mprolab

SMART CONTRACTS REVIEW

zokyo

April 11th 2024 | v. 1.0

Security Audit Score

PASS

Zokyo Security has concluded that these smart contracts passed a security audit.

ZOKYO AUDIT SCORING METAPRO

1. Severity of Issues:
    - Critical: Direct, immediate risks to funds or the integrity of the contract. Typically, these would have a very high weight.
    - High: Important issues that can compromise the contract in certain scenarios.
    - Medium: Issues that might not pose immediate threats but represent significant deviations from best practices.
    - Low: Smaller issues that might not pose security risks but are still noteworthy.
    - Informational: Generally, observations or suggestions that don't point to vulnerabilities but can be improvements or best practices.
2. Test Coverage: The percentage of the codebase that's covered by tests. High test coverage often suggests thorough testing practices and can increase the score.
3. Code Quality: This is more subjective, but contracts that follow best practices, are well-commented, and show good organization might receive higher scores.
4. Documentation: Comprehensive and clear documentation might improve the score, as it shows thoroughness.
5. Consistency: Consistency in coding patterns, naming, etc., can also factor into the score.
6. Response to Identified Issues: Some audits might consider how quickly and effectively the team responds to identified issues.

HYPOTHETICAL SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: -1 point

Starting with a perfect score of 100:
- 1 Critical issues: 1 resolved = 0 points deducted
- 1 High issue: 1 resolved = 0 points deducted
- 2 Medium issues: 2 resolved = 0 points deducted
- 1 Low issue: 1 resolved = 0 points deducted
- 3 Informational issues: 3 resolved = 0 points deducted

Thus, score is 100

TECHNICAL SUMMARY

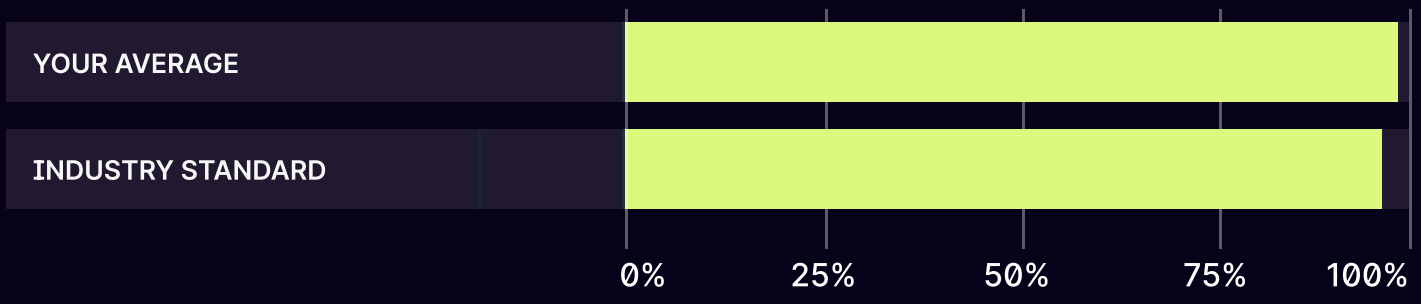
This document outlines the overall security of the Metapro smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

There was 1 critical issue found during the review.

Testable Code



100% of the code is testable, 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/s but rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that can withstand the Ethereum network's fast-paced and rapidly changing environment, we recommend that the Metapro team put in place a bug bounty program to encourage further active analysis of the smart contract/s.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The source code of the smart contract was taken from the Metapro repository:
Repo: https://github.com/Augmented-Life-Studio/mpro-token

Last commit: 220c6847dc1793ea0709eecd5d325d908606b777

Within the scope of this audit, the team of auditors reviewed the following contract(s):

- MPRO.sol
- MPROLight.sol
- MPROMasterDistributor.sol
- MPROMasterDistributorLight.sol
- MPROVesting.sol

During the audit, Zokyo Security ensured that the contract:

- Implements and adheres to the existing standards appropriately and effectively;
- The documentation and code comments match the logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices, efficiently using resources without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the most recent vulnerabilities;
- Meets best practices in code readability, etc.

