Matrixdock Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Hans

April 9, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Matrixdock is a Real World Asset (RWA) Tokenization protocol which has tokenized US Treasuries as STBT and physical gold as XAUm.

Currently Matrixdock uses Chainlink CCIP to allow token holders to bridge their tokens from Ethereum mainnet to Binance Smart Chain (BSC). The purpose of this audit is to perform a security review on its new LayerZero bridging capability and its existing Chainlink CCIP bridging mechanism.

5 Audit Scope

The audit scope is limited to:

```
contracts/MTokenMessager.sol
contracts/MTokenMessagerBase.sol
contracts/MTokenMessagerLZ.sol
contracts/MTokenMessagerV2.sol
```

We did however examine several other files and included some findings for them as an additional deliverable.

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the Matrixdock smart contracts provided by Matrixdock. In this period, a total of 17 issues were found.

The findings consist of 4 Low severity issues with the remainder being informational and gas optimizations. Of the 4 Low issues:

- 2 Lows were related to the ability for users to evade token blocklists

- 1 Low resulted in holders of the LINK token paying 10% more in bridging fees than they otherwise would
- 1 Low recommended preventing the ability for users to directly transfer ETH to the bridging contracts

The new bridging contracts:

- are immutable so no one including the owner can change them, giving users confidence that the same immutable code will execute every time
- feature very limited admin powers; the ability to pause bridging and for Chainlink to allow only bridging between trusted contracts which is a required security measure
- have no additional fee beyond the normal CCIP / LayerZero fee
- require the bridging fee to be paid in the native gas token
- appear to correctly follow CCIP & LayerZero integration guidelines

Summary

Project Name	Matrixdock
Repository	RWA-Contracts
Commit	0a83a96aab62...
Audit Timeline	Apr 2nd - Apr 4th, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	0
Low Risk	4
Informational	10
Gas Optimizations	3
Total Issues	17

Summary of Findings

[L-1] Forcing CCIP native fee payment results in 10 percent higher costs for `LINK` holders	Acknowledged
[L-2] Users can use transfer and bridging to evade having their tokens frozen via the blocklist	Acknowledged
[L-3] Missing `receive` function to reject direct ETH transfers in messager contracts	Acknowledged
[L-4] Cross-chain blocked recipients aren't properly handled	Acknowledged
[I-1] Only emit events when state actually changes	Acknowledged

[I-2] Use named mappings	Resolved
[I-3] Emit missing events for important state changes	Resolved
[I-4] LayerZero integration can be paused but CCIP integration can't be paused	Acknowledged
[I-5] Don't allow pausing for LayerZero receive, only send	Resolved
[I-6] Use consistent prefix for `internal` function names	Acknowledged
[I-7] Use named imports	Resolved
[I-8] Consider renaming `MTokenMessagerBase::ccipClient` as it is used by LayerZero integration and actually refers to `MToken`	Resolved
[I-9] Unused event `OwnershipTransferRequested` in `MTokenMessagerLZ`	Resolved
[I-10] Unnecessary code duplication in `MTokenMessager::sendDataToChain`	Resolved
[G-1] Use `immutable` for storage slots only set once in the constructor of non-upgradeable contracts	Resolved
[G-2] Use named returns especially for `memory` outputs	Resolved
[G-3] Cache amount and use Solady `SafeTransferLib::safeTransferETH` when refunding excess fee	Acknowledged

7 Findings

7.1 Low Risk

7.1.1 Forcing CCIP native fee payment results in 10 percent higher costs for `LINK` holders

Description: CCIP allows users to pay using either `LINK` or native gas token. By hard-coding `EVM2AnyMessage::feeToken = address(0)` the protocol forces all users to pay using the native gas token.

This results in higher costs for `LINK` holders as CCIP offers a 10% discount for paying using `LINK`, though this does simplify the protocol implementation.

Matrixdock: Acknowledged.

7.1.2 Users can use transfer and bridging to evade having their tokens frozen via the blocklist

Description: One unconventional application of regular transfers or cross-chain transfers via CCIP / LayerZero bridging is to evade the blocklist:

- user sees operator call to `MToken::addToBlockedList` in mempool which would block their address
- user front-runs this transaction by a normal transfer or a CCIP / LayerZero cross-chain transfer to bridge their tokens to a new `receiver` address on another chain
- if the operator attempts to call `MToken::addToBlockedList` on the other chain for the new `receiver` address, the user can bridge back to another new address again

To prevent this the operator can:

- pause bridging (pausing has been implemented for LayerZero but not CCIP) prior to calling `MToken::addToBlockedList`
- use a service such as flashbots when calling `MToken::addToBlockedList` so the transaction is not exposed in a public mempool

Matrixdock: Acknowledged.

7.1.3 Missing `receive` function to reject direct ETH transfers in messager contracts

Description: The messager contracts (`MTokenMessager`, `MTokenMessagerLZ`, `MTokenMessagerV2`) are designed to receive the bridging fee in native token but none of them implemented a `receive()` function to handle direct ETH transfers. Without this function, users can accidentally send ETH to the contract address where it will be permanently locked since there's no mechanism to withdraw it.

Recommended Mitigation: Add a `receive()` function that reverts to explicitly reject any direct ETH transfers to the contract:

```
contract MTokenMessagerBase {

    address public ccipClient;//@audit-info MToken

    constructor(address _ccipClient){
        ccipClient = _ccipClient;
    }
+
+     receive() external payable {
+         revert("ETH transfers not accepted");
+     }
}
```

Matrixdock: Acknowledged.

7.1.4 Cross-chain blocked recipients aren't properly handled

Description: The `MToken` contract implements a blocking mechanism to prevent certain addresses from interacting with the token. However, the cross-chain functionality doesn't properly handle blocked addresses.

There are two key issues:

1. In `MToken::msgOfCcSendToken`, the contract checks if the `receiver` is blocked on the source chain, but this check is invalid since the receiver exists on the destination chain.

```
    function msgOfCcSendToken(
        address sender,
        address receiver,
        uint256 value
    ) public view returns (bytes memory message) {
        _checkBlocked(sender);
        _checkBlocked(receiver);//@audit-issue receiver is not on the same chain, so this check
↪   does not make sense
        return abi.encode(TagSendToken, abi.encode(sender, receiver, value));
    }
```

2. In `MToken::ccReceiveToken`, there's no check to verify if the `receiver` is blocked on the current (destination) chain before minting tokens to them.

```
    function ccReceiveToken(bytes memory message) internal {
        (address sender, address receiver, uint value) = abi.decode(
            message,
            (address, address, uint)
        );
        _mint(receiver, value);//@audit-issue should check if receiver is blocked, might need to
↪   manage the funds sent to the blocked address
        emit CCReceiveToken(sender, receiver, value);
    }
```

These issues could allow blocked addresses to receive tokens via cross-chain transfers, bypassing the security controls intended by the protocol.

Impact: The blocking mechanism can be bypassed using cross-chain transfers. Malicious or sanctioned addresses that are blocked on one chain can still receive tokens through cross-chain transfers, undermining the security feature of the protocol.

Proof Of Concept:

```
    // Test cross-chain sending to a blocked address
    function testCrossChainSendToken_ToBlockedAddress() public {
        // Mint some tokens to user1
        uint256 amount = 100 * 10**18;
        mintTokens(user1, amount);

        // Block user2 on the destination chain
        vm.prank(operator);
        remoteChainMToken.addToBlockedList(user2);

        // User1 tries to send tokens cross-chain to blocked user2
        vm.startPrank(user1);
        mtoken.approve(address(mockMessager), amount);

        // When sending to a blocked address, the send may succeed but the tokens should never reach
        ↪   the destination
        mockMessager.sendTokenToChain{value: 0.01 ether}(
            CHAIN_SELECTOR_2,
            address(remoteChainMToken),
            user2,
            amount,
            ""
        );
        vm.stopPrank();

        // Check that user1's tokens are gone (burned in the sending process)
        assertEq(mtoken.balanceOf(user1), 0, "Tokens should be burned on source chain");

        // The blocked user should NOT receive any tokens
        // assertEq(remoteChainMToken.balanceOf(user2), 0, "Blocked user should not receive tokens");
    }
```

Recommended Mitigation:

1. Remove the receiver check in `msgOfCcSendToken` as it's not relevant to the source chain:

```
function msgOfCcSendToken(
    address sender,
    address receiver,
    uint256 value
) public view returns (bytes memory message) {
    _checkBlocked(sender);
-   _checkBlocked(receiver);
    return abi.encode(TagSendToken, abi.encode(sender, receiver, value));
}
```

2. Add a blocked address check in `ccReceiveToken` and implement a mechanism to handle tokens sent to blocked addresses:

```
function ccReceiveToken(bytes memory message) internal {
    (address sender, address receiver, uint value) = abi.decode(
        message,
        (address, address, uint)
    );
+   if (isBlocked[receiver]) {
+       // Option 1: Send to a recovery address
+       _mint(operator, value);
+       emit CCReceiveBlockedAddress(sender, receiver, value);
+   } else {
        _mint(receiver, value);
+   }
    emit CCReceiveToken(sender, receiver, value);
}
```

Matrixdock: Acknowledged.

