



AUDIT REPORT




August 2025

For



TOKEN
METRICS

Table of Content

Executive Summary	03
Number of Security Issues per Severity	04
Summary of Issues	05
Checked Vulnerabilities	06
Techniques and Methods	08
Types of Severity	09
Types of Issues	11
Severity Matrix	12
 High Severity Issues	13
1. An attacker can bypass contract pause via transferWithAuthorization	13
 Low Severity Issues	14
2. Recommended to Use disableInitializers	14
 Informational Issues	15
3. Avoid using Floating Pragma	15
Functional Tests	16
Automated Tests	16
Closing Summary & Disclaimer	17



Executive Summary

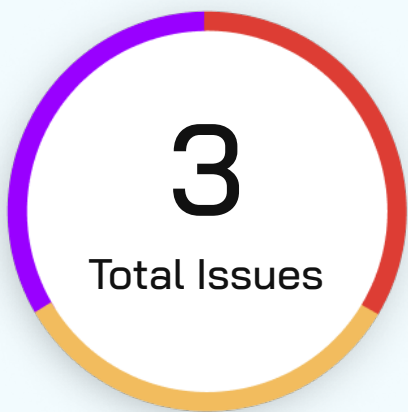
Project Name	TMAI
Project URL	https://www.tokenmetrics.com/tmai
Overview	Upgradeable contract with EIP-3009 support.
Audit Scope	The scope of this Audit was to analyze the TMAIV2Smart Contracts for quality, security, and correctness.
Source Code link	https://github.com/token-metrics/tmai-solidity-contracts/tree/feature/EIP-3009-support
Branch	EIP-3009-support
Contracts in Scope	TMAIV2.sol
Commit Hash	e8cd03a04bb4dca5b12ac96c13e2c6e7dbdd4724
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	5th August 2025 - 11th August 2025
Updated Code Received	13th August 2025
Review 2	13th August 2025 - 14th August 2025
Fixed In	https://github.com/token-metrics/tmai-solidity-contracts/blob/feature/EIP-3009-support/src/core/token/TMAIV2.sol

Verify the Authenticity of Report on QuillAudits Leaderboard:

<https://www.quillaudits.com/leaderboard>



Number of Issues per Severity



Critical	0 (0%)
High	1 (33.3%)
Medium	0 (0%)
Low	1 (33.3%)
Informational	1 (33.3%)

		Severity				
		Critical	High	Medium	Low	Informational
Issues	Open	0	0	0	0	0
	Acknowledged	0	0	0	0	0
	Partially Resolved	0	0	0	0	0
	Resolved	0	1	0	1	1



Summary of Issues

Issue No.	Issue Title	Severity	Status
1	An attacker can bypass contract pause via transferWithAuthorization	High	Resolved
2	Recommended to Use disableInitializers	Low	Resolved
3	Avoid using Floating Pragma	Informational	Resolved



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

✓ Style guide violation

✓ Unsafe type inference

✓ Implicit visibility level



Techniques and Methods

Throughout the audit of smart contracts, 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
- Implementation of ERC standards
- 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 analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.



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 analyzed, 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 behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms Used for Audit

Remix IDE, Foundry, Solhint, Mythril, Slither, Solidity Static Analysis.



Types of Severity

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

■ **Critical: Immediate and Catastrophic Impact**

Critical issues are the ones that an attacker could exploit with relative ease, potentially leading to an immediate and complete loss of user funds, a total takeover of the protocol's functionality, or other catastrophic failures. Critical vulnerabilities are non-negotiable; they absolutely must be fixed.

■ **High (H): Significant Risk of Major Loss or Compromise**

High-severity issues represent serious weaknesses that could result in significant financial losses for users, major malfunctions within the protocol, or substantial compromise of its intended operations. While exploiting these vulnerabilities might require specific conditions to be met or a moderate level of technical skill, the potential damage is considerable. These findings are critical and should be addressed and resolved thoroughly before the contract is put into the Mainnet.

