M^0 Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

0kage

Immeas

Assisting Auditors

November 26, 2024

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

Mˆ0 Labs has developed a cross-chain token bridging protocol that enables users to transfer M tokens between different EVM-compatible blockchains using Wormhole as the messaging layer. The protocol follows a Hub-and-Spoke architecture:

Hub Portal: Deployed on Ethereum mainnet, serves as the central bridging point for M tokens

Spoke Portals: Deployed on various L2s (Base & Optimism currently), handle bridging on respective chains

Spoke Vault: Bridges excess M tokens on L2s back to the hub

Wormhole NttManager & Transceivers: Uses Wormhole's Native Token Transfer (NTT) Manager and transceivers for secure cross-chain messaging

Governance: Includes configurator and migrator contracts for protocol management and upgrades

The protocol is designed with upgradeability in mind, using ERC-1967 proxy pattern and supporting migrations through governance controls

5 Audit Scope

Following files were in the scope of the current audit

- src/Portal.sol
- src/HubPortal.sol
- src/SpokePortal.sol
- src/SpokeVault.sol
- src/governance/Governor.sol
- src/governance/Migrator.sol

- src/governance/Configurator.sol
- src/libs/PayloadEncoder.sol
- src/libs/RegistrarReader.sol
- src/libs/TypeConverter.sol

6 Executive Summary

Over the course of 7 days, the Cyfrin team conducted an audit on the M^0 smart contracts provided by M^0 Foundation. In this period, a total of 3 issues were found.

Mˆ0 Labs has developed a cross-chain token bridging protocol that enables seamless transfer of M tokens between Ethereum mainnet and various L2 networks (Base, Optimism) using Wormhole as the messaging layer. The protocol implements a Hub-and-Spoke architecture where a central HubPortal on mainnet coordinates with SpokePortals on L2s. A key component, the SpokeVault, manages excess M tokens on L2s and facilitates their return to the hub chain.

The audit identified three issues of varying severity, primarily centered around gas fee handling and token approvals. Two medium-risk findings relate to gas fee management and token approval mechanisms. All reported issues were successfully mitigated.

The test suite provides satisfactory coverage of the codebase, however, there are notable limitations in the current testing approach. Mock contracts implement minimal functionality and all tests are primarily verifying message calls. Consequently, complex interactions like token transfers and state changes are not accurately simulated by the existing tests. We recommend enhancing mock contracts to more closely mirror production behavior.

The documentation provided is adequate, clearly explaining the protocol's architecture, components, and intended functionality. Inline code comments and interface definitions are comprehensive and helpful.

Overall, the protocol demonstrates high code quality and maintains a limited attack surface through careful design choices. The core bridge architecture and cross-chain messaging implementation show robust security considerations.

Summary

Project Name	M^0
Repository	m-portal
Commit	8f909076f4fc...
Audit Timeline	Oct 31st - Nov 8th
Methods	Manual Review, Stateful Fuzzing

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	2
Low Risk	0
Informational	1
Gas Optimizations	0
Total Issues	3

Summary of Findings

[M-1] Cross-chain token transfer from `SpokeVault` fails due to approval from implementation contract instead of proxy	Resolved
[M-2] SpokeVault transfers will frequently revert due to Wormhole gas fee fluctuations	Resolved
[I-1] `ExcessMTokenSent` event logs `messageSequence_` as `0` incorrectly	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Cross-chain token transfer from SpokeVault fails due to approval from implementation contract instead of proxy

Description: SpokeVault is a contract that holds excess M tokens from Smart M and can transfer this excess to HubVault on mainnet via SpokeVault::transferExcessM.

To enable this, an infinite approval is set in the SpokeVault constructor:

```
// Approve the SpokePortal to transfer M tokens.
IERC20(mToken).approve(spokePortal_, type(uint256).max);
```

However, SpokeVault is deployed as a proxy, as seen in DeployBase::_deploySpokeVault:

```
spokeVaultImplementation_ = address(
    new SpokeVault(spokePortal_, hubVault_, destinationChainId_, migrationAdmin_)
);

spokeVaultProxy_ = _deployCreate3Proxy(address(spokeVaultImplementation_), _computeSalt(deployer_,
↪   "Vault"));
```

Where _deployCreate3Proxy deploys an ERC1967Proxy.

As a result, the approval set in the constructor applies only to the implementation contract, not to the proxy, which holds the tokens and initiates the transfers.

Impact: SpokeVault::transferExcessM will not function as intended because the Portal requires approval to transfer tokens. The M tokens in SpokeVault will be effectively stuck, though not permanently, as the contract is upgradeable.

Proof of Concept: Adding a token transfer in MockSpokePortal::transfer will cause SpokeVaultTests::test_-transferExcessM to fail:

```
+import { IERC20 } from "../../lib/common/src/interfaces/IERC20.sol";
...
    function transfer(
        uint256 amount,
        uint16 recipientChain,
        bytes32 recipient,
        bytes32 refundAddress,
        bool shouldQueue,
        bytes memory transceiverInstructions
    ) external payable returns (uint64) {
+       IERC20(mToken).transferFrom(msg.sender, address(this), amount);
    }
```

Recommended Mitigation: Consider setting token approvals only as needed in transferExcessM and removing the approval from the constructor:

```
+ IERC20(mToken).approve(spokePortal_, amount_);
  messageSequence_ = INttManager(spokePortal).transfer{ value: msg.value }(
      amount_,
      destinationChainId,
      hubVault_,
      refundAddress_,
      false,
      new bytes(1)
```

```
    );

```

In addition, this approach eliminates the infinite approval to an upgradeable contract, enhancing security.

