
# SMART CONTRACT AUDIT

ZOKYO.

May 13, 2021 | v. 1.0

## PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges

# TECHNICAL SUMMARY

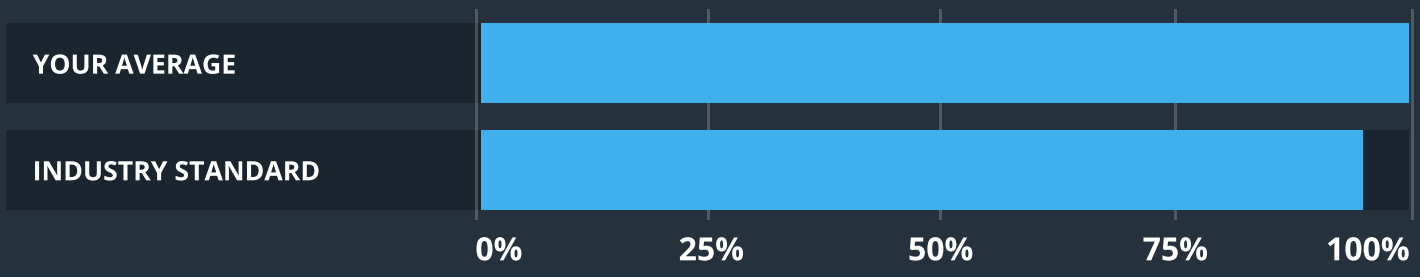
This document outlines the overall security of the Unmarshal smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Unmarshal smart contract codebase for quality, security, and correctness.

## Contract Status


There were 5 critical issues found during the audit.

## Testable Code




The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Unmarshal team put in place a bug bounty program to encourage further and active analysis of the smart contract.

# TABLE OF CONTENTS

# AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Unmarshal repository – https://github.com/DefiWizard/unmarshal-contracts.git.
Commit – a0aeeed9882cc391809816731c4168e52375d031.
Last commit – 9693aa5ba040ffd71e7c7e991999d939e0db7eb0.

**Throughout the review process, care was taken to ensure that the token contract:**

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Unmarshal smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

| | | | |
|---|---|---|---|
| 1 | Due diligence in assessing the overall code quality of the codebase. | 3 | Testing contract logic against common and uncommon attack vectors. |
| 2 | Cross-comparison with other, similar smart contracts by industry leaders. | 4 | Thorough, manual review of the codebase, line-by-line. |

# EXECUTIVE SUMMARY

There were 5 critical issues found during the audit. All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner.

Contracts are well written and structured. The findings during the audit have a slight impact on contract performance or security and are reproduced under specific conditions.

The score is based on the findings, code style, overall security level. Some of the issues (including critical ones) were fixed by Unmarshal team. We also took that into consideration while evaluating the contract score.

## STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

### Critical

The issue affects the ability of the contract to compile or operate in a significant way.

### High

The issue affects the ability of the contract to compile or operate in a significant way.

### Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

### Low

The issue has minimal impact on the contract’s ability to operate.

### Informational

The issue has no impact on the contract’s ability to operate.

# MANUAL REVIEW

## Contract owner can transfer tokens allocated for investors

**CRITICAL** | UNRESOLVED

Contract owner can invoke the smart contract PrivateDistributions's method recoverToken and pass argument _token as UnmarshalToken address to withdraw tokens allocated for investors.

**Snippet:**

```
function recoverToken(address _token, uint256 amount) external onlyOwner {
  IERC20(_token).safeTransfer(_msgSender(), amount);
  emit RecoverToken(_token, amount);
}
```

**Recommendation:**
Forbid to pass argument _token equal to UnmarshalToken address.

## Critical: Method releaseTokens has multiple issues

CRITICAL | RESOLVED

Method releaseTokens of contract PrivateDistribution, that is invocable only by the contract owner, has set of issues at the moment:

- it releases tokens for first 256 registered investors, because of uint8 overflow, if the number of investors is greater than 256;
- it does not increase investor.withdrawnTokens amount, that lets investors withdraw the same withdrawable amount later on;
- it does not generate event WithdrawnTokens;
- it does not track that amount of tokens should be more than zero.