■ **Medium (M): Potential for Moderate Harm Under Specific Circumstances**

Medium-severity bugs are loopholes in the protocol that could lead to moderate financial losses or partial disruptions of the protocol's intended behavior. However, exploiting these vulnerabilities typically requires more specific and less common conditions to occur, and the overall impact is generally lower compared to high or critical issues. While not as immediately threatening, it's still highly recommended to address these findings to enhance the contract's robustness and prevent potential problems down the line.

■ **Low (L): Minor Imperfections with Limited Repercussions**

Low-severity issues are essentially minor imperfections in the smart contract that have a limited impact on user funds or the core functionality of the protocol. Exploiting these would usually require very specific and unlikely scenarios and would yield minimal gain for an attacker. While these findings don't pose an immediate threat, addressing them when feasible can contribute to a more polished and well-maintained codebase.

■ **Informational (I): Opportunities for Improvement, Not Immediate Risks**

Informational findings aren't security vulnerabilities in the traditional sense. Instead, they highlight areas related to the clarity and efficiency of the code, gas optimization, the quality of documentation, or adherence to best development practices. These findings don't represent any immediate risk to the security or functionality of the contract but offer valuable insights for improving its overall quality and maintainability. Addressing these is optional but often beneficial for long-term health and clarity.



Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

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.

Severity Matrix

		Impact		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Impact

- **High** - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- **Low** - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

Likelihood

- **High** - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- **Medium** - only a conditionally incentivized attack vector, but still relatively likely.
- **Low** - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.



High Severity Issues

An attacker can bypass contract pause via `transferWithAuthorization`

Resolved

Path

TMAIV2.sol

Function

`transferWithAuthorization`, `receiveWithAuthorization`, `cancelAuthorization`

Description

The contract implements `PausableUpgradeable` but the EIP-3009 meta-transaction functions (`transferWithAuthorization`, `receiveWithAuthorization`, and `cancelAuthorization`) do not check if the contract is paused.

This means that even when the contract is paused for emergency reasons, users can still transfer tokens through meta-transactions, bypassing the pause mechanism.

Impact

In case of contract pause, transfer of tokens is still possible

Likelihood

High likelihood as the function can be called by anyone.

Recommendation

Add the `whenNotPaused` modifier to all EIP-3009 functions



Low Severity Issues

Recommended to Use disableInitializers

Resolved

Path

TMAIV2.sol

Description

The contract does not call `_disableInitializers()` in its constructor. This is a recommended practice for upgradeable contracts to prevent the implementation contract (not the proxy) from being initialized directly.

While this doesn't pose an immediate security risk since the implementation contract being initialized doesn't affect the proxy's state, it's considered a best practice for defense in depth.

Recommendation

Consider using `_disableInitializer` in the constructor.



Informational Issues

Avoid using Floating Pragma

Resolved**Path**

TMAIV2.sol

Description

The contract uses a floating pragma `pragma solidity ^0.8.20`; which allows compilation with any version from 0.8.20 up to 0.9.0. This can lead to different behavior if compiled with different compiler versions and makes it harder to ensure consistent behavior across deployments.

Recommendation

Consider using fixed pragma



Functional Tests

Some of the tests performed are mentioned below:

- ✓ `should_initialize_with_correct_token_name_and_symbol`
- ✓ `should_transfer_tokens_between_valid_accounts`
- ✓ `should_execute_transferWithAuthorization_with_valid_signature`
- ✓ `should_mark_nonce_as_used_after_cancellation`

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 TMAI. We performed our audit according to the procedure described above.

Issues of High, Low and Informational severity were found. The TMAI team resolved all the issues mentioned.

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 leading name in Web3 security, offering top-notch solutions to safeguard projects across DeFi, GameFi, NFT gaming, and all blockchain layers.

With seven years of expertise, we've secured over 1400 projects globally, averting over \$3 billion in losses. Our specialists rigorously audit smart contracts and ensure DApp safety on major platforms like Ethereum, BSC, Arbitrum, Algorand, Tron, Polygon, Polkadot, Fantom, NEAR, Solana, and others, guaranteeing your project's security with cutting-edge practices.

**7+**

Years of Expertise

1M+

Lines of Code Audited

50+

Chains Supported

1400+

Projects Secured

Follow Our Journey



AUDIT REPORT

August 2025

For



TOKEN
METRICS



Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillaudits.com