SMART CONTRACTS REVIEW

zokyo

December 2, 2025 | v1.0

Security Audit Score

PASS

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

ZOKYO AUDIT SCORING MAIV

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.

SCORING CALCULATION:

Let's assume each issue has a weight:
- Critical: -30 points
- High: -20 points
- Medium: -10 points
- Low: -5 points
- Informational: 0 points

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

Thus, the score is 100

TECHNICAL SUMMARY

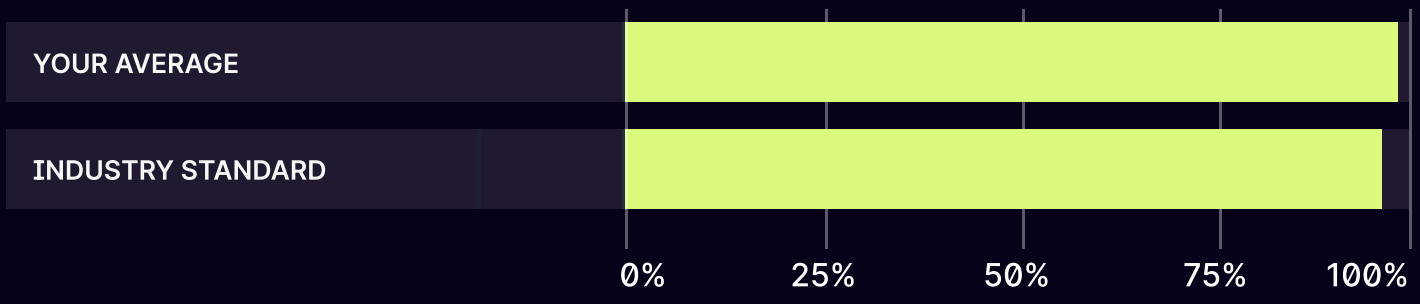
This document outlines the overall security of the MAIV smart contract/s evaluated by the Zokyo Security team.

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

Contract Status

There were 0 critical issues found during the review. (See Complete Analysis)

Testable Code



98,5% 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 MAIV 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 MAIV repository:
Repo: https://github.com/Baboons-dev/maiv-staking-contract

Last commit - `06a2fad6dfe07c5e32ad1d63470e1b90701ef395`

Contracts under the scope:

- MAIVStaking.sol
- Membership.so

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 MAIV 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.
02	Cross-comparison with other, similar smart contract/s by industry leaders.
03	Testing contract/s logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase line by line.

Executive Summary

MAIV (Multi Asset Investment Vehicle) is innovating in the crypto space through it's revolutionary platform that allows access to institutional-grade markets through integrating DeFI with traditional finance. It takes advantage of web3 technologies to create opportunities for wider users and allow them to participate in secure, high-yield investment opportunities that were once gate kept by hedge funds and institutional investors.

Zokyo was responsible for the security audit portion of the project, specifically the Membership and MAIVStaking contracts. The Membership contract represents investor access for the MAIV ecosystem. The Membership contract supports tiered rewards and allows the access to tokenized investment opportunities. Staking uses a Synthetix style rewards system (linear distribution over admin configured duration) and integrates the membership tier system within the contract.

Overall the code is well structured and well documented through in-line comments. The code itself is coherent and makes sense in order to achieve the goals of MAIV. The security review yielded findings which ranged from High severity issues down to Informational. These issues mainly revolved around rewards distribution and gas optimizations in order to improve contract efficiency. In addition to this a suite of unit tests was provided in order to test various edge cases and ensure that invariants are not broken during the updating of the contracts.

Zokyo recommends that the issues are carefully reviewed for which a fix review will be provided to ensure that the aforementioned issues have been resolved and that no further vulnerabilities have been introduced through the fixes. We at Zokyo with MAIV the best of luck deploying to a mainnet environment.

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 MAIV team and the MAIV 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	Rewards Are Calculated Based On New Effective Stake For Past Durations	High	Resolved
2	Membership expiry wipes past boosted rewards in MAIVStaking	High	Resolved
3	No Reasonable Upper Limit For Duration When Adding A New Period May Cause Temporary Stuck Funds	Low	Resolved
4	Using custom errors and if statements can improve gas efficiency of the contract	Informational	Resolved
5	Both contracts operate directly on storage variables	Informational	Resolved

Rewards Are Calculated Based On New Effective Stake For Past Durations

Description:
When a tier is upgraded for a user the earned rewards for the time period between previous update and current time is calculated with the inflated multiplier i.e. rewards are earned as if the inflated multiplier was applied over the entire time duration of the user's stake.
Consider the following scenario →