Zokyo Security has followed best practices and industry-standard techniques to verify the implementation of Metapro smart contract/s. To do so, the code was reviewed line by line by our smart contract developers, who documented even minor issues as they were discovered. Part of this work includes writing a test suite using the Foundry testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.	03	Testing contract/s logic against common and uncommon attack vectors.
02	Cross-comparison with other, similar smart contract/s by industry leaders.	04	Thorough manual review of the codebase line by line.

Executive Summary

The Zokyo team has performed a security audit of the provided codebase. The contracts submitted for auditing are well-crafted and organized. Detailed findings from the audit process are outlined in the "Complete Analysis" section.

STRUCTURE AND ORGANIZATION OF THE DOCUMENT

For the ease of navigation, the following sections are arranged from the most to the least critical ones. Issues are tagged as “Resolved” or “Unresolved” or “Acknowledged” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the Metapro team and the Metapro team is aware of it, but they have chosen to not solve it. The issues that are tagged as “Verified” contain unclear or suspicious functionality that either needs explanation from the Client or remains disregarded by the Client. 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 contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

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.

COMPLETE ANALYSIS

FINDINGS SUMMARY

#	Title	Risk	Status
1	Burn method allows anyone to burn anyone tokens	Critical	Resolved
2	Inaccurate Token Deduction and Burn Mechanism in transferFrom Function	High	Resolved
3	ETH Cannot Be Withdrawn From The Vesting Contract	Medium	Resolved
4	DoS for distribute and distributeBulk methods	Medium	Resolved
5	Beneficiary Might Lose Existing Claimable Reward If Registered Again	Low	Resolved
6	Introduce Sanity Checks	Informational	Resolved
7	Inefficient Looping Over Dynamic Array	Informational	Resolved
8	Unnecessary Initialization of Variables to Zero in Solidity	Informational	Resolved

Burn method allows anyone to burn anyones' tokens

In Contract MPRO.sol and MPROLight.sol, the burn method has the following logic:

```
function burn(address account, uint256 amount) external virtual {
        _burn(account, amount);
    }
```

Here, any malicious user can pass any other users address and all their token balance and burn it all leading to loss of funds for users and the protocol.

Recommendation:

Update the burn() method to burn only msg.senders' tokens.

Inaccurate Token Deduction and Burn Mechanism in `transferFrom` Function

Location: MPRO line 296

Issue Summary:
The current implementation of the `transferFrom` function in the smart contract incorrectly calculates the amount of tokens burned and transferred, leading to potential fund loss for the token approver. This occurs due to the burn mechanism being applied after the token transfer approval but before the actual token transfer, affecting the total amount deducted from the approver's balance.

Detailed Explanation:
In the provided contract logic, when a user (User A) approves another user (User B) to transfer a specific amount of tokens (e.g., 1 million tokens) on their behalf, and User B initiates a `transferFrom` operation with a burn-on-transfer fee, the actual amount transferred plus the burned amount exceeds the approved amount. This results in an incorrect total deduction from User A's account.
The function `_burnOnTransfer` calculates a burn amount based on the transfer amount and subtracts this from the total amount to be transferred. However, if User B transfers an amount such that the sum of the transferred amount and the burn amount exceeds User A's approved amount, User A incurs an unintended loss of funds beyond their approval.

Impact:
This behavior leads to a higher deduction from the approver's account than approved, causing potential fund losses and undermining the trust in the contract's token transfer and burn mechanisms.

Recommendation:
To rectify this issue, we recommend adjusting the `transferFrom` and `_burnOnTransfer` functions to ensure that the burn amount is considered within the total approved amount, not in addition to it. This adjustment should enforce that the total amount deducted from the approver's balance (including any burned tokens) does not exceed the initially approved amount.

