

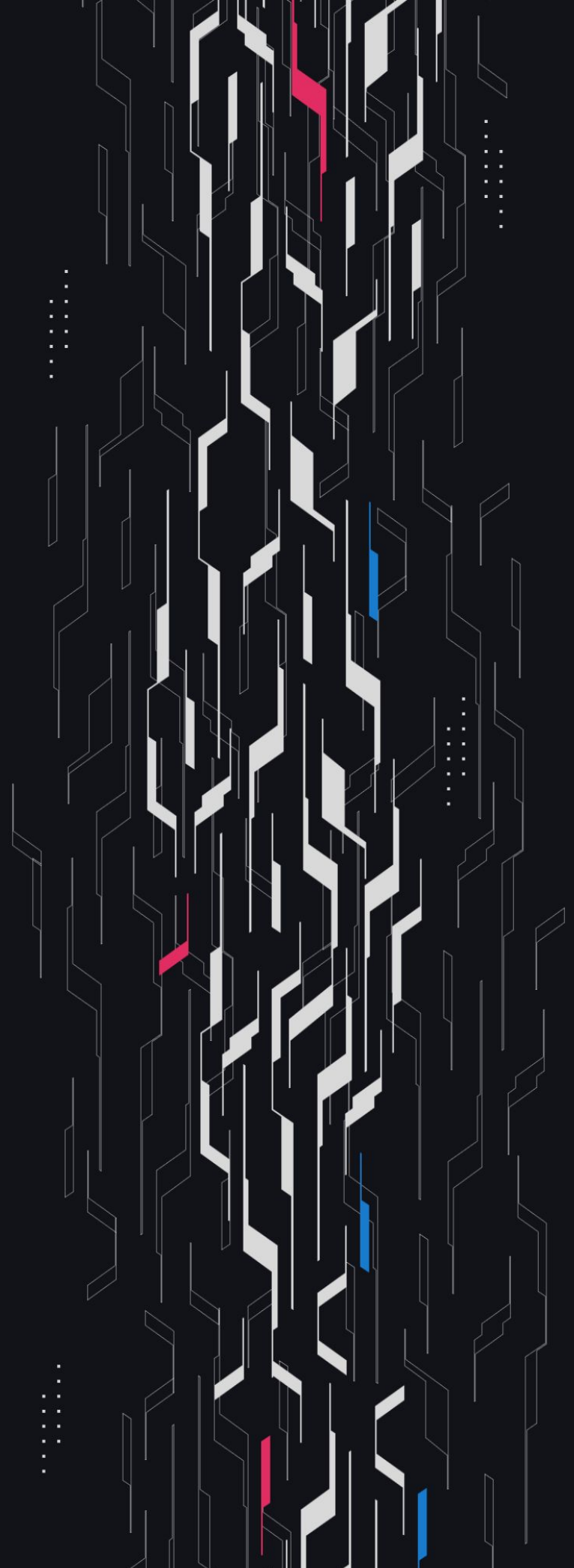
GA GUARDIAN

M0

**Liquidity Delivery
Review**

Security Assessment

January 30th, 2026



Summary

Audit Firm Guardian

Prepared By CuriousApple, Zdravko Hristov, Alex Lazar

Client Firm M0

Final Report Date January 30, 2026

Audit Summary

M0 engaged Guardian to review the security of their M0's Liquidity Delivery. From the 8th of December 2025 to the 7th of January 2026, a team of 3 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of High and Critical issues detected, Guardian assigns a Confidence Ranking of 2 to the protocol. Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment. Guardian strongly advises that the protocol undergo a full follow-up audit at a finalized and fully remediated commit before any mainnet deployment. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

Note: Fixes to the findings uncovered in the remediation review have not been reviewed by Guardian. Guardian recommends the client either perform an amended follow-up review period to focus on the fixes to remediations findings or seek a follow-up audit from another team.

Security Recommendation: Guardian recommends a follow-up security review of the protocol at a finalized frozen commit.

✅ Verify the authenticity of this report on Guardian's GitHub: <https://github.com/guardianaudits>

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only 1 Significant issue was uncovered per week. Design patterns are sound, and test coverage is strong. Recommendation: Suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity issues. Varied Low severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. Recommendation: Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1-2 High/Critical findings per engagement week.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities. Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. Recommendation: Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week. Or additional High/Critical findings uncovered in remediation review which have not been resolved and confirmed by Guardian.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥ 5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. Recommendation: Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥ 5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Information

Project Overview 5

Audit Scope & Methodology 7

Smart Contract Risk Assessment

Findings & Resolutions 12

Addendum

Disclaimer 76

About Guardian 77

Project Overview

Project Summary

Project Name	M0
Language	Solidity
Codebase(s)	https://github.com/m0-foundation/m-portal-v2 , https://github.com/m0-foundation/solana-portal , https://github.com/m0-foundation/liquidity-delivery
Commit(s)	Main Review Commits: M-Portal-V2 Main Review Commit: b9799699cab6986e09beee4833423510d3128c1d Solana-Portal Main Review Commit: 687ba581e11593abc975f48d296a39b28c5a4661 Liquidity-Delivery Main Review Commit: 15a972bc51a559824b699461c9cdd706badd8491 Remediation Review Commits: M-Portal-V2 Remediation Commit: fc69ff4cf1b68017f45f3d53242f57ed3e5e4973 Solana-Portal Remediation Commit: eaa029e11c493d909149cd8ea8e2dd4c473eae4d Liquidity-Delivery Remediation Commit: 53510e6dbf5ca5ae2fae9fd3f79c7f4f75210566

Project Overview

Audit Summary

Delivery Date	January 30, 2026
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	3	0	0	0	0	3
● High	5	0	0	0	0	5
● Medium	10	0	0	1	0	9
● Low	14	0	0	6	0	8
● Info	27	0	0	15	0	12

Audit Scope & Methodology

M-Portal-V2 Scope:

```
contract,source,total,comment
m-portal-v2/evm/src/HubPortal.sol,123,223,46
m-portal-v2/evm/src/Portal.sol,307,546,126
m-portal-v2/evm/src/SpokePortal.sol,62,113,30
m-portal-v2/evm/src/utils/ReentrancyLock.sol,11,19,4
m-portal-v2/evm/src/libraries/BytesParser.sol,46,64,7
m-portal-v2/evm/src/libraries/PayloadEncoder.sol,100,209,76
m-portal-v2/evm/src/libraries/TypeConverter.sol,29,70,30
m-portal-v2/evm/src/bridgeAdapters/BridgeAdapter.sol,94,152,22
m-portal-v2/evm/src/bridgeAdapters/wormhole/WormholeBridgeAdapter.sol,57,96,19
m-portal-v2/evm/src/bridgeAdapters/hyperlane/HyperlaneBridgeAdapter.sol,44,73,15
m-portal-v2/evm/src/bridgeAdapters/wormhole/libraries/ExecutorMessages.sol,8,22,11
m-portal-v2/evm/src/bridgeAdapters/wormhole/libraries/RelayInstructions.sol,11,27,13
source count: {
  total: 1614,
  source: 892,
  comment: 399,
  single: 399,
  block: 0,
  mixed: 0,
  empty: 323,
  todo: 1,
  blockEmpty: 0,
  commentToSourceRatio: 0.44730941704035876
}
```

Audit Scope & Methodology

Solana-Portal Scope:

file,blank,comment,code
programs/wormhole-adapter/src/instructions/resolve_execute.rs,33,9,202
programs/portal/src/instructions/receive_message.rs,21,6,169
packages/common/src/accounts.rs,22,3,157
programs/portal/src/instructions/send_token.rs,28,8,157
packages/common/src/payloads.rs,14,0,155
programs/hyperlane-adapter/src/instructions/receive_message.rs,22,6,143
packages/common/src/adapter_accounts.rs,11,0,126
programs/wormhole-adapter/src/instructions/send_message.rs,21,9,124
packages/common/src/receive metas.rs,14,11,123
programs/hyperlane-adapter/src/instructions/send_message.rs,18,8,114
programs/wormhole-adapter/src/instructions/mod.rs,16,8,114
programs/portal/src/instructions/mod.rs,5,3,103
programs/wormhole-adapter/src/instructions/receive_message.rs,16,7,92
programs/hyperlane-adapter/src/state.rs,8,0,87
programs/portal/src/lib.rs,18,3,80
programs/hyperlane-adapter/src/instructions/transfer_admin.rs,15,0,67
programs/portal/src/instructions/transfer_admin.rs,15,0,67
programs/wormhole-adapter/src/instructions/transfer_admin.rs,15,0,67
programs/hyperlane-adapter/src/instructions/mod.rs,8,0,63
programs/hyperlane-adapter/src/lib.rs,15,0,62
programs/hyperlane-adapter/src/instructions/initialize.rs,9,0,58
programs/portal/src/instructions/send_merkle_root.rs,9,1,55
programs/wormhole-adapter/src/lib.rs,14,0,55
programs/portal/src/instructions/send_fill_report.rs,8,1,54
programs/wormhole-adapter/src/state.rs,8,0,48
programs/portal/src/instructions/send_index.rs,8,1,45
programs/hyperlane-adapter/src/instructions/sync_extensions.rs,7,0,44
packages/common-macros/src/lib.rs,6,4,41
programs/hyperlane-adapter/src/instructions/pause.rs,7,0,38
programs/portal/src/instructions/pause.rs,7,0,38
programs/wormhole-adapter/src/instructions/pause.rs,7,0,38
programs/hyperlane-adapter/src/instructions/set_peer.rs,6,1,37
programs/wormhole-adapter/src/instructions/set_peer.rs,6,1,37
packages/common/src/errors.rs,1,0,34
programs/portal/src/state.rs,7,0,34
programs/portal/src/instructions/initialize.rs,6,0,30
packages/common/src/lib.rs,3,0,29
programs/wormhole-adapter/src/consts.rs,2,0,29
programs/wormhole-adapter/src/instructions/initialize.rs,6,0,28
packages/common/src/conversions.rs,7,1,26
packages/common/src/interfaces.rs,6,0,18
programs/hyperlane-adapter/src/consts.rs,3,0,14
TOTAL,478,91,3102