1. rewardDuration = 100 , reward = 100 therefore rewardRate = 1
2. In Membership contract User1 stakes for tier1 and period of 90 days → getMinimumStakeForTier() = 2e6 *1e18
3. User1 calls stakeFlex() with 10 as amount → flexStakes[] = 10 , totalFlexStaked = 10 , userEffectiveStake[] = 10 , totalEffectiveStaked = 10
4. After t = 20 , User1 upgrades to tier 3 duration = 90 days → syncTierMultiplierFor() → updateRewards() → rewardPerToken() = 20/10 = 2 = rewardPerTokenStored , earned() = getEffectiveStake() = 20 , rewardDelta = 2 , earned = 20*2 = 40 which is wrong since this means the user earned rewards as if he was staked in tier3 for the entire duration of t=20.

Impact:
Any user can upgrade to a higher tier during the end of his stake and he will earn rewards for the entire stake duration as if he was staked with the higher multiplier throughout.

Recommendation:
Consider updating the logic such that when a tier is upgraded then the rewards earned for the past duration is calculated with the old multiplier.

Membership expiry wipes past boosted rewards in MAIVStaking

Description:
MAIVStaking.earned() uses the current tier from Membership.getUserTier() via getEffectiveStake().
When a user's membership lock expires, Membership.getUserTier() returns tier `0`, so getEffectiveStake() drops to 1x. At that point, earned() recomputes the entire rewardPerToken delta using the downgraded effective stake, so all historical periods where the user actually had a higher tier (e.g. 1.5x or 2x) are paid out as if they were only 1x.

Meanwhile, rewardPerToken() was accumulated using the boosted totalEffectiveStaked (old multiplier), so the "missing" boosted rewards remain in the contract and are never claimable by the user.

Impact:
Users who stake at a boosted tier but only claim after membership expiry do not receive multiplied rewards (lose all historical boost if they haven't interacted since staking).

Recommendation:
Consider updating the logic such that earned rewards of a user are calculated based on the multiplier that existed within the duration.

LOW-1 | RESOLVED

No Reasonable Upper Limit For Duration When Adding A New Period May Cause Temporary Stuck Funds

Description

The addNewPeriod function takes a new periodId, the duration in days and discounted percent and adds a new locking period for the user stakes. There is no upper bound limit on the duration in days which can allow admins to set unnecessarily large locking periods.

Impact

Funds could theoretically be locked in the contract. This was rated a low in severity because an admin can remove the configuration and add a new one.

Recommendation:

It's recommended that upper bound validation is done to ensure that there are reasonable bounds on the durationInDays.

INFORMATIONAL-1 | RESOLVED

Using custom errors and if statements can improve gas efficiency of the contract.

Description:

Currently, the contract uses strings in conjunction with required statements in order to perform validation. Using custom errors instead of strings uses a fixed amount of gas but the longer the string the more gas is used when raising an error.

Recommendation:

Use custom errors in conjunction with if statements to further optimize the contract's gas.

Both contracts operate directly on storage variables

Description:
Reading from storage using sload with each iteration of an array should be cached whenever possible as it can become quite costly when read multiple times. When a storage variable is accessed for the first time (cold access), this can cost 2100 gas units. Each time after that warm access will cost an additional 100 gas units thereafter.

Recommendation:
It's recommended that storage variables are cached in memory variables (3 gas units for each read), operated on, then updated once all operations are complete.

	MAIVStaking.sol Membership.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 MAIV 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 MAIV contract/s requirements for details about issuance amounts and how the system handles these.

Ran 1 test for .audit/test/MAIVTierFindingPoC.t.sol:MAIVTierFindingPoC
[PASS] test_ProofTierCorruption() (gas: 498889)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 3.41ms (2.03ms CPU time)

