# Swarm Markets Nifty Minter Security Analysis

by Pessimistic

This report is public

March 28, 2022

## Abstract

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

## Disclaimer

## Summary

In this report, we considered the security of Swarm Markets Nifty Minter smart contracts. We performed our audit according to the procedure described below.

The initial audit showed four issues of medium severity: ERC20 standard violation, Overpowered roles, Documentation issue and Tests issues. Also, several low-severity issues were found.

After the initial audit, the code base was updated. The developers fixed most of the issues and provided comments.

## General recommendations

We recommend using multisig wallets for important roles.

# Project overview

## Project description

For the audit, we were provided with Swarm Markets Nifty Minter project on a private GitHub repository, commit 5948df7eb71200890347ea14173ed9e78d76e671.

The scope of audit included only the following files:

- **AuthorizeManager.sol**
- **RoleManager.sol**
- **RoyaltyDistributor.sol**
- **SX1155NFT.sol**

The developers provided the documentation with a brief description of a system (**NFT Minter spec for auditor.docx** file, sha1sum acee40d98b4939c198f35e9c501a26f26620522e). The code base has detailed NatSpec comments.

All 34 tests pass, the overall code coverage is 44.83%.

The total LOC of audited sources is 346.

## Code base update

After the initial audit, the code base was updated. For the recheck we were provided with commit ccf56bc286756c16c97d060a32a34209a3fd603a.

In this update, most of the issues were fixed. Also, developers provided additional description and comments regarding the update.

The developers implemented new tests. As a result, all 50 tests were passed and overall code coverage increased to 83.33%.

## Procedure

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

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices.

We perform our audit according to the following procedure:

- Automated analysis
  - We scan the project's code base with the automated tool Slither.
  - We manually verify (reject or confirm) all the issues found by the tool.

- Manual audit
  - We manually analyze the code base for security vulnerabilities.
  - We assess the 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 project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

**The audit showed no critical issues.**

## Medium severity issues

Medium issues can influence project operation in the current implementation. Bugs, loss of potential income, and other non-critical failures fall into this category, as well as potential problems related to incorrect system management. We highly recommend addressing them.

### M01. ERC20 standard violation (fixed)

ERC-20 standard states:

```
Callers MUST handle false from returns (bool success). Callers MUST
NOT assume that false is never returned!
```

However, in **RoyaltyDistributor** contract, the returned value from `transfer` call is not checked at line 62.

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

### M02. Overpowered roles

The project has roles with excessive powers.

1. The `ISSUER_ROLE` can:
   - Grant and revoke other roles to any address.
   - Mint tokens to any address.
2. The `AGENT_ROLE` is able to set authorization contracts. These contracts apply restrictions to token transfers.
3. The `Owner` of **RoyaltyDistributor** contract can add/remove beneficiaries without any restrictions.

In the current implementation, the system depends heavily on these roles. Thus, there are scenarios that can lead to undesirable consequences for the project and its users, e.g., if private keys for any of these roles become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> *Comment from the developers: Not addressed, required multi-sig support.*

## M03. Documentation issue (fixed)

The documentation states:

```
ContractURI is set once per token and contains fixed attributes.
```

However, the code allows changing `ContractURI` at any moment to any value.

*The issue has been fixed and is not present in the latest version of the code. `ContractURI` variable can be set once per contract and cannot be changed afterwards.*

## M04. Tests issue (fixed)

The project has tests. However, the overall code coverage is only 44.83%. Testing is crucial for the security of the project, and the audit does not replace tests in any way. We highly recommend covering the code with tests and ensuring that all tests pass and the code coverage is sufficient.

*The developers have implemented additional tests. There are 83.33% of branches covered in the latest version of the code.*

## Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

### L01. Code quality

The **RoleManager** contract implements functions for granting and revoking roles. However, the contract inherits from **AccessControl** contract of OpenZeppelin library that already includes this functionality. Consider using `grantRole` and `revokeRole` functions directly.

> *Not addressed, role grant can be done by only issuer, but openzeppelin code does have control on this, so overriding.*

### L02. Code quality (fixed)

In **SX1155NFT** contract, `mintBatch`, `burnBatch` and `_beforeTokenTransfer` functions consider that arrays passed to these functions as arguments have the same lengths. We recommend adding proper checks to these functions that will verify that the lengths of the arrays match.

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

### L03. Code quality (fixed)

In **SX1155NFT** contract, `mint` and `mintBatch` functions should verify that quantity arguments are greater than `0`. Otherwise, these functions can mint empty tokens with non-empty metadata.

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

### L04. Code quality (fixed)

Consider declaring `id` argument of the `TokenAuthContractSet` event as `indexed` to ease any contract integrations and simplify single token tracking.

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

### L05. Misleading NatSpec comment (fixed)

In **SX1155NFT** contract, `setTokenAuthContract` function has a misleading NatSpec comment. The comment states that only `ISSUER` can call this function. However, the function has `onlyAgent` modifier.

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

### L06. Typo (fixed)

In **RoyaltyDistributor.sol**, the word `beneficiary` is misspelled throughout the contract.

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

## Notes

### N01. Gas consumption

In **RoyaltyDistributor** contract, `distributeRoyalty` function iterates over `benificiaries` array. Note that the function can exceed the block gas limit when this array reaches a certain length.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Pavel Kondratenkov, Security Engineer
Nikita Kirillov, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

March 28, 2022