CANTINA

Relay protocol

Security Review

Cantina Managed review by:
Sujith Somraaj, Security Researcher

April 22, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Relay is a cross-chain payments system that enables instant, low-cost bridging and cross-chain execution.

From Mar 12th to Mar 18th the Cantina team conducted a review of relay-protocol on commit hash 7cc3ecac. The team identified a total of **29** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	1	1	0
High Risk	1	1	0
Medium Risk	4	3	1
Low Risk	4	3	1
Gas Optimizations	1	0	1
Informational	18	16	2
Total	**29**	**24**	**5**

3 Findings

3.1 Critical Risk

3.1.1 Broken access control in `claim()` function of `BridgeProxy.sol`

Severity: Critical Risk

Context: BridgeProxy.sol#L67

Description: The `onlyRelayPool` modifier, which is intended to restrict access to the `claim()` function, contains a logical error that allows unauthorized accounts to drain funds from the contract. The vulnerability exists in the following code:

```
modifier onlyRelayPool() {
  if (msg.sender != RELAY_POOL && block.chainid != RELAY_POOL_CHAIN_ID) {
    revert NotAuthorized(msg.sender, block.chainid);
  }
  _;
}
```

This modifier uses `AND` (`&&`) logic where `OR` (`||`) logic was intended, creating a critical security flaw. The current implementation will only revert if BOTH of these conditions are true:

- The sender is not the `RELAY_POOL`, `AND`.
- The current chain `ID` is not the `RELAY_POOL_CHAIN_ID`.

Any arbitrary address can call functions with this modifier while operating on the chain with an `ID` matching `RELAY_POOL_CHAIN_ID`, completely bypassing the access control check.

Likelihood Explanation: This issue is very likely, as anyone can call the `claim()` function to drain the bridged funds before the relay pool utilizes them.

Impact Explanation: The impact is HIGH, as the issue allows stealing LP funds from the proxy contract.is very likely, as anyone can call the `claim()` function to drain the bridged funds before the relay pool utilizes them.

Recommendation: The modifier should be updated to use proper logical conditions:

```
modifier onlyRelayPool() {
  if (msg.sender != RELAY_POOL || block.chainid != RELAY_POOL_CHAIN_ID) {
    revert NotAuthorized(msg.sender, block.chainid);
  }
  _;
}
```

Relay Protocol: Fixed in PR 244.

Cantina Managed: Fix verified.

3.2 High Risk

3.2.1 Missing `receive()` function in Bridge Proxy contracts

Severity: High Risk

Context: *(No context files were provided by the reviewer)*

Description: The contracts `BridgeProxy.sol`, `ArbitrumOrbitNativeBridgeProxy.sol`, and `ZkSyncBridgeProxy.sol` enable the bridging of native tokens (ETH or native chain currency) between various blockchain networks. However, they are missing the crucial `receive()` or `fallback()` functions to accept incoming native token transfers. The affected contracts implement logic to bridge native tokens out from their chain (preferably an L2):

```
/// ArbitrumOrbitNativeBridgeProxy.sol
ARB_SYS.withdrawEth{value: amount}(L1_BRIDGE_PROXY);

/// ZkSyncBridgeProxy.sol
L2_BASE_TOKEN.withdraw{value: amount}(L1_BRIDGE_PROXY);
```

These functions correctly forward native tokens in the `bridge()` function. However, without a `receive()` or `fallback()` function, these contracts deployed on the L1 chain (Ethereum) cannot accept incoming native tