Swarm Markets Nifty Bundles Security Analysis

by Pessimistic

This report is private

November 22, 2021

Abstract

In this report, we consider the security of smarts contracts of Swarm Markets Nifty Bundles project. Our task is to find and describe security issues in the 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 Swarm Markets Nifty Bundles smart contracts. We performed our audit according to the procedure described below.

The audit showed eight critical issues that completely break the economy of the system. There are also multiple other issues of various severity.
Both the code and the documentation are of low quality, and there are major discrepancies between them. The project has no tests, and many of the discovered issues could be detected by proper testing.

The contracts should not be deployed as is or with minor fixes.

General recommendations

We recommend reworking the project's documentation, simplifying contract system's architecture. Consider putting extra effort into testing, i.e. writing tests and achieving high code coverage, manually checking a wide variety of complex scenarios.
We also recommend getting independent review of the bundles logic and economy.

Project overview

Project description

For the audit, we were provided with Swarm Markets Nifty Bundles project on a private GitHub repository, commit 7741b6875f471eb7223a9a85b283834dd45a4d99.

The following documentation was provided:

- **README.md** file in the repo.
- NiftyAsset Bundles v0.1 (Shared with PF) document.

The project has no tests.

The total LOC of audited sources is 1577.

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 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 manually analyze code base 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 project security. They can lead to loss of funds or other catastrophic consequences. The contracts should not be deployed before these issues are fixed.

Economy breach

When a user wins an auction, (s)he receives SBT, and his/her ERC20 tokens are transferred to the seller. However, there are no restrictions for the seller to win the auction. In this case, the winner will not only get back the deposit in ERC20 tokens as a seller but will also receive SBT. This makes SBT worthless and breaks the economy of the system.

Auction lots for free

In **AuctionManager** contract, `withdrawBids()` function does not check whether the caller is the winner of the auction. As a result, the winner can get the lot and then withdraw his/her bid. Consider implementing the proper check that restricts withdrawals for the auction winner.

Multiple withdrawals of the same bid

In **AuctionManager** contract, when an auction is settled or cancelled, users can withdraw their bids using `withdrawBids()` function. However, the function does not change the `bids[_bid_id].amount` value. Thus, users can withdraw same bids until there are no tokens left on **Escrow** contract.

We recommend adding a check if a user's bid has been withdrawn already.

Price manipulation

In **NftBundle** contract, `setPrice()` function is public and has no modifiers. Thus, anyone can call this function and change the nominal price of SBT tokens.

We recommend limiting access to this functionality.

Non-cancelled auctions

In **AuctionManager** contract, if there are no participants in the auction, this auction is not cancelled automatically. This can have the following consequences:

- An NFT will be transferred from seller to **NFTBundle** contract, but the seller will not get anything in return.
- An NFT will be transferred from **NFTBundle** to **Escrow** contract, but `transferNFt()` function of **AuctionManager** contract will revert at line 243. In this case, the NFT will remain on **Escrow** contract, but the seller will not get the deposit back.

Instantaneous auctions

In **AuctionManager** contract, the duration of auction is assigned with a number of days from `randomNumber()` function. However, this function can return `0`. In this case, the auction will end within the same block that results in Non-cancelled auctions issue.

We recommend modifying the `randomNumber()` function function so that it cannot return `0`.

Stuck lot

When proposal is made without auction, an NFT will be transferred from the seller to **Escrow** contract. When the auction is settled, NFT will remain on **Escrow** contract, and the seller will not be able to get it back.

Auction end by request

In **AuctionManager** contract, anyone can end the auction using `endAuction()` function since it does not verify the caller and does not check whether `endDate` is reached. This allows a wide range of malicious actions including auction blocking, winning auction with minimal bid using front-running attack, etc.

We recommend restricting access to this function and adding a check that the `endDate` of the auction has accrued.

Medium severity issues

Medium issues can influence project operation in 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.

No tests

The project has no tests. Testing is crucial for code security and audit does not replace tests in any way. We highly recommend covering the code base with tests and making sure that all tests pass and the code coverage is sufficient.

Overpowered roles

There are special roles with extended powers in the project:

- The owner of **Escrow** contract can withdraw all NFTs and bids before the end of the auction using `transferNFt()`, `transferCurrency()`, `transferWithdrawDeposit()`, and `transferFee()` function.
- The owner of **NftBundle** contract can provide his/her own address as an auction and then mint any amount of SBT tokens to this address using `mintBundleShare()` and `transferBundleShare()` functions consequently.
- Admin can authorize users who did not pass KYC in `PermissionManager`.

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 these roles become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

Misleading documentation

There are many statements in the documentation that are violated in the code. E.g.:

- Minimal cost of the lot is not used in auctions.
- ERC1155 standard is not supported.
- Most of NAT logic is not implemented.
- etc.

Overall, the documentation is of inadequate quality.

DoS

In **PermissionManager** contract, the functions that call `PermissionItems.mint()` inside `for`-loops will revert if any of receiving accounts does not accept ERC1155 token.

ERC20 standard violation

ERC-20 standard states:

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

However, returned values from `transfer()` and `transferFrom()` calls are not checked.

Bug

The condition inside `if` statement at line 82 in `removeVetos()` function of **NftBundle** contract is probably wrong: should be `account == vetoAddress` instead of `account != vetoAddress`. As a result, the function deletes all the accounts from `vetos` except `vetoAddress`. We assume it should only delete the `vetoAddress` element from vetoes list.

Weak source of randomness

In `randomNumber()` function of **AuctionManager** contract, `block.difficulty` and `block.timestamp` values should not be used as a source of randomness since the result can be predicted.

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality

- **ERCFunction** library is redundant since it does not add any functionality but complicates the system.
- The following variables are not used:
  - `duration` variable at line 46 of **AuctionStorage** contract.
  - `startPrice` variable at line 72 of **AuctionStorage** contract.
- CEI (checks-effects-interactions) pattern is violated in:
  - **AuctionManager** contract at line 310.
  - **NFTMarketMinter** contract at lines 97 and 115.
  - **BundleToken** contract at line 250.

  We highly recommend following CEI pattern to increase the predictability of the code execution and protect from some types of re-entrancy attacks.
- Consider declaring a variable as `constant` when its value is not intended to change:
  - Variable `baseMetadataURI` in **NFTMarketMinter** contract at line 22.
  - Variables `_name` and `_symbol` in **BundleToken** contract at lines 32–33.
  - Variable `AUCTION_DURATION` in **AuctionStorage** contract at line 8.
- Consider declaring a variable as `immutable` when it is only assigned during the contract deployment and its value is not intended to change later:
  - Variable `priceDetermination` in **BundleStorage** contract at line 37.
  - Variable `initialMintedAmount` in **NftBundle** contract at line 42.
  - Variables `escrow` and `permissionAddress` in **AuctionManager** contract at lines 30–31.
  - Variable `feeAddress` in **AuctionStorage** contract at line 60.
- The code base has lots of typos in names of functions and variables, in comments, etc. Consider using spell checker.
- Consider declaring visibility of variables explicitly to improve code readability, e.g., `proxyRegistryAddress` variable in **NFTMarketMinter** contract at line 21.

- Variable `proxyRegistryAddress` is not assigned with initial value at line 21 of **NFTMarketMinter** contract.
- Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption.
- Consider implementing minimal proxy pattern in **BundleFactory** contract.
- In **AuctionManager** contract, consider using a constant with value `10000` instead of plain number at lines 169, 230, 237, and 272.
- In **AuctionManager** contract, consider indexing addresses in events at lines 408–413 to simplify off-chain processing.
- In **PermissionManager** contract, consider processing user's address and user's proxy address in the same way.
- In **AuctionManager** contract, consider using `require()` instead of `revert()` inside `if` block at line 216.
- In **AuctionManager** contract, the time check at line 145 is redundant since it always evaluates to `false`.
- The following allowance checks in **AuctionManager** contract are redundant for NFT:
  - At line 172, since NFT is transferred from `_bundleAddress`.
  - At line 73, since NFT is transferred from `msg.sender`.

This analysis was performed by Pessimistic:

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

November 22, 2021