**Snippet:**

```
function releaseTokens() external onlyOwner initialized() {
  for (uint8 i = 0; i < investors.length; i++) {
    uint256 availableTokens = withdrawableTokens(investors[i]);
    _oddzToken.safeTransfer(investors[i], availableTokens);
  }
}
```

**Recommendation:**
It is expected that releaseTokens method should have similar behavior as for `withdrawTokens`, also it should include changes;

- variable `i` should be of type `uint256`;
- method should accept two arguments `offset` & `limit` to release tokens for a portion of investors;
- skip transferring of tokens if withdrawable amount of tokens is 0;
- generate event WithdrawnTokens on successful release (transfer) of tokens.

**Resolution:**
Method is removed.

## Investors will not be able to withdraw tokens after the end of the vesting duration for Ecosystem in UnmarshalTokenVesting

CRITICAL | RESOLVED

Array ecosystemVesting length is equal to 50. If more than 50 months have passed since the initial time function _calculateAvailablePercentage will lead to error.

**Snippet:**

```
if (currentTimeStamp > _initialTimestamp) {
        if (distributionType == DistributionType.ECOSYSTEM) {
          return ecosystemVesting[noOfMonths];
```

**Recommendation:**
Add check for noOfMonths > 50, if it evaluates to true, then returns last value of ecosystemVesting.

## Investors will not be able to withdraw tokens after the end of the vesting duration for STAKING, MARKETING and RESERVES in UnmarshalTokenVesting

CRITICAL | RESOLVED

Array marketingReserveVesting length is equal to 38. If more than 38 months have passed since the initial time function _calculateAvailablePercentage will lead to error.

**Snippet:**

```
if (currentTimeStamp > _initialTimestamp) {
        if (distributionType == DistributionType.ECOSYSTEM) {
            return ecosystemVesting[noOfMonths];
        } else if (
            distributionType == DistributionType.MARKETING ||
            distributionType == DistributionType.RESERVES ||
            distributionType == DistributionType.STAKING
        ) {
            return marketingReserveVesting[noOfMonths];
```

**Recommendation:**
Add check for noOfMonths > 38 if true return last value of marketingReserveVesting.

## Incorrect withdraw logic for team distribution type

CRITICAL | RESOLVED

Function _calculateAvailablePercentage for team distribution will always return 100% if current timestamp is before of initial cliff (240 days).

**Snippet:**

```
if (currentTimeStamp > initialCliff && currentTimeStamp < vestingDuration) {
  uint256 noOfDays = BokkyPooBahsDateTimeLibrary.diffDays(initialCliff, currentTimeStamp);
  if (noOfDays == 1) {
    return uint256(15).mul(1e18);
  } else if (noOfDays > 1) {
    uint256 currentUnlockedPercentage = noOfDays.mul(everyDayReleasePercentage);
    // console.log("Current Unlock %: %s", currentUnlockedPercentage);
    return uint256(15).add(currentUnlockedPercentage);
  }
} else {
  return uint256(100).mul(1e18);
}
```

**Recommendation:**
Extend `else` block with the condition to check if the current timestamp is greater than initial cliff.

## PrivateDistribution does not guarantee the allocation of tokens for investors

HIGH | UNRESOLVED

Method _addInvestor does not allocate tokens for investors during their creation and requires the governance address manager to transfer tokens manually to PrivateDistribution contract address.

**Snippet:**

```
function _addInvestor(
    address _investor,
    uint256 _tokensAllotment,
    uint256 _allocationType
) internal onlyOwner {
    require(_investor != address(0), "Invalid address");
    require(_tokensAllotment > 0, "the investor allocation must be more than 0");
    Investor storage investor = investorsInfo[_investor];

    require(investor.tokensAllotment == 0, "investor already added");

    investor.tokensAllotment = _tokensAllotment;
    investor.exists = true;
    investors.push(_investor);
    investor.allocationType = AllocationType(_allocationType);

    _totalAllocatedAmount = _totalAllocatedAmount.add(_tokensAllotment);
    emit InvestorAdded(_investor, _msgSender(), _tokensAllotment);
}
```

**Recommendation:**
Invoke transferFrom to method _addInvestor on the amount specified in method argument _tokensAllotment.