A potential solution involves revising the logic to:

- Calculate the burn amount upfront.
- Ensure the sum of the amount to be transferred and the burn amount does not exceed the approved amount.

Deduct the burn amount from the transfer amount before calling `_transferFrom`.

MEDIUM-1 | RESOLVED

ETH Cannot Be Withdrawn From The Vesting Contract

There's a receive() function in the MPROVesting.sol and it is mentioned that the contract should be able to receive ether (for reasons not mentioned) , but this ether cannot be withdrawn since there is no withdraw functionality for ether . Therefore any ether sent to this contract would be stuck inside the contract.

Recommendation:

Either have a withdrawal functionality or remove the receive() function if not needed.

MEDIUM-2 | RESOLVED

DoS for distribute and distributeBulk method

In Contract MPROMasterDistributor.sol, the method getAllTokenDistribution() calculates the tokens so far available to distribute. In this method, if a reduction is added and a user calls it before the reductionTimestamp, the transaction revers due to underflow issue as `index–` in the for loop will try to `0 - 1`.

Since this method getAllTokenDistribution() is called by both distribute() and distributeBulk() methods, these methods will revert as well.

Recommendation:

Update the logic to avoid this scenario by updating the for loop.

Beneficiary Might Lose Existing Claimable Reward If Registered Again

A beneficiary can be registered through the registerBeneficiaries() function in MPROVesting.sol , if the beneficiary already had an amount then this code is executed

```
if (beneficiary.amount > 0) {
                if (beneficiary.claimed > _amounts[i]) {
                    beneficiary.amount = beneficiary.claimed;
                } else {
                    beneficiary.amount = _amounts[i];
                }
```

This means that if the beneficiary already had an amount and a new amount is being assigned then he would lose the previous claimable amount .

Recommendation:
Make sure the beneficiary can not be updated unless he has claimed his remaining rewards.

INFORMATIONAL-1 | RESOLVED

Introduce Sanity Checks

There should be sanity checks performed while assigning values to the tgeUnlockPercent and vestingUnlockPercentPerPeriod in MPROVesting.sol since these values should not go beyond `10000`.

Recommendation:
While assigning these values in the constructor make sure they are less than `10000`.

INFORMATIONAL-2 | RESOLVED

Inefficient Looping Over Dynamic Array

Location: MPROVesting line 179, MPROMasterDistributor line 222, 376, 502, MPRO line 91

Summary: Iterating over a dynamic array using its length property in each loop iteration may lead to excessive gas consumption.

Description: When looping through an array with `for (uint256 i = 0; i < _beneficiaries.length; i++)`, the `.length` property is read from storage on every iteration. In Solidity, storage reads are costly in terms of gas usage. This pattern can significantly increase the transaction cost, especially with larger arrays.

Recommendation:
Cache the array length in a local variable before the loop starts, and use this variable for comparison in the loop condition. This reduces the number of storage reads to just one, lowering the overall gas cost of the transaction.

Unnecessary Initialization of Variables to Zero in Solidity

Location: MPROMasterDistributor Line 133

Summary: In Solidity, variables are initialized to their default values automatically, with `uint256` types defaulting to `0`. Explicit initialization to `0` is redundant and can be omitted for brevity and clarity.

Context: The `distributedTokens` variable is explicitly initialized to `0`, which is unnecessary as Solidity automatically initializes `uint256` variables to `0`.

Recommendation:
Simplify the declaration by omitting the explicit initialization:

```
uint256 private distributedTokens;
```

Removing the explicit initialization makes the code cleaner and emphasizes the use of Solidity's default initialization behavior, without any impact on functionality or gas costs.

