MixBytes()

Akropolis

VESTING SMART CONTRACT AUDIT REPORT

AUGUST 5

2019

FOREWORD
TO REPORT

A small bug can cost you millions. **MixBytes** is a team of experienced blockchain engineers that reviews your codebase and helps you avoid potential heavy losses. More than 10 years of expertise in information security and high-load services and 11 000+ lines of audited code speak for themselves.

This document outlines our methodology, scope of work, and results.

We would like to thank **Akropolis** for their trust and opportunity to audit their smart contracts.

CONTENT
DISCLAIMER

This report was made public upon consent of **Akropolis**. **MixBytes** is not to be held responsible for any damage arising from or connected with the report.

Smart contract security audit does not guarantee a comprehensive inclusive analysis disclosing all possible errors and vulnerabilities but covers the majority of issues that represent threat to smart contract operation, have been overlooked or should be fixed.

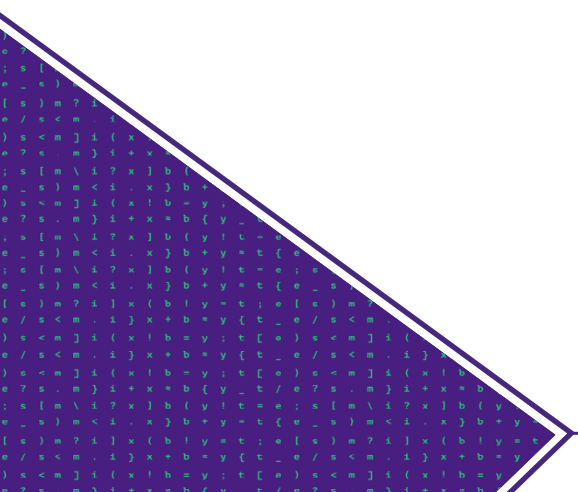


TABLE OF CONTENTS

01 INTRODUCTION TO THE AUDIT

GENERAL PROVISIONS

The **Akropolis** team asked **MixBytes Blockchain Labs** to audit their vesting smart contracts. The code was located in the following **github repository**.

SCOPE OF THE AUDIT

The scope of the audit is smart contracts at **https://github.com/akropolisio/akropolis-vesting/tree/7f4f4543b08d3749b92839c85e1d77a33d917a37/contracts**.

The audited commit is **7f4f4543b08d3749b92839c85e1d77a33d917a37**.

02 SECURITY ASSESSMENT PRINCIPLES

CLASSIFICATION OF ISSUES

CRITICAL

Bugs leading to Ether or token theft, fund access locking or any other loss of Ether/tokens to be transferred to any party (for example, dividends).

MAJOR

Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.

WARNINGS

Bugs that can break the intended contract logic or expose it to DoS attacks.

COMMENTS

Other issues and recommendations reported to/acknowledged by the team.

SECURITY ASSESMENT METHODOLOGY

The audit was performed with triple redundancy by three auditors. Stages of the audit were as follows:

1. “Blind” manual check of the code and its model
2. “Guided” manual code review
3. Checking the code compliance with customer requirements
4. Automated security analysis using the internal solidity security checker
5. Automated security analysis using public analyzers
6. Manual checklist system inspection
7. Discussion of independent audit results
8. Report preparation

03 DETECTED ISSUES

CRITICAL

1. TokenTimelock.sol#L48

Public read-write access to the beneficiary is without any restrictions.

Solution:

We suggest making `TokenTimelock.changeBeneficiary` internal.

Status:

FIXED - at 18474eabd96a6dda2abb39e90493d95e2cb5da9a

MAJOR

1. AkropolisTimeLock.sol#L34

The `changeBeneficiary` method of `TokenTimelock.sol` was incorrectly overridden in `AkropolisTimeLock.sol` that results in an infinite recursive call.

Solution:

We recommend using `super.changeBeneficiary(_newBeneficiary);` or making a base method `internal` and call it this way: `_changeBeneficiary(_newBeneficiary);`.

Status:

FIXED - at 18474eabd96a6dda2abb39e90493d95e2cb5da9a

2. AkropolisVesting.sol#L40

The contract tries to override a non-existing `changeBeneficiary` method of the parent `TokenVesting.sol` that results in an infinite recursive call.

Solution:

If the `changeBeneficiary` method is essential, we recommend trying out the solution described in **Major issue #1 section**.

Status:

FIXED - at **18474eabd96a6dda2abb39e90493d95e2cb5da9a**

3. TimelockProxy.sol#L18, TokenVestingProxy.sol#L18

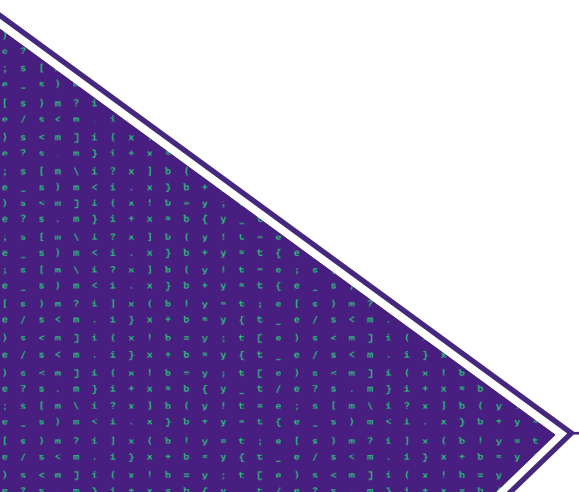
Upgradability operation is broken. Most contract methods are not upgradable because they are handled by the proxy itself. This was caused by the inclusion of `TokenVesting` and `TokenTimelock` into proxy contracts.