Audit Scope & Methodology

EVM Liquidity-Delivery Scope:

```
contract,source,total,comment
liquidity-delivery/evm/src/OrderBook.sol,373,554,92
source count: {
  total: 554,
  source: 373,
  comment: 92,
  single: 84,
  block: 8,
  mixed: 3,
  empty: 92,
  todo: 0,
  blockEmpty: 0,
  commentToSourceRatio: 0.24664879356568364
}
```

SVM Liquidity-Delivery Scope:

```
file,blank,comment,code
liquidity-delivery/svm/programs/order_book/src/instructions/fill.rs,65,42,292
liquidity-delivery/svm/programs/order_book/src/instructions/open.rs,25,9,179
liquidity-delivery/svm/programs/order_book/src/state/orders.rs,24,3,157
liquidity-delivery/svm/programs/order_book/src/instructions/report_fill.rs,24,15,117
liquidity-delivery/svm/programs/order_book/src/instructions/claim_refund.rs,20,6,111
liquidity-delivery/svm/programs/order_book/src/error.rs,1,0,56
liquidity-delivery
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: <i>High</i>	Impact: <i>Medium</i>	Impact: <i>Low</i>
Likelihood: <i>High</i>	● Critical	● High	● Medium
Likelihood: <i>Medium</i>	● High	● Medium	● Low
Likelihood: <i>Low</i>	● Medium	● Low	● Low

Impact

- High** Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
- Medium** A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
- Low** Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

- High** The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
- Medium** An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
- Low** Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.
Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Orderbook Drained Due To Cancelled Orders	Validation	● Critical	Resolved
C-02	Anyone Can Replay The M0 Transfer Between Spokes	Access Control	● Critical	Resolved
C-03	Cross Chain Orders Can Be Canceled From Source	Unexpected Behavior	● Critical	Resolved
H-01	Crosschain Orders Cannot Be Filled On EVM	DoS	● High	Resolved
H-02	Portals Cannot Send Cancel Reports	DoS	● High	Resolved
H-03	Stuck Funds Due To Non-isolated Spokes	DoS	● High	Resolved
H-04	Incorrect Scaling Of Token Amount	Unexpected Behavior	● High	Resolved
M-01	Portal Initialize Logic Cannot Be Executed	Upgradeability	● Medium	Resolved
M-02	Merkle Root Updates Cannot Be Send To SVM	Unexpected Behavior	● Medium	Resolved
M-03	Loss For Solvers Due To A Cancel Race Condition	Unexpected Behavior	● Medium	Resolved
M-04	Hardcoded Chain ID Vulnerable To Replay	Configuration	● Medium	Resolved
M-05	SVM: Race Condition For Earner Merkle Roots	DoS	● Medium	Resolved
M-06	SVM: M Supply Inflation Due To Missing Burn	Logical Error	● Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-01	Users Can Receive Native M Token	Access Control	● Low	Acknowledged
L-02	Gasless Order VERSION Is Not Verified	Validation	● Low	Resolved
L-03	Do Not Reset Finality On Removed Support	Unexpected Behavior	● Low	Resolved
L-04	Risk Of Broken Accounting After Disabling Portal	Unexpected Behavior	● Low	Acknowledged
L-05	Index Should Be Updated During All xChain Interactions	Unexpected Behavior	● Low	Resolved
L-06	Missing Check If Default Adapter Is Supported	Validation	● Low	Resolved
L-07	Incorrect msgSender() When Tokens Are Received	Logical Error	● Low	Acknowledged
L-08	Orderbooks Should Have Minimum Fill Amount	Validation	● Low	Acknowledged
L-09	Other Programs Can't CPI SendFillReport	Unexpected Behavior	● Low	Acknowledged
L-10	Sender Cancellation For Cross-Chain Orders	Suggestion	● Low	Acknowledged
L-11	Inaccurate Comment Regarding Chain ID Definition	Best Practices	● Low	Resolved
L-12	Incorrect Comment _revertIfTokenTransferDisabled	Best Practices	● Low	Resolved
I-01	The Sanity Check In _fillOrder() Is Needed	Warning	● Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-02	Order Cannot Be Filled If Solver == Recipient	Warning	● Info	Resolved
I-03	Index Mismatch Can Cause Dormant Funds & Losses	Informational	● Info	Acknowledged
I-04	Out-of-order Message Finalization	Informational	● Info	Acknowledged
I-05	Tighten Constraints For Filling Orders	Best Practices	● Info	Resolved
I-06	Fee On Transfer Tokens Can Be Sent To Order Book	Unexpected Behavior	● Info	Resolved
I-07	Rounding Difference May Not Always Be Covered	Warning	● Info	Acknowledged
I-08	Having Shared Lists May Not Be Ideal	Informational	● Info	Acknowledged
I-09	HubPortal Is Giving Out More Value	Informational	● Info	Acknowledged
I-10	TokenReceived NatSpec Could Be Improved	Best Practices	● Info	Resolved
I-11	Existing Griefing Vector In _receiveToken()	Informational	● Info	Acknowledged
I-12	Payload Gas Limit Could Be Validated	Best Practices	● Info	Resolved
I-13	Outdated Comment	Best Practices	● Info	Resolved
I-14	Portal Lacks On Chain Replay Protection	Validation	● Info	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
I-15	Potential For Temp Stuck Messages With Hyperlane	Informational	● Info	Acknowledged
I-16	The Name Solana-earners Is Misleading	Best Practices	● Info	Acknowledged
I-17	User Can Omit Overhead Even If It's Configured	Informational	● Info	Acknowledged
I-18	Unused Message_account.consumed Property	Superfluous Code	● Info	Acknowledged
I-19	Nonce Inconsistency For Message Ids	Best Practices	● Info	Acknowledged
I-20	Orderbooks: Version Upgrade	Best Practices	● Info	Resolved
I-21	Cannot Close Order Accounts	Gas Optimization	● Info	Acknowledged
I-22	Lacking Token Validation For Filling Local Order	Validation	● Info	Resolved
I-23	SVM Chain ID Assignment And EVM Collision Risk	Configuration	● Info	Acknowledged
I-24	Use Of The Wrong Multiplier	Best Practices	● Info	Acknowledged

C-01 | Orderbook Drained Due To Cancelled Orders

Category	Severity	Location	Status
Validation	● Critical	OrderBook.sol: 431-446C10	Resolved

Description [PoC](#)

OrderBook._fillOrder() doesn't check the status of the order to be filled. It just transfers tokenOut from the solver to the recipient and gives the appropriate amount of tokenIn to the solver. This works because the address that requested the order already transferred these tokens when the order was opened.

However, orders can be cancelled by calling requestCancelOrder() which will trigger the logic in _claimRefund() and the order requester will receive back the tokenIn they have put upon creating the order.

Notice how the filled amounts stay unchanged as well. Therefore, an order can be filled after it was cancelled, which makes the orderbook double spend the tokenIn for local orders.

A malicious user can leverage this to drain the orderbook by creating a local order with amountIn = tokenIn.balanceOf(orderBook) and a small amountOut. The user can then first cancel the order to receive their tokens back and then fill it themselves to drain all the tokens out of the orderbook.

Recommendation

Revert _fillOrder() if the status of the local order is different than Created.

Resolution

M0 Team: The issue was resolved in commit [b36bd28](#).

C-02 | Anyone Can Replay The M0 Transfer Between Spokes

Category	Severity	Location	Status
Access Control	● Critical	WormholeBridgeAdapter.sol: 118-119	Resolved

Description

VAAAs are multicast by default in Wormhole. There is no default target chain for a given message - the application developer must enforce destination validation.

From Wormhole docs:

"VAAAs are multicast by default. This means there is no default target chain for a given message. The application developer decides on the format of the message and its treatment upon receipt."

Looking at `parseAndVerifyVM` in Wormhole's core library, the hash verified against guardian signatures does *not* include the destination chain:
<https://github.com/wormhole-foundation/wormhole/blob/c113791abd5241bc7a23655e3a7475085d51dab7/ethereum/contracts/Messages.sol#L50>

Attack Scenario
If the protocol is live on ETH, ARB, and OP:

- A legitimate message is sent from ETH → ARB
- If guardians for ARB and OP overlap, the same VAA could potentially be replayed on OP
- Guardians can indeed overlap across chains (e.g., Chorus One guardian appears on multiple chains with same address)
- See guardian overlap:

<https://wormhole-foundation.github.io/wormhole-dashboard/#/?endpoint=Mainnet>

Recommendation

Consider passing destination chain ID and destination address in the payload, then verify in `executeVAAv1`:

- Destination chain ID should match `block.chainid`
- Destination address should match `address(this)`

Resolution

M0 Team: The issue was resolved in [PR#19](#).

C-03 | Cross Chain Orders Can Be Canceled From Source

Category	Severity	Location	Status
Unexpected Behavior	● Critical	OrderBook.sol: 323-336	Resolved

Description [PoC](#)

The finality buffer design was removed and currently the action of canceling an order must be initiated on the destination chain.

This will mark the order as `Cancelled` and solvers won't be able to fill it anymore. If it was a crosschain one, `sendCancelReport()` will notify the source chain and the funds will be released to the original sender.

However, there is no validation performed to ensure the current chain is the same as the destination chain for the order. Because of this, any order that originated on the current chain can be instantly canceled, which will execute `_claimRefund()`.