	MPRO.sol MPROLight.sol MPROMasterDistributor.sol MPROMasterDistributorLight.sol MPROVesting.sol
Re-entrancy	Pass
Access Management Hierarchy	Pass
Arithmetic Over/Under Flows	Pass
Unexpected Ether	Pass
Delegatecall	Pass
Default Public Visibility	Pass
Hidden Malicious Code	Pass
Entropy Illusion (Lack of Randomness)	Pass
External Contract Referencing	Pass
Short Address/ Parameter Attack	Pass
Unchecked CALL Return Values	Pass
Race Conditions / Front Running	Pass
General Denial Of Service (DOS)	Pass
Uninitialized Storage Pointers	Pass
Floating Points and Precision	Pass
Tx.Origin Authentication	Pass
Signatures Replay	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security

As a part of our work assisting Metapro in verifying the correctness of their contract/s code, our team was responsible for writing integration tests using the Foundry testing framework.

The tests were based on the functionality of the code, as well as a review of the Metapro contract/s requirements for details about issuance amounts and how the system handles these.

Running 14 tests for test/MPRO.t.sol:MPROTest
[PASS] test_FailsIfMintingMoreThanCapAmount() (gas: 80060)
[PASS] test_FailsToMintWhenCapExceeded() (gas: 90747)
[PASS] test_FailsToTransferFromAmountMoreThanApproved() (gas: 52285)
[PASS] test_ShouldApproveSuccessfully() (gas: 49130)
[PASS] test_ShouldInitializeCorrectly() (gas: 62369)
[PASS] test_ShouldMintSuccessfully() (gas: 139359)
[PASS] test_ShouldReturnCapCorrectly() (gas: 8475)
[PASS] test_ShouldTransferFromSuccessfullyAndBurnTokensOnTransfer() (gas: 83318)
[PASS] test_ShouldTransferSuccessfullyAndBurnTokensOnTransfer() (gas: 72476)
[PASS] test_failsToApproveIfEitherUserIsBlacklisted() (gas: 77006)
[PASS] test_failsToTransferFromIfEitherSenderOrFromOrToIsBlacklisted() (gas: 101575)
[PASS] test_failsToTransferIfEitherSenderOrToIsBlacklisted() (gas: 77282)
[PASS] test_failsWhenNonDistributorAddressCalls() (gas: 16599)
[PASS] test_shouldBurnTokensCorrectly() (gas: 40050)
Test result: ok. 14 passed; 0 failed; 0 skipped; finished in 6.90ms

Running 10 tests for test/MPROLight.t.sol:MPROLightTest
[PASS] test_FailsToTransferFromAmountIfNotApproved() (gas: 294961)
[PASS] test_FailsToTransferFromAmountMoreThanApproved() (gas: 329732)
[PASS] test_ShouldApproveSuccessfully() (gas: 49210)
[PASS] test_ShouldBurnTokens() (gas: 287167)
[PASS] test_ShouldSendFromOneChainToAnotherSuccessfully() (gas: 279146)
[PASS] test_ShouldTransferFromSuccessfullyAndBurnTokensOnTransfer() (gas: 343857)
[PASS] test_ShouldTransferSuccessfullyAndBurnTokensOnTransfer() (gas: 316749)
[PASS] test_failsToApproveIfEitherUserIsBlacklisted() (gas: 77827)
[PASS] test_failsToTransferFromIfEitherSenderOrFromOrToIsBlacklisted() (gas: 101242)
[PASS] test_failsToTransferIfEitherSenderOrToIsBlacklisted() (gas: 346517)
Test result: ok. 10 passed; 0 failed; 0 skipped; finished in 7.98ms

