PROTOFIRE

SMART CONTRACT AUDIT REPORT

SwarmMarkets

Protofire
August, 2022

The SwarmMarkets audit report

The audit report was prepared for **SwarmMarkets** and includes only the "**AssetTokenIssuer.sol**" file (not all repository contracts).

Git repository:
https://github.com/SwarmMarkets/asset-token-contract/blob/73fd8d696a3a9d71afd4c4acffb629dd10cbf836/contracts/assetToken/AssetTokenIssuer.sol

Audited commit:
73fd8d696a3a9d71afd4c4acffb629dd10cbf836

Update

The audit report has been updated after recheck commit:
8115cae264a3ffc795c3ae2541f354f34304937c

Procedure

The audit includes the following procedures:

- code analysis by the Slither static analyzer, followed by manual analysis and selection of problems;
- manual code analysis, including logic check;
- checking the business logic for compliance with the documentation (or white paper).

In addition, during the audit, we also provide recommendations on optimizing smart contracts and using best practices.

Summary of audit findings

Severity	Count
HIGH	1
MEDIUM	1
LOW	1
INFORMATIONAL	2
TOTAL	**5**

Severity classification

High severity issues

High severity issues can lead to a direct or indirect loss of funds by both users and owners, a serious violation of the logic of the contract, including through the intervention of third parties. Such issues require immediate correction.

Medium severity issues

Medium severity issues may cause contract functionality to fail or behave incorrectly. Also, medium severity issues can cause financial damage. Such issues require increased attention.

Low severity issues

Low severity issues do not have a major security impact. It is recommended that such issues be taken into account.

Informational severity issues

These issues provide general guidelines for writing code as well as best practices.

Smart contract AssetTokenIssuer

The contract allows users to mint an *assetToken*. The price is determined on the basis of data from the *priceFeed* contracts (Chainlink). The contract administrator can determine which assets can be paid for by the user for minting. At the same time, each new asset and price feed must be added carefully by the administrator so that the price feeds are denominated in same currency.

In addition, the contract has several admin functions for setting up configurations.

It is assumed that the contract will be deployed using a proxy pattern.

ID: **SM01-01**	Severity: **High**	Status: **Fixed**

Unlimited fees - FIXED

The AssetTokenIssuer contract has the *feeBPS* variable. It determines the amount of fees charged from the user when buying tokens in the *mint()* function.

The contract administrator can set the *feeBPS* variable to any value (L58, L99). Thus, the amount of commissions can be 100% or even exceed this value.

We strongly recommend limiting the value of the *feeBPS* variable.

Recheck status: the issue has been fixed.

ID: **SM01-02**	Severity: **Medium**	Status: **Fixed**

Changing asset without changing priceFeed

The contract administrator can change the addresses of the *assetTokenAddress* and the *assetTokenPriceFeedAddress* variables. Such changes are performed separately by calling functions *setAssetTokenAddress()* and *setAssetTokenPriceFeedAddress()*.
An attacker can take advantage of this in the following way. First, it detects a transaction by changing the address of the *assetTokenAddress*. If the new token (asset) has a higher market value than the previous one, then the attacker will take advantage of the fact that the *assetTokenPriceFeedAddress* has not yet been updated and will make a purchase at the price of the previous token. Such a transaction can be executed using the front-run pattern.

We recommend combining the *setAssetTokenPriceFeedAddress()* and *setAssetTokenAddress()* functions into one. And also recommend writing more tests even for incorrect behavior.
At the same time, any change to the token or price feed should be done with caution and be tested in advance.

Additionally, it should be noted if the main token (*assetTokenAddress*) and its price feed change. And the new price feed will be denominated in a different currency, then before such a change, the administrator must clear the *authorizedAssetsPriceFeedAddresses* mapping.

Recheck status: the issue has been fixed.

ID: **SM01-03**	Severity: **Low**	Status: **Fixed**

Gas saving

1) The functions *initialize()*, *getInterestRate()* could be declared with '*external*' visibility type to save gas.

2) The imported library *DSMath.sol* (L12) is never used in the contract and can be removed.

Recheck status: the issue has been fixed.

ID: **SM01-04**	Severity: **Info**	Status: **Fixed**

Code quality

a. **Events for setter functions**
The contract contains several setter functions that do not emit events. Events make it easier to keep track of important changes to a contract (also off-chain).
For example, this may be needed to track all active price feeds.
Consider adding additional events.

b. **Documentation**
Documenting your code is a good practice and also makes it easier to read and refactor your code.
We recommend adding documentation for the contract code and business logic.

c. **Tests**
Consider going beyond unit tests and using functional tests. We highly recommend inventing more complex test-flows to cover all edge-cases with tests.
For example the *mint()* function must be tested after every changing asset or price feed. Also, it should be done with different price feeds.

Recheck status: the issue has been fixed.

ID: **SM01-05**	Severity: **Info**	Status: **Not relevant**

Minting tokens to users

The minting functionality is implemented in the AssetToken contract, which is not part of the audit scope.

However, we recommend that you make sure that the minting cannot be blocked even if the owner (administrator) of the contracts loses the private keys. Or provide an emergency refund functionality to the user.

Recheck status: the issue is not relevant.

Conclusion

The owner (administrator) of the contract must be very careful to add or change price feeds because each price feed should be denominated in the same currency.
The contract administrator must have detailed documentation.
We highly recommend making tests on the fork of the mainnet network with different assets and price feeds to cover all edge cases (even with price feeds denominated with different currencies).

In the updated code, the developer fixed the founded issues.

Disclaimer

Note that smart contract audit provided by Protofire is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, Protofire recommends proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Smart contract audit provided by Protofire shall not be used as investment or financial advice.