VIDMA

MEGAFANS
eSports for All!

SMART CONTRACT AUDIT

Project: MEGAFANS
Date: April 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.

MegaFans is a multiplayer tournament platform. The audited scope included the MegaNFT.sol contract that implements the ERC721 NFT standard. The NFT contract allows users to buy NFT by paying the price in a native coin that is settled in the contract. There is some max supply for the token sale that can be changed by the contract owner, the same as price and baseTokenURI. The NFT contact inherits the Ownable functionality. Except listed above owner right, the owner has the ability to withdraw native coin from the SC that was paid as the price for each token. The owner of the contract can pause the token sale at any time, the same as unpause it back. The owner of the contract has an opportunity to mint NFT to any address without paying the required price.

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

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	2 issues	2 issues	0 issues
Low	5 issues	5 issues	0 issues
Informational	6 issues	6 issues	0 issues
Total	**13 issues**	**13 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:

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: 98.00

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

Bringing the online gaming community together, MEGAFANS strives to provide a fun and competitive environment to gamers of all levels. Their esports game platform also gives gamers a chance to win cash prizes by participating and have even more fun when playing mobile games online. It's hard to find a mobile game system that combines fun and monetary incentives, but MEGAFANS does exactly that.

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 on Thursday, April 20, 2023.

The initial review of the fixes was conducted on Friday, April 21, 2023. The second review of the fixes is conducted on Monday, April 24th. Two new issues are added to the list of findings. Both issues were caused during the fixing process.

The final review of the fixes was conducted on April 25th, 2023. During this review the Vidma audit team stated that all found issues are resolved.

The outcome is disclosed in this document.

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

- MegaNFT.

The source code was taken from the following **source**:
https://github.com/megafans/SmartContracts

Last commit reviewed by the auditing team:
797bba0cd5f7f6ef56db9a3a288c24b81c1f11b5

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 MEGAFANS 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

Unsecuring ether transfer

Medium | MM – 01 | Resolved

In MegaNFT contract the function *withdraw()* use *send* method to transfer ether. Both send and transfer forward a stipend of 2300 gas, which is just enough to log an event at the receiving contract. In case the receiver requires a larger amount of gas to handle the reception of ether, *call.value* has to be used, as it forwards all remaining gas, unless specified otherwise with the help of the *.gas()* parameter.

Read more about this issue:

https://fravoll.github.io/solidity-patterns/secure_ether_transfer.html

Recommendation:

Use *call* to transfer ether.

```
(bool sent, ) = payable(msg.sender).call{value: amount}("");
require(sent, "Failed");
```

Useless require

Medium | MM – 02 | Resolved

In the contract, there is a case when the owner can mint a token and not pay the price. In this case, *msg.value* will be zero. But there is a require in the *whenNotPausedAndValidSupply()* modifier to check if the sent ether value by the user is above 0. That's why the owner won't be able to call *internalMint()* function at any time.

Recommendation:

Consider removing the require in line 53.

Absent indexing in parameters of events

Low | ML – 01 | Resolved

In MegaNFT contract events are emitted without *indexed* parameter, which helps to log parameters in a transaction. The reason it is important to add *indexed* parameter to events is that it allows for more efficient and selective event filtering. By making certain parameters as indexed, Solidity can create an index of these values, making it easier and faster to search for events with the specific parameter value.

Recommendation:

Add *indexed* parameter in events.

```
event Withdraw(uint256 amount, address indexed addr);
```

Initialization with default values

Low | ML – 02 | Resolved

In MegaNFT in constructor *contractPaused* value set to *false*. This is a useless step that increases the use of gas, as a bool value by default is false.

Recommendation:

Remove the initialization of the variable from constructor.

Declaring default visibility

Low | ML – 03 | Resolved

In MegaNFT the global values *baseTokenURI* and *maxSupply* are created without declaring their visibility. By default it is internal, but the best practice is to set visibility directly.

Recommendation:

Add visibility to *baseTokenURI* and *maxSupply* variables.

Require for inputs

Low | ML – 04 | Resolved

In MegaNFT contract most of the functions have absent to check input parameters for zero value.

Recommendation:

Add require for inputs.

Never reached require

Low | ML – 05 | Resolved

Require require(_tokenId != 0, "ERC721Metadata: token must not be 0") in *tokenURI()* function line 137 is never reached. The starting token id is 1 so it is impossible that the token with id 0 will ever exist. Its condition is checked in the require in line 133. So that is why the mentioned previously require in line 137 is useless.

Recommendation:

Consider deleting the require in line 137.

Inappropriate function visibility

Informational | MI – 01 | Resolved

In MegaNFT smart contract the functions *mint()*, *mint_to()*, *internalMint()*, *pauseContract()* and *unpauseContract()* have public visibility. The visibility can be changed to external, as it consumes less gas, and functions are not called outside the contract.

Recommendation:

Change functions visibility.

Re-Audit:

After re-audit, the function *setBaseURI()* still has public visibility, which can be changed to external.

