// Security Assessment

09.22.2025 - 09.24.2025

M Portal

MO

HALBORN

M Portal - M0

Prepared by: HALBORN

Last Updated 10/01/2025

Date of Engagement: September 22nd, 2025 - September 24th, 2025

Summary

100% OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

ALL FINDINGS	CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
3	0	0	1	0	2

TABLE OF CONTENTS

1. Introduction

`M0` engaged `Halborn` to conduct a security assessment on their smart contracts beginning on September 22, 2025 and ending on September 24, 2025. The security assessment was scoped to the smart contracts provided to Halborn. Commit hashes and further details can be found in the Scope section of this report.

The `M0` M Portal implements a cross-chain token bridge using Wormhole's Native Token Transfer (NTT) framework. The system enables M token transfers between Ethereum (Hub) and Layer 2 chains (Spokes). The earning index from Ethereum is propagated via Wormhole, allowing Solana holders to benefit from the same underlying collateral yield.

The reviewed changed were based on the following pull request: **https://github.com/m0-foundation/m-portal/pull/62**, which were later rebased to **https://github.com/m0-foundation/m-portal/pull/68**.

2. Assessment Summary

`Halborn` was provided with 3 days for this engagement and assigned 1 full-time security engineer to review the security of the smart contracts PR in scope. The assigned engineer possess deep expertise in blockchain and smart contract security, including hands-on experience with multiple blockchain protocols.

The objectives of this assessment were to:

- Identify potential security vulnerabilities within the smart contracts modified by the PR.
- Ensure that the smart contracts function as intended.

In summary, `Halborn` identified several areas for improvement to reduce the likelihood and impact of security risks, which were acknowledged by the `M0 team`. The main ones were:

- `Implement access control on administrative earning toggling functions`.
- `Use sequence verification for registrar modificiation messages`.
- `Reset allowance after use.`

3. Test Approach And Methodology

`Halborn` performed a combination of a manual review of the source code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program assessment. While manual testing is recommended to uncover flaws in business logic, processes, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices.

The following phases and associated tools were used throughout the term of the assessment:

- Research into the architecture and purpose of the smart contracts.
- Manual code review and walkthrough of the smart contracts.
- Manual assessment of critical Solidity variables and functions to identify potential vulnerability classes.
- Manual testing using custom scripts.
- Static security analysis of the scoped contracts and imported functions using Slither.
- Local deployment and testing with Foundry.

4. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

4.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

ATTACK COMPLEXITY (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

METRICS:

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2

EXPLOITABILITY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

4.2 IMPACT

CONFIDENTIALITY (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

INTEGRITY (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

AVAILABILITY (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

DEPOSIT (D):

Measures the impact to the deposits made to the contract by either users or owners.

YIELD (Y):

Measures the impact to the yield generated by the contract for either users or owners.

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Integrity (I)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1
Availability (A)	None (A:N) Low (A:L) Medium (A:M) High (A:H) Critical (A:C)	0 0.25 0.5 0.75 1
Deposit (D)	None (D:N) Low (D:L) Medium (D:M) High (D:H) Critical (D:C)	0 0.25 0.5 0.75 1
Yield (Y)	None (Y:N) Low (Y:L) Medium (Y:M) High (Y:H) Critical (Y:C)	0 0.25 0.5 0.75 1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

4.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

SCOPE (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

METRICS:

SEVERITY COEFFICIENT (C)	COEFFICIENT VALUE	NUMERICAL VALUE
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4

SEVERITY	SCORE VALUE RANGE
Informational	0 - 1.9

5. SCOPE

REPOSITORY

(a) Repository: m-portal

(b) Assessed Commit ID: 91bc4a1

(c) Items in scope:

- src/ExecutorEntryPoint.sol
- src/HubExecutorEntryPoint.sol
- src/Portal.sol
- src/HubPortal.sol

Out-of-Scope: Third party dependencies, economic attacks and everything besides https://github.com/m0-foundation/m-portal/pull/62

Out-of-Scope: New features/implementations after the remediation commit IDs.

6. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	1	0	2

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
STALE REGISTRAR MESSAGES CAN BE USED TO DISRUPT STATE INTEGRITY BETWEEN THE CHAINS.	MEDIUM	FUTURE RELEASE - 09/30/2025
PORTAL SHOULD RESET ALLOWANCE AFTER MTOKEN WRAP	INFORMATIONAL	ACKNOWLEDGED - 09/30/2025

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF ACCESS CONTROL IN HUBPORTAL EARNING ENABLING FUNCTIONS LEAD TO EARNING DOS	INFORMATIONAL	NOT APPLICABLE - 09/30/2025

7. FINDINGS & TECH DETAILS

7.1 STALE REGISTRAR MESSAGES CAN BE USED TO DISRUPT STATE INTEGRITY BETWEEN THE CHAINS.

// MEDIUM

Description

The M0 **Portal** system synchronizes the earners set across chains using Wormhole messages. However, the registrar update processing lacks sequence validation, allowing stale messages to overwrite newer state.

The attack scenario is as follows:

1. Attacker calls `sendRegistrarListStatus()` when `status = false` while supplying a Wormhole transceiver instruction where `shouldSkipRelayerSend = true`; the relayer is bypassed and the attacker retains the emitted VAA off-chain.
2. Someone calls `sendRegistrarListStatus()` when `status = true` (account was added to earners, for example the Hub itself) using the default instruction (`shouldSkipRelayerSend = false`) so the standard relayer delivers this update immediately.
3. After the "add" message executes on the spoke, the **HubPortal** is recognized as an authorized earner.
4. Attacker manually delivers the withheld "remove" VAA using `WormholeTransceiver.receiveMessage()`.
5. **SpokePortal** accepts the stale message and removes **HubPortal** from earners list.

Code Location

The `HubPortal::sendRegistrarListStatus` function sends whether an account is in the earners set to a destination chain, and does not use a sequence parameter.

```
88  /// @inheritdoc IHubPortal
89  function sendRegistrarListStatus(
90      uint16 destinationChainId_,
91      bytes32 listName_,
92      address account_,
93      bytes32 refundAddress_,
94      bytes memory transceiverInstructions_
95  ) external payable returns (bytes32 messageId_) {
96      // Sending Registrar list status to SVM chains is not supported at this time.
97      // To propagate earners to SVM chains call `sendEarnersMerkleRoot`.
98      if (_isSVM(destinationChainId_)) revert UnsupportedDestinationChain(destinationChainId_);
99
100     bool status_ = IRegistrarLike(registrar).listContains(listName_, account_);
101     bytes memory payload_ = PayloadEncoder.encodeListUpdate(listName_, account_, status_, destinati
102     messageId_ = _sendCustomMessage(destinationChainId_, refundAddress_, payload_, transceiverInstr
103
104     emit RegistrarListStatusSent(destinationChainId_, messageId_, listName_, account_, status_);
105 }
```

On the receiving end, the `SpokePortal::_updateRegistrarList` updates the earners list with the incoming message data (`Portal::_handleMsg` calls `Portal::_receiveCustomPayload` that calls

`SpokePortal::_updateRegistrarList`):

```
/// @notice Adds or removes an account from the Registrar List based on the message from the Hub chain.
function _updateRegistrarList(bytes32 messageId_, bytes memory payload_) private {
    (bytes32 listName_, address account_, bool add_, uint16 destinationChainId_) = payload_.decodeListUp

    _verifyDestinationChain(destinationChainId_);

    emit RegistrarListStatusReceived(messageId_, listName_, account_, add_);

    if (add_) {
        IRegistrarLike(registrar).addToList(listName_, account_);
    } else {
        IRegistrarLike(registrar).removeFromList(listName_, account_);
    }
}
```

BVSS

AO:A/AC:L/AX:L/R:P/S:U/C:N/A:N/I:C/D:N/Y:N (5.0)

Recommendation

A monotonically increasing `sequence` number must be included in registrar update messages. The `SpokePortal` must store and verify the last-seen `sequence` for each `(listName, account, destinationChain)` tuple. Any message with a `sequence` less than or equal to the stored value must be rejected to prevent stale overwrites.

Remediation Comment

FUTURE RELEASE: The **MO team** will address this issue in a future release.

References

https://wormhole.com/docs/products/token-transfers/native-token-transfers/reference/transceivers/evm/#encodewormholetransceiverinstruction

7.2 PORTAL SHOULD RESET ALLOWANCE AFTER MTOKEN WRAP

// INFORMATIONAL

Description

The **Portal** `_wrap` function does approve an amount so that the wrapped token contract can pull and escrow the canonical tokens. In case of failure, the approval is reset to 0, but in case of success, the approval is not zeroed.