M0 Foundation Fixed in commit 78ac49b

Cyfrin Verified

7.1.2 SpokeVault transfers will frequently revert due to Wormhole gas fee fluctuations

Description: The `SpokeVault.transferExcessM()` function forwards ETH to pay for Wormhole gas fees, but any excess ETH sent will cause the transaction to revert since `SpokeVault` lacks capability to receive ETH refunds. Current implementation only works if the user sends a fee that is exactly equal to the wormhole gas fee at the time of transaction.

This creates a significant usability problem because Wormhole gas fees can fluctuate between the time a user calculates the fee off-chain and when their transaction is actually executed.

The following code in `ManagerBase::_prepareForTransfer` shows that any gas fee shortfall will revert the transaction. Also, any excess gas fee over and above the delivery fee is refunded back to the sender.

```
// In ManagerBase.sol
function _prepareForTransfer(...) internal returns (...) {
    // ...
    if (msg.value < totalPriceQuote) {
        revert DeliveryPaymentTooLow(totalPriceQuote, msg.value);
    }

    uint256 excessValue = msg.value - totalPriceQuote;
    if (excessValue > 0) {
        _refundToSender(excessValue); // Reverts as SpokeVault can't accept ETH
    }
}

  function _refundToSender(
        uint256 refundAmount
    ) internal {
        // refund the price quote back to sender
        (bool refundSuccessful,) = payable(msg.sender).call{value: refundAmount}("");
         //@audit excess gas fee sent back to msg.sender (SpokeVault)

        // check success
        if (!refundSuccessful) {
            revert RefundFailed(refundAmount);
        }
    }

```

As a result, `transferExcessM` can only run if the user sends the exact fee. Doing so would be challenging because:

- Since this function can be called by anyone, it is likely that an average user would not know how to calculate the delivery fees
- Even if a user calculates the exact fee off-chain, it is highly likely that transaction fails due to natural gas fee fluctuation

Impact: If gas fee increases or decreases even slightly between quote and execution, transaction reverts.

Proof of Concept Make following changes to `MockSpokePortal` and run the test below in `SpokeVault.t.sol`:

```
      contract MockSpokePortal {
          address public immutable mToken;
```

```
        address public immutable registrar;

        constructor(address mToken_, address registrar_) {
            mToken = mToken_;
            registrar = registrar_;
        }

        function transfer(
            uint256 amount,
            uint16 recipientChain,
            bytes32 recipient,
            bytes32 refundAddress,
            bool shouldQueue,
            bytes memory transceiverInstructions
        ) external payable returns (uint64) {

            // mock return of excess fee back to sender
            if(msg.value > 1) {
                payable(msg.sender).transfer(msg.value-1);
            }

        }
    }

contract SpokeVaultTests is UnitTestBase {
     function testFail_transferExcessM() external { //@audits fails with excess fee
        uint256 amount_ = 1_000e6;
        uint256 balance_ = 10_000e6;
        uint256 fee_ = 2;
        _mToken.mint(address(_vault), balance_);
        vm.deal(_alice, fee_);

        vm.prank(_alice);
        _vault.transferExcessM{ value: fee_ }(amount_, _alice.toBytes32());
    }
}
```

Recommended Mitigation: Wormhole has a specific provision to refund excess gas fee back to the sender. This is put in place to ensure reliability of transfers even with natural gas fee fluctuations.

Consider updating the `SpokeVault.transferExcessM()` function to handle excess ETH refunds by adding a `payable receive()` function in `SpokeVault`. Also, please make sure the logic forwards any received ETH back to the original caller - not doing so would result in the excess gas fee stuck inside `SpokeVault`.

M0 Foundation Fixed in commit 78ac49b

Cyfrin Verified

7.2 Informational

7.2.1 `ExcessMTokenSent` event logs `messageSequence_` as `0` incorrectly

Description: When calling `SpokeVault::transferExcessM` to transfer excess M tokens from `SpokeVault`, an `ExcessMTokenSent` event is emitted:

```
) external payable returns (uint64 messageSequence_) {
    ...

    // @audit messageSequence_ not assigned yet
    emit ExcessMTokenSent(destinationChainId, messageSequence_, msg.sender.toBytes32(), hubVault_,
    ↪  amount_);

    messageSequence_ = INttManager(spokePortal).transfer{ value: msg.value }(
        ...
    );
}
```

However, at the time the event is emitted, the `messageSequence_` value hasn't been assigned and will therefore always be 0.

Recommended Mitigation: Consider emitting the event after the `transfer` call, once `messageSequence_` has been assigned a valid value.

M0 Foundation Fixed in commit 78ac49b

Cyfrin Verified