VIDMA

EG

SMART CONTRACT AUDIT

Project: EGMigrate
Date: January 25th, 2023

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

EG implemented the migration mechanism that allows holders to update their old tokens for new ones when the project upgrades to a new contract on the EVM chain. It helps projects with contract upgrades, supply consolidations, mergers/acquisitions etc.

An audited contract allows adding the token with consolidation ratio to be supported on the platform, doing the actual migration and token exchange with selected amount of coins, updating data of supported migration token, disable this token, refunding non-migrated tokens to the project team by owner. Also, the contract has ownership functionality to limit user access to the admin functions and transfer control to another address.

During the audit process, the Vidma team found several issues. A detailed summary and the current state are displayed in the table below.

Vulnerability Summary

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	0 issues	0 issues	0 issues
Low	6 issues	6 issues	0 issues
Informational	3 issues	3 issues	0 issues
Total	**9 issues**	**9 issues**	**0 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Vidma Points System

To set the codebase quality mark, our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent evaluation codebase quality system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Codebase quality: 98.80

Evaluating the **initial commit** and the **last commit with the fixes**, Vidma audit team set the following **codebase quality** mark.

Score

Based on the **overall result of the audit** and the state of the final reviewed commit, the Vidma audit team grants the following **score**:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

EG

EGMigrate mission is to leverage community action and blockchain technologies to grow a global movement that defies the status quo and makes profitability intrinsically linked to positive social impact.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from January 10, 2023 to January 12, 2023. The review of the fixes was conducted from January 16, 2023 to January 17, 2023. The latest commit was reviewed and approved on January 25, 2023. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contract:

- eg-migrate;

The source code was taken from the following **source**:
https://github.com/EG-Ecosystem/eg-migrate

Initial commit submitted for the audit:
5f92aa1ef2a6bc151317a4ccdf55e70e6e6b26d8

Last commit reviewed by the auditing team:
9a554d563763d56e931ecb21b6c1991ef32248ee

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contract.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by EG or not. The issues with “Not Relevant” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Inappropriate function visibility

Low | ML – 01 | Resolved

OwnableUpgradeable smart contract is a modified version of openzeppelin standard contract (*renounceOwnership()* function removed). There is *transferOwnership()* function that is not called in the contract itself so its visibility can be improved from public to external.

Recommendation:

Consider changing visibility from public to external to safe gas usage on calling this function.

Usage of *msg.sender*

Low | ML – 02 | Resolved

In EGMigrate smart contract could be used *_msgSender()* function instead of *msg.sender* because it is inherited from ContextUpgradeable smart contract. In addition, the *_msgSender()* function call requires less gas than *msg.sender*.

Recommendation:

Consider changing *msg.sender* to *_msgSender()*.

Too much import code

Low | ML – 03 | Resolved

Imports of IERC20 and ERC20 could be replaced by import of IERC20Metadata that already includes the IERC20 interface and does not contain unnecessary functionality of ERC20 smart contract.

Recommendation:

Consider changing imports:

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
```

to:

```
import "@openzeppelin/contracts/token/ERC20/extensions/
IERC20Metadata.sol";
```

Also change ERC20 type usage to IERC20Metadata for *decimals()* calls.

Incorrect address stored during migration

Low | ML – 04 | Resolved

In *migrate()* function of EGMigrate smart contract there is storing Migration to *_userMigrations* of *_msgSender()* but *toAddress* is not stored:

```
Migration({
migrationId: migrationCounter,
toAddress: _msgSender(), //! should be toAddress input variable
timestamp: block.timestamp,
amountOfSourceToken: amount,
amountOfTargetToken: migrationAmount
})
```

Recommendation:

Change *toAddress: _msgSender()* to *toAddress: toAddress*.

Lack of require statements

Low | TL – 01 | Resolved

The functions *addMigrationToken()* and *updateMigrationTokenInfo()* are not provided with input parameters validation of *sourceToken* and *targetToken* parameters, can be passed the same addresses for source and target tokens.

Recommendation:

Consider adding next require statement (*require(sourceToken != targetToken)*).

Lack of transfer amount validation

Low | TL – 02 | Resolved

The function *returnTokens()* isn't provided with validation amount tokens to transfer, contract balance of target token can be less than the passed amount which will lead to an error in transfer function.

Recommendation:

Consider adding require for check contract balance of target token (*require(amount => IERC20(migrationToken.targetToken).balanceOf(address(this))*).

Solidity source file contains two contract definitions

Informational | MI – 01 | Resolved

Usually, one source file contains one smart contract. OwnableUpgradeable smart contract implements a module of certain functionality and can be placed in an independent source file.

Recommendation:

Consider moving OwnableUpgradeable smart contract to separate source file.

Missed NatSpec

Informational | MI – 02 | Resolved

NatSpec comments are missed before EGMigrate smart contract declaration and *initialize()* function. A public state variable is equivalent to a function for the purposes of NatSpec. Events also need NatSpec notations.

Recommendation:

Consider adding NatSpec for EGMigrate SC, *initialize()* function and events. Also, change comments of public variables to NatSpec comments. View NatSpec Format — Solidity 0.8.17 documentation.

Private variable naming style

Informational | MI – 03 | Resolved

Private and internal state variable names should use underscore and mixedCase (*_initialSupply*, *_account*, *_recipientAddress* etc).
View Style Guide — Solidity 0.8.17 documentation.

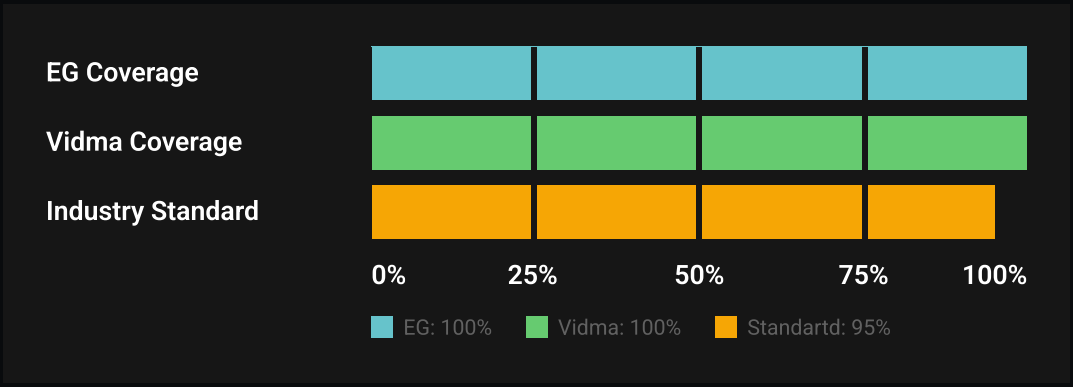
Recommendation:

Consider changing *userMigrations* to *_userMigrations*.

TEST RESULTS

To verify the security of contracts and the performance, a number of integration tests were carried out using the Hardhat testing framework.

In this section, we provide both tests written by EG and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the EG repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests Written by EG

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	77.78	100.00	100.00
eg-migrate.sol	100.00	77.78	100.00	100.00
All Files	**100.00**	**77.78**	**100.00**	**100.00**

Test Results

