spiko × PESSIMISTIC

SECURITY ANALYSIS

by Pessimistic

This report is public

April 1, 2025

ABSTRACT

In this report, we consider the security of smart contracts of Spiko ATM 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. A single 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, a security audit is not investment advice.

SUMMARY

In this report, we considered the security of Spiko ATM smart contracts. We described the audit process in the section below.

The initial audit showed two issues of medium severity: Project roles, Unchecked timestamp of a price. Also, two low-severity issues were found.

After the audit the developers provided us with a new version of the code. In that update, the developers fixed Unchecked timestamp of a price issue. In addition to this, we moved Project roles issue to the notes section.

The overall code quality is good.

GENERAL RECOMMENDATIONS

We do not have any additional recommendations.

PROJECT OVERVIEW

Project description

For the audit, we were provided with Spiko ATM project on a public repository, commit **dfc554f6318fe8e588afdcf571be023a26b5e0ad**.

The scope of the audit included the following files:

- **contracts/token/ATM.sol**;
- **contracts/token/ATM2.sol**;
- **contracts/permissions/PermissionManaged.sol**.

The documentation for the project consisted of private notion document.

All 187 tests pass successfully. The code coverage is 95.45%.

The total LOC of audited sources is 157.

Codebase update

After the initial audit, we were provided with commit 678807393652bc08121ea9566d28a5f37e7ec89d. In that update, the developers fixed the Unchecked timestamp of a price issue and implemented tests for the new code.

All 199 tests pass successfully. The code coverage is 95.5%.

AUDIT PROCESS

We started the audit on March 20, 2025 and finished on March 24, 2025.

We inspected the materials provided for the audit. Based on the discussion with the developers and the provided documentation, we identified the following parts of the project that require additional attention:

- The correctness of price rounding;
- The possibility of a sandwich attack;
- Contract interactions with the **Multicall** contract and meta-transactions;
- Possible malicious actions by non-whitelisted users.

We scanned the project with the following tools:

- Static analyzer Slither;
- Our plugin Slitherin with an extended set of rules;
- AI-powered tool Savant Chat.

We ran tests and calculated the code coverage.

We combined in a private report all the verified issues we found during the manual audit or discovered by automated tools.

On March 27, 2025, the developers provided us with an updated version of the code. In that update, they fixed the Unchecked timestamp of a price issue. In addition to this, they provided comments on other issues. As a result of a discussion, we moved the Project roles issue to the notes section.

We reviewed the updated codebase, updated the statuses of the issues, and inserted developer comments in the respective sections.

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 the 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.

M01. Unchecked timestamp of a price (fixed)

In both **ATM** and **ATM2** contracts timestamp of the prices from the oracle are not checked. We recommend checking them for the case of the oracle misfunctioning.

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

Low severity issues

Low severity issues do not directly affect project operation. However, they might lead to various problems in future versions of the code. We recommend fixing them or explaining why the team has chosen a particular option.

L01. Revert due to underflow (commented)

If there is only one price in the ChainLink, buy and sell transactions in the **ATM2.sol** will be reverting due to underflow. We recommend adding an extra check for the round zero.

> `Comment from the developers:`
>
> Noted. We will never deploy this contract with an Oracle that provides fewer than two price datapoints. No fix is required from our perspective.

L02. Incorrect address is checked (commented)

ATM and **ATM2** contracts support meta-transactions. This implies the use of `_msgSender()` instead of `msg.sender` in the code. Note, that this behavior does not work for restricted functions, since **PermissionManaged** contract uses `msg.sender`.

If this is the expected behavior, ensure that the **_trustedForwarder** contract does not have any roles.

> `Comment from the developers:`
>
> The only restricted function is drain. We will not use a meta-transaction for this function. No roles are assigned to the forwarder. No fix is required from our perspective.

Notes

N01. ATM is susceptible to a sandwich attack (commented)

ATM contract has the same buy and sell prices. That means, it is possible to frontrun Oracle price update transaction and buyout the entire pool in order to sell tokens to it again after for a profit. Note, that according to the developers, **ATM.sol** is not supposed to be used in the production.

> `Comment from the developers:`
> This is precisely why we implemented ATM2. The ATM contract will not be deployed as a standalone component. No fix is required from our perspective.

N02. Documentation mismatch (addressed)

In the documentation, the developers requested verification that "There is no way for an investor who is not allowed to transfer **USTBL** to interact with the **ATM** in any way." However, the code does not check whether `_msgSender` is whitelisted - only the token sender and receiver are verified. This means that a non-whitelisted user can buy tokens on behalf of whitelisted users.

Additionally, some pools are structured in a way that allows pre-sending tokens for sale. If such a pool would be added to the whitelist, a non-whitelisted user could buy tokens using a stablecoin and send them to the pool to sell them instantly.

According to the developers, at the time of writing this report, no such pools are included in the whitelist.

> `Comment from the developers:`
> This is indeed a documentation mismatch. The described behavior is expected, but we acknowledge that the documentation does not currently reflect this. We have corrected it.

N03. DoS by buying out the tokens (commented)

ATM.sol allows user to buy and sell tokens for a fixed price. Since the price changes only through the oracle and buy and sell prices are the same, there is a possible DoS attack on the users by buying out all tokens from the pool.

ATM2.sol has different buy and sell prices, so the attack has a non-zero cost. However, it is still hypothetically possible and the cost of the attack depends on the pool size, the size of the trade to DoS and on the difference between `round` and `round - 1` prices.

> `Comment from the developers:`
>
> In our view, buying the entire supply is not a denial-of-service issue. The limited liquidity for sale is an intentional aspect of the contract's design. No fix is required from our perspective.

N04. The price of stablecoin (commented)

According to the documentation, price of USDC is assumed equal to the price of USD.

> `Comment from the developers:`
>
> This is by design.

N05. Project roles (addressed)

In the current implementation, the system depends heavily on the owners of the project:

- There are responsible for liquidity provision;
- They are able to withdraw tokens from the pool.

Thus, certain scenarios can lead to undesirable consequences for the project and its users, such as if the owners's private keys become compromised. We recommend designing contracts in a trustless manner or implementing proper key management, e.g., setting up a multisig.

> `Comment from the developers:`
>
> As an RWA issuer and regulated entity, our system is intentionally highly centralized. Administrative functions are managed via a multisig secured with hardware wallets.

This analysis was performed by Pessimistic:

Pavel Kondratenkov, Senior Security Engineer
Yhtyyar Sahatov, Security Engineer
Irina Vikhareva, Project Manager
Alexander Seleznev, CEO

April 1, 2025