Swarm Markets Security Analysis

by Pessimistic

This report is public.

Published: April 15, 2021

Abstract....2
Disclaimer....2
Summary....2
General recommendations....2
Project overview....3
Project description....3
Latest version of the code....3
Procedure....4
Manual analysis....5
Critical issues....5
Medium severity issues....6
Use of tx.origin (fixed)....6
Typos (fixed)....6
Underflow (fixed)....6
Bug....7
ERC20 standard violation....7
Low severity issues....8
Code quality....8
Code logic....9
Gas consumption....9
Project management....10

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 audit showed several issues of medium severity, including Use of tx.origin, Typos, Underflow, Bug, and ERC20 standard violation, and many issues of low severity, mostly of Code quality and Code logic types.

The project has the documentation.

After the audit, the code base was updated to the latest version. Most of the issues were either fixed or commented.

General recommendations

We do not have any further recommendations.

Project overview

Project description

For the audit, we were provided with two projects on a private GitHub repository:

- Swarm Markets project, commit 01c9046eff8e378b70681b3f22fe974d260169e8.
- Balancer fork, commit 5205d06135006b0a5764e666c2af2690af5be8b0.

The project has tests and documentation. All tests pass without any issues, code coverage is above 90%.

The scope of the audit included:

- swarm-markets-smart-contracts/contracts/token/*
- swarm-markets-smart-contracts/contracts/authorization/*
- swarm-markets-smart-contracts/contracts/balancer/BPoolProxy.sol
- swarm-markets-smart-contracts/contracts/balancer/ProtocolFee.sol
- swarm-markets-smart-contracts/contracts/permissioning/*
- swarm-markets-balancer-core/*

Latest version of the code

For the recheck, we were provided with two pull requests:

- https://github.com/Altoros/swarm-markets-smart-contracts/pull/55
- https://github.com/Altoros/swarm-markets-balancer-core/pull/7

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 smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence project operation in current implementation. We highly recommend addressing them.

Use of tx.origin (fixed)

- In `transfer()` function of **XToken** contract, consider using `msg.sender` instead of `tx.origin` at line 178 as `tx.origin` might not be sending tokens.
- `unwrap()` function of **XTokenWrapper** contract sends ether to `tx.origin`. If the user uses smart wallet and sends meta-transactions via relayers, the function will send ether to the relayer instead of the user.

  Consider sending ether to `msg.sender`.

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

Typos (fixed)

There are a few typos in names of the functions, which results in wrong signatures:

- In **authorization/Authorization.sol** at line 145, there is `setTradingLimint` instead of `setTradingLimit`.
- In **balancer/ProtocolFee.sol** at lines 138, 148, and 153, there is `toatlSwapFeeAmount` instead of `totalSwapFeeAmount`.
- All over the code there is used word `Setted` instead of `Set`.

We recommend fixing these typos to avoid integration issues.

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

Underflow (fixed)

Using assets with `decimals` greater than 18 will result in an underflow in function `calculateAmount()` of **EurPriceFeed** contract at line 181:

```
return _amount.mul(10**uint256(18 - assetDecimals)).mul(assetPrice);
```

In this case, the function will return `0`.

Consider adding checks that arithmetical operations are safe.

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

Bug

In `multihopBatchSwapExactIn()` function of **BPoolProxy** contract, the condition at line 397 should be `k >= 1` instead of `k == 1` to prevent any leftovers for longer hop sequences.

> *Comment from developers: This was in the original contract version from Balancer. There is a risk in changing this because there are no unit tests for this function, the code was taken from the verified contract on etherscan. Although the specific behavior added to this function is covered by unit test.*

ERC20 standard violation

EIP-20 states:

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

However, in **BPoolProxy** contract, the returned values of `ERC20.approve()` calls are not checked.

We highly recommend following ERC20 standard to minimize integration issues.

> *Comment from developers: all ERC20 used within the **BPoolProxy** are going to be **xTokens**, which implements the `approve` function with `revert`. Same case for utility token.*

Low severity issues

Low severity issues can influence project operation in future versions of code. We recommend taking them into account.

Code quality

- Consider declaring functions as `external` instead of `public` where possible.

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

- In **XToken** contract, consider obtaining function signature via selector, e.g. `ERC20Pausable.transfer.selector`.

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

- In **BPoolExtend** contract, consider using selectors (`BPool.joinPool.selector`) instead of hardcoded values.

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

- Natspecs for `setXTokenWrapper()` and `setOperationsRegistry()` functions of **XTokenFactory** contract are copied from `setEurPriceFeed()` function and therefore are misleading.

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

- Natspec for `swapSequences` parameter of `multihopBatchSwapExactOut()` function in **BPoolProxy** contract should mention that the parameter is restricted to include only one- or two-hop sequences.

- There are many blocks of code with the following structure:

  ```
  if (A) {return true;} else {return false;}
  ```

  E.g., at lines 320–323 in `_hasItem()` function of **PermissionManager** contract. Consider replacing such blocks with `return A;` expressions.

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

- Functions `setEurPriceFeed()`, `allowAsset()`, and `disallowAsset()` of **OperationsRegistry** contract always return `true` which is never used. Also, internal `transferFrom(address, uint)` function of **BPoolProxy** contract always returns `false` and its return value is never checked.

  Consider removing return value and `returns (bool)` part from these functions.

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

Code logic

- In **BPoolProxy** contract, consider using SafeERC20 library for transfers since many tokens do not fully comply with ERC20 standard and do not return `bool success` value, e.g. USDT.

  > *Comment from developers: All ERC20 used within the **BPoolProxy** are going to be **xTokens**, which implements safe transfer, returning `true`/`false`. Same case for utility token.*

- Internal `transfer()` function of **BPoolProxy** contract does not call `token.transfer(msg.sener, amount)` and therefore does not emit ERC20 Transfer event if `amount == 0`. Thus, there might be fewer events than expected.

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

- Consider checking that `_operationsRegistry`, `_permissionManager`, and `_exchProxy` are initialized prior to new **BPool** deployment in `newBPool()` function of **BFactory** contract.

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

- In **BToken** contract, consider declaring name and symbol different from those of **Balancer** contract.

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

Gas consumption

- The checks at lines 301–303, 349–351, 404–406, and 454–456 of **BPoolProxy** contract do not protect from front-running attacks.

  We recommend removing these lines and giving an infinite approve to the pool.

  > *Comment from developers: This was in the original contract version from Balancer. There is a risk in changing this because there are no unit tests for this function, the code was taken from the verified contract on etherscan. Although the specific behavior added to this function is covered by unit test.*

- Consider comparing the second return value of `viewSplitExactIn()` call with `minTotalAmountOut` value to save gas in case of revert in `smartSwapExactIn()` and `smartSwapExactOut()` functions of **BPoolProxy** contract.

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

- Consider applying protocol fee `getProtocolFeeAmount` after the loop at line 120 in `batchFee()` function of **ProtocolFee** contract to optimize gas consumption.

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

- In **BToken** contract, initialization of variables at lines 60–62 has no effect on pools.

Project management

- The project compiles with warnings.
- Scripts in `package.json` require hardhat to be installed globally though it is used inside `devDependencies` block.

This analysis was performed by Pessimistic:

Evgeny Marchenko, Senior Security Engineer
Vladimir Tarasov, Security Engineer
Daria Korepanova, Security Engineer
Boris Nikashin, Analyst
Alexander Seleznev, Founder

April 15, 2021