7.2 Informational

7.2.1 Only emit events when state actually changes

Description: Only emit events when state actually changes, for example in `MTokenMessager::setAllowedPeer`:

```
    function setAllowedPeer(
        uint64 chainSelector,
        address messager,
        bool allowed
    ) external onlyOwner {
+       require(chainSelector][messager] != allowed, "No state change");
        allowedPeer[chainSelector][messager] = allowed;
        emit AllowedPeer(chainSelector, messager, allowed);
    }
```

Also affects:

- `MTokenMessagerV2::setAllowedPeer`

Matrixdock: Acknowledged.

7.2.2 Use named mappings

Description: Use named mappings to explicity indicate purpose of index => value:

```
MTokenMessager.sol
16:     mapping(uint64 => mapping(address => bool)) public allowedPeer;
//      mapping(uint64 chainSelector => mapping(address messager => bool allowed)) public allowedPeer;

MTokenMessagerV2.sol
28:     mapping(uint64 => mapping(address => bool)) public allowedPeer;
//      mapping(uint64 chainSelector => mapping(address messager => bool allowed)) public allowedPeer;
```

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerV2`.

Cyfrin: Resolved.

7.2.3 Emit missing events for important state changes

Description: Emit missing events for important state changes:

- `MTokenMessagerLZ::setLZPaused`

Matrixdock: Fixed in commit f3fbe97.

Cyfrin: Verified.

7.2.4 LayerZero integration can be paused but CCIP integration can't be paused

Description: `MTokenMessagerLZ` has a `bool lzPaused` storage slot and uses `onlyLZNotPaused` modifier to make LayerZero send/receive revert when paused.

In contrast `MTokenMessager` and `MTokenMessagerV2` have no similar pausing functionality for CCIP send/receive.

Consider whether this asymmetry is intentional or whether the CCIP send/receive should similarly be able to be paused.

Matrixdock: Acknowledged.

7.2.5 Don't allow pausing for LayerZero receive, only send

Description: `MTokenMessagerLZ` has the `onlyLZNotPaused` modifier on both the receiving function `_lzReceive` and the two sending functions `lzSendTokenToChain` / `lzSendMintBudgetToChain`.

Consider removing the `onlyLZNotPaused` modifier from `_lzReceive` as the sender has already burned their tokens when sending, so don't want receiving to revert in this case.

Matrixdock: Fixed in commit f3fbe97.

Cyfrin: Verified.

7.2.6 Use consistent prefix for `internal` function names

Description: Some of the `internal` functions use a _ prefix character but others don't. Use _ as a consistent prefix for all `internal` function names:

- `MTokenMessager::sendDataToChain`
- `MTokenMessagerLZ::sendThroughLZ`
- `MTokenMessagerV2::sendDataToChain`

Matrixdock: Acknowledged.

7.2.7 Use named imports

Description: The contracts mostly use named imports but strangely some import statements don't; use named imports everywhere:

`MTokenMessager`:

```
import "./interfaces/ICCIPClient.sol";
```

`MTokenMessagerLZ`:

```
import "./MTokenMessagerBase.sol";
import "./interfaces/ICCIPClient.sol";
```

`MTokenMessagerV2`:

```
import "./interfaces/ICCIPClient.sol";
import "./MTokenMessagerLZ.sol";
```

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerLZ` and `MTokenMessagerV2`.

Cyfrin: Verified.

7.2.8 Consider renaming `MTokenMessagerBase::ccipClient` as it is used by LayerZero integration and actually refers to `MToken`

Description: `MTokenMessager::ccipClient` and `MTokenMessagerBase::ccipClient` are used by both LayerZero (`MTokenMessagerLZ`) and CCIP (MTokenMessagerV2‘).

But they actually simply reference the `MToken` contract. Calling them `ccipClient` is initially confusing especially when reading the LayerZero integration and wondering why it is calling `ccipClient`.

Consider renaming `MTokenMessager::ccipClient` and `MTokenMessagerBase::ccipClient` to `mToken` and simply adding the additional functions to `IMToken` then deleting `ICCIPClient`.

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerBase`.

Cyfrin: Verified.