Running 75 tests for test/MPROMasterDistributor.t.sol:MPROMasterDistributorTest

[PASS] test_FailsIfAddingDistributionReductionAmountIsGreaterThanDoubleOfTheLastReductionAmount() (gas: 127261)
[PASS] test_FailsIfAddingDistributionReductionAmountIsLessThanHalfOfTheLastReductionAmount() (gas: 127164)
[PASS] test_FailsIfAddingDistributionReductionByNonRoleAccount() (gas: 40374)
[PASS] test_FailsIfAddingDistributionReductionLessThan183DaysAfterLastReductions() (gas: 127016)
[PASS] test_FailsIfCallerIsBlacklistedForApproveAllowed() (gas: 52316)
[PASS] test_FailsIfCallerIsBlacklistedForTransferAllowed() (gas: 52633)
[PASS] test_FailsIfFromIsBlacklistedForTransferAllowed() (gas: 50527)
[PASS] test_FailsIfMintAllowedCalledNotByDistributorOnly() (gas: 13481)
[PASS] test_FailsIfNonOwnerSettingDistributionStartTime() (gas: 18004)
[PASS] test_FailsIfSettingDistributionStartTimeInPast() (gas: 13536)
[PASS] test_FailsIfSettingDistributionStartTimeIsHigherThanDistributionStartTimeDeadline() (gas: 15738)
[PASS] test_FailsIfSpenderIsBlacklistedForApproveAllowed() (gas: 50020)
[PASS] test_FailsIfToIsBlacklistedForTransferAllowed() (gas: 52458)
[PASS] test_FailsToDistributeIfAmountExceedsDistributionLimit() (gas: 24598)
[PASS] test_FailsToDistributeIfAmountIsNotMoreThanZero() (gas: 15794)
[PASS] test_FailsToDistributeIfDistributionNotStarted() (gas: 18064)
[PASS] test_FailsToDistributeIfNotMasterDistributorRole() (gas: 45591)
[PASS] test_FailsToDistributeInBulkIfNotMasterDistributorRole() (gas: 51136)
[PASS] test_FailsToDistributeInBulkIfUsersAndAmountListsDoNotMatch() (gas: 98452)
[PASS] test_FailsToGrantRoleIfGrantedMoraThanOnce() (gas: 67936)
[PASS] test_FailsToGrantRoleToABlacklistedAccount() (gas: 51203)
[PASS] test_FailsToGrantRoleToAddressZero() (gas: 14451)
[PASS] test_FailsToRenounceRoleForAccountWhichDoesNotHaveThatRole() (gas: 17423)
[PASS] test_FailsToRenounceRoleForAddressZero() (gas: 12643)
[PASS] test_FailsToRenounceRoleForAnyOtherAccount() (gas: 16903)
[PASS] test_FailsToSetBurnRateIfBurnRateMoreThan10Percent() (gas: 11300)
[PASS] test_FailsToSetBurnRateIfNonOwnerAccount() (gas: 15815)
[PASS] test_FailsToSetDistributorTimeAdministratorRoleIfRoleAlreadyBurned() (gas: 117528)
[PASS] test_FailsToSetDistributorTimeAdministratorRoleIfSetByNonAuthorisedAccount() (gas: 102631)
[PASS] test_FailsToSetDistributorTimeAdministratorRoleManagerIfSetByNonOwner() (gas: 17884)
[PASS] test_FailsToSetDistributorTimeAdministratorRoleManagerMoreThanOnce() (gas: 65353)

