QuillAudits

AUDIT REPORT

April, 2025

For

TOKEN METRICS

Table of Content

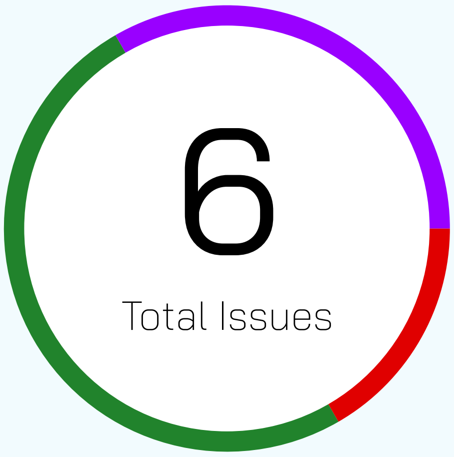
Executive Summary

Project Name	TMAI
Overview	Token Metrics AI is a data-driven cryptocurrency platform offering advanced analytics, ratings, and AI-powered tools for traders and investors. It introduces the $TMAI token to enhance platform governance, offer discounts, and unlock gamified rewards, ensuring both technical and economic engagement within its ecosystem.
Method	Manual Review, Functional Testing, Automated Testing, etc. All the raised flags were manually reviewed and re-tested to identify any false positives.
Audit Scope	The scope of this audit was to analyze the contracts in scope for quality, security, and correctness.
Source Code	https://github.com/token-metrics/tmai-solidity-contracts Commit Log: Efd1a6cf8feaab9ecc35064c0b9ec64895a0c9a8 224779d1804451d990785e615409a925ae4e6fcb be41cb3b2dbc5ffeb0b6768ef9c14b828639941b 3ebb11c478ca75da6431024a9c2979387e2389fb
Branch	Master

Executive Summary

Contracts In-Scope	src/core/token/veTMAI.sol src/core/governance/GovernorAlpha.sol src/periphery/launchpad/PublicLaunchpad.sol src/periphery/launchpad/PrivateLaunchpad.sol src/periphery/launchpad/Whitelist.sol src/periphery/launchpad/crossChainSaleManager.sol src/periphery/launchpad/launchpadConfiguration.sol src/periphery/launchpad/launchpadFactory.sol src/periphery/launchpad/launchpadVesting.sol src/periphery/astrobot-bridge/AstrobotBridgeBase.sol src/periphery/astrobot-bridge/AstrobotBridgeEthereum.sol src/periphery/astrobot-bridge/AstrobotPlanMapping.sol
Timeline	31 Jan 25 - 07 Feb 2025
Blockchain	EVM
Fixed In	https://github.com/token-metrics/tmai-solidity-contracts/pull/8

Number of Issues per Severity



🟥	High	1 (16.67%)
🟧	Medium	0 (0.00%)
🟩	Low	3 (50.00%)
🟪	Informational	2 (33.33%)

Severity

Issues

	🟥 High	🟧 Medium	🟩 Low	🟪 Informational
Open	0	0	0	0
Resolved	0	0	**1**	**1**
Acknowledged	**1**	0	**2**	**1**
Partially Resolved	0	0	0	0

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Computations Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

- ✓ Using inline assembly
- ✓ Unsafe type inference
- ✓ Style guide violation
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of Solana Programs, care was taken to ensure:

- The overall quality of code.
- Use of best practices
- Code documentation and comments, match logic and expected behavior
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper
- Efficient use of gas
- Code is safe from re-entrancy and other vulnerabilities

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of Solana programs. A thorough check was done to ensure the Solana program is structured in a way that will not result in future problems.

Static Analysis

Static analysis of Solana programs was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of Solana programs.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, and their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of Solana programs in production. Checks were done to know how much gas gets consumed and the possibilities of optimising code to reduce gas consumption.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below

🟥 High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

🟧 Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

🟩 Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

🟪 Informational Issues

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open Security vulnerabilities identified that must be resolved and are currently unresolved.	**Resolved** Security vulnerabilities identified that must be resolved and are currently unresolved.
Acknowledged Vulnerabilities which have been acknowledged but are yet to be resolved.	**Partially Resolved** Considerable efforts have been invested to reduce the risk/ impact of the security issue, but are not completely resolved.