7.2.9 Unused event `OwnershipTransferRequested` in `MTokenMessagerLZ`

Description: The `MTokenMessagerLZ` contract declares an `OwnershipTransferRequested` event but never emits it anywhere in the contract. This suggests there might have been plans to implement a timelock mechanism for ownership transfer, but it was not completed. The event is defined but remains unused, which could indicate incomplete functionality.

```
    event OwnershipTransferRequested(address indexed from, address indexed to);
```

Matrixdock: Removed in commit f3fbe97.

Cyfrin: Verified.

7.2.10 Unnecessary code duplication in `MTokenMessager::sendDataToChain`

Description: The `sendDataToChain` function creates a message object and calculates fees, duplicating logic that already exists in the `getFeeAndMessage` function. This creates redundancy in the codebase, which can lead to inconsistencies during future updates and increases gas costs.

Recommended Mitigation: Refactor the `sendDataToChain` function to use the existing `getFeeAndMessage` function:

```
    function sendDataToChain(
        uint64 destinationChainSelector,
        address messageReceiver,
        bytes calldata extraArgs,
        bytes memory data
    ) internal returns (bytes32 messageId) {
-        Client.EVM2AnyMessage memory evm2AnyMessage = Client.EVM2AnyMessage({
-            receiver: abi.encode(messageReceiver),
-            data: data,
-            tokenAmounts: new Client.EVMTokenAmount[](0),
-            extraArgs: extraArgs,
-            feeToken: address(0)
-        });
-        uint256 fee = IRouterClient(getRouter()).getFee(
-            destinationChainSelector,
-            evm2AnyMessage
-        );
+        (uint256 fee, Client.EVM2AnyMessage memory evm2AnyMessage) = getFeeAndMessage(
+            destinationChainSelector,
+            messageReceiver,
+            extraArgs,
+            data
+        );
        if (msg.value < fee) {
            revert InsufficientFee(fee, msg.value);
        }
        messageId = IRouterClient(getRouter()).ccipSend{value: fee}(
            destinationChainSelector,
            evm2AnyMessage
        );
        if (msg.value - fee > 0) {
            payable(msg.sender).sendValue(msg.value - fee);
        }
        return messageId;
    }
```

The same issue is also present in `MTokenMessagerV2::sendDataToChain`.

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerV2`.

Cyfrin: Verified.

7.3 Gas Optimization

7.3.1 Use `immutable` for storage slots only set once in the constructor of non-upgradeable contracts

Description: Use `immutable` for storage slots only set once in the constructor:

- `MTokenMessager::ccipClient`
- `MTokenMessagerBase::ccipClient`

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerBase`.

Cyfrin: Verified.

7.3.2 Use named returns especially for `memory` outputs

Description: Use named returns especially for `memory` outputs, eg in `MTokenMessager::calculateCCSendTokenFeeAndMessage`

```
    function calculateCCSendTokenFeeAndMessage(
        uint64 destinationChainSelector,
        address messageReceiver,
        address sender,
        address recipient,
        uint value,
        bytes calldata extraArgs
    )
        public
        view
        returns (uint256 fee, Client.EVM2AnyMessage memory evm2AnyMessage)
    {
        bytes memory data = ccipClient.msgOfCcSendToken(
            sender,
            recipient,
            value
        );
-       return
+       (fee, evm2AnyMessage) =
            getFeeAndMessage(
                destinationChainSelector,
                messageReceiver,
                extraArgs,
                data
            );
    }
```

Also applies to:

- `MTokenMessager::calculateCcSendMintBudgetFeeAndMessage`
- `MTokenMessager::sendDataToChain` where obsolete `return` can be removed
- the same functions in `MTokenMessagerV2`

Matrixdock: Fixed in commit f3fbe97 for `MTokenMessagerV2`.

Cyfrin: Verified.

7.3.3 Cache amount and use Solady `SafeTransferLib::safeTransferETH` when refunding excess fee

Description: In `MTokenMessager::sendDataToChain` and `MTokenMessagerV2::sendDataToChain`, cache the amount and use Solady `SafeTransferLib::safeTransferETH` when refunding excess fee:

```
+ import {SafeTransferLib} from "@solady/utils/SafeTransferLib.sol";

-       if (msg.value - fee > 0) {
```

```
-           payable(msg.sender).sendValue(msg.value - fee);
-       }
+       uint256 excessFee = msg.value - fee;
+       if(excessFee > 0) {
+           SafeTransferLib.safeTransferETH(msg.sender, excessFee);
+       }
```

Matrixdock: Acknowledged.