[PASS] test_FailsToSetDistributorTimeAdministratorRoleManagereIfRoleAlreadyBurned() (gas: 63089)
[PASS] test_FailsToSetDistributorTimeAdministratorRoleMoreThanOnce() (gas: 119856)
[PASS] test_FailsToSetMPROIfAlreadySet() (gas: 15564)
[PASS] test_FailsToSetupDistributionStartTimeIfDistributionAlreadyStarted() (gas: 85117)
[PASS] test_FailsToShouldSetMPROIfNonOwnerAccount() (gas: 17962)
[PASS] test_ShouldBlocklist() (gas: 49368)
[PASS] test_ShouldDistributeCorrectly() (gas: 84438)
[PASS] test_ShouldDistributeCorrectlyInBulk() (gas: 262399)
[PASS] test_ShouldGrantRole() (gas: 66793)
[PASS] test_ShouldInitializeCorrectly() (gas: 28404)
[PASS] test_ShouldRenounceRole() (gas: 16925)
[PASS] test_ShouldReturnBurnAmountAs0IfUserWhitelisted() (gas: 42288)
[PASS] test_ShouldReturnCorrectBurnAmount() (gas: 18740)
[PASS] test_ShouldReturnDistributionReductions() (gas: 195863)
[PASS] test_ShouldReturnFalseForNonDistributorAccount() (gas: 10948)
[PASS] test_ShouldReturnFalseForNonListerAccount() (gas: 11046)
[PASS] test_ShouldReturnFalseIfNotBlocklisted() (gas: 15229)
[PASS] test_ShouldReturnFalseIfNotWhitelisted() (gas: 17720)
[PASS] test_ShouldReturnTrueForDistributorAccount() (gas: 10925)
[PASS] test_ShouldReturnTrueForListerAccount() (gas: 13179)
[PASS] test_ShouldReturnTrueIdMintAllowedCalledByDistributorOnly() (gas: 8655)
[PASS] test_ShouldReturnTrueIfBlocklisted() (gas: 49347)
[PASS] test_ShouldReturnTrueIfCheckingAddressZeroWhitelisted() (gas: 8867)
[PASS] test_ShouldReturnTrueIfNoAccountBlacklistedForApproveAllowed() (gas: 19775)
[PASS] test_ShouldReturnTrueIfNoAccountBlacklistedForTransferAllowed() (gas: 20410)
[PASS] test_ShouldReturnTrueIfWhitelisted() (gas: 41849)
[PASS] test_ShouldRevokeRole() (gas: 18426)
[PASS] test_ShouldSetBurnRate() (gas: 17653)
[PASS] test_ShouldSetDistributionStartTime() (gas: 22141)
[PASS] test_ShouldSetDistributorTimeAdministratorRole() (gas: 119616)
[PASS] test_ShouldSetDistributorTimeAdministratorRoleManager() (gas: 65350)
[PASS] test_ShouldWhitelist() (gas: 39798)
[PASS] test_ShuldReturnCorrectDistributionAmountBasedOnTimePassedWithMultpleReductionsAdded() (gas: 258418)
[PASS] test_ShuldReturnCorrectDistributionAmountBasedOnTimePassedWithNoReductionsAdded() (gas: 25099)
[PASS] test_ShuldReturnCorrectDistributionAmountBasedOnTimePassedWithOneReductionAdded() (gas: 213270)
[PASS] test_failsToBlocklistAddressZero() (gas: 14158)

[PASS] test_failsToBlocklistIfCalledByNonLister() (gas: 45571)
[PASS] test_failsToBlocklistOwnerOrDistributorAccountOrListerAccount() (gas: 24916)
[PASS] test_failsToGrantRoleIfCalledByNonOwner() (gas: 18885)
[PASS] test_failsToRevokeRoleIfCalledByNonOwner() (gas: 18925)
[PASS] test_failsToRevokeRoleIfRevokingFromAccountWhichDoesNotHaveRole() (gas: 18810)
[PASS] test_failsToRevokeRoleIfRevokingFromAddressZero() (gas: 12230)
[PASS] test_failsToWhitelistAddressZero() (gas: 14158)
[PASS] test_failsToWhitelistIfCalledByNonLister() (gas: 45615)
Test result: ok. 75 passed; 0 failed; 0 skipped; finished in 8.71ms