High Severity Issues

NFT's cannot be recovered if revert happens in destination chain

Acknowledged

Path
AstrobridgeEthereum.sol

Function Name
bridgeNFT

Description
The contract first burns user's NFT followed by calling Axelar's gateway contract to initiate execution on the destination chain where lockFor is used.
The risk with this setup is revert on destination chain.

If a revert happens on destination chain for any reason, there is no way for a user to recover their NFT.

Recommendation
Implement a 2 step process where instead of directly burning, send NFT to an admin controlled address. They can be recovered by the admin in case of a mishappening.

TMAI team's Comment
Although this should not happen, but if it does, we will manually stake the tokens for user.

Low Severity Issues

Lack of Fund Transfer Verification During Launchpad Creation plus Unrestricted Token Withdrawal

Acknowledged

Path

src/periphery/launchpads/LaunchpadFactory.sol
src/periphery/launchpads/PrivateLaunchpad.sol
src/periphery/launchpads/PublicLaunchpad.sol

Description

The LaunchpadFactory contract allows whitelisted users to request a new launchpad, and once approved by the factory owner, a new launchpad instance is created. However, there is no mechanism to ensure that the amount specified in the request is actually transferred to the newly created launchpad. Instead, the current architecture assumes that the launchpad creator will manually top up the required amount after deployment. This introduces a major risk where a launchpad could be created with an expected token allocation, but the actual funds never arrive, leading to misleading or fraudulent launches.

Additionally, in PrivateLaunchpad, there are two withdrawal functions: one for the base token (used for buying) and another for the sale token. The function withdrawBaseTokens correctly ensures that the sale must have ended before funds can be withdrawn. However, the withdrawTokens function, which allows withdrawal of the sale token, does not enforce this check. This oversight allows the launchpad owner to withdraw unsold sale tokens before the sale ends, potentially leading to early liquidation or manipulation.

Impact

- The absence of fund transfer verification during launchpad creation means that a launchpad can be approved and advertised with a predefined token supply, yet the actual tokens might never be sent. This creates an opportunity for malicious launchpad creators to mislead buyers or run deceptive campaigns.
- By using withdrawal, a dishonest launchpad owner could drain unsold tokens before the sale concludes, affecting investor confidence and market stability.

Code Effected

```
function withdrawTokens(
    address _tokenAddress,
    uint256 _amount
) external onlyOwner {
    require(_tokenAddress != baseToken, "Cannot withdraw base token");
    IERC20(_tokenAddress).safeTransfer(msg.sender, _amount);
    emit TokensWithdrawn(_tokenAddress, msg.sender, _amount);
}
```

Recommendation

- The factory contract should explicitly verify that the specified sale token amount is transferred to the newly created launchpad during deployment even if the vesting is not enabled.
- The withdrawTokens function should include a check to ensure that the sale has ended before allowing withdrawals of the sale token.

Use Ownable2Step instead

Acknowledged

Path

-

Function

-

Description

The contract uses the single-step Ownable pattern for ownership management, which transfers ownership immediately upon calling transferOwnership(). This creates a risk that ownership could be accidentally transferred to an incorrect or inaccessible address, permanently locking admin functions.

Recommendation

Replace the current Ownable implementation with OpenZeppelin's Ownable2Step, which implements a two-step ownership transfer.

Data poisoning due to redundant state updates

Resolved

Target

src/periphery/launchpads/LaunchpadFactory.sol

Description

The addLaunchpad function, which is restricted to the contract owner via the onlyOwner modifier, is designed to create a new launchpad and record its details. However, upon reviewing the code, it is evident that certain operations within this function are redundantly executed, causing unnecessary state updates. Specifically, the following redundant operations have been identified:

1. Duplicate Insertion into the launchpads Array:
   The newly created launchpad address is pushed into the launchpads array within both the addLaunchpad function and the _createLaunchpad function.
2. Redundant Mapping Update for tokenToLaunchpad:
   The mapping tokenToLaunchpad is updated to link the token address to the newly created launchpad address in both functions. Since the mapping update has already occurred in _createLaunchpad.

Impact

- These redundant state updates not only lead to inefficient use of gas but also increase the complexity of the contract logic without providing any additional functionality.
- The launchpads array will contain duplicate entries, making data management more cumbersome for off-chain applications or when iterating over the array on-chain.