It is a best practice to reset the approval after user so that no malicious contract (in that case, a wrapping contract) could pull only a fraction of the tokens during the wrapping, and be free to use the rest of the allowance after that.

Code Location

In `Portal::_wrap` function, the allowance is not reset on success:

```
function _wrap(address mToken_, address destinationWrappedToken_, address recipient_, uint256 amount_) pr
    IERC20(mToken_).approve(destinationWrappedToken_, amount_);

    // Attempt to wrap $M token
    // NOTE: the call might fail with out-of-gas exception
    //       even if the destination token is the valid wrapped M token.
    //       Recipients must support both $M and wrapped $M transfers.
    bool success = destinationWrappedToken_.safeCall(
        abi.encodeCall(IWrappedMTokenLike.wrap, (recipient_, amount_))
    );

    if (!success) {
        emit WrapFailed(destinationWrappedToken_, recipient_, amount_);
        // reset approval to prevent a potential double-spend attack
        IERC20(mToken_).approve(destinationWrappedToken_, 0);
        // transfer $M token to the recipient
        IERC20(mToken_).transfer(recipient_, amount_);
    }
}
```

BVSS

AO:A/AC:L/AX:H/R:N/S:U/C:N/A:N/I:N/D:M/Y:N (1.6)

Recommendation

It is recommended to zero the allowance after the token wrapping.

Remediation Comment

ACKNOWLEDGED: The **MO team** acknowledged this finding.

7.3 LACK OF ACCESS CONTROL IN HUBPORTAL EARNING ENABLING FUNCTIONS LEAD TO EARNING DOS

// INFORMATIONAL

Description

The `enableEarning()` and `disableEarning()` functions of the **HubPortal** contract control global protocol state affecting yield distribution across all chains, yet they lack any access control mechanisms. These functions interact with the irreversible `EarningCannotBeReenabled` constraint, making improper state changes permanent.

A single malicious user could call the `disableEarning()` function during active earning periods, permanently halting yield accrual.

Proof of Concept

The following unit test shows that after enabling the earning for the **HubPortal**, any user can manually call `disableEarning`, and turning the earning on again will be forbidden by the contract.

```
0  // SPDX-License-Identifier: UNLICENSED
1
2  pragma solidity 0.8.26;
3
4  import "./HubPortal.t.sol";
5
6  contract HubPortalDisableAccessTest is HubPortalTests {
7      address internal constant _GOVERNANCE = address(0xA11CE);
8      address internal constant _RANDOM_USER = address(0xBEEF);
9
10     function test_randomUserDisables_afterGovernanceRemoval_cannotReenable() external {
11         uint128 index = 1_100000068703;
12
13         // Governance sets up earning eligibility.
14         _mToken.setCurrentIndex(index);
15         vm.prank(_GOVERNANCE);
16         _registrar.setListContains(_EARNERS_LIST, address(_portal), true);
17         _portal.enableEarning();
18
19         // Any random user can now call disableEarning even though they cannot touch the registrar.
20         vm.prank(_RANDOM_USER);
21         _portal.disableEarning();
22
23         // Governance attempts to re-enable but the toggle is irreversible.
24         vm.expectRevert(IHubPortal.EarningCannotBeReenabled.selector);
25         _portal.enableEarning();
26     }
27 }
```

The success shows that the governance could not turn the earning back after the exploit:

```
0  Ran 1 test for test/unit/HubPortalDisableAccess.t.sol:HubPortalDisableAccessTest
1  [PASS] test_randomUserDisables_afterGovernanceRemoval_cannotReenable() (gas: 99731)
2  Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 1.38ms (113.83µs CPU time)
```

BVSS

AO:A/AC:L/AX:H/R:F/S:U/C:N/A:N/I:N/D:N/Y:C (0.8)

Recommendation

It is recommended to add access control to the earning toggling functions.

Remediation Comment

NOT APPLICABLE: The issue was considered as not applicable due to out-of-scope components preventing the arbitrary change of the earning states.

Halborn strongly recommends conducting a follow-up assessment of the project either within six months or immediately following any material changes to the codebase, whichever comes first. This approach is crucial for maintaining the project's integrity and addressing potential vulnerabilities introduced by code modifications.