```
if (orderData_.originChainId == chainId) {
  // Local orders can be immediately refunded
  _claimRefund(orderId_, order);
}
```

This opens up a griefing vector where the creator of the order can cancel their order as soon as it's filled on the destination chain, resulting in their profit and loss for the solver.

Recommendation

Add validation to `_cancelOrder()` that reverts if the chain id doesn't match the order destination chain.

Resolution

M0 Team: The issue was resolved in commit [e945bc6](#).

H-01 | Crosschain Orders Cannot Be Filled On EVM

Category	Severity	Location	Status
DoS	● High	OrderBook.sol: 459-469	Resolved

Description

An order on the destination chain is filled by executing the internal `OrderBook._fillOrder()` function. In case the order originated from a different chain, a fill report is sent back to that chain.

The `IMessenger.sendFillReport()` function signature accepts 3 arguments.

The external `Portal.sendFillReport()` function has two overloads - one of them accepts 4 arguments and the other 5, but none accepts 3.

The first is:

```
function sendFillReport(
uint32 destinationChainId,
IOrderBookLike.FillReport calldata report,
bytes32 refundAddress,
bytes calldata bridgeAdapterArgs
) external;
```

And the second is:

```
function sendFillReport(
uint32 destinationChainId,
IOrderBookLike.FillReport calldata report,
bytes32 refundAddress,
address bridgeAdapter,
bytes calldata bridgeAdapterArgs
) external;
```

Because of this, attempts to fill any crosschain order will be always failing.

In addition to that, `sendFillReport` should be called with an appropriate `value` to pay for crosschain fees. The current implementation doesn't forward any value which will result in all messages failing, even if the right interface was used.

Recommendation

Use one of the correct overloads when sending the fill report and make sure to specify an appropriate amount value sent.

Resolution

M0 Team: The issue was resolved in [PR#11](#).

H-02 | Portals Cannot Send Cancel Reports

Category	Severity	Location	Status
DoS	● High	Portals	Resolved

Description

When a crosschain order is cancelled, `IMessenger(messenger).sendCancelReport()` is called. The messenger is the `Portal`, where such a function doesn't exist. Because of this the cancel feature won't work as the transaction will always be reverting.

The same is true for the Solana portal - there isn't an instruction that sends the cancel report.

Recommendation

Implement the needed functionality in the portals.

Resolution

M0 Team: The issue was resolved in [PR#25](#).

H-03 | Stuck Funds Due To Non-isolated Spokes

Category	Severity	Location	Status
DoS	● High	HubPortal, SpokePortal	Resolved

Description

The new `crossSpokeTokenTransferEnabled` flag shows if a spoke can send tokens to other spokes or only to the hub. If the flag is set to `false`, cross spoke transfers are not enabled and each time tokens are sent from the hub to that spoke, `bridgedPrincipal` for that spoke is increased.

Similarly, when tokens are sent from that spoke to the hub, the hub decreases the `bridgedPrincipal` with the appropriate amount. However, this flag limits the spokes only for sending tokens, not receiving. Because of this it's possible to send tokens from a non-isolated spoke to an isolated one.

This will result in the second spoke having more principal than recorded in the hub. The Hub reverts if a spoke tries to unlock more principal than the bridged amount.

```
uint248 principalAmount = IndexingMath.getPrincipalAmountRoundedDown(uint240(amount),
_currentIndex());
// Prevents unlocking more than was bridged to the Spoke
if (principalAmount > totalBridgedPrincipal) revert InsufficientBridgedBalance();
```

In result, the tokens that have been sent will remain stuck and the transfer message will be reverting.

Recommendation

Consider fully isolating these spokes. For this to be possible, each spoke must track the flag for the other spokes and not allow sending tokens to an isolated one.

Resolution

M0 Team: The issue was resolved in [PR#29](#).

H-04 | Incorrect Scaling Of Token Amount

Category	Severity	Location	Status
Unexpected Behavior	● High	send_token.rs: 181	Resolved

Description

On Solana, `send_token` forwards `m_amount` derived from `ext_swap::unwrap` directly into the bridge payload.

The unwrap path computes and transfers principal units of \$M (not scaled UI units).

On the EVM side, the `HubPortal` releases the exact payload amount with no scaling. Because the payload carries principal units, the EVM unlock will be under-issued by the current multiplier (e.g., multiplier=2 → 500 principal becomes 500 released instead of 1000).

This creates a cross-chain supply/amount mismatch whenever the multiplier ≠ 1, leading to losses for users.

Recommendation

Ensure the payload amount is scaled/UI units before sending to EVM.

Resolution

M0 Team: The issue was resolved in [PR#31](#).

M-01 | Portal Initialize Logic Cannot Be Executed

Category	Severity	Location	Status
Upgradeability	● Medium	HubPortal.sol: 56-61,35-37	Resolved

Description

The portals are upgradeable contracts with `initialize()` functions. The `initialize()` function in `HubPortal()` sets the value of the `disableEarningIndex`.

```
function initialize(address owner, address pauser, address operator) external initializer {
  _initialize(owner, pauser, operator);
  HubPortalStorageStruct storage $ = _getHubPortalStorageLocation();
  $.disableEarningIndex = IndexingMath.EXP_SCALED_ONE;
}
```

This contract is intended to become the new implementation of the current [HubPortal](#). The portal is currently initialized at version 4. After its logic is updated, attempts to call `initialize()` will revert because of the initializer modifier. If left uninitialized, `_isEarningEnabled()` will return false even when earning is enabled and lead to wrong index updates.

```
function _isEarningEnabled() internal view returns (bool) {
  return wasEarningEnabled() && disableEarningIndex() == IndexingMath.EXP_SCALED_ONE;
}
```

If the storage of the old portal is cleared before the upgrade, to let `initialize()` be called again, then the initialized version of the contract would be reset from 4 to 1.

Recommendation

Use the `reinitializer()` modifier instead.

Resolution

M0 Team: The issue was resolved in [PR#15](#).

M-02 | Merkle Root Updates Cannot Be Send To SVM

Category	Severity	Location	Status
Unexpected Behavior	● Medium	Global	Resolved

Description

The `receive_message` instruction in the `solana-portal` program decodes the received payload and if its type is `EarnMerkleRoot`, the merkle root for the earners is updated.

```
Payload::EarnMerkleRoot(payload) => {
  msg!("Received EarnMerkleRoot Payload");
  ctx.accounts.portal_global.update_index(payload.index);
  return Self::handle_index_payload(&ctx, payload);
}
```

However, it's impossible to send such a message from the EVM portal because there is no function for it and the payload type doesn't exist.

```
enum PayloadType {
  TokenTransfer,
  Index,
  RegistrarKey,
  RegistrarList,
  FillReport
}
```

Recommendation

Implement a way to send the merkle root from EVM to SVM.

Resolution

M0 Team: The issue was resolved in [PR#14](#).

M-03 | Loss For Solvers Due To A Cancel Race Condition

Category	Severity	Location	Status
Unexpected Behavior	● Medium	OrderBook.sol	Resolved

Description

Orders are both filled and canceled from the destination chain. It's done by sending crosschain messages to the source chain where the order status is handled. If the order is fulfilled, it can't be canceled anymore because its status is set to `Completed`.

On the other hand, for partial fills the status is set to `Created`, which means the order is still cancelable. Previously, there was a finality buffer which protected solvers from instant cancellations.

Now that it's removed, a race condition resulting in a loss for the solver and gain for the order creator can happen. For example:

- Alice opens order on OP to be filled on Arb: 3000 USDC in, 1 WETH out.
- Bob fills half of it on Arb: pays 0.5 WETH to Alice on Arb => fill report is sent
- Alice cancels the order on Arb => cancel report is sent
- If the cancel report arrives on OP before the fill report, Alice will receive back 3000 USDC and any attempts to fill the order will fail. In result, Alice will have stolen 1500 USDC from Bob.

Recommendation

You can:

- Include the filled amounts in the cancel report
- Use these amounts in `reportCancel()` and forward them to `_claimRefund()` instead of reading them from storage. This ensures the user will be refunded only what's left for refund after all inflight fills.
- Allow `reportFill()` to be executed for orders where `status == Cancelled`. This would allow all of the inflight fills to be executed after the cancel.

Resolution

M0 Team: The issue was resolved in [PR#31](#).

M-04 | Hardcoded Chain ID Vulnerable To Replay

Category	Severity	Location	Status
Configuration	● Medium	OrderBook.sol: 67-68	Resolved

Description

The contract caches the chain ID in an immutable variable during construction:

```
constructor(uint32 chainId_, address messenger_) {
  chainId = chainId_;
  messenger = messenger_;
}
```

This cached chainId is used throughout the contract for critical logic including:

- Order creation and ID generation (line 191)
- Cross-chain vs same-chain detection (line 323, line 385, line 439)
- Destination chain validation (line 579)

If a blockchain undergoes a contentious hard fork (like Ethereum/Ethereum Classic or Ethereum PoW/PoS), both chains will initially have the same chain ID (cached).

This creates a replay attack vulnerability similar to the Omni Bridge exploit during the ETHPoW fork. <https://chainbulletin.com/ethw-replay-exploit-caused-by-omni-contract-vulnerability>

The cached chain ID doesn't update after a fork, so both chains believe they are the "correct" chain with that ID.

Recommendation

Consider implementing dynamic chain ID validation to detect forks: Option 1 - Read block.chainid directly

Option 2 - Add runtime validation to check cached == block.chainid

Resolution

M0 Team: The issue was resolved in [PR#34](#).

M-05 | SVM: Race Condition For Earner Merkle Roots

Category	Severity	Location	Status
DoS	● Medium	send_merkle_root.rs: 13	Resolved

Description

The SVM portal implementation allows any account to call `send_merkle_root` and `send_index` instructions without access control checks. These functions enable any SVM chain in the system to broadcast earner merkle root updates to any other SVM chain.

When a message with `PayloadData::EarnerMerkleRoot` is received, the portal calls `earn::cpi::propagate_index()` which updates the earn program's state. While the earn program protects against stale index updates using `new_multiplier >= current_multiplier`, the merkle root itself doesn't have ordering protection.

Attack Scenario

Consider a system with Hub (EVM), SVM Chain 1, and SVM Chain 2:

- At T0: Hub propagates fresh merkle root `ROOT_100` (index = 100) to SVM Chain 1
- At T1: SVM Chain 1 successfully updates: { index: 100, merkle_root: `ROOT_100` }
- At T2: Hub propagates newer merkle root `ROOT_100_v2` (index = 100, different earner set) to SVM Chain 1
- At T3: SVM Chain 1 successfully updates: { index: 100, merkle_root: `ROOT_100_v2` }
- At T4: Attacker on SVM Chain 2 calls `send_merkle_root` with their stale snapshot `ROOT_100` (index = 100) and sends it to SVM Chain 1
- At T5: SVM Chain 1 receives message from SVM Chain 2 with { index: 100, merkle_root: `ROOT_100` }
- At T6: Earn program checks: `new_multiplier` (from index 100) >= `current_multiplier` (from index 100) → PASSES
- At T7: Since `merkle_root != [0; 32]`, it overwrites the current root

SVM Chain 1 now has stale `ROOT_100` instead of the fresh `ROOT_100_v2`, even though the index remains 100.

This means legitimate earners in the newer merkle tree can no longer claim rewards, while earners only in the old tree can still claim (potentially including removed malicious earners).

Recommendation

Consider allowing propagation of merkle earner root only from hub instead of even allowing between spokes for svm.

Resolution

M0 Team: The issue was resolved in [PR#32](#).

M-06 | SVM: M Supply Inflation Due To Missing Burn

Category	Severity	Location	Status
Logical Error	● Medium	send_token.rs: 107	Resolved

Description

The SVM portal implementation doesn't burn M as its done in EVM spoke portals.

User's wrapped token (e.g., mUSDC) is transferred to portal

Portal calls `ext_swap::cpi::unwrap()` which burns the wrapped token and releases underlying M tokens.

M tokens are deposited into portal's custody account.(`send_token.rs:148-161`)

Bridge message is sent with token transfer payload

No burn operation occurs - M tokens remain locked in portal's `m_token_account`

On the receiving chain, `receive_message.rs:126-158` mints new M tokens to the recipient.

This leads to permanent supply inflation.

Every SVM-to-SVM transfer permanently increases the total M token supply.

In a system with N SVM chains, each transfer inflates the supply by the transfer amount. After K transfers totaling T tokens, the excess supply is exactly T tokens locked across all portals' custody accounts.

Recommendation

Consider adding burn inside send on SVM as well.

Resolution

M0 Team: The issue was resolved in [PR#24](#).

L-01 | Users Can Receive Native M Token

Category	Severity	Location	Status
Access Control	● Low	Portal.sol: 429-430	Acknowledged

Description

Portals allow users to set the destination token as M token and receive M token directly. This behavior bypasses the swap facility's restrictions which controls who can obtain M tokens.

Even if users receive native M token through this path, they cannot do anything meaningful with it since:

- All DeFi integrations are expected to use wrapped versions of M
- Users cannot swap native M into wrapped versions because `swapInM` enforces `_revertIfNotApprovedSwapper` on the swap facility

Recommendation

Consider revisiting whether users should be allowed to hold native M token:

1. If allowing native M is acceptable - No changes needed
2. If allowing native M is not acceptable - Consider:
 - Whether there are any legal repercussions of users holding native M
 - If legal concerns exist: Block setting M token as destination token in portal transfers
 - If no legal concerns: Add a warning when users set destination token as M Token in portal transfers to inform them of the limited utility

Resolution

M0 Team: Acknowledged.

L-02 | Gasless Order VERSION Is Not Verified

Category	Severity	Location	Status
Validation	● Low	OrderBook.sol: 253-256	Resolved

Description

The `GaslessOrderParams` struct is used to create an order for another user that has signed a message for it. In `_openOrderFor()`, the `originChainId` and `nonce` are verified against the actual current values of the contract.

```
if (orderParams_.originChainId != chainId) revert InvalidOriginChain();
OrderBookStorageStruct storage $ = _getOrderBookStorageLocation();
// Requiring a nonce in the order provides replay protection for the sender
if (orderParams_.nonce != $.senderNonces[orderParams_.sender]) revert InvalidNonce();
```

The `version` field is never verified. This may allow executing the message with another version of the contract, not the one the user specified.

Although, if the contract is properly updated, this should not happen, because users sign the result of `_getDigest()`, which includes the domain separator, and the contract version is encoded inside.

Recommendation

Either remove the version field of the struct (because it's already included in the domain separator), or validate it against the actual contract `VERSION`.

Resolution

M0 Team: The issue was resolved in [PR#12](#).

L-03 | Do Not Reset Finality On Removed Support

Category	Severity	Location	Status
Unexpected Behavior	● Low	OrderBook.sol: 540-541	Resolved

Description

Each destination chain can be disabled in the Orderbook by calling `setDestinationConfig()`. The `newFinalityBuffer` will then be reset to 0, and will be used as the new effective buffer after the old buffer expires.

This can lead to a situation where a solver fulfills an order on a destination chain that was disabled on the source and the order creator cancels it immediately to grief the solver. For example:

- Day1: Alice opens an order, `finalityBuffer` = 1 day
- Day2: The destination chain is disabled, effective buffer is still 1 day, but the new buffer is 0
- Day3: Solver fills the order on the destination chain. Alice is now able to immediately cancel her order because the new finality is 0. Now `reportFill` will revert and the solver won't receive their tokens.

The same behavior is present on Solana

Recommendation

Consider not resetting the finality buffer when a chain is being disabled, and when enabling it later not applying `newFinalityBufferEffectiveTimestamp`

Resolution

M0 Team: The issue was resolved in [PR#7](#).

L-04 | Risk Of Broken Accounting After Disabling Portal

Category	Severity	Location	Status
Unexpected Behavior	● Low	HubPortal.sol	Acknowledged

Description

In the event that HubPortal is removed from the list of earners, the permissionless HubPortal.disableEarning() function will be executed. It will call mToken.stopEarning() and will save the current index in the disableEarningIndex variable.

This shows earning is disabled, therefore _currentIndex() will be capped to the latest index before the portal was disabled.

This ensures index doesn't continue growing on spoke chains.

Once an earner is disabled, mToken.stopEarning(address) can be called by anyone to stop its earning. Meaning there may be a period of time between calling mToken.stopEarning() and HubPortal.disableEarning().

In this period, the index of the mToken will be increasing, which will result in inflated disableEarningIndex propagated to all the spokes. If this happens, users are incentivized to send back their funds to the hub, because:

- they will receive more tokens on the hub than they should
- sending tokens to the hub earlier reduces the risk of being unable to withdraw due to portal insolvency

Recommendation

Since the earners list is controlled by the M0 team, make sure that in the event of the hub portal being disabled, the action is always batched together with a call to HubPortal.disableEarning().

Resolution

M0 Team: Acknowledged.

L-05 | Index Should Be Updated During All xChain Interactions

Category	Severity	Location	Status
Unexpected Behavior	● Low	Portals	Resolved

Description

There are 3 custom payloads that can be sent from the hub to a spoke. The first, of type `Index`, fetches the `MToken` index on the hub and sends it to a spoke in order for that spoke to update its index. The other two - `RegistrarKey` and `RegistrarList` - update the registry values.

This can lead to unexpected situations if the registry is updated, but the index is not. For example, a user is not an earner and has 1000 tokens on a spoke.

The index on the hub has grown from 1 to 1.1 and the user is approved as earner after that. Now a registry update can be sent to the spoke, which would allow the user to enter the system with index 1, leading to instant profit.

On the other side, users may end up losing as well. If they are removed from the earners list and the index is not updated, when `stopEarning` is called for them, their principal amount will be calculated using the old index.

The same is true for `reportFill` for `Hub ⇒ Spoke`. The amount transferred to the solver would (if earner) would be more.

Since the index would not be updated, the solver would receive more `mTokens`

Recommendation

Consider refreshing the index as well when a registry key or list is updated.

Resolution

M0 Team: The issue was resolved in [PR#34](#).

L-06 | Missing Check If Default Adapter Is Supported

Category	Severity	Location	Status
Validation	● Low	Global	Resolved

Description

All the functions for sending crosschain messages through the portals - `sendToken()`, `sendFillReport()`, `sendMTokenIndex()`, `sendRegistrarKey()` and `sendRegistrarListStatus()` - have two overrides. One of them allows the user to specify a desired `bridgeAdapter`, while the other uses the default one.

In case a bridge adapter is specified, the following function will check if it's enabled in the `supportedBridgeAdapter` mapping for that `chainID`.

```
_revertIfUnsupportedBridgeAdapter(destinationChainId, bridgeAdapter);
```

However, this check is not present in the other override - the one using the default adapter. The only check performed there is about the default adapter not being `address(0)`.

It's important to highlight that in order to disable a default adapter, the right course of action is not to set it to `address(0)` because then `address(0)` will be irreversibly as a supported adapter.

What should be done instead is to disable the default adapter by calling `setSupportedBridgeAdapter()`. This will modify the `supportedBridgeAdapter` mapping setting the flag to `false`.

Because one of the two overrides of these functions doesn't check that mapping, a disabled adapter may end up still being usable, leading to unexpected behaviors.

Recommendation

Add the following check to all the overrides using the default bridge adapter.