```
Contract: EGMigrate
  EGMigrate test case
    Deployment
      ✓ should set owner correctly (138ms)
    checked all functions
      ✓ checked addMigrationToken EGMigrate: source token
        address is zero (2076ms)
      ✓ checked addMigrationToken EGMigrate: source token
        already exists (174ms)
      ✓ checked addMigrationToken EGMigrate: target token
        address is zero (331ms)
      ✓ checked addMigrationToken EGMigrate: rate is zero (173ms)
      ✓ checked addMigrationToken success (137ms)
      ✓ checked setStatusOfMigrationToken (690ms)
      ✓ checked updateMigrationTokenInfo (920ms)
      ✓ checked migration EGMigrate: source token does not
        exist (206ms)
      ✓ checked migration EGMigrate: migration is disabled for
        this token (623ms)
      ✓ checked migration EGMigrate: transfer to the zero
        address is not allowed (144ms)
      ✓ checked migration EGMigrate: amount is zero (159ms)
```

```
    ✓ checked migration EGMigrate: insufficient balance
      of source token in holder wallet (605ms)
    ✓ checked migration EGMigrate: holder has insufficient
      approved allowance for source token (1470ms)
    ✓ checked migration EGMigrate: insufficient balance of
      target token (744ms)
    ✓ checked migration success (7997ms)
    ✓ checked userMigrationsLength (158ms)
    ✓ checked userMigration (140ms)
    ✓ checked migration failed EG: Transfer amount exceeds
      the maxTransactionCoolDownAmount (2180ms)
    ✓ checked migration with whitelist and
      maxTransactionAmount (3062ms)
    ✓ checked returnTokens Amount should be greater than
      Zero (125ms)
    ✓ checked returnTokens ERC20: transfer to the zero address
      is not allowed (158ms)
    ✓ checked returnTokens ERC20: source token does not
      exist (156ms)
    ✓ checked returnTokens success (971ms)

  24 passing (37s)
```

Tests Written by Vidma Auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
eg-migrate.sol	100.00	100.00	100.00	100.00
All Files	**100.00**	**100.00**	**100.00**	**100.00**

Test Results

```
Contract: EGMigrate
  deploy
    ✓ should set owner correctly
    ✓ should initialize only once
  addMigrationToken
    ✓ cannot add if caller is not owner
    ✓ cannot add if source token address is zero
    ✓ cannot add if target token address is zero
    ✓ cannot add if rate is zero
    ✓ should add correctly
    ✓ cannot add if source token already exists
  setStatusOfMigrationToken
    ✓ cannot set if caller is not owner
    ✓ cannot set if source token does not exist
    ✓ should set correctly
  updateMigrationTokenInfo
    ✓ cannot update if caller is not owner
    ✓ cannot update if source token does not exist
    ✓ cannot update if target token address is zero
    ✓ cannot update if rate is zero
    ✓ should update correctly (43ms)
  migrate
```

```
    ✓ cannot migrate if source token does not exist
    ✓ cannot migrate if migration is disabled for this token
    ✓ cannot migrate to the zero address
    ✓ cannot migrate if amount is zero
    ✓ cannot migrate if insufficient balance of source token
      in holder wallet
    ✓ cannot migrate if holder has insufficient approved
      allowance for source token
    ✓ cannot migrate if insufficient balance of target token (42ms)
    ✓ should migrate correctly (87ms)
    ✓ should migrate multiple times correctly
  returnTokens
    ✓ cannot return if caller is not owner
    ✓ cannot return if amount is zero
    ✓ cannot return if address for transfer is zero
    ✓ cannot return if source token does not exist
    ✓ should return correctly (51ms)

30 passing (6s)
```

We are delighted to have a chance to work with the EG team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**