Running 39 tests for test/
MPROMasterDistributorLight.t.sol:MPROMasterDistributorTest
[PASS] test_FailsIfCallerIsBlacklistedForApproveAllowed() (gas: 49812)
[PASS] test_FailsIfCallerIsBlacklistedForTransferAllowed() (gas: 50238)
[PASS] test_FailsIfFromIsBlacklistedForTransferAllowed() (gas: 48122)
[PASS] test_FailsIfSpenderIsBlacklistedForApproveAllowed() (gas: 47629)
[PASS] test_FailsIfToIsBlacklistedForTransferAllowed() (gas: 50088)
[PASS] test_FailsToGrantRoleIfAlreadyGranted() (gas: 23126)
[PASS] test_FailsToGrantRoleToABlacklistedAccount() (gas: 48662)
[PASS] test_FailsToGrantRoleToAddressZero() (gas: 14298)
[PASS] test_FailsToRenounceRoleForAccountWhichDoesNotHaveThatRole() (gas: 17262)
[PASS] test_FailsToRenounceRoleForAddressZero() (gas: 12490)
[PASS] test_FailsToRenounceRoleForAnyOtherAccount() (gas: 16796)
[PASS] test_FailsToSetBurnRateIfBurnRateMoreThan10Percent() (gas: 11189)
[PASS] test_FailsToSetBurnRateIfNonOwnerAccount() (gas: 15682)
[PASS] test_ShouldBlocklist() (gas: 46841)
[PASS] test_ShouldInitializeCorrectly() (gas: 18619)
[PASS] test_ShouldRenounceRole() (gas: 16789)
[PASS] test_ShouldReturnBurnAmountAs0IfUserWhitelisted() (gas: 42131)
[PASS] test_ShouldReturnCorrectBurnAmount() (gas: 18551)
[PASS] test_ShouldReturnFalseForNonListerAccount() (gas: 11002)
[PASS] test_ShouldReturnFalseIfNotBlocklisted() (gas: 15139)
[PASS] test_ShouldReturnFalseIfNotWhitelisted() (gas: 17530)
[PASS] test_ShouldReturnTrueForListerAccount() (gas: 13090)
[PASS] test_ShouldReturnTrueIfBlocklisted() (gas: 46776)
[PASS] test_ShouldReturnTrueIfCheckingAddressZeroWhitelisted() (gas: 8772)
[PASS] test_ShouldReturnTrueIfNoAccountBlacklistedForApproveAllowed() (gas: 19753)
[PASS] test_ShouldReturnTrueIfNoAccountBlacklistedForTransferAllowed() (gas: 20398)
[PASS] test_ShouldReturnTrueIfWhitelisted() (gas: 41670)
[PASS] test_ShouldRevokeRole() (gas: 18239)
[PASS] test_ShouldSetBurnRate() (gas: 17563)
[PASS] test_ShouldWhitelist() (gas: 41735)

[PASS] test_failsToBlocklistAddressZero() (gas: 14124)
[PASS] test_failsToBlocklistIfCalledByNonLister() (gas: 45537)
[PASS] test_failsToBlocklistOwnerOrDistributorAccountOrListerAccount() (gas: 24723)
[PASS] test_failsToGrantRoleIfCalledByNonOwner() (gas: 18757)
[PASS] test_failsToRevokeRoleIfCalledByNonOwner() (gas: 18764)
[PASS] test_failsToRevokeRoleIfRevokingFromAccountWhichDoesNotHaveRole() (gas: 18729)
[PASS] test_failsToRevokeRoleIfRevokingFromAddressZero() (gas: 12076)
[PASS] test_failsToWhitelistAddressZero() (gas: 14124)
[PASS] test_failsToWhitelistIfCalledByNonLister() (gas: 45581)
Test result: ok. 39 passed; 0 failed; 0 skipped; finished in 8.74ms