```
_revertIfUnsupportedBridgeAdapter(destinationChainId, bridgeAdapter);
```

Resolution

M0 Team: The issue was resolved in [PR#1cc8a120be2ddb635abf14918fe4f79746ace80e](#).

L-07 | Incorrect msgSender() When Tokens Are Received

Category	Severity	Location	Status
Logical Error	● Low	Portal.sol: 165-182	Acknowledged

Description

All permissionless functions in SwapFacility implement the isNotLocked modifier.

It serves two purposes - one is guarding against reentrancy - if the locker is currently not address(0), the call will revert, so any attempt for reentrancy will fail. The second is that SwapFacility.msgSender() will return the address of the locker.

As it can be seen, if the actual msg.sender is a trusted router, the locker will be set to whatever that router returns as msgSender(). The intended routers for the system are the portals. So every time the portal calls SwapFacility, the locker of the facility will be set to Portal.msgSender().

The Portal implements the same pattern as the facility - its msgSender() function returns the current locker.

There are two flows where the portal interacts with the facility - when sending and receiving tokens. In case of sending tokens, sendToken implements the whenNotLocked modifier, so msgSender() will return the address that initiated the action.

However, this modifier is not applied to the receiveMessage function or any of the functions called inside it. This will lead to the locker in the portal staying address(0) and in result:

- there will be no reentrancy protection in the portal
- the locker in the facility will be set to address(0), leading to wrong address usage by the extension the portal is wrapping into and potentially causing unexpected behaviors, depending on the exact implementation
- the swap facility reentrancy guard will not work as intended and technically reentrancy would be also possible at the facility level.

Recommendation

Set the locker to an appropriate address. You should choose what that address is. If you apply the modifier to the receiveMessage() function, the locker will be set as the bridge adapter.

Another approach you can take is to set the locker manually to the sender of the message. However, there should be a workaround for addresses that don't fit in 20 bytes, for example for tokens coming from Solana.

Resolution

M0 Team: Acknowledged.

L-08 | Orderbooks Should Have Minimum Fill Amount

Category	Severity	Location	Status
Validation	● Low	OrderBook.sol: 418,367-377	Acknowledged

Description

Orders created with permissionless solvers can be fulfilled by anyone. Several solvers may compete for getting their transaction executed first.

Because there is no minimum amount enforced on the tokens spent or the tokens received, a solver may end up fulfilling a foreign order with minuscule amount and still pay for crosschain fees.

For example, there is an order for 1000e18 tokens. Solver A sends a fill transaction for all of the 1000e18 tokens, but solver B's transaction for 1000e18 - 1 is executed first. Now Solver A will fill only 1 wei, but still pay for the crosschain message.

Recommendation

Consider adding minimum amount in in the filler params and use it when filling orders.

Resolution

M0 Team: Acknowledged.

L-09 | Other Programs Can't CPI SendFillReport

Category	Severity	Location	Status
Unexpected Behavior	● Low	lib.rs: 114	Acknowledged

Description

The Order Book's FillForeignOrder instruction performs a deep chain of Cross-Program Invocations (CPI).

The current flow is:

1. order_book::fill_foreign_order() at https://github.com/GuardianOrg/liquidity-delivery-team1-1764637085192/blob/cc3e9f649123575e2ee262ecf817426f299d744c/svm/programs/order_book/src/lib.rs#L114
2. portal::send_fill_report() at https://github.com/GuardianOrg/liquidity-delivery-team1-1764637085192/blob/cc3e9f649123575e2ee262ecf817426f299d744c/svm/programs/order_book/src/instructions/fill.rs#L444 and <https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcca81/programs/portal/src/lib.rs#L92>
3. hyperlane_adapter::send_message() at <https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcca81/programs/portal/src/instructions/mod.rs#L103> and https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcca81/programs/hyperlane_adapter/src/lib.rs#L80
4. IGP's PayForGas at https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcca81/programs/hyperlane_adapter/src/instructions/send_message.rs#L271 and <https://github.com/hyperlane-xyz/hyperlane-monorepo/blob/main/rust/sealevel/programs/hyperlane-sealevel-igp/src/processor.rs#L268>
5. IGP sending lamports at <https://github.com/hyperlane-xyz/hyperlane-monorepo/blob/main/rust/sealevel/programs/hyperlane-sealevel-igp/src/processor.rs#L358>

That makes it already hit the current CPI depth ceiling of 4 CPIs. This may not be a problem for your average solver using a Phantom wallet, for example. But it is indeed a problem if another program attempts to solve (this could happen for aggregators, automated solver and more commonly multi-sigs and smart wallets).

Solving foreign orders is essentially bricked for any such user.

Recommendation

I don't honestly know what to recommend without you having to undergo significant architectural change. Potentially adapter logic could be more coupled to the portal? Or you could accept this limitation.

Note that with SIMD-0268 having been approved, this may not be a problem soon as the depth will increase from 4 to 8. (<https://github.com/solana-foundation/solana-improvement-documents/blob/main/proposals/0268-raise-cpi-nesting-limit.md>)

Resolution

M0 Team: Acknowledged.

L-10 | Sender Cancellation For Cross-Chain Orders

Category	Severity	Location	Status
Suggestion	● Low	OrderBook.sol: 282-283	Acknowledged

Description

In the `cancelOrder` function at lines 287-291, the authorization logic prevents senders from canceling cross-chain orders before the deadline, only allowing the recipient or sender of same-chain orders to cancel.

The suspected intention is to handle address incompatibility between different chain types (e.g., EVM addresses vs Solana addresses in EVM↔SVM scenarios). However, this restriction impacts legitimate use cases:

EVM↔EVM transfers: Users with the same key pair controlling the same address on both chains cannot cancel their cross-chain orders

SVM↔SVM transfers: Similarly restricted despite having the same address derivation

Recommendation

Consider implementing a more nuanced cancellation policy that allows sender cancellation for compatible chain pairs.

Resolution

M0 Team: Acknowledged.

L-11 | Inaccurate Comment Regarding Chain ID Definition

Category	Severity	Location	Status
Best Practices	● Low	OrderBook.sol: 56-57	Resolved

Description

The comment for the `chainId` state variable at line 56 is incorrect:

```
/// @notice the chain ID of this chain according to the messaging network used by this
contract
uint32 public immutable chainId;
```

The comment states the chain ID is "according to the messaging network," but the implementation shows that this is actually the chain ID according to consensus (i.e., the EVM's chain identifier), not a messaging-network-specific identifier.

Recommendation

Consider correcting the comment.

Resolution

M0 Team: The issue was resolved in [PR#34](#).

L-12 | Incorrect Comment _revertIfTokenTransferDisabled

Category	Severity	Location	Status
Best Practices	● Low	SpokePortal.sol: 157-158	Resolved

Description

The NatSpec comment for the `_revertIfTokenTransferDisabled` function in `SpokePortal.sol:157` incorrectly describes the revert condition.

Current comment:

```
/// @dev Reverts if the destination chain is the Hub chain
```

Actual behavior: The function reverts when the destination chain is NOT the Hub chain (and cross-spoke transfers are disabled).

Recommendation

Consider correcting the comment.

Resolution

M0 Team: The issue was resolved in commit [c2661a4](#).

I-01 | The Sanity Check In _fillOrder() Is Needed

Category	Severity	Location	Status
Warning	● Info	OrderBook.sol: 395-396	Resolved

Description

There is the following sanity check in `OrderBook._fillOrder()` which validates that the passed `orderData` matches the `orderId`.

```
// Ensure the provided order ID matches the computed order ID from the order data
// This check is not strictly required, but it is a useful sanity check for solvers
// to ensure they have the order data correct
if (orderId_ != getOrderId(orderData_)) revert OrderIdMismatch();
```

The comment above the check says it's not strictly required, but it's useful. However, if the check wasn't there, anyone could have provided arbitrary order data (including tokens and amounts) with existing `orderId` and easily drain the contract.

Recommendation

Remove that part of the comment that says the check is not needed and be careful to keep the validation if the code is changed in the future.

```
// Ensure the provided order ID matches the computed order ID from the order data
•      // This check is not strictly required, but it is a useful sanity check for solvers
•      // to ensure they have the order data correct
if (orderId_ != getOrderId(orderData_)) revert OrderIdMismatch();
```

Resolution

M0 Team: The issue was resolved in [PR#14](#).

I-02 | Order Cannot Be Filled If Solver == Recipient

Category	Severity	Location	Status
Warning	● Info	OrderBook.sol: 449-453	Resolved

Description

During order fulfillment, tokenOut is transferred from the solver to the recipient by calling safeTransferExactFrom().

