PRIVATE

Code Assessment

of the MUSD

Smart Contracts

August 26, 2025

Produced for

M^ZERO LABS_

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help M^ZERO Labs with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of MUSD according to Scope to support you in forming an opinion on their security risks.

M^ZERO Labs implements MUSD, an extension of `MYieldToOne` with pausable and forced transfers capabilities.

The most critical subjects covered in our audit are asset solvency, functional correctness, and precision of arithmetic operations. Security regarding all the aforementioned subjects is high.

The general subjects covered are documentation, gas efficiency, and the integration of the wrapper into the existing system. Security regarding all the aforementioned subjects is generally good.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the MUSD repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	13 Aug 2025	b62fab7c3e867b700bd81dad2ab140e074d98f32	Initial Version
2	25 Aug 2025	8230a530019b0c63c2deda8b656d9a8b0b259291	Version after fixes

For the solidity smart contracts, the compiler version `0.8.26` was chosen.

The following contracts are in the scope of the review:

```
src:
    MUSD.sol
```

2.1.1 Excluded from scope

Any contracts not explicitly listed above are out of the scope of this review. Third-party libraries are out of the scope of this review. More specifically, `openzeppelin-contracts-upgradeable` is expected to work as intended and is out of the scope of this review. The code related to the base contract (`MYieldToOne`) is out of the scope of this review but a dedicated review can be found at https://www.chainsecurity.com/security-audit/m-zero-m-extensions-smart-contracts.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

M^ZERO Labs offers MUSD, an extension of `MYieldToOne` (see https://www.chainsecurity.com/security-audit/m-zero-m-extensions-smart-contracts) with pausable and forced transfers capabilities.

2.2.1 MUSD

MUSD is an extension of the `MYieldToOne` that is an `MExtension` where all the yield goes to a single, configurable `yieldRecipient` address. It also features a freeze-list, restricting token movements among ordinary users.

A privileged `yieldRecipientManager` address can set the `yieldRecipient` at will; another privileged `freezeManager` address can add and remove arbitrary addresses to/from the freeze-list. When a new `yieldRecipient` recipient is set, the yield is automatically claimed for the old recipient.
The `claimYield()` function computes the "pending" yield as the difference between the contract's own `$M` balance and the `totalSupply()` of wrapped tokens: this amount of *wrapped* tokens is then minted to the `yieldRecipient`.
All four hooks are implemented to enforce the freeze-list on every user action (`wrap()`, `unwrap()`, `approve()`, `transfer()`, `transferFrom()`, `permit()`, `transferWithAuthorization()`).
This extension can seamlessly adapt to the case where the `$M` governance revokes its earning status; no particular action is required to update its internal accounting.
On top of the functionalities of the `MYieldToOne` contract, `MUSD` implements the following:

- pausability: the `PAUSER_ROLE` can pause and unpause the contract. When paused, the `MUSD` token cannot be transferred, wrapped, unwrapped, and no allowance modification is possible.
- force transfer: the `FORCED_TRANSFER_MANAGER_ROLE` can access the funds of a frozen address and transfer them to arbitrary addresses. Note that forced transfers can be done even if the contract is paused, and allows the recipient to be a frozen address as well.
- yield can be claimed only by the `YIELD_RECIPIENT_MANAGER_ROLE`.

2.2.2 Changes in V2

- Allowance modifications can be done even when the contract is paused.

2.3 Trust Model

- Users: not trusted.
- Bearers of `DEFAULT_ADMIN_ROLE` in general: fully trusted. They are trusted to manage the internal roles of the contracts in the best interest of the users and in a non-adversarial manner. If malicious or compromised, they can, for example, distribute critical roles to lock (DOS) users' funds and collect their yield.
- Bearers of the `FREEZE_MANAGER_ROLE`: fully trusted. They are trusted to manage the freeze-list in the best interest of the users and the entity collecting the yield, and in a non-adversarial manner. If malicious or compromised, they can lock (DOS) user's funds or freeze the `feeRecipient`.
- Bearers of the `PAUSER_ROLE`: fully trusted. They are trusted to manage the pause status of the contract in the best interest of the users and the entity collecting the yield, and in a non-adversarial manner. If malicious or compromised, they can wrongfully DOS the contract.
- Bearers of the `FORCED_TRANSFER_MANAGER_ROLE`: fully trusted. They are trusted to manage frozen funds in the best interest of the system. If malicious or compromised, they can redirect frozen tokens to an address they control.
- Bearers of the `YIELD_RECIPIENT_MANAGER_ROLE`: semi-trusted. They are trusted to manage the yield recipient address in the best interest of the entity collecting the yield. If malicious or compromised, they can redirect the yield to an address they control but cannot DOS users.

2.3.1 Other assumptions

- `FREEZE_MANAGER_ROLE` and `FORCED_TRANSFER_MANAGER_ROLE` will not collude to steal users' funds
- The address of the `SwapFacility` will never be frozen. This would block all the swapping actions involving the `MUSD`.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

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

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe our findings. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

6.1 Force Transfer of Tainted Funds

Informational Version 1 Acknowledged

CS-MUSD-001

If some frozen funds have been tainted, for instance after a hack, and then frozen, seizing them with a forced transfer might also taint the recipient.

Acknowledged:

M^ZERO Labs has acknowledged the issue and stated:

```
This is a legal issue that will have to be addressed if such a situation arises.
```

6.2 Gas Optimizations

Informational Version 1 Acknowledged

CS-MUSD-002

1. The checks `_requireNotPaused()` in the function `MUSD._beforeUnwrap()` is unnecessary as the tokens will have to be transferred to the `SwapFacility` first. If the contract is paused, the transaction will revert during the transfer and the flow will never reach the unwrap logic.

Acknowledged:

M^ZERO Labs has acknowledged the issue and stated:

```
We will keep the check in case the current unwrapping mechanism
via the SwapFacility changes in the future and unwrapping becomes possible at
the extension level.
```

7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

7.1 Contract Can Be Temporarily Insolvent

Note Version 1

Due to the rebasing nature of the `MToken`, the wrapper contract may be temporarily insolvent. In practice, this should not impact security or UX as the rounding error should balance itself over time with users wrapping and unwrapping, and the yield accumulated should be able to cover for this small difference. However, it is important to note that this is a potential risk that should be monitored.

7.2 Freezelist Front-running

Note Version 1

If the transaction freezing a user is published in a public mempool, the user might be able to avoid being frozen by front-running the transaction, transferring their funds to a new address.

Developers should be aware of this possibility when implementing block lists for compliance purposes.