Running 31 tests for test/MPROVesting.t.sol:MPROVestingTest
[PASS] test_FailsToClaimIfCalledByNonAuthorizedAccount() (gas: 143565)
[PASS] test_FailsToClaimIfTGENotUnlocked() (gas: 145602)
[PASS] test_FailsToEmergencyWithdrawIfNotOwner() (gas: 13525)
[PASS] test_FailsToInitializeWithInvalidParams() (gas: 130247)
[PASS] test_FailsToRegisterBeneficiariesIfAnyBeneficaryIsAddressZero() (gas: 166475)
[PASS] test_FailsToRegisterBeneficiariesIfCalledByNonOwner() (gas: 37369)
[PASS] test_FailsToRegisterBeneficiariesIfLengthMismatched() (gas: 60401)
[PASS] test_FailsToReleaseAmountIfCalledByNonAuthorizedAccount() (gas: 11510)
[PASS] test_FailsToReleaseTimestampIfCalledByNonAuthorizedAccount() (gas: 11444)
[PASS] test_FailsToSetTgeUnlockTimestampIfNotOwner() (gas: 13593)
[PASS] test_FailsToSetTgeUnlockTimestampIfTGEUnlockTimeHasPassed() (gas: 15923)
[PASS] test_FailsToSetTgeUnlockTimestampIfTimestampIsInPast() (gas: 11238)
[PASS]
test_FailsToSetTgeUnlockTimestampIfTimestampIsMoreThanTgeUnlockTimestampDeadline()
(gas: 11298)
[PASS] test_FailsToSetVestingTokenIfSetByNonOwnerAccount() (gas: 13551)
[PASS] test_FailsToSetVestingTokenIfTokenIsAddressZero() (gas: 11077)
[PASS] test_FailsToSetVestingTokenIfTokenIsAlreadySet() (gas: 15362)
[PASS] test_InitializeCorrectly() (gas: 31233)
[PASS] test_MaxReleaseShouldBeBeneficaryAmount() (gas: 154062)
[PASS] test_ShouldClaimSuccessfullyForUser1() (gas: 294857)
[PASS] test_ShouldClaimSuccessfullyForUser2() (gas: 294834)
[PASS] test_ShouldEmergencyWithdraw() (gas: 33658)
[PASS] test_ShouldRegisterBeneficaries() (gas: 151212)
[PASS] test_ShouldReturnCorrectReleaseAmountIfTimestampMoreThanCliffTimestamp() (gas:
166261)
[PASS]
test_ShouldReturnCorrectReleaseAmountIfTimestampMoreThanTGEUnlockTimestampButLes
sThanCliffTimestamp() (gas: 151795)

```
[PASS]
test_ShouldReturnCorrectReleaseTimestampWhenCurrentTimestampIsLessThanCliffTimesta
mp() (gas: 14956)
[PASS]
test_ShouldReturnCorrectReleaseTimestampWhenCurrentTimestampIsLessThanTGEUnlockTi
mestamp() (gas: 12759)
[PASS]
test_ShouldReturnCorrectReleaseTimestampWhenCurrentTimestampIsMoreThanCliffTimesta
mp() (gas: 22307)
[PASS]
test_ShouldReturnZcorrectNextReleaseAllaocationWhenCurrentTimestampIsGreaterThanTGE
UnlockTimestamp() (gas: 168620)
[PASS]
test_ShouldReturnZcorrectNextReleaseAllaocationWhenCurrentTimestampIsLessThanTGEUnl
ockTimestamp() (gas: 147233)
[PASS] test_ShouldReturnZeroReleaseAmountIfTimestampLessThanTGEUnlockTimestamp()
(gas: 148091)
[PASS] test_ShouldSetTgeUnlockTimestamp() (gas: 20471)
Test result: ok. 31 passed; 0 failed; 0 skipped; finished in 8.45ms

Ran 5 test suites: 169 tests passed, 0 failed, 0 skipped (169 total tests)
```

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
MPRO.sol	100.00% (32/32)	100.00% (8/8)	100.00% (11/11)	100.00% (25/25)	
MPROLight.sol	100.00% (26/26)	100.00% (4/4)	100.00% (9/9)	100.00% (20/20)	
MPROMasterDistributor.sol	100.00% (96/96)	100.00% (42/42)	100.00% (24/24)	100.00% (81/81)	
MPROMasterDistributorLight.sol	100.00% (34/34)	100.00% (18/18)	100.00% (12/12)	100.00% (28/28)	
MPROVesting.sol	100.00% (81/81)	97.22% (35/36)	100.00% (10/10)	100.00% (64/64)	
All Files	**100.00% (269/269)**	**99.07% (107/108)**	**100.00% (66/66)**	**100.00% (218/218)**	

We are grateful for the opportunity to work with the Metapro team.

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

Zokyo Security recommends the Metapro team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo