SOLIDIFIED

Audit Report for DAN NFT Marketplace - April 5, 2022

Summary

Audit Report prepared by Solidified covering the DAN NFT Marketplace smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief was on 5 April 2022.

Audited Files

The source code has been supplied in the form of a GitLab repository:

https://gitlab.com/undeveloped/digital-assets-backend.git

Commit hash: `d84198be8bac32ee67e86cc15892185f57c642c5`

```
contracts
├── Marketplace.sol
├── Migrations.sol
├── lib
│   └── LibOrder.sol
└── proxy
    ├── ITransferProxy.sol
    ├── OperatorRole.sol
    └── TransferProxy.sol
```

Intended Behavior

The smart contracts implement a marketplace for non-fungible tokens.

Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases have their limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

Note that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

SOLIDIFIED

Issues Found

Solidified found that the DAN NFT contracts contain no critical issues, no major issues, 3 minor issues and 3 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	Marketplace.sol: A malicious NFT which is not transferring the asset can still get payment	Minor	Pending
2	TransferProxy.sol: Successful erc20 transfers can sometime revert	Minor	Pending
3	Use of dependencies with known security vulnerabilities in build system	Minor	Pending
4	Cancel function is susceptible to front-running	Note	-
5	Miscellaneous comments	Note	-
6	Use of floating pragma	Note	-

Critical Issues

No issues found

Major Issues

No issues found

Minor Issues

1. `Marketplace.sol`: A malicious NFT which is not transferring the asset can still get payment

In the `marketplace` contract, when a sell and buy order is matched and fulfilled, a malicious ERC721 or ERC1155 token with a modified transfer method can simply not transfer the NFT. This could remain uncaught and not reverted, and payment could be transferred as normal resulting in the seller getting the money without transferring the NFT.

Recommendation

Any other validation like checking the current owner of the NFT can also be manipulated by the malicious NFT contract to result in an expected output to get the funds. Hence it is always recommended to approach this with caution and do enough due diligence before allowing an NFT to be sold through this.

2. `TransferProxy.sol`: Successful erc20 transfers can sometime revert

In the `TransferProxy` contract, the method `erc20safeTransferFrom` always validates the return value to make sure the assets are transferred. Though this will work fine for most tokens, some ERC20 implementations do not return a value and instead throw on failures. A token not returning any value on successful transfer can cause this method to revert.

```
require(
   token.transferFrom(from, to, value),
   "failure while transferring"
);
```

Recommendation
Consider using the OpenZeppelin's `SafeERC20` contract which handles this case by checking the return value only when it's present.

3. Use of dependencies with known security vulnerabilities in build system

The dependency tree of the build and test system in the project shows 23 known vulnerabilities. Out of those vulnerabilities, 11 are considered of high severity. Whilst these do not directly affect the smart contracts, they could lead to accidental key exposure in some scenarios. During deployment

Recommendation
Consider upgrading the reported dependencies to the newest version or replacing them with secure alternatives. Run `npm audit` for details.

Notes

4. Cancel function is susceptible to front-running

The cancel method in the marketplace contract is susceptible to front running. An order that is meant to be canceled can be positioned right before the cancel message in the block to execute the order.

Recommendation
This note is here to ensure the team is aware of such edge cases.

5. Miscellaneous comments

The following are some recommendations to improve the overall code quality and readability.

- `Marketplace.sol`: Visibility of the method pause, unpause and hashOrder can be made external to save some gas during execution.
- `Marketplace.sol` and `OperatorRole.sol`: The use of unchained initializer functions is not recommended by openzeppelin unless necessary. In this case they perform the same job as their chained counterparts. Consider replacing `__*_init_unchained()` with `__*_init()`.
- `Marketplace.sol` and `ITransferProxy.sol`: ABI coder v2 is activated by default on Solidity v0.8.0 and latest. Hence explicit activation can be removed.
- `Marketplace.sol`: Consider adding a method to uncancel a transaction. This can help with any accidental cancellation which can never be reverted.
- `LibOrder.sol`: Consider removing unused variable - `ORDER_KIND_FIXED_PRICE`

Recommendation
Consider addressing these notes to increase the overall code quality and readability.

6. Use of floating pragma

Using a floating pragma version allows any Solidity compiler sub version to be used. Several important compiler bugs have been recently fixed. With the use of floating pragma, these will most probably be compiled with the latest version, which is `0.8.13`.

Recommendation
Consider locking the version pragma to a specific compiler version, preferably the one used to test the contracts, above `0.8.4` and below `0.8.13`.

Disclaimer