Ran 54 tests for .audit/test/MembershipTest.t.sol:MembershipStakeTest
[PASS] testAddNewPeriod_RevertsWhenDiscountInvalid() (gas: 22257)
[PASS] testAddNewPeriod_RevertsWhenDurationZero() (gas: 14738)
[PASS] testAddNewPeriod_RevertsWhenPeriodExists() (gas: 14673)
[PASS] testAddNewPeriod_Success() (gas: 63222)
[PASS] testConstructor_RevertWhenZeroMaiv() (gas: 92730)
[PASS] testDowngrade_RevertsWhenInvalidTier() (gas: 219154)
[PASS] testDowngrade_RevertsWhenLockPeriodNotEnded() (gas: 221280)
[PASS] testDowngrade_RevertsWhenNoActiveStake() (gas: 32268)
[PASS] testDowngrade_RevertsWhenNotADowngrade() (gas: 222251)
[PASS] testDowngrade_Success() (gas: 249149)
[PASS] testGetInvestmentLimit_ReturnsCorrectLimit() (gas: 16675)
[PASS] testGetInvestmentLimit_RevertsOnInvalidTier() (gas: 12871)
[PASS] testGetMinimumStakeForTier_ReturnsCorrectAmount() (gas: 26785)
[PASS] testGetMinimumStakeForTier_RevertsOnInvalidTier() (gas: 18371)
[PASS] testGetPeriodInfo_ReturnsCorrectData() (gas: 11680)
[PASS] testGetTierRequirements_ReturnsCorrectValues() (gas: 13115)
[PASS] testGetUserStake_ReturnsCorrectData() (gas: 216328)
[PASS] testGetUserTier_Scenarios() (gas: 221120)
[PASS] testIsPeriodActive_ReturnsFalse() (gas: 8720)
[PASS] testIsPeriodActive_ReturnsTrue() (gas: 8764)
[PASS] testRemovePeriod_RevertsWhenPeriodNotFound() (gas: 13850)
[PASS] testRemovePeriod_Success() (gas: 19407)
[PASS] testSetMaivStakingContract_Success() (gas: 33397)
[PASS] testStake_RevertTokenWithFee() (gas: 4920854)
[PASS] testStake_RevertsWhenActiveStakeExists() (gas: 217695)
[PASS] testStake_RevertsWhenInsufficientBalance() (gas: 71645)

[PASS] testStake_RevertsWhenInvalidPeriod() (gas: 53284)
[PASS] testStake_RevertsWhenPaused() (gas: 225374)
[PASS] testStake_RevertsWhenTierTooHigh() (gas: 51021)
[PASS] testStake_RevertsWhenTierTooLow() (gas: 51041)
[PASS] testStake_Success() (gas: 3396159)
[PASS] testUpdateInvestmentLimit_RevertsOnInvalidTier() (gas: 16925)
[PASS] testUpdateInvestmentLimit_Success() (gas: 22168)
[PASS] testUpdateTierRequirement_RevertsOnInvalidTier() (gas: 16971)
[PASS] testUpdateTierRequirement_RevertsOnZeroAmount() (gas: 12198)
[PASS] testUpdateTierRequirement_RevertsWhenNotOwner() (gas: 14752)
[PASS] testUpdateTierRequirement_Success() (gas: 26493)
[PASS] testUpdateWithdrawalGracePeriod() (gas: 24756)
[PASS] testUpgrade_RevertsWhenInsufficientBalance() (gas: 226682)
[PASS] testUpgrade_RevertsWhenInvalidPeriod() (gas: 27888)
[PASS] testUpgrade_RevertsWhenInvalidTier() (gas: 23501)
[PASS] testUpgrade_RevertsWhenNoActiveStake() (gas: 39822)
[PASS] testUpgrade_RevertsWhenNotAnUpgrade() (gas: 219858)
[PASS] testUpgrade_Success() (gas: 269308)
[PASS] testUpgrade_WhileUpdatingBaseRequirements() (gas: 264637)
[PASS] testWithdraw_RevertsWhenAlreadyWithdrawn() (gas: 184559)
[PASS] testWithdraw_RevertsWhenLockPeriodNotEnded() (gas: 225550)
[PASS] testWithdraw_RevertsWhenNoActiveStake() (gas: 18800)
[PASS] testWithdraw_Success() (gas: 196688)
[PASS] test_SweepERC20_RevertsInvalidRecipient() (gas: 14468)
[PASS] test_SweepERC20_RevertsWhenSweepingMAIV() (gas: 199673)
[PASS] test_SweepERC20_RevertsZeroAmount() (gas: 16669)
[PASS] test_SweepERC20_SuccessForExcessMAIV() (gas: 233026)
[PASS] test_SweepERC20_SuccessForOtherToken() (gas: 63840)

Suite result: ok. 54 passed; 0 failed; 0 skipped; finished in 14.19ms (67.91ms CPU time)

Ran 44 tests for .audit/test/MAIVStakingTest.t.sol:MAIVStakingTest
[PASS] testConstructor_RevertWhenZeroMaivToken() (gas: 276473)
[PASS] testStakeFlex_UnPaused() (gas: 210465)
[PASS] testTopUpRewards_BeforePeriodFinish() (gas: 320227)
[PASS] testTopUpRewards_RevertForVeryLowRewards() (gas: 73437)
[PASS] testTopUpRewards_RevertForZeroRewards() (gas: 43208)
[PASS] test_ClaimRewards_WithNoRewards() (gas: 218961)
[PASS] test_ClaimRewards_WithRewards() (gas: 370458)
[PASS] test_CurrentAPR_ReturnsZero() (gas: 287683)