Incorrect order of layout, functions

Informational | MI – 02 | Resolved

In MegaNFT smart contract the order of functions and declared variables are not the same as in solidity documentation.

Recommendation:

Layouts, functions, and modifier order should be grouped according to Solidity documentation:

Layouts:

- interfaces;
- libraries;
- contracts.

Functions:

- *constructor*;
- *receive*;
- *fallback*;
- *external*;
- *public*;
- *internal*;
- *private*;
- *view*/*pure*.

Pay attention to the order of global variables and internal and public functions.

Re-Audit:

After re-audit there is still left one function *setBaseURI()*, which is public, between two external functions. It should be declared after all external and before internal.

Add Natspec

Informational | MI – 03 | Resolved

MegaNFT smart contract has no explanation for functions and variables, which makes it harder to understand the contract's logic.

Recommendation:

Add Natspec even for internal and private functions in all contracts that do not have it.

Optimization for requires

Informational | MI – 04 | Resolved

In MegaNFT smart contract has *mint* and *mint_to* functions with the same requires. To reduce repetitive code, checking conditions can be moved to a separate function. At the same time this require *require(!contractPaused, "Sale Paused!");* is used three times and can be moved to a *modifier*.

Recommendation:

Create a modifier for checking if the contract is paused, and add a separate function for requires.

Naming convention for *mint_to()*

Informational | MI – 05 | Resolved

In MegaNFT smart contract the function *mint_to()* doesn't follow naming conventions from official documentation:

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-styles

Recommendation:

Rename function to *mintTo()*.

Floating pragma version

Informational | MI – 06 | Resolved

In MegaNFT smart contract the current pragma Solidity directive is "^0.8.16". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Recommendation:

Change the version to "0.8.16".

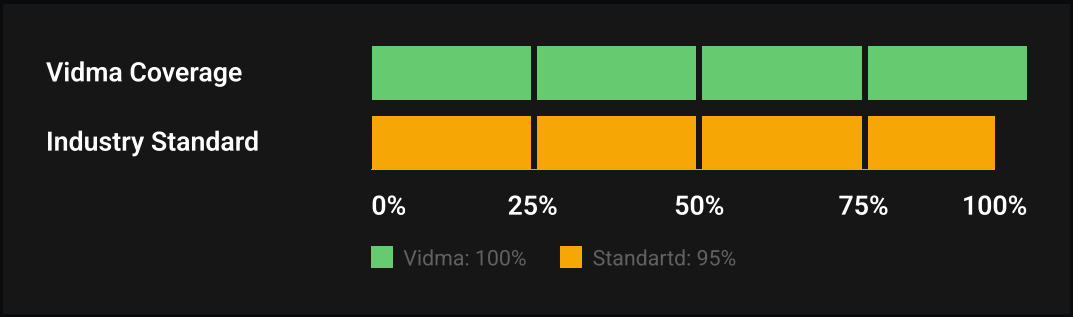
Re-Audit:

The project team didn't change the pragma version or leave any comment about this issue. Hence, the issue is till unresolved.

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 MEGAFANS 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 MEGAFANS repository. We write totally separate tests with code coverage of a minimum of 95% to meet the industry standards.

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	95.00	100.00	100.00
MegaNFT.sol	100.00	95.00	100.00	100.00
All Files	**100.00**	**95.00**	**100.00**	**100.00**

Test Results

```
Contract: MegaNFT
  initialization
    ✓ should has a name
    ✓ should has a symbol
    ✓ should set price correct
    ✓ should set max supply correct
    ✓ should set base token URI correct
    ✓ should not be paused once just deployed
  mint
    ✓ should mint token correct
    ✓ should mint to token correct
    ✓ shouldn't mint token when contract is paused (110ms)
    ✓ shouldn't mint token if max supply is reached (1048ms)
    ✓ shouldn't mint token if not enought eth is sent
    ✓ shouldn't mint zero token (39ms)
    ✓ shouldn't call internal mint by not the owner
    ✓ should call internal mint by the ownercorrect
    ✓ should return tokenURI correct (56ms)
  withdraw
    ✓ should withdraw ether by the owner (57ms)
    ✓ shouldn't withdraw ether by not the owner (51ms)
  pause
```

```
    ✓ should pause contract by the owner
    ✓ should unpause contract by the owner (52ms)
    ✓ shouldn't pause contract by not the owner (45ms)
    ✓ should't unpause contract by not the owner (45ms)
  setter
    ✓ should changePrice correct
    ✓ shouldn't changePrice by not the owner
    ✓ shouldn't set price to zero
    ✓ should changeMaxSupply correct
    ✓ shouldn't changeMaxSupply by not the owner (49ms)
    ✓ shouldn't set max supply to zero
    ✓ should setBaseURI correct (39ms)
    ✓ shouldn't setBaseURI by not the owner
    ✓ shouldn't setBaseURI as empty string

30 passing (5s)
```

We are delighted to have a chance to work with the MEGAFANS 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**

VIDMA