makersplace

MakersPlace Security Analysis

by Pessimistic

This report is public.

Published: October 7, 2021

Abstract

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

Disclaimer

The audit does not give any warranties on the security of the code. One 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, security audit is not an investment advice.

Summary

In this report, we considered the security of smart contracts of MakersPlace project. We performed our audit according to the procedure described below.

The initial audit showed violations of CEI pattern that can lead to Reentrancy, and many issues of low severity.

The project has no public documentation. The code coverage is low.

After the initial audit, the code base was updated. In this update, most of the issues were fixed, and new functionality was added to the project. However, during the recheck we found two new issues of medium severity: Bug and No payout. Also, a few low severity issues were found.

After the recheck, code base update #2 was made. Some issues were fixed in this update.

General recommendations

We recommend fixing the mentioned issues, adding public documentation to the project, and improving code coverage. We also recommend adding CI to compile the project, run tests, calculate code coverage, and analyze code with linters and security tools.

Project overview

Project description

For the audit, we were provided with MakersPlace project on private GitHub repository, commit 03afffc2f3540dce41ed1f8f92d4c94e1437b712.

The scope of the audit included **DigitalMediaSaleCore.sol** file and its dependencies.

The documentation for the project was provided as a separate file:
Documentation MakersPlace July.docx,
sha1sum is f0fc1fffb3e972c2cf6b090b4806f05c656bab23.

The project compiles with warnings, `evmVersion` in `truffle` config file is outdated.

All 28 tests pass, the coverage is 42%.

The total LOC of audited sources is 893.

Code base update #1

After the initial audit, the code base was updated. For the recheck, we were provided with commit 5bef1c72bb54781ea2f6384b499b8e6ea2cc50b1. In this update, most of the issues from the initial audit were fixed, and new functionality was added to the project.

Code base update #2

After the recheck, another code base update was performed. For the recheck #2, we were provided with commit cb4a829bd091a13eb4b32376f81c8f85c88066b5. This update includes fixes. However, in this update, some tests do not pass.

Procedure

In our audit, we consider the following crucial features of the code:

1. Whether code logic corresponds to the specification.
2. Whether the code is secure.
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan project's code base with automated tools: Slither and SmartCheck.
  - We manually verify (reject or confirm) all the issues found by tools.
- Manual audit
  - We compare the project's code to Compound's codebase and check whether the changes are consistent with documentation.
  - We manually analyze new code for security vulnerabilities.
  - We assess overall project structure and quality.
- Report
  - We reflect all the gathered information in the report.

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 smart contracts security. We highly recommend fixing them.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

Reentrancy

In **DigitalMediaSaleCore** contract, there are functions that pass execution to an external address. In some cases, CEI pattern is violated. The contract has multiple entry points and complex logic, but it does not implement proper reentrancy protection. It is hard to prove whether the contract can be exploited.

An attacking contract might be able to intercept execution when:

- `acceptPartialBid()` function calls `transferFunds()` at line 667, which, in turn, executes `msg.call()`.

  The function has been removed from the contract's code.

- Functions `purchase()`, `purchaseOBO()`, `bidOnToken()`, `acceptBid()`, `acceptBidOBO()`, and `cancelBid()` call `transferFunds()` and `_transferFromTo()` functions internally, which perform external calls to user-controlled addresses.

Misleading comment (fixed)

The comment at line 17 of **CommissionControl** contract states:

```
Full week wait period before new percentage is in effect
```

However, `commissionAddressWaitPeriod` constant is set to 3 days at line 18.

The issue has been fixed and is not present in the latest version of the code.

Tests

The project has tests. However, important scenarios are not covered. Therefore, we consider the code coverage insufficient. Also, in the latest version of the code, some tests do not pass.

Testing is crucial for code security and audit does not replace tests in any way.

We highly recommend checking and improving code coverage regularly as the project grows.

Bug

Custom commission rules in `getCommissionPercentageForToken()` function of **DigitalMediaSaleCore** contract never apply due to incorrect token type check.

No payout (fixed)

In `acceptBid()` function of **DigitalMediaSaleCore** contract, the seller can accept a bid. In this case, NFT token will be transferred to the buyer. However, the seller will not get a payout in return since this part of the code is commented out.

The issue has been fixed and is not present in the latest version of the code.

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Code quality

- Sale and Resale fee management logic is identical, and the code only differs with names of variables. Consider combining `createdAt`, `default`, and `intermediate` variables into two struct variables and use them to avoid code duplication.

  The issue has been fixed and is not present in the latest version of the code.

- In `getSale()` and `getPartialBidById()` functions of **DigitalMediaSaleCore** contract, consider returning a `struct` instead of individual values.

  The issue has been fixed and is not present in the latest version of the code.

- Consider declaring visibility level for variables explicitly in:
  - **CommissionControl** contract, lines 24 and 25.

    The issue has been fixed and is not present in the latest version of the code.
  - **DigitalMediaFixedSale** contract, line 62.

    The issue has been fixed and is not present in the latest version of the code.
  - **DigitalMediaSaleCore** contract, lines 95, 96, and 100.
- Consider replacing expressions for time intervals like `60 * 60 * 24 * 2` with time units like `2 days` in:
  - **OBOControl** contract, line 9
  - **DigitalMediaFixedSale** contract, line 15
  - **CommissionControl** contract, line 18
  - **WithdrawFundsControl** contract, line 16

  The issues have been fixed and are not present in the latest version of the code.

- In **DigitalMediaSaleCore** contract, `DigitalMediaFixedSale._cancelSale()` calls at lines 216 and 280 should be replaced with `DigitalMediaFixedSale._cancelSaleIfSaleExists()`. This will also allow to inline `_cancelSale()` function.

  The issue has been fixed and is not present in the latest version of the code.

- Since Solidity version 0.8.0, SafeMath library is not required as overflow checks are added by compiler.

  The issue has been fixed and is not present in the latest version of the code.

- In **DigitalMediaFixedSale** and **DigitalMediaSaleBase** contracts, consider declaring functions as `external` instead of `public` where possible.

  The issue has been fixed and is not present in the latest version of the code.

- Many internal functions consist of a single line or used only once. Consider inlining these functions.

  The issues have been fixed and are not present in the latest version of the code.

- There is no consistent logic for placing checks in the code. As a result, some checks are performed multiple times which increases gas consumption.

  The issues have been fixed and are not present in the latest version of the code.

- The implementation of `VaultCore` integration is inconsistent.

  The issue has been fixed and is not present in the latest version of the code.

- In **DigitalMediaSaleCore** contract, `customCommisssion` variable has a typo in its name. Consider renaming it to `customCommission` at lines 66–67.

  The issue has been fixed and is not present in the latest version of the code.

- Consider using `indexed` event parameters when events filtering is required.
- Consider separating `internal` and `external` methods in the code visually to improve code readability.
- Getters should not revert transactions. Best practice is to perform checks on getter's return values. Consider moving checks to `external` functions.
- Consequential initialization of struct's fields is considered a bad practice. Consider assigning `sale` struct in a single action in **DigitalMediaSaleCore** contract:
  - At lines 104–109.
  - At lines 251–254.

Gas consumption

- In **DigitalMediaSaleCore** contract, `purchase()` function checks whether the contract is valid, whether the token is on sale, and whether salecontract is approved. However, all these checks are performed inside internal functions. Consider removing duplicating checks.

  The issue has been fixed and is not present in the latest version of the code.

- In **DigitalMediaSaleBase** contract, `getDigitalMedia()` function calls getters that read unnecessary data from storage at lines 91 and 96.

Project design

The logic distribution between the contracts is confusing. The inheritance does not help to structure the code.

- The inheritance graph of the project is complicated and requires non-trivial C3 linearization. The final inheritance list for **DigitalMediaSaleCore** contract includes **Ownable**, **OBOControl**, **Context**, **Pausable**, **DigitalMediaSaleBase**, **CommissionControl**, **WithdrawFundsControl**, and **DigitalMediaFixedSale** contracts. In the current implementation, it does not result in any issues. However, this approach is risky when methods overriding is used.
- The Sale logic is arbitrary distributed between **DigitalMediaSaleCore**, **DigitalMediaSaleBase**, and **DigitalMediaFixedSale** contracts. The inheritance makes no sense in this case. These contracts can be combined into a single contract.

No public documentation

The project has no public documentation. Considering the complexity of the project, it can be confusing for the users.

We strongly recommend adding public documentation to the project.

Dependency management (fixed)

- The **package.json** file lacks `ethereumjs-abi` dependency.
- The code of OpenZeppelin libraries is copied into the repository. Consider managing it as a dependency.

The issues have been fixed and are not present in the latest version of the code.

Notes

Overpowered roles

Operators can cancel any existing bid or sale.

All users who choose custodial approach, fully depend on marketplace operators.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

October 7, 2021