[PASS] testConstructor_RevertWhenZeroMaivToken() (gas: 276473)
[PASS] testStakeFlex_UnPaused() (gas: 210465)
[PASS] testTopUpRewards_BeforePeriodFinish() (gas: 320227)
[PASS] testTopUpRewards_RevertForVeryLowRewards() (gas: 73437)
[PASS] testTopUpRewards_RevertForZeroRewards() (gas: 43208)
[PASS] test_ClaimRewards_WithNoRewards() (gas: 218961)
[PASS] test_ClaimRewards_WithRewards() (gas: 370458)
[PASS] test_CurrentAPR_ReturnsZero() (gas: 287683)
[PASS] test_CurrentAPR_WithActiveStaking() (gas: 287940)
[PASS] test_Earned() (gas: 293351)
[PASS] test_Exit() (gas: 384796)
[PASS] test_GetEffectiveStakeForUser() (gas: 209537)
[PASS] test_GetUserInfo() (gas: 322316)
[PASS] test_GetUserTier() (gas: 42982)
[PASS] test_GetUserTier_ReturnsZer0() (gas: 49269)
[PASS] test_LastTimeRewardApplicable_AfterPeriodFinish() (gas: 150060)
[PASS] test_LastTimeRewardApplicable_BeforePeriodFinish() (gas: 149338)
[PASS] test_RewardForDuration_AfterPeriodFinish() (gas: 150324)
[PASS] test_RewardForDuration_BeforePeriodFinish() (gas: 151691)
[PASS] test_RewardPerToken_WithStaking() (gas: 289423)
[PASS] test_RewardPerToken_ZeroStaked() (gas: 12688)
[PASS] test_SetMembershipContract() (gas: 193033)
[PASS] test_SetRewardDuration_RevertsWhenDurationInvalid() (gas: 33390)
[PASS] test_SetRewardDuration_RevertsWhenPeriodNotFinished() (gas: 159666)
[PASS] test_SetRewardDuration_Success() (gas: 34270)
[PASS] test_SetTierMultiplier_RevertsWhenMultiplierOutOfBounds() (gas: 37876)
[PASS] test_SetTierMultiplier_RevertsWhenTierInvalid() (gas: 24720)
[PASS] test_SetTierMultiplier_Success() (gas: 74002)
[PASS] test_SetTierMultipliersEnabled() (gas: 35993)
[PASS] test_StakeFlex_RevertTokenWithFee() (gas: 4449277)
[PASS] test_StakeFlex_RevertsWhenAmountIsZero() (gas: 77610)
[PASS] test_StakeFlex_RevertsWhenPaused() (gas: 68089)
[PASS] test_StakeFlex_Success() (gas: 222998)
[PASS] test_SweepERC20_RevertsInvalidRecipient() (gas: 17175)
[PASS] test_SweepERC20_RevertsWhenSweepingMAIV() (gas: 291702)
[PASS] test_SweepERC20_RevertsZeroAmount() (gas: 19376)
[PASS] test_SweepERC20_SuccessForExcessMAIV() (gas: 316100)
[PASS] test_SweepERC20_SuccessForOtherToken() (gas: 67692)
[PASS] test_SyncTierMultiplierFor_NoStake() (gas: 44157)
[PASS] test_SyncTierMultiplierFor_TierDowngrade() (gas: 228827)
[PASS] test_SyncTierMultiplierFor_TierUpgrade() (gas: 245886)
[PASS] test_WithdrawFlex_RevertsWhenAmountIsZero() (gas: 206762)

```
[PASS] test_WithdrawFlex_RevertsWhenInsufficientStake() (gas: 210791)
[PASS] test_WithdrawFlex_Success() (gas: 256731)
Suite result: ok. 44 passed; 0 failed; 0 skipped; finished in 14.20ms (78.98ms CPU
time)
```

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

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	%UNCOVERED LINES
MAIVStaking.sol	98.24% (167/170)	92.16% (47/51)	96.43% (27/28)	96.95% (159/164)	3.05%
Membership.sol	98.86% (173/175	94.05% (79/84)	96.43% (27/28)	98.33% (177/180)	1.67%
All Files	**98.5% (340/345)**	**93.3% (126/135)**	**96.43%**	**97.7% (336/344)**	

We are grateful for the opportunity to work with the MAIV 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 MAIV team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo