90c8aa 2. https://snowtrace.io/address/0xf0b5d0f2c999f95e03a363a58eb44e88cb620404
Technical Documentation	YES
JS tests	NO
Website	https://metagamz.io/
Timeline	11 FEBRUARY 2022 - 18 FEBRUARY 2022
Changelog	18 FEBRUARY 2022 - Initial Audit

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

Table of contents

Introduction 4

Scope 4

Executive Summary 5

Severity Definitions 6

Audit overview 7

Conclusion 9

Disclaimers 10

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

Introduction

Hacken OÜ (Consultant) was contracted by Metagamz (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 February 11th, 2022 - February 18th, 2022.

Scope

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

Repository:
https://github.com/avnishmishra04/MetagamZ
Commit:
bbcc0341d584ddbfe46d0e9e5ae957a100158b6b
Technical Documentation: Yes – https://metagamz.io/wp-content/uploads/2022/01/Metagamez-Whitepaper-2.7_WEB.pdf
JS tests: No
Contracts:
METAG.sol
Vesting.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 secured but the gas usage could be improved.

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 **5** low severity issues.

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

No high severity issues were found.

Medium

No medium severity issues were found.

Low

1. Using SafeMath on solidity >=0.8.0

Starting Solidity v0.8.0 there's no need to check for uint to overflow while mathematical operations because this check is already built-in.

Recommendation: Please either discard SafeMath or use its version updated to use with Solidity 0.8 or later.

2. Vesting rules not described

While the provided whitepaper contains very specific vesting rules, the smart contract itself doesn't specify them. It waits that the admin will add each rule additionally with separate transactions which unclear if added rules would match the witepaper or not. At the time of the audit, the deployed smart contract didn't have any vesting rules added.

Contract: TokenVesting

Recommendation: Please consider specifying vesting rules either in the constructor or initialization function.

3. Access the state variables in the loop

It is not recommended to have read or write access to the state in the loops because it costs a lot of gas. Right not the internal function "createVesting" is being called from the "createMultipleVesting" in the loop. The "createVesting" function is accessing the "totalVestings" variable multiple times: three times for reading and once for writing.

Contract: TokenVesting

Functions: createMultipleVesting, createVesting

Recommendation: We'd recommend making the "createVesting" function totally pure and eliminating any state access from it by reading the "totalVestings" value in the "createMultipleVesting" before the loop

and assigning it back after that. Accessing and incrementing should be done on the local variable, which will save you tons of gas.

4. External calls in the loop

Like as above, there is an external call to the “ERC20Interface.allowance” and “ERC20Interface.safeTransferFrom” functions each time the “createVesting” is accessed, which is definitely not needed

Contract: TokenVesting

Functions: createMultipleVesting, createVesting

Recommendation: Please consider summing together all amounts and do the allowance check and transfer after the loop.

5. Using storage variable

Using the storage variable instead of memory one without any writings only for multiple reads will just burn excess gas.

Contract: TokenVesting

Functions: claim, suspendLockTransferToReceiver

Recommendation: Please consider using the memory variable placing instead.

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.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

As a result of the audit, security engineers found **5** low severity 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 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.