MixBytes()

TOKEN MANAGER
SMART CONTRACT
AUDIT REPORT

NOVEMBER 20
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 15 000+ lines of audited code speak for themselves. This document outlines our methodology, scope of work, and results. We would like to thank **Autark** for their trust and opportunity to audit their smart contracts.

CONTENT
DISCLAIMER

This report is public upon the consent of **Autark**. **MixBytes** is not to be held responsible for any damage arising from or connected with the report. Smart contract security audit does not guarantee an 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

Aragon is software allowing to freely organize and collaborate without borders or intermediaries. Create global, bureaucracy-free organizations, companies, and communities.

Autark is an Aragon Network organization building open source tools that serve digital cooperatives and aims to revolutionize work by leveraging the corresponding challenges.

With this in mind, **MixBytes** team is willing to contribute to **Autark** development initiatives by providing security assessment of the Token Manager smart contract and its dependencies.

SCOPE OF THE AUDIT

Audited code:

1. **TokenManager version 72fa119**
2. **WhitelistOracle version 72fa119**

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 ASSESSMENT METHODOLOGY

The audit was performed by 2 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

None found.

MAJOR

1. TokenManager.sol#L272

```
bytes memory input = new bytes(0); // TODO: Consider input for this
```

Comment:

We recommend receiving `input` from function arguments, otherwise it will be impossible to create scripts with arguments.

Status:

ACKNOWLEDGED

> Client: As long as the inputs are already encoded in the _evmScript this behaves the same. **Since the forwarder interface doesn't expect input** and the arguments are already being encoded in the _evmScript this will behave as intended.
>
> I agree at some point it would make sense to expand the forwarders to leverage input, but that's currently out of scope for this contract as it would require architectural changes to the way forwarders behave as well as changes to the wrapper that composes those _evmScripts.

2. TokenManager.sol#L72-L76

```
modifier vestingExists(address _holder, uint256 _vestingId) {
    // TODO: it's not checking for gaps that may appear because of deletes in revokeVesting function
    require(_vestingId < vestingsLengths[_holder], ERROR_NO_VESTING);
    _;
}
```

Comment:

We suggest appending `bool exist` to the `TokenVesting` struct. This would not lead to the struct size increase because of packing.

Status:

FIXED at c2278f6

WARNINGS

1. TokenManager.sol#L24-L29

```
bytes32 public constant MINT_ROLE = keccak256("MINT_ROLE");
bytes32 public constant ISSUE_ROLE = keccak256("ISSUE_ROLE");
bytes32 public constant ASSIGN_ROLE = keccak256("ASSIGN_ROLE");
bytes32 public constant REVOKE_VESTINGS_ROLE = keccak256("REVOKE_VESTINGS_ROLE");
bytes32 public constant BURN_ROLE = keccak256("BURN_ROLE");
bytes32 public constant SET_ORACLE = keccak256("SET_ORACLE");
```

* **WhitelistOracle.sol#L17-L18**

```
bytes32 public constant ADD_SENDER_ROLE = keccak256("ADD_SENDER_ROLE");
bytes32 public constant REMOVE_SENDER_ROLE = keccak256("REMOVE_SENDER_ROLE");
```

Comment:

We advise to use the following snippet:

```
bytes32 public constant MINT_ROLE = 0x154c00819833dac601ee5ddded6fda79d9d8b506b911b3dbd54cdb95fe6c3686;

constructor() public {
    require(MINT_ROLE == keccak256("MINT_ROLE"));
}
```

Status:

FIXED at **78bca05**

2. TokenManager.sol#L196-L205

```
TokenVesting storage v = vestings[_holder][_vestingId];
require(v.revokable, ERROR_VESTING_NOT_REVOKABLE);

uint256 nonVested = _calculateNonVestedTokens(
    v.amount,
    getTimestamp(),
    v.start,
    v.cliff,
    v.vesting
);
```

* **TokenManager.sol#L303-L308**

```
TokenVesting storage tokenVesting = vestings[_recipient][_vestingId];
amount = tokenVesting.amount;
start = tokenVesting.start;
cliff = tokenVesting.cliff;
vesting = tokenVesting.vesting;
revokable = tokenVesting.revokable;
```

* **TokenManager.sol#L416-L423**

```
TokenVesting storage v = vestings[_holder][i];
uint256 nonTransferable = _calculateNonVestedTokens(
    v.amount,
    _time,
    v.start,
    v.cliff,
    v.vesting
);
```

Comment:

We recommend replacing `storage` with `memory` to perform exactly 2 SLOADs instead of 4 or 5, since the struct is packed to two 256-bit slots.

```
TokenVesting memory v = ...;
...
```

Status:

FIXED at **6609575**

3. TokenManager.sol#L167

```
require(vestingsLengths[_receiver] < MAX_VESTINGS_PER_ADDRESS, ERROR_TOO_MANY_VESTINGS);
```

Comment:

We suggest having an array of actual vesting id's for each holder, manage it without gaps and use MAX_ACTIVE_VESTINGS_PER_ADDRESS instead of MAX_VESTINGS_PER_ADDRESS. You can store all holders' vesting in a single mapping.

```
uint256 nextVestingId = 1;
mapping(uint256 => TokenVesting) public allVesting;

mapping (address => mapping (uint256 => uint256)) internal vestingIds;
mapping (address => uint256) public vestingsLengths;

function assignVested(...) {
    uint256 vestingId = nextVestingId++;
    uint256 vestingIndex = vestingsLengths[_receiver]++;
    vestingIds[_receiver][vestingIndex] = vestingId;
    allVesting[vestingId] = TokenVesting(...);
}
```

Besides, it's possible to maintain `mapping(address => mapping(uint256 => uint256)) vestingIndexById` for each receiver to perform a reverse lookup of vestingIndex by vestingId for O(1).