```
// Transfer tokens from the solver to the recipient
IERC20(orderData_.tokenOut.toAddress()).safeTransferExactFrom(
msg.sender,
orderData_.recipient.toAddress(),
uint256(amountOutToFill_)
);
```

If the solver is the same address as the recipient, the net change in their balance will be <= 0 and the transaction will revert.

Recommendation

Consider reverting if solver == recipient when an order is being opened.

Resolution

M0 Team: The issue was resolved in [PR#13](#).

I-03 | Index Mismatch Can Cause Dormant Funds & Losses

Category	Severity	Location	Status
Informational	● Info	Global	Acknowledged

Description

M token is not a standard ERC20 - it's yield-bearing with an index that grows over time. The index updates on different chains are not real time but stepwise, which creates edge cases during cross-chain transfers.

Scenario 1: Hub to Spoke (ETH → ARB)

- Lock X on ETH, release X on ARB
- User plays with X on ARB, balance grows to X+1
- Index on ETH updates, locked amount becomes X+2 (but not yet propagated to ARB)
- User bridges X+1 from ARB back to ETH
- Contract releases X+1, leaving +1 dormant inside the contract

This doesn't cause insolvency on ETH since ARB index is always $\leq$ ETH index (monotonically increasing). At worst: small net loss for users and dormant funds accumulating on hub.

Scenario 2: Spoke to Spoke (ARB → OP)

- Index on ARB = A
- Index on OP = O, where $O > A$
- Transfer from ARB to OP
- Since OP index is higher, receive doesn't update the index
- Principal burned on ARB: Amount/A
- Principal received on OP: Amount/O
- Since $O > A$: $\text{principal}_{OP} < \text{principal}_{ARB}$
- End balance transferred is the same, but principal differs
- Users of OP (and OP locker) take a small loss as index progresses

Recommendation

Be aware of these edge cases around index mismatch between chains.

Resolution

M0 Team: Acknowledged.

I-04 | Out-of-order Message Finalization

Category	Severity	Location	Status
Informational	● Info	Global	Acknowledged

Description

In general, the Portal doesn't enforce ordering of operations since it doesn't need to - token transfer for User B happening before User A (even if User A initiated the cross-chain transfer first) doesn't change anything.

However, for protocol state updates like:

- Index updates
- Registrar key updates
- Registrar List updates

Order does matter when two updates are sent at the same time. The stale update could be finalized later, resulting in outdated state on the destination chain overwriting the newer state.

Recommendation

Be aware of this edge case. If out-of-order finalization occurs for state updates, remember to initiate a fresh update afterward to ensure destination chain has the correct current state.

Resolution

M0 Team: Acknowledged.

I-05 | Tighten Constraints For Filling Orders

Category	Severity	Location	Status
Best Practices	● Info	fill.rs: 317	Resolved

Description

Malicious solvers could attempt to operate on a native order in the `fill_foreign_order` instruction. Since there is no validation that the originating chain id is not the current one, it's possible to do that.

The first three fields of `NativeOrder` can be decoded into `ForeignOrder`

There:

- `ForeignOrder.status = NativeOrder.status`
- `ForeignOrder.amount_in_released = version + sender[0..14]`
- `ForeignOrder.amount_out_released = sender[16..30]`

The following if statement won't be executed for an already existing order since its status is `Created`

The execution will proceed to compute `amount_out_remaining` and `amount_in_remaining` which will most likely result in revert, due to `amount_out_released` and `amount_in_released` being very large numbers.

P.S.: Note this is also the case for cancelling. A simple and quick example would be that a grifter can operate on a `NativeOrder` with `CancelForeignOrder` once it expires which would allow them to cancel it without refunding the user. The only thing that stops them in this case is that the portal would (should?) disallow same chain transfers.

Recommendation

1. Add a check for origin chain to not be equal to dest chain in `fill_foreign_order` (same for cancelling functions)
2. Add checks for `order_type` in both `fill_native_order` and `fill_foreign_order` (same for cancelling functions)
3. Explore separating `Order<ForeignOrder>` and `Order<NativeOrder>` into something like `OrderForeign` and `OrderNative` so that discriminators would actually be different. Maybe even explore having a different order type for each possible scenario (`Sol → Sol`, `Sol → Eth`, `Eth → Sol`).

Resolution

M0 Team: The issue was resolved in [PR#35](#).

I-06 | Fee On Transfer Tokens Can Be Sent To Order Book

Category	Severity	Location	Status
Unexpected Behavior	● Info	open.rs: 195	Resolved

Description

On Solana, a user can open an order where `token_out` is a fee-on-transfer (Token-2022) mint. This differs from the EVM implementation, which prevents opening and filling FoT orders by using `safeTransferExact()` to enforce exact received amounts.

In the Solana flow, FoT behavior can make an order impossible to fully fill, since each transfer reduces the received amount by a fee. As a result:

- Orders may only be partially fillable, up to approximately `initial_amount - fee`, unless additional tokens are transferred into the order’s ATA beyond what was deposited in `open_order()`.
- Any remaining portion may become stuck, leading to degraded UX and solver confusion.
- For cross-chain orders, this can cause downstream failures if the remote side expects the order to be fully fillable or relies on exact amounts, potentially resulting in failed fills or reverted execution on the destination chain.

Recommendation

Check for `token2022` tokens that have fees and disallow them, as you do on the EVM side.

Resolution

M0 Team: The issue was resolved in [PR#36](#).

I-07 | Rounding Difference May Not Always Be Covered

Category	Severity	Location	Status
Warning	● Info	HubPortal.sol	Acknowledged

Description

During MToken transfers from non-earners to earners, the transferred amount is converted to principal amount before it's added to the recipient balance. The calculation rounds down and because of that the receiver may bear losses.

This is applicable to the HubPortal because it's an approved earner. When tokens are sent crosschain, the portal checks uses `_getTransferAmount()`, which tolerates `_getMaxRoundingError()` of rounding error in favor of the user. Which means that if the actual amount received by portal is in these bounds, the user will still receive their original requested amount. There is a comment saying that yield will cover that difference.

This is true because if the recipient on the destination chain is an earner, the calculation will use the same index and the principal amount will increase with the same amount on the two chains. On the other hand, if the recipient is not an earner, as times goes on the yield will be accumulating unilaterally, only on the hub side, therefore the system won't be left insolvent.

However, there are some edge cases that may leave the portal underfunded. For example, let's say `index = 2` and the user is not an earner

1. User bridges 5 MTokens from hub to spoke:
 - $\text{hub principal} = 5 / 2 = 2$
 - $\text{user balance on spoke} = 5$
2. User bridges 3 more MTokens from hub to spoke
 - $\text{hub principal} = 2 + 3 / 2 = 3$
 - $\text{user balance on spoke} = 5 + 3 = 8$
3. User is approved as an earner so their raw balance is converted to principal
 - $\text{user principal on spoke} = 8 / 2 = 4$
4. At this point the principal of the user is more than the principal of the HubPortal, therefore the portal won't be able to pay out all of the tokens, no matter the index. For example, let's say the index becomes 20 on both chains
5. User bridges 80 MToken back to hub
 - $\text{user principal on spoke} = 4 - 80 / 20 = 0$
6. Hub has to transfer 80 MTokens to the user
 - $\text{hub has to burn } 80 / 20 = 4 \text{ principal, but it has only } 3$

Recommendation

This case should be extremely rare, especially because the hub portal may be holding additional funds due to natural user losses, for example the issue described in [I-03](I-03%202c78bda5828c81e4aac9d34c6af9b937.md). Having said that, the team may be monitoring the portal and ensure it's balance is enough to cover transfers.

Resolution

M0 Team: Acknowledged.

I-08 | Having Shared Lists May Not Be Ideal

Category	Severity	Location	Status
Informational	● Info	Global	Acknowledged

Description

The current design of the system stores all keys and lists in the registrar on the Hub and the data is being propagated to the spokes.

This means that once an address is added to a list, it's in that list on every chain. This may lead to unexpected behavior, for example, two different contract have the same address across chains, leading to unauthorized earning by one of them.

Recommendation

Consider whether this is acceptable.

Resolution

M0 Team: Acknowledged.

I-09 | HubPortal Is Giving Out More Value

Category	Severity	Location	Status
Informational	● Info	HubPortal.sol: 255	Acknowledged

Description

Because the HubPortal is enabled as an earner, when `_mintOrUnlock()` is executed, the amount of principal transferred will be rounded up, which can lead to losses up to `currentIndex / 1e12`.

This should not cause insolvency because it's expected that portal will have dust values due to the stepwise index behavior.

Recommendation

Keep this behavior in mind.

Resolution

M0 Team: Acknowledged.

I-10 | TokenReceived NatSpec Could Be Improved

Category	Severity	Location	Status
Best Practices	● Info	IPortal.sol: 53	Resolved

Description

The `TokenReceived()` event is emitted when a token transfer is received by the portal. One of the parameters there, the `index`, is the index on the chain the transfer originated to.

The documentation of this parameter is vague. It could be understood that this is the new token index on the current chain, which may not be the case, especially if the transfer is from spoke to hub.

```
/// @param index The \M token index
```

Recommendation

Consider explaining that this index is on the source chain.

Resolution

M0 Team: The issue was resolved in commit [db90e53](#).

I-11 | Existing Griefing Vector In _receiveToken()

Category	Severity	Location	Status
Informational	● Info	Portal.sol: 453-459	Acknowledged

Description

The `Portal._receiveToken()` function is executed when a crosschain token transfer is received. If the desired destination token is an extension token, `_wrap()` will execute a low level call to the `swapFacility` in order to swap the received `mToken` into an extension and send it to the recipient. If the operation fails, the raw `mToken` will be transferred to the recipient instead.

There are currently two ways to execute this flow - through Hyperlane or Wormhole. For example, with hyperlane the flow is `Mailbox.process() -> HyperlaneBridgeAdapter.handle() -> Portal.receiveMessage() -> Portal._receiveToken()`. In theory, this flow can be called with such a gas value that 63/64 of it are less than the gas used by the `swapFacility`, but the 1/64 of it is enough to execute the fallback logic.

In this case, anyone would be able to grief crosschain transfers for normal users resulting in these users receiving `mToken` instead. The current fallback logic needs roughly ~33000 gas. This means $1/64 * totalGas \geq 33000 \Rightarrow totalGas \geq 2112000$. Since 63/64 of that would be forwarded to the `swapFacility`, the facility must use at least $63/64 * 2112000 = 2079000$ gas in order for the attack to be executed.

When the currently deployed [SwapFacility](#) was tested with the [MUSD](#) extension, the gas spent was 166731. This shows that under normal circumstances the issue should not be exploitable. However, if the `SwapFacility` logic or the extension's `wrap()` function introduce new gas-heavy features, the probability of exploiting the vector increases.

Recommendation

Monitor gas usage of the facility and the extensions to avoid this situation.

Resolution

M0 Team: Acknowledged.

I-12 | Payload Gas Limit Could Be Validated

Category	Severity	Location	Status
Best Practices	● Info	Portal.sol: 335	Resolved

Description

The `_sendMessage()` function in the Portal calls `payloadGasLimit()` to determine the gas that should be forwarded for the current call.

```
function _sendMessage(
uint32 destinationChainId,
PayloadType payloadType,
bytes32 refundAddress,
bytes memory payload,
address bridgeAdapter,
bytes calldata bridgeAdapterArgs
) internal {
  IBridgeAdapter(bridgeAdapter).sendMessage{ value: msg.value }(
    destinationChainId, payloadGasLimit(destinationChainId, payloadType), refundAddress, payload,
    bridgeAdapterArgs
  );
}
```

It's possible for messages to be sent before such a value is configured. This will result in the call using 0.

Recommendation

Consider whether additional validation or a fallback is needed before sending the message.

Resolution

M0 Team: The issue was resolved in commit [45b1400](#).

I-13 | Outdated Comment

Category	Severity	Location	Status
Best Practices	● Info	Portal.sol: 374	Resolved

Description

There is a comment left above the call to `_burnOrLock()` that says there is bridged principal amount tracked for the hub portal. In this version of the project, there is no such feature.