## Owner can set initial timestamp several times in UnmarshalTokenVesting

HIGH | RESOLVED

Function setInitialTimestamp does not change the value of isInitialized to true.

**Snippet:**

```
function setInitialTimestamp(uint256 _timestamp) external onlyOwner() notInitialized() {
        // isInitialized = true;
        _initialTimestamp = _timestamp;
```

**Recommendation:**
Remove comment.

## Require function can't throw exceptions in UnmarshalTokenVesting

MEDIUM | RESOLVED

Function _addDistribution is internal. All distributions are declared when deploying the contract including tokens allotment and beneficiary address, so require can't throw exceptions for _tokensAllotment > 0 and distribution.tokensAllotment == 0.

**Snippet:**

```
require(_tokensAllotment > 0, "the investor allocation must be more than 0");
require(distribution.tokensAllotment == 0, "investor already added");
```

**Recommendation:**
Remove those exceptions.

## Extra debug events are present in PrivateDistribution smart contract

LOW | UNRESOLVED

Typo in error message in PrivateDistribution smart contract.

**Snippet:**

```
require(tokensAvailable > 0, "no tokens available for withdrawl");
```

**Recommendation:**
Make `withdrawl` to become `withdrawal`.

## Extra debug events are present in PrivateDistribution smart contract

LOW | RESOLVED

PrivateDistribution utilizes library hardhat/console.sol for debugging events.

**Snippet:**

```
console.log("Withdrawable Tokens: %s", tokensAvailable);
console.log("Everyday Percentage: %s, Days: %s, Current Unlock %: %s",everyDayReleasePercentage,
noOfDays, currentUnlockedPercentage);
```

**Recommendation:**
Remove debug events.

## Incorrect comment at Smart contract UnmarshalToken

INFORMATIONAL | RESOLVED

Comment states that `MAX_CAP` is equal to 125mln instead of 100mln.

**Snippet:**

```
uint256 public constant MAX_CAP = 100 * (10**6) * (10**18); // 125 million
```

**Recommendation:**
Remove comment or change it to point to the correct amount.

## Events and variables which are not used in UnmarshalTokenVesting

INFORMATIONAL | RESOLVED

Event DistributionRemoved and variable isFinalized declared but never used.

**Recommendation:**
Remove them if they are not needed.

## Variable that is not used in PrivateDistribution

INFORMATIONAL | RESOLVED

Variable isFinalized declared but never used.

**Recommendation:**
Remove if not needed.

| | UnmarshalToken | PrivateDistribution |
|---|---|---|
| Re-entrancy | Not affected | Not affected |
| Access Management Hierarchy | Not affected | Not affected |
| Arithmetic Over/Under Flows | Not affected | Not affected |
| Unexpected Ether | Not affected | Not affected |
| Delegatecall | Not affected | Not affected |
| Default Public Visibility | Not affected | Not affected |
| Hidden Malicious Code | Not affected | Not affected |
| Entropy Illusion (Lack of Randomness) | Not affected | Not affected |
| External Contract Referencing | Not affected | Not affected |
| Short Address/ Parameter Attack | Not affected | Not affected |
| Unchecked CALL Return Values | Not affected | Not affected |
| Race Conditions / Front Running | Not affected | Not affected |
| General Denial Of Service (DOS) | Not affected | Not affected |
| Uninitialized Storage Pointers | Not affected | Not affected |
| Floating Points and Precision | Not affected | Not affected |
| Tx.Origin Authentication | Not affected | Not affected |
| Signatures Replay | Not affected | Not affected |
| Pool Asset Security (backdoors in the underlying ERC-20) | Not affected | Not affected |

| | UnmarshalTokenVesting |
|---|---|
| Re-entrancy | Not affected |
| Access Management Hierarchy | Not affected |
| Arithmetic Over/Under Flows | Not affected |
| Unexpected Ether | Not affected |

| | |
|---|---|
| Delegatecall | Not affected |
| Default Public Visibility | Not affected |
| Hidden Malicious Code | Not affected |
| Entropy Illusion (Lack of Randomness) | Not affected |
| External Contract Referencing | Not affected |
| Short Address/ Parameter Attack | Not affected |
| Unchecked CALL Return Values | Not affected |
| Race Conditions / Front Running | Not affected |
| General Denial Of Service (DOS) | Not affected |
| Uninitialized Storage Pointers | Not affected |
| Floating Points and Precision | Not affected |
| Tx.Origin Authentication | Not affected |
| Signatures Replay | Not affected |
| Pool Asset Security (backdoors in the underlying ERC-20) | Not affected |

