# ATM Audit Report

Version 1.0.0

Serial No. 2021042200022017

Presented by Fairyproof

April 22, 2021

# 01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the ATM project, at the request of the ATM team.

**Audited Code's Github Repository:**

N/A

**Audited Code's Github Commit Number:**

N/A

**Audited Contract Files' HECO Onchain Address:**

0xE5ed86704819b5C5ec41eeeb52BB739fd36Aec20

**Audited Contract Files:**

There is only one file that has been audited. The calculated SHA-256 value for the file is as follows:

```
ATM.sol:   0x69acc32445a6862a0360f3586e0ba33fbabdf8e793425e0b936957ac2a6a07f9
```

The goal of this audit is to review ATM's solidity implementation for its token issurance function, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

# — Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding smart contract security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. Risks or issues introduced by using data feeds from offchain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

# — Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's smart contracts.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

# — Structure of the document

This report contains a list of issues and comments on all the above contract files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

# — Documentation

For this audit, we used the following sources of truth about how the ATM system should work:

http://www.atm-chain.com/

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the ATM team or reported an issue.

## — Comments from Auditee

No vulnerabilities with critical, high or medium-severity were found in the above contract files.

One vulnerability with low-severity was found in the above contract files.

# 02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying smart contract systems.

# 03. Introduction to ATM

ATM is a blockchain platform for the next generation of global e-commerce. It intends to improve supply chain efficiency, reduce transaction costs, bridge isolated information islands and create a grand business ecosystem.

# 04. Major functions of audited code

The audited code only implements a token issurance function.
Name of the token: Autonomy Chain
Symbol of the token: ATM
Decimals of precision: 18
Total supply:1 billion.  22% of the total supply has been locked and the vesting period is one year. The owner of the contract or an address which is authorized by the owner can burn the locked tokens.

# 05. Key points in audit

During the audit, we worked closely with the ATM team, checked the possible vulnerabilities associated with token issurance, especially checked the following two areas:

- max supply of the token and mint functions, and
- whether or not the locking function works as expected.

# 06. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- DDos Attack
- Integer Overflow
- Function Visibility
- Logic Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Shadow Variable
- Design Vulnerability
- Token Issurance
- Asset Security
- Access Control

# 07. Severity level reference

Every issue in this report was assigned a severity level from the following:

**Critical** severity issues need to be fixed as soon as possible.

**High** severity issues will probably bring problems and should be fixed.

**Medium** severity issues could potentially bring problems and should eventually be fixed.

**Low** severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

# 08. Major areas that need attention

Based on the provided contract files the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

## - Token Issurance

We checked whether or not the contract file can mint tokens at will and whether or not the total supply is designed as expected.
We didn't find issues or risks in this function or area at the time of writing.

## - Locking Function

We checked whether or not the locking function works as expected

We found an issue. For more detailes please refer to section 11.

## - Miscellaneous

We didn't find issues or risks in other functions or areas at the time of writing.

# 09. List of issues by severity

## A. Critical

**- N/A**

### B. High

**- N/A**

### C. Medium

**- N/A**

### D. Low

**- ATM.sol**

Locked Tokens Can be Burned

## 10. List of issues by contract file

### - ATM.sol

**Locked Tokens Can be Burned: Low**

## 11. Issue descriptions and recommendations by contract file

### - ATM.sol

## Locked Tokens Can be Burned: Low

Source and Description:
The owner of the contract or an address authorized by the owner can burn the locked tokens. This is not what the locking function intends to work.

Recommendation:

Consider not setting the locked token's address to be burned by the owner or an address authorized by the owner.
**Update**: Acknowledged by the ATM team. The team will not set the locked token's address to be burned by the owner or an address authorized by the owner.

# 12. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- In line 406, the `getLockBalance` function has a constant `31536000` . This constant can be replaced by the defined `one_year` to avoid mistakes.
- The `start_time` variable can be set to `public` such that it can be read externally.
- In line 375, the `_lock` function can use a local variable instead of `lock_amount` to save gas.
- In line 331, the `lock_total` variable is never used. It can be deleted.
- In line 349, the `Lock` event is never used. It can be deleted or commented out.