```
// In case of Hub, only update the bridged principal amount as tokens already transferred.
_burnOrLock(transferAmount);
```

Recommendation

Update the comment.

Resolution

M0 Team: Resolved.

I-14 | Portal Lacks On Chain Replay Protection

Category	Severity	Location	Status
Validation	● Info	Portal.sol	Resolved

Description

There is no replay guard in Portal for processed `messagelds`. If a bridge adapter is buggy, misconfigured, or compromised and re-delivers (or allows re-delivery of) the same payload, `receiveMessage` will mint/unlock tokens, update the registry or re-execute fill reports again. Relying solely on adapters for replay protection creates a single point of failure.

Recommendation

Consider adding replay protection in the portal as well.

Resolution

M0 Team: The issue was resolved in commit [9ab37d8](#).

I-15 | Potential For Temp Stuck Messages With Hyperlane

Category	Severity	Location	Status
Informational	● Info	send_message.rs: 141	Acknowledged

Description

Right now you pay the IGP (interchain gas payments) as part of the transaction where you also dispatch the message with no direct easy way of either bumping the payment or paying another Relayer for the same message.

The trust assumptions of Hyperlane are that a Relayer could take your IGP funds and still not relay the message.

This should usually not be a problem unless the Relayer chosen by a user is dishonest which I'd say is not very likely. But, because on the SVM implementation the gas amount to send is globally set (in `hyperlane_global.igp_gas_amount`) it could be that it's simply not enough (unless you set it so you always overpay) and as such even an honest Relayer may not relay.

If a Relayer doesn't relay, the user always has the option of manually constructing a `PayForGas` transaction to bump the payment or pay another Relayer, but this would be a bit of a nuisance.

References:

- <https://docs.hyperlane.xyz/docs/protocol/core/interchain-gas-payment#trust-assumptions>
- https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcc81/programs/hyperlane-adapter/src/instructions/send_message.rs#L238

Recommendation

Make the gas amount for IGP dynamic and/or implement a way for the user to repay a dispatched message without having to manually figure it out.

Resolution

M0 Team: Acknowledged.

I-16 | The Name Solana-earners Is Misleading

Category	Severity	Location	Status
Best Practices	● Info	HubPortal.sol: 49	Acknowledged

Description

The SVM_EARNER_LIST constant in the HubPortal is set to solana-earners

```
bytes32 public constant SVM_EARNER_LIST = bytes32("solana-earners");
```

The earner list may be sent to other SVM chains as well, not only Solana. In this case, solana-earners will be misleading.

Recommendation

Consider changing the list name to svm-earners.

Resolution

M0 Team: Acknowledged.

I-17 | User Can Omit Overhead Even If It's Configured

Category	Severity	Location	Status
Informational	● Info	send_message.rs: 130	Acknowledged

Description

The `hyperlane_global.igp_overhead_account == Some(igp_overhead_account.key()) @ BridgeError::InvalidIgpAccount` constraint only runs when a user provides an `igp_overhead_account` regardless of if `hyperlane_global.igp_overhead_account` is set or not.

A user could still choose to not pass an account at all even if you want them to do so which gives them a risk of underpaying and the message not getting relayed.

The message would as such get temporarily stuck and the user would have to find another way to pay for it to be relayed.

Recommendation

Add a constraint that the `igp_overhead_account` must be provided when it's set in `hyperlane_global.igp_overhead_account`.

Resolution

M0 Team: Acknowledged.

I-18 | Unused Message_account.consumed Property

Category	Severity	Location	Status
Superfluous Code	● Info	receive_message.rs: 85	Acknowledged

Description

The message_account.consumed property is set to true when receiving a message, but it's never checked anywhere.

This still doesn't allow for replay because the message account uses init (https://github.com/GuardianOrg/solana-portal-team1-1764637060861/blob/d9bc5194d2ce78d50a80d411f6143e6034bcca81/programs/portal/src/instructions/receive_message.rs#L29) which would fail if someone tries to receive an already received message ID.

Recommendation

Unless message_account.consumed is somehow used off-chain, I believe it can be safely removed.

Resolution

M0 Team: Acknowledged.

I-19 | Nonce Inconsistency For Message Ids

Category	Severity	Location	Status
Best Practices	● Info	state.rs: 41-52	Acknowledged

Description

Each message sent by the portal is assigned an id, which is equal to the keccak256 hash of the originating chain, the destination chain and the nonce of the portal.

There is an inconsistency between EVM and SVM. On EVM, the nonce is first used, then incremented.

```
function _getMessageId(uint32 destinationChainId) internal returns (bytes32) {
    return keccak256(abi.encode(currentChainId, destinationChainId,
    _getPortalStorageLocation().nonce++));
}
```

While on SVM, it's first incremented and used afterwards

```
pub fn generate_message_id(&mut self, destination_chain_id: u32) -> [u8; 32] {
    self.message_nonce += 1;
    let mut encoded = [0u8; 96];
    // ABI encode: each value is padded to 32 bytes (left-padded for integers)
    encoded[28..32].copy_from_slice(&self.chain_id.to_be_bytes());
    encoded[60..64].copy_from_slice(&destination_chain_id.to_be_bytes());
    encoded[88..96].copy_from_slice(&self.message_nonce.to_be_bytes());
    keccak::hash(&encoded).to_bytes()
}
```

This will result in SVM messages having higher nonces than ones coming from EVM and may be unexpected for external integrators.

Recommendation

Consider sticking to the same approach on both VMs.

Resolution

M0 Team: Acknowledged.

I-20 | Orderbooks: Version Upgrade

Category	Severity	Location	Status
Best Practices	● Info	OrderBook.sol: 45-46	Resolved

Description

The contract implements a versioning system (VERSION constant at line 45) and validates order versions during fills (line 387):

```
uint16 public constant VERSION = 1;
// In _fillOrder
if (orderData_.version != VERSION) revert InvalidOrderVersion();
```

However, there is no mechanism to handle in-flight orders during a version upgrade, which could lead to denial of service or loss of assets if not handled gracefully.

Recommendation

Consider adding pause on new orders before version upgrade.

Let all old orders settle or cancel, and then upgrade.

Resolution

M0 Team: The issue was resolved in [PR#25](#).

I-21 | Cannot Close Order Accounts

Category	Severity	Location	Status
Gas Optimization	● Info	open.rs: 93	Acknowledged

Description

Right now you do not close order accounts but simply let them live in perpetuity even once successfully cancelled or filled.

At first sight, I believe closing them will be safe (no replay vector) because of the way the order ID is encoded (contains nonce, created_at, fill_deadline).

This should result in deleting ~257 Order bytes from the network which (at current SOL and rent prices) should be roughly ~\$0.27 (\$0.65 if you also close the order's ATA).

Not much for a single order, but a high volume user/solver could have an observable amount of money over time if they can get this back.

This has the added benefit of keeping the network and program state cleaner, potentially making indexing and off-chain account fetching easier/faster.

Recommendation

Implement account closing for orders once they are successfully cancelled or fully filled.

Resolution

M0 Team: Acknowledged.

I-22 | Lacking Token Validation For Filling Local Order

Category	Severity	Location	Status
Validation	● Info	fill.rs: 132-135	Resolved

Description

The `token_in_mint` account in the `FillNativeOrder` struct is not validated against the `token_in` of the `order_data`

```
#[account(
mint::token_program = token_in_program,
)]
pub token_in_mint: InterfaceAccount<'in fo, Mint>
```

Therefore, any token mint can be provided here. In result, the `solver_token_in_account` and `order_token_in_ata` may not be the tokens specified in the order. This should be generally safe, but if the order account holds other real tokens, they can be stolen.

Even if such tokens don't exist, malicious user can create a fake token and use it instead of the real one. This will lead to a loss for them because they receive fake token instead of the real one.

Recommendation

Validate that the provided `token_in_mint` is the right token account specified in the order.

Resolution

M0 Team: The issue was resolved in commit [f94d1c0](#).

I-23 | SVM Chain ID Assignment And EVM Collision Risk

Category	Severity	Location	Status
Configuration	● Info	Global	Acknowledged

Description

SVMs technically do not have native chain IDs like EVM chains do. In the M0 Portal cross-chain system, M0 must assign custom chain IDs to SVM chains for routing and peer identification purposes.

However, this creates a collision risk where a custom SVM chain ID assigned by M0 could conflict with a real EVM chain ID that M0 may want to support in the future.

Example: M0 assigns chain ID 888 to a new SVM chain. Later, Nasdaq launches an L2 with the official EVM chain ID 888. Now:

EVM contracts use `block.chainid` directly, which would return 888 on the Nasdaq L2

M0's system already has 888 mapped to the SVM chain in peer configurations

Ambiguity arises in cross-chain message routing and validation

Breaking changes would be required to remap the SVM chain to a different ID

Recommendation

Consider defining a high-value range for SVM chains that is extremely unlikely to conflict with EVM chain IDs and beware of possibility of this scenario.

Resolution

M0 Team: Acknowledged.

I-24 | Use Of The Wrong Multiplier

Category	Severity	Location	Status
Best Practices	● Info	receive_message.rs: 185	Acknowledged

Description

When getting the principal amount of \$M you use `new_multiplier` instead of what seems to be the better practice of using `ScaledUiAmountConfig::current_multiplier()`.

I don't see a strong reason to do this. I haven't unfortunately had the time to explore exploitability (hence marking it as an info now), but intuitively it could result in a stale multiplier (or even an empty 0 multiplier if `new_multiplier` has a clearing mechanism) which can be problematic.

Recommendation

Use `ScaledUiAmountConfig::current_multiplier()` instead of `new_multiplier`.

Resolution

M0 Team: Acknowledged.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	SVM Payload Is Malformed	Compatibility	● High	Resolved
M-01	Stuck Orders During Orderbook Upgrades	DoS	● Medium	Acknowledged
M-02	Delivery Of Signed Cancellations Can Be Griefed	Gaming	● Medium	Resolved
M-03	Isolated Spokes Can Receive Tokens From SVM	Validation	● Medium	Resolved
M-04	Cancel Race Condition Reintroduced On SVM	Validation	● Medium	Resolved
L-01	Lack Of Address(0) Check For Default Adapter	Validation	● Low	Resolved
L-02	Unscaled Amount Emitted In TokenSent	Events	● Low	Resolved
I-01	Index Can Be Updated Before Calling Orderbook	Best Practices	● Info	Resolved
I-02	Incomplete NatSpec	Best Practices	● Info	Resolved
I-03	A Note About FOT Tokens	Informational	● Info	Acknowledged

H-01 | SVM Payload Is Malformed

Category	Severity	Location	Status
Compatibility	● High	payloads.rs	Resolved

Description

The PayloadHeader on SVM doesn't include the index