Solution:

We suggest rewriting upgradability. A proxy contract should not have any fields but contain methods related to proxy operation. Initialization of a proxy should be conducted using the second argument of the `UpgradabilityProxy` constructor.

Status:

REMOVED - at **18474eabd96a6dda2abb39e90493d95e2cb5da9a**



WARNINGS

1. AkropolisVesting.sol#L33, AkropolisTimeLock.sol#L27

Comment why index 1 was used: `changeBeneficiary(beneficiaries[1]);`. Make sure it is correct.

Solution:

Try using the first element (at index `0`) as an alternative.

Status:

FIXED - at 18474eabd96a6dda2abb39e90493d95e2cb5da9a

2. BeneficiaryOperations.sol#L141

Nested call logic is not working if there are two or more consistent nested calls.

Solution:

We recommend using a stack of calls.

COMMENTS

1. Proxy-ready versions of OpenZeppelin smart contracts

Proxy-ready versions of OpenZeppelin smart contracts with `initialize` method instead of `constructor` may be used:

* **TokenTimelock.sol**
* **TokenVesting.sol**

Status:

REMOVED - at **18474eabd96a6dda2abb39e90493d95e2cb5da9a**

2. `BeneficiaryOperations.sol`

`bereficiarys` should be replaced with `bereficiaries`, the event name should be capitalized. `Ship` inside the event name and throughout the code should be decapitalized:

```
event beneficiaryShipTransferred(
    address[] previousbeneficiaries,
    uint howManyBeneficiariesDecide,
    address[] newBeneficiarys,
    uint newHowManybeneficiarysDecide
);
```

3. `BeneficiaryOperations.sol`

Since the smart contract is a slightly modified version of the **Multiownable smart contract**, some comments about logic changes could be added: **https://www.diffchecker.com/KDsfgVmt**.

4. `BeneficiaryOperations.sol`

No need to implement `function beneficiaryIndices(address wallet) public view returns(uint256)`, since there is a public member `mapping(address => uint) public beneficiariesIndices;` which leads to automatic generation of such public getter by Solidity compiler.

Status:

FIXED - at **18474eabd96a6dda2abb39e90493d95e2cb5da9a**

5. `TokenVesting.sol`

OpenZeppelin-solidity TokenVesting can be imported from:
https://www.diffchecker.com/aJPz04bc

Please note that the developer might have forgotten to implement `changeBeneficiary`(see the **Major issue #2**).

6. BeneficiaryOperations.sol#L73

Multisig wallets with offchain signature aggregation or sidechain signature aggregation can be used instead of unoptimized logic of onlyManyBeneficiaries. Here is a good **example**.

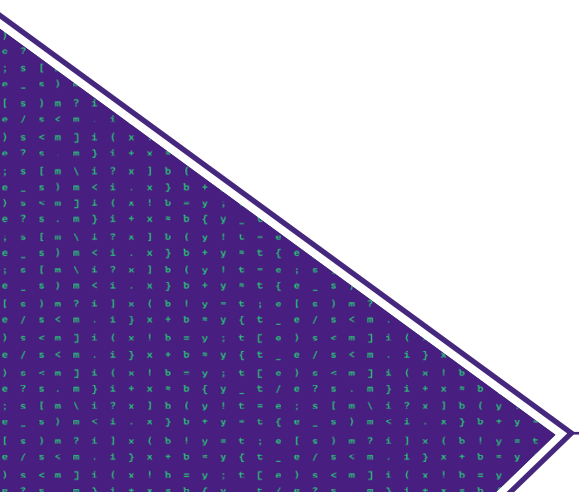
04 CONCLUSION AND RESULTS

We recommend rewriting the upgradability initialization code and conduct a full-scale testing of smart contract logic. We also suggest working out regression tests for the issues described above.

We recommend replacing the `TokenTimelock.sol` and `TokenVesting.sol` smart contracts with the ones implemented in the `openzeppelin-eth` library.

We suggest creating a separate multisig contract and making it the `owner` of the timelock and vesting contracts. The original `Multiownable` library better suits for the advanced multisig smart contracts development.

Major and critical issues are fixed at **18474eabd96a6dda2abb39e90493d95e2cb5da9a**. This version is recommended to deploy at mainnet.



ABOUT MIXBYTES

MixBytes is a team of experienced developers providing top-notch blockchain solutions, smart contract security audits and tech advisory.

JOIN US