# CODE COVERAGE AND TEST RESULTS FOR ALL FILES

## Tests written by Zokyo Secured team

As part of our work assisting Unmarshal in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Unmarshal contract requirements for details about issuance amounts and how the system handles these.

**Contract: PrivateDistribution**

PrivateDistribution Tests

- ✓ should return initial timestamp correct (63ms)
- ✓ shouldn't change initialized timestamp (80ms)
- ✓ shouldn't be initialized if timestamp wasn't set (268ms)
- ✓ should add investors correct (183ms)
- ✓ shouldn't add investors by not the owner (46ms)
- ✓ shouldn't add investors with the zero address (75ms)
- ✓ shouldn't add investors with token allocation 0 (74ms)
- ✓ shouldn't add investors without proper token allocations (38ms)
- ✓ shouldn't add investor twice (43ms)
- ✓ should display investor initial withdrawable tokens correct (73ms)
- ✓ shouldn't withdraw tokens if not investor (39ms)
- ✓ shouldn't withdraw without available tokens (174ms)
- ✓ should recover tokens correct (112ms)
- ✓ shouldn't recover tokens if amount exceed token balance (43ms)
- ✓ should withdraw tokens after initial cliff period correct (516ms)
- ✓ should withdraw tokens after vesting duration correct (259ms)
- ✓ shouldn't withdraw tokens if initial timestamp wasn't set (388ms)

**Contract: Unmarshal**

Unmarshal Token Tests

- ✓ should deploy with correct name
- ✓ should deploy with correct symbol
- ✓ should deploy with correct decimals
- ✓ should deploy with correct initial total supply

✓ shouldn't set new governance from not the current governance (56ms)
✓ should set a  governance correct (105ms)
✓ should set a governance to the zero address (111ms)
✓ should recover funds correct (83ms)
✓ shouldn't recover funds to token address
✓ shouldn't permit with expired deadline
✓ shouldn't permit with wrong signature
✓ should permit correct (95ms)

**Contract: UnmarshalTokenVesting**
UnmarshalTokenVesting Tests
✓ should display initial timestamp correct
✓ shouldn't change initial timestamp if it is initialized (remove comment)
✓ shouldn't deploy with not valid token address (113ms)
✓ shouldn't deploy with not valid beneficiary address (111ms)
✓ shouldn't withdraw tokens if initial timestamp wasn't set (357ms)
✓ shouldn't withdraw tokens if amount equal to zero
✓ should withdraw tokens for ecosystem distribution type correct (545ms)
✓ should withdraw tokens for staking distribution type correct (425ms)
✓ should withdraw tokens for marketing distribution type correct (464ms)
✓ should withdraw tokens for reserves distribution type correct (446ms)
✓ shouldn't withdraw tokens for team distribution type before initial cliff
✓ should withdraw tokens for team distribution type correct (470ms)
✓ should recover excess Tokens correct (84ms)

42 passing (28s)

| FILE | % STMTS | % BRANCH | % FUNCS | % LINES | UNCOVERED LINES |
|---|---|---|---|---|---|
| ERC20Permit.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| IERC2612Permit.sol | 100.00 | 100.00 | 100.00 | 100.00 | |
| PrivateDistribution.sol | 100.00 | 92.31 | 100.00 | 100.00 | |
| UnmarshalToken.sol | 100.00 | 83.33 | 100.00 | 100.00 | |
| UnmarshalTokenVesting.sol | 98.24 | 90.91 | 100.00 | 98.28 | |
| **All files** | **99.24** | **91.38** | **100.00** | **99.25** | |

We are grateful to have been given the opportunity to work with the Unmarshal team.

**The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.**

Zokyo's Security Team recommends that the Unmarshal team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.