Status:

ACKNOWLEDGED

> Client: I think this change could potentially break backwards compatibility so I'm not sure it's worth it. I'm also concerned around the check in `_transferableBalance` (which while noted as not necessary still currently exists).

COMMENTS

1. TokenManager.sol#L332

```
// Must use transferFrom() as transfer() does not give the token controller full control
require(token.transferFrom(address(this), _receiver, _amount), ERROR_ASSIGN_TRANSFER_FROM_REVERTED);
```

Comment:

We recommend using the transfer and rename `ERROR_ASSIGN_TRANSFER_FROM_REVERTED` to `ERROR_ASSIGN_TRANSFER_REVERTED`. If this method is used when transfersEnabled is switched off, this should be mentioned in the comment above.

```
require(token.transfer(_receiver, _amount), ERROR_ASSIGN_TRANSFER_REVERTED);
```

Status:

ACKNOWLEDGED

2. TokenManager.sol#L395-L399

```
// vestedTokens = tokens * (time - start) / (vested - start)
// In assignVesting we enforce start <= cliff <= vested
// Here we shortcut time >= vested and time < cliff,
// so no division by 0 is possible
uint256 vestedTokens = tokens.mul(time.sub(start)) / vested.sub(start);
```

Comment:

We advise to use SafeMath whenever possible without forcing a user/auditor to read the context.

Status:

ACKNOWLEDGED

3. TokenManager.sol#L171-L177

```
vestings[_receiver][vestingId] = TokenVesting(
    _amount,
    _start,
    _cliff,
    _vested,
    _revokable
);
```

Comment:

We recommend using a safer syntax for struct initialization. Please notice that the variable name `_vested` instead of `_vesting` may have been mistyped.

```
vestings[_receiver][vestingId] = TokenVesting({
    amount: _amount,
    start: _start,
    cliff: _cliff,
    vesting: _vested,
    revokable: _revokable
});
```

Status:

ACKNOWLEDGED

4. WhitelistOracle.sol#L29-L35

```
function addSender(address _sender) external auth(ADD_SENDER_ROLE){
    validSender[_sender] = true;
}

function removeSender(address _sender) external auth(REMOVE_SENDER_ROLE) {
    validSender[_sender] = false;
}
```

Comment:

We recommend adding checks to prevent silent mistakes and events.

```
string private constant ERROR_SENDER_ALREADY_ADDED = "WO_ERROR_SENDER_ALREADY_ADDED";
string private constant WO_ERROR_SENDER_NOT_EXIST = "WO_ERROR_SENDER_NOT_EXIST";

event ValidSenderAdded(address indexed sender);
event ValidSenderRemoved(address indexed sender);

function addSender(address _sender) external auth(ADD_SENDER_ROLE){
    require(!validSender[_sender], WO_ERROR_SENDER_ALREADY_ADDED);
    validSender[_sender] = true;
    ValidSenderAdded(_sender);
}

function removeSender(address _sender) external auth(REMOVE_SENDER_ROLE) {
    require(validSender[_sender], WO_ERROR_SENDER_NOT_EXIST);
    validSender[_sender] = false;
    ValidSenderRemoved(_sender);
}
```

Status:

ACKNOWLEDGED

5. WhitelistOracle.sol#L37

```
function getTransferability(address _from, address _to, uint256 _amount)
external returns (bool) {
```

Comments:

We suggest commenting or removing the names of unused variables if the function must correspond to the signature `getTransferability(address,address,uint256)`. Also, make sure that only `_from` value is enough to determine the function return value.

```
function getTransferability(address _from, address /*_to*/, uint256 /*_
amount*/) external returns (bool) {
```

Status:

ACKNOWLEDGED

6. TokenManager.sol#L140-L143

```
function burn(address _holder, uint256 _amount) external authP(BURN_
ROLE, arr(_holder, _amount)) {
    // minime.destroyTokens() never returns false, only reverts on failure
     token.destroyTokens(_holder, _amount);
}
```

Comments:

We recommend removing the `burn` method. The concern about vestings is about fairness of the already vested amount, but this method allows the admin to burn the holder's balance.

Status:

ACKNOWLEDGED

7. TokenManager.sol#L55-L61

```
MiniMeToken public token;
ITransferOracle public oracle;
uint256 public maxAccountTokens;

// We are only mimicking an array in the inner mapping and use a mapping instead to make an app upgrade more graceful
mapping (address => mapping (uint256 => TokenVesting)) internal vestings;
mapping (address => uint256) public vestingsLengths;
```

Comments:

Make sure that the upgradability framework does not affect the positions of storage variables or use low-level calls with UnstructuredStorage or EthernalStorage.

```
using UnstructuredStorage for bytes32;


bytes32 public constant MINIME_TOKEN = 0x39526e419af036dca68e4194c2c904991e4eed0cdc629df81de1a74466ce9284;


constructor() public {
    require(MINIME_TOKEN == keccak256(“MINIME_TOKEN”));
}


function method() public {
    MiniMeToken token =  MiniMeToken(MINIME_TOKEN.getStorageAddress());
    // do something with token
}
```

Status:

ACKNOWLEDGED

04 CONCLUSION AND RESULTS

Overall security level of the system was rated “Average”. No critical flaws were found. However, there was quite a number of issues worth paying attention to. Some of them can be regarded as a known expected behavior, but others require fixes (e.g. not implemented to-do points and gas cost optimizations).

The fixed contracts of **TokenManager** and **WhitelistOracle** don’t have any vulnerabilities according to our analysis.

ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build open-source solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, consult universities and enterprises, do research, publish articles and documentation.

Stack

Blockchains

JOIN US