Swarm Markets
SX1411 - AssetToken
Security Analysis

by Pessimistic

This report is public

December 30, 2021

Abstract

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

The initial audit showed no critical issues. However, two bugs and incorrect usage of OpenZeppelin library were discovered. The contracts also depend heavily on users with special roles and include a number of low severity issues.

The details of interaction between the contracts and off-chain code are not documented.

After the initial audit, the code base was updated. In this update,Overpowered role and most of low severity issues were fixed. Also, code coverage was significantly improved.

After the recheck #1, another code base update was performed. In this update, all medium severity issues were fixed.

General recommendations

Almost all of the issues were fixed. Thus, we do not have any further recommendations.

Project overview

Project description

For the audit, we were provided with Swarm Markets project on a GitLab repository, commit 3f4aa34f9d12cca3dea86baa50173548928253bc.

The project has README.md file, and private documentation.

The project compiles with warnings and the compiler version in the config file is set incorrectly.

Not all tests pass (15/16), the code coverage is 20.56%.

The total LOC of audited sources is 405.

Code base update #1

For the recheck #1, we were provided with Swarm Markets project on a private GitHub repository, commit 23f1f08c6d5a944dede093d40d9ffcdf30ceb0e5.

In this update, a few issues were fixed. Also, new tests were added to the project, the code coverage increased up to 99.56%.

Code base update #2

After the recheck #1, another code base update was performed. For the recheck #2, we were provided with commit 272df8639084bf53fb1eda826352ad8b38bcc806.

In this update, issues of medium severity were fixed and new tests were added to the project. However, the code coverage decreased to 99.2%. Also, developers added new functionality and provided a documentation.

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.

The audit showed no critical issues.

Medium severity issues

Medium severity 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.

Bug (fixed)

- When a user stakes their wrapped tokens in **xToken** contract, their tokens are consumed twice. First, the tokens are transferred to the contract balance during `stake()` function call. Second, the same amount of tokens are burned from the users balance when the Guardian approves redemption request.
- The same token rate is applied for mint and for burn in **xToken** contract. Consider the following example:

  Mint operation was initiated with `amount = 100`, and `rate = 2`, which updates user's balance `balanceOf(user) = 50`.

  For simplicity, rate does not change, and burn operation with `amount = 25` uses all the user's tokens.

  That is, off-chain `amount` values are `100` for direct operation and `25` for the reverse (`amount → amount / rate ^ 2`). However, one can reasonably expect these operations use the same value.

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

Incorrect integration with OpenZeppelin (fixed)

Function `_beforeTokenTransfer` of **xToken** contract at line 90 should be `virtual` and call `super._beforeTokenTransfer(from, to, amount)` as described in OpenZeppelin documentation.

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

Overpowered role (fixed)

During staking in **xToken** contract, the user gives a Guardian approval for amount at lines 224 and 238, which is not required by the contract. Therefore, the Guardian can withdraw additional amount of tokens from the user.

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

Low severity issues

Low severity issues don't directly affect project's operations. However, they might lead to various problems in the future versions of the code. We recommend taking them into account.

Code quality

- `update` function of **CompoundRateKeeper** contract has TODO comment at line 48 and this is not implemented in the code.

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

- Storage variables of **xToken** contract are considered `internal` by default. Declare visibility of `mintRequestID` , `redemptionRequestID`, `hundredPercent` variables explicitly to improve code readability.

  Same issues at line 5 in **AdminManager** contract and at lines 21, 14 in **MembersManager** contract.

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

- Variables of **xToken** contract at line 48 are set via constructor and never changed later. Declare them as `immutable` to reduce gas consumption.

  This issue is not relevant anymore in the latest version of the code.

- **AgentManager** contract has `equalStrings` and `notEqualStrings` functions at lines 27-33 which are almost identical.

  This part of the code has been removed from the code base.

- Mark `transferOwnership` function of **AdminManager** contract as `external`. It improves readability and saves a bit of gas.

  This part of the code has been removed from the code base.

- Literal `21979553151239153027` at line 42 in `setInterestRate` function of **CompoundRateKeeper** contract should be declared as constant. Also, consider adding a comment in the code that explains where this value comes from.

  > *Comment from developers: Regarding the constant, this is a number intended to limit the possible interest rate that can be set per year. The original developer indicated we should limit it as at some point the high numbers would cause an error. The limit we selected was 100% per year, which when calculated per second, derives the constant in the contract. You can also see that using this formula, which converts the annual interest rate to a per-second interest rate. Using the value 2 to represent 100%, we get the same constant as is in the contract.*

- Consider replacing `10 ** 27` with `1e27` at line 17 in **CompoundRateKeeper** contract.

  Same issue in **xToken** contract at line 46.

> *Comment from developers: Given in the **DSMath** library the `RAY` definition it's written like this `10**27`, I replicated that in the **CompoundRateKeeper**.*

Gas consumption

Additional storage variable assignments are redundant in **xToken** contract:

- at line 172.

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

- at line 127.

Project management (fixed)

- `package-lock.json` is in `.gitignore`. Best practice is to commit it to the repository.
- Some tools are added to the `dependencies` list rather than in `devDependencies`.

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

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Daria Korepanova, Security Engineer
Irina Vikhareva, Project Manager

December 30, 2021