Code Effected

```
launchpads.push(launchpadAddress); // Redundant as it's already done in _createLaunchpad
tokenToLaunchpad[_tokenAddress] = launchpadAddress; // Redundant operation as it's already
done in _createLaunchpad
```

Recommendation

The redundant operations should be removed from the addLaunchpad function. Since _createLaunchpad already handles the insertion of the launchpad address into the launchpads array and the mapping update for tokenToLaunchpad, these lines can be safely deleted from addLaunchpad.

Lack of Input Validation and Limit Checks for Critical Parameters Can Lead to Misconfigurations

Resolved

Path

src/periphery/launchpads/PublicLaunchpad.sol
src/periphery/launchpads/PrivateLaunchpad.sol
src/periphery/launchpads/LaunchpadConfiguration.sol
src/periphery/launchpads/LaunchpadFactory.sol

Description

Across multiple smart contracts in the launchpad system, critical functions responsible for configuring sensitive parameters such as fees, base amounts, minimum investment limits, and contract addresses lack proper input validation. This absence of validation creates significant risks, as incorrect or malicious configurations could severely disrupt the platform's functionality, potentially leading to financial loss or system instability.

Functions like setSetupFee, setCompletionFee, updateBaseAmount, updateMinAmount, and others allow the contract owner to modify values without enforcing any upper or lower bounds. For example, the setSetupFee and setCompletionFee functions do not cap the fee percentage, theoretically allowing these fees to be set as high as 100% (or even higher if not properly constrained). This could result in investors losing their entire contributions to fees, effectively nullifying the purpose of the sale.

```
function setSetupFee(uint256 _SETUP_FEE) external onlyOwner {
    SETUP_FEE = _SETUP_FEE; // ❌ No upper limit - could be set to 100% or more
    emit SetupFeeUpdated(SETUP_FEE);
}

function setCompletionFee(uint256 _COMPLETION_FEE) external onlyOwner {
    COMPLETION_FEE = _COMPLETION_FEE; // ❌ No cap, enabling exploitative fee settings
    emit CompletionFeeUpdated(COMPLETION_FEE);
}

function updateBaseAmount(uint256 _baseAmount) external onlyOwner {
    baseAmount = _baseAmount; // ❌ No lower or upper bound - could be set to unrealistic
values
    emit BaseAmountUpdated(_baseAmount);
}

function updateMinAmount(uint256 _minAmount) external onlyOwner {
    minAmount = _minAmount; // ❌ No check for reasonable minimums, risking DoS or unfair

participation
    emit MinimumAmountUpdated(_minAmount);
}

function setTreasuryAddress(address _TREASURY_ADDRESS) external onlyOwner {
    TREASURY_ADDRESS = _TREASURY_ADDRESS; // ❌ No validation - could be set to address(0)
    emit TreasuryAddressUpdated(_TREASURY_ADDRESS);
}
```

Recommendation

To mitigate these risks, input validation must be implemented consistently across all configuration functions.

Redundant safeTransferfrom

Acknowledged

Description

Using safeTransferFrom to transfer to a burn address is unnecessary since it's not a contract that needs to implement receiver hooks.

Recommendation

Use normal transfer

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the TMAI Contracts. We performed our audit according to the procedure described above.

6 issues were found, TMAI team resolved few and acknowledged others.

Disclaimer

At QuillAudits, we have spent years helping projects strengthen their smart contract security. However, security is not a one-time event—threats evolve, and so do attack vectors. Our audit provides a security assessment based on the best industry practices at the time of review, identifying known vulnerabilities in the received smart contract source code.

This report does not serve as a security guarantee, investment advice, or an endorsement of any platform. It reflects our findings based on the provided code at the time of analysis and may no longer be relevant after any modifications. The presence of an audit does not imply that the contract is free of vulnerabilities or fully secure.

While we have conducted a thorough review, security is an ongoing process. We strongly recommend multiple independent audits, continuous monitoring, and a public bug bounty program to enhance resilience against emerging threats.

Stay proactive. Stay secure.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem

QuillAudits

7+ Years of Expertise	**1M+** Lines of Code Audited
$30B+ Secured in Digital Assets	**1400+** Projects Secured

Follow Our Journey

AUDIT REPORT

April, 2025

For

TOKEN METRICS

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com audits@quillaudits.com