Swarm Markets dOTC Security Analysis

by Pessimistic

This report is public

May 11, 2022

Abstract

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

The initial audit showed one critical issue with Access control, four issues of medium severity, including an Overpowered role, No check for offer expiration, a Bug, and Tests issues, and several low-severity issues.

After the initial audit, the codebase was updated. In this update, the developers fixed most of the issues and added new functionality to the project. However, we discovered new issues in the updated codebase.

After the recheck, the codebase update #2 was performed. This update included fixes and new functionality. We discovered a critical Front-run issue in this update.

After the recheck #2, the codebase was updated again. The developers fixed the Front-run critical issue and one low-severity issue and added new functionality and new tests.

General recommendations

We recommend fixing the rest of the issues, implementing CI to run tests and analyze the codebase with linters and security tools.

Project overview

Project description

For the audit, we were provided with Swarm Markets dOTC project on a private GitHub repository, commit 2a5eaf6189d2859e286adab804badf656d49ae22.

The scope of the audit includes only:

- **contracts/P2POTC** folder.
- **contracts/Escrow/escrow.sol** file.
- **contracts/interfaces/IEscrow.sol** file.
- **contracts/interfaces/IdOTC.sol** file.
- Dependencies for these files.

The private documentation for the project was provided as a Google Docs link.

The project includes 434 tests. Some of them do not pass. For the scope of the audit, the code coverage is 75.1%.

The total LOC of audited sources is 498.

Codebase update #1

After the initial audit, the codebase was updated. For the recheck #1, we were provided with the commit 8f7cc89996e4eb4c9763538bac488e574f97160e.

In this update, the developers fixed most of the issues, added new functionality and a new contract. The code coverage of the scope decreased to 69.4%.

Codebase update #2

After the recheck, another codebase update was performed. For the recheck #2, we were provided with commit 24e94400abbe5f0b3e03b309129a5b796d4cdb56.

In this update, the developers fixed a few issues and added new functionality. However, we found one critical issue and one low-severity issue in this version of the code. Some tests still do not pass. The code coverage of the scope decreased to 65.9%.

Codebase update #3

After the recheck #2, the developers performed a new update. For the recheck #3 we were provided with commit ea6e0aa6f97feca4c7bfe0493dfd594f132a62e4.

In this update, the developers fixed the issues from the previous recheck, added new functionality and new tests, and updated the documentation.

Two tests out of 449 do not pass. The code coverage of the scope decreased to 55.63%.

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 codebase with the automated tools Slither and SmartCheck.
    - We manually verify (reject or confirm) all the issues found by the tools.
- Manual audit
    - We manually analyze the codebase 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.

Access control (fixed)

The architecture of the project considers that `cancelDeposit` function of **SwarmDOTCEscrow** contract should only be called from **DOTCManager** contract. However, the function does not verify the caller. As a result, anyone can cancel any offer. Also, since the function does not verify other arguments, anyone can also request a transfer of any amount of any tokens from **SwarmDOTCEscrow** contract to any address.

We recommend adding a check to verify that `cancelDeposit` function is called by **DOTCManager** contract with proper parameters.

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

Front-run (fixed)

In **DOTCManager** contract, a `maker` of an offer can front-run users who call `takeOffer` function. A `maker` can call `updateOffer` function and change the `amountOUT` parameter. This will change `amountOut / amoutIn` ratio. As a result, a `taker` will receive less tokens than expected.

We recommend passing minimal expected return value to `takeOffer` function as a parameter.

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

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 check for offer expiration (fixed)

In **DOTCManager** contract, `takeOffer` function does not check whether the offer has expired. As a result, users can take offers at any moment. Consider using `can_buy` modifier in this function.

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

Bug (fixed)

In **DOTCManager** contract, `takeOffer` function decreases `allOffers[offerId].amountOut` by `amountToReceiveByTaker` value at line 163. Instead, the function should subtract `_amount` value from `amountOut` since `amountOut` is the value that the maker can receive.

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

Tests issues

Some tests do not pass. The overall code coverage is low. Also, the tests do not cover important scenarios, e.g., escrow unfreezing deposit, offer removal, the whole **NFTDOTCManager** contract etc.

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.

Overpowered role

The admin role is overpowered. Particularly, an admin can:

- Change the fee to any value at any moment.
- Withdraw fees to an admin's address.
- Change `permissionAddress` to any value. A wrong address can break the authorization process for users.
- Change the escrow address, which is used for deposits transfers.
- Pause/unpause the escrow.

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

After the recheck, the developers added new roles and modified special powers and their distribution between new roles:

1. `dOTC_Admin_ROLE` of **DOTCManager** and **NFTDOTCManager** contracts can:
    - Freeze/unfreeze the escrow.
    - Change token list manager address.
    - Freeze/unfreeze any offers.
    - Remove any offers.
    - Change fees at any moment.
    - Change the address for fee withdrawals.
    - Change `permissionAddress` to any value. A wrong value can break the authorization process for users.

2. `ESCROW_MANAGER_ROLE` and `NFT_ESCROW_MANAGER_ROLE` of **DOTCManager** and **NFTDOTCManager** contracts can:
    - Change the escrow address for deposit transfers at any moment. This address receives funds from new offers. Thus, a sudden change of the address to a wrong value can result in a loss of funds.
    - Change the address of the manager contract in the escrow.

3. `FEE_MANAGER_ROLE` with `dOTC_Admin_ROLE` can:
    - Change fees at any moment.
    - Change the address for fee withdrawals.

4. `PERMISSION_SETTER_ROLE` with `dOTC_Admin_ROLE` can:
    - Change `permissionAddress` to any value. A wrong address can break the authorization process for users.

5. `DEFAULT_ADMIN_ROLE` of **SwarmDOTCEscrow**, **DOTCManager**, and **NFTDOTCManager** contracts can:
    - Can grant any role to any address.

We still consider these roles overpowered. Thus, our recommendations still apply.

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

1. (fixed) In **SwarmDOTCEscrow** contract, consider adding `emit` keyword at line 78 to emit `Withdraw` event.

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

2. (fixed) Using `indexed` properties for events is useful when they are considered for further search. The following events have no `indexed` properties:

   - In **DOTCManager** contract, lines 409–437.
   - In **SwarmDOTCEscrow** contract, lines 217–222.

   As a result, searching through these events is slow and expensive.

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

3. (fixed) In **DOTCManager** contract, consider replacing the checks at lines 241, 252, and 266 with `isAdmin` modifier since `DOTCManager` inherits from `AdminFunctions` contract.

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

4. Consider declaring functions as `external` instead of `public` when possible to improve code readability and optimize gas consumption in **DOTCManager**, **NFTDOTCManager**, **SwarmDOTCEscrow**, and **AdminFunctions** contracts.

5. (fixed) In **SwarmDOTCEscrow** contract, most of the functions always return `true`. Moreover, other contracts never check these returned values. Consider removing unused return parameters.

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

6. (fixed) In **IdOTC** interface, `takenAmount` field of `Offer` structure is only assigned and updated but never read within the project. It is also redundant for off-chain processing since it can be calculated as `amountIn - amountOut`. Consider removing `takenAmount` field from `Offer` struct.

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

7. (fixed) In `makeNFTOffer` function of **DOTCManager** contract, using a ternary operator in the assignment expression at line 236 makes no sense. Consider assigning a new value without any additional conditions.

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

8. Comments in the codebase contain many typos.

9. **NFTDOTCManager** contract inherits from **ERC1155Holder** contract and therefore can receive ERC1155 tokens. However, there is no way to withdraw tokens from this contracts. Thus, if anyone sends ERC1155 tokens to **NFTDOTCManager**, these tokens will be stuck.

Gas consumption

1. (fixed) The following checks for the sender's balance are redundant since the balance is checked in **ERC20** contract:
    - In **DOTCManager** contract, checks at lines 298, 311, and 329.
    - In **SwarmDOTCEscrow** contract, check at line 109.

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

2. (fixed) In **AdminFunctions** contract, consider declaring `BPSNUMBER` and `DECIMAL` variables as constants.

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

3. Optimizing structs allows decreasing the number of occupied slots in storage. Consider packing into a single slot a field of `address` type with fields of `bool` type in:
    - `MakerDeposit` and `NftDeposit` structs of **SwarmDOTCEscrow** contract at lines 14–20 and 22–28 (after updates, these structs were combined into a single `Deposit` struct).
    - `Offer` and `NftOffer` structs of **IdOTC** interface at lines 16–31 and 40–52 (after updates, `NftOffer` struct was renamed to `Offer` struct and moved to **INFTdOTC** interface).

4. (fixed) Updating fields of a struct consequentially consumes more gas than a single assignment. Consider assigning `offer` struct in **DOTCManager** contract as `NftOffer({...})` instead of lines 230–238.

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

Code style (fixed)

The code does not follow the Solidity Style Guide. Multiple recommendations are ignored, including Naming Convention, Order of Layout, etc.

We recommend following the Style Guide.

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

Notes

Unsafe transfer

`takeNftOffer` and `cancelNftOffer` functions of **NFTDOTCManager** contract perform external calls to `IERC1155Receiver(to).onERC1155BatchReceived()` through a `safeBatchTransferFrom` call. Note that the receiving contract can have any logic.

This analysis was performed by Pessimistic:

Daria Korepanova, Security Engineer
Nikita Kirillov, Junior Security Engineer
Evgeny Marchenko, Senior Security Engineer
Ivan Gladkikh, Junior Security Engineer
Boris Nikashin, Analyst
Irina Vikhareva, Project Manager

May 11, 2022