



## Audit Report for DAN NFT Marketplace - May 24, 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. The fixes were verified on 27 April 2022.

### Audited Files

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

<https://gitlab.com/undeveloped/marketplace-smart-contracts>

Final Commit hash: `a10bd34f62c437c0b1363372fe9426ef8ef54072`

```
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	-



## Audit Report for DAN NFT Marketplace - May 24, 2022

### 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	Acknowledged
2	TransferProxy.sol: Successful erc20 transfers can sometime revert	Minor	Fixed
3	Use of dependencies with known security vulnerabilities in build system	Minor	Fixed
4	Cancel function is susceptible to front-running	Note	Acknowledged
5	Miscellaneous comments	Note	Fixed
6	Use of floating pragma	Note	Fixed

## 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.

**Update:** Acknowledged

#### Team Reply

*"We are aware of the risks and would advise any buyer of a NFT to always perform their own due diligence before using any trading service that facilitates ownership transfers. We will accommodate users interacting with our contract through the marketplace frontend by pre-filtering any malicious NFTs that we are aware of."*

*Currently, there is no solution that would prevent this issue unless we work with an allow list on chain which goes against the decentralized nature of NFTs. We will keep track of new*

*developments around NFT verification to solve this issue in a way that allows for that decentralized nature.”*

## 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.

**Update:** Fixed

### Team Reply

*“We implemented the suggestion of using SafeERC20.”*

## 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.

**Update:** Fixed

**Team Reply**

*"We forced our project to use the updated versions of the indirect dependencies."*

## 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.

**Update:** Acknowledged

**Team Reply**

*"We are aware of this possibility."*

### 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.

**Update:** Fixed

- `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()`.

**Update:** Fixed

- `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.

**Update:** Fixed

- `Marketplace.sol`: Consider adding a method to uncanceled a transaction. This can help with any accidental cancellation which can never be reverted.

- `LibOrder.sol`: Consider removing unused variable - `ORDER_KIND_FIXED_PRICE`

**Update:** Fixed

## Recommendation

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

## Team Reply

- *"We changed the visibility of those methods."*
- *"We fixed the init functions. We also renamed the initializers to 'initialize' to be more in line with the convention."*
- *"We removed the pragma abicoder v2."*
- *"Uncancelling is not necessary because the orders are stored off-chain so creating a new one comes for free while un-cancelling an on-chain cancelled order would cost gas fees."*
- *"We removed the unused variable."*

## 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`.

**Update:** Fixed. Using version `0.8.13` now.

## Team Reply

*"We set the compiler version pragma to `=0.8.13`."*



Audit Report for DAN NFT Marketplace - May 24, 2022

## Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of DAN or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

*Oak Security GmbH*