```
#[derive(Debug, Clone)]
pub struct PayloadHeader {
    pub payload_type: u8,
    pub destination_chain_id: u32,
    pub destination_peer: [u8; 32],
    pub message_id: [u8; 32],
}
```

Instead, it treats the index as part of the message body. But as it can be seen from the EVM part, the index is encoded in the header.

This will lead to unexpected behavior for crosschain messages.

For example, decoding a token transfer on Solana will load wrong data from the payload, as the index is the first element, not the last.

This will most likely lead to a failure and the user tokens will be stuck on the EVM side. But if the data matches, there exists a possibility of exploiting the malformed payload and minting a large amount of tokens.

Recommendation

Fix the payload.rs file to decode the data correctly and update the index for all actions, like you do on the EVM side.

Resolution

M0 Team: The issue was resolved in [PR#38](#).

M-01 | Stuck Orders During Orderbook Upgrades

Category	Severity	Location	Status
DoS	● Medium	OrderBook.sol: 393	Acknowledged

Description

As described in [PR#25](#), the plan of action for OrderBook upgrades is to pause all the orderbooks, wait for inflight messages to be processed and execute the upgrade. Pausing the contracts ensures no new crosschain messages will be sent.

However, `_cancel()` and `fillOrder()` both implement the `whenNotPaused` modifier. This means existing orders can be neither filled nor cancelled, even if they expire.

After the upgrade, the order cannot be filled and if a major change in the cancelation logic has been made, the order may become stuck forever.

The same is true for the SVM part - there the cancel instructions are paused.

Recommendation

Remove the `whenNotPaused` modifier from `_cancel()` and the cancel instructions. After the orderbook is paused, give a grace period to users to cancel their orders. Execute the upgrade after this period.

Resolution

M0 Team: Acknowledged.

M-02 | Delivery Of Signed Cancellations Can Be Griefed

Category	Severity	Location	Status
Gaming	● Medium	OrderBook.sol: 773	Resolved

Description

The `OrderBook` allows users to sign cancel messages for their orders that can be executed by anyone providing the given signature. Besides the domain separator, the digest consists only of the `orderId` to be cancelled. This allows execution of `cancelOrderFor()` with arbitrary `bridgeAdapter_` and `bridgeAdapterArgs_`.

This enables griefing of the delivery of all signed requests once the wormhole adapter is enabled in the Portal. Since both of the above parameters are arbitrary, all cancellations can be routed through Wormhole. In `WormholeBridgeAdapter`, the `bridgeAdapterArgs_` are used as `signedQuote`.

Therefore, the user calling `cancelOrderFor` must pay only gas for execution on the current chain, as well as `coreBridgeFee` to the `coreBridge`, but as it can be seen on [Etherscan](#), this fee is currently disabled.

Any user can direct cancellations to Wormhole and send no `msg.value`, choose themselves as quoter or both. In result, the message will be successfully recorded and the order will be marked as cancelled, but it won't be automatically delivered to the source chain and the order will be stuck until someone executes it.

Recommendation

Partial validation where only the bridge adapter is signed by the user may be helpful.

Full validation is more complex and will likely cause increase in the price of the transaction, but it would look like this:

- if chosen adapter is hyperlane, no further validation is needed
- otherwise, validate `bridgeAdapterParams` with the following steps:
 1. Make sure the user signed the exact same `signedQuote`
 2. Fetch the `gasLimit` from the portal
 3. Use the data in the `signedQuote` to calculate how much `msg.value` has to be sent.

Resolution

M0 Team: The issue was resolved in [PR#44](#).

M-03 | Isolated Spokes Can Receive Tokens From SVM

Category	Severity	Location	Status
Validation	● Medium	Portal.sol	Resolved

Description

Additional check was added in response to [H-03](#). Each spoke tracks the isolated flag of every other spoke. The check ensures a spoke cannot send tokens to an isolated spoke.

However, this design is not present on the SVM side. SVM spokes can send tokens to any supported chain. Isolated spokes are protected by additional check in `handle_token_transfer_payload` that reverts if the transfer doesn't come from the Hub.

Because this check is not present on the EVM side, once an SVM becomes a non-isolated spoke, the limits of any other EVM isolated spoke can be exploited. For example:

- Users have bridged 10 000 tokens from Hub to the isolated OP portal.
- A malicious user bridges 10 000 tokens in total from Hub -> SVM -> OP
- This decreases the bridged principal of OP in the `HubPortal` to 0.
- M0 has to make OP non-isolated in order to allow users to bridge their tokens back

This attack can be repeated for any EVM spoke, nullifying the isolated spokes feature.

Recommendation

Revert in `_receiveToken()` on the EVM side if the current spoke is isolated.

If you implement this, you can get away without having each spoke tracking the state of the other spokes, which will also reduce complexity, because if a spoke status changes, you will have to update only two chains, not all of them.

One downside of this approach is that accidental transfers to an isolated spoke will be stuck until the spoke is toggled to a non-isolated.

Resolution

M0 Team: The issue was resolved in [PR#48](#).

M-04 | Cancel Race Condition Reintroduced On SVM

Category	Severity	Location	Status
Validation	● Medium	close_order_token_account.rs: 83	Resolved

Description

The cancel/fill race condition described in [M-03](#) was fixed by encoding the amount to be refunded in the `CancelReport` and allowing orders to be filled even if their status is Cancelled to let fill reports arriving later to be executed.

A new instruction - `close_order_token_account.rs`- was introduced to the orderbook program. It's permissionless and it closes the `order_token_in_ata` by sweeping any leftover tokens to the `order.sender` and refunding `order.payer` because he initialized the account.

This instruction can be executed if the order is either Completed or Cancelled. A malicious user can execute the same steps as in M-03 and immediately close the account once the cancel report arrives. They will receive the full amount back + what the solver gave them on the destination chain, while the expected fill report will later fail.

Recommendation

You can encode the `amountOut - amountOutFilled` to the `CancelReport`. Then in `close_account` you can infer if an order is really cancelled not by its status, but if `filled_amount_out + remaining_out_when_cancelled == order.amount_out`.

Resolution

M0 Team: The issue was resolved in [PR#51](#).

L-01 | Lack Of Address(0) Check For Default Adapter

Category	Severity	Location	Status
Validation	● Low	Portal.sol: 239-269	Resolved

Description

Portal.setDefaultBridgeAdapter() doesn't check if the passed bridgeAdapter is address(0). Because of this, it's technically possible to set address(0) as a default bridge adapter.

The problem is that setSupportedBridgeAdapter() reverts if the bridgeAdapter to be modified is address(0). In result, once set as a default adapter, address(0) will always stay supported.

Recommendation

Add the address(0) check in setDefaultBridgeAdapter() as well.

Resolution

M0 Team: The issue was resolved in [PR#50](#).

L-02 | Unscaled Amount Emitted In TokenSent

Category	Severity	Location	Status
Events	● Low	send_token.rs: 232	Resolved

Description

The principal `m_amount` is now scaled to normal token units when the payload for `send_token.rs` is constructed.

```
let scaled_m_amount = common::principal_to_amount_down(  
  m_amount,  
  common::get_scaled_ui_config(&ctx.accounts.m_mint.to_account_info())?  
    .multiplier  
    .into(),  
);
```

However, the `TokenSent` event is emitted with the principal value.

```
emit!(TokenSent {  
  ...  
  amount: m_amount as u128,  
  ...  
});
```

On the EVM side, this event is emitted with the actual value transferred.

Recommendation

Use `scaled_m_amount` for the event emission.

Resolution

M0 Team: The issue was resolved in [PR#39](#).

I-01 | Index Can Be Updated Before Calling Orderbook

Category	Severity	Location	Status
Best Practices	● Info	Portal.sol	Resolved

Description

Portal._receiveFillReport() and Portal_receiveCancelReport() first call the OrderBook and then update the M index. If the orderbook works with M tokens, wrong amounts may be transferred.

Recommendation

It's better to first update the index and then call the orderbook.

Resolution

M0 Team: The issue was resolved in [PR#51](#).

I-02 | Incomplete NatSpec

Category	Severity	Location	Status
Best Practices	● Info	OrderBook.sol: 67-68	Resolved

Description

The NatSpec of OrderBook.portal describes how the portal is responsible for sending fill reports, but doesn't mention that it also handles cancel reports.

```
/// @dev sends crosschain messages to report fills on this chain to other chains
///      receive crosschain messages to report fills on other chains to this chain
```

Recommendation

Include information about cancel reports as well.

Resolution

M0 Team: The issue was resolved in [PR#c4fd6833a82d509e5bf1b8096fbaa44c50e564de](#).

I-03 | A Note About FOT Tokens

Category	Severity	Location	Status
Informational	● Info	OrderBook	Acknowledged

Description

When cancel or fill is received, exact transfers are not used and the following comment says it's to not DOS bridges. This implies there can be tokens with fee on transfer feature that is being turned on/off. If the system works with these tokens, it may lead to unexpected behaviors, like incorrect event emissions, etc...

```
// We do not check exact amount received here to avoid DoS refund / bridge message
if amount_in_to_refund > 0 {
  transfer_tokens_from_program(
    &ctx.accounts.order_token_in_ata,
    &ctx.accounts.sender_token_in_ata,
    amount_in_to_refund,
    &ctx.accounts.token_in_mint,
    &ctx.accounts.order.to_account_info(),
    [&[
      ORDER_SEED_PREFIX,
      &cancel_report.order_id,
      &[ctx.accounts.order.bump],
    ]],
    &ctx.accounts.token_in_program,
  )?;
} else {
  return err!(OrderBookError::OrderFilled);
}
```

Recommendation

Be aware of these quirks of FOT on/off tokens.

Resolution

M0 Team: Acknowledged.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit <https://guardianaudits.com>

To view our audit portfolio, visit <https://github.com/guardianaudits>

To book an audit, message <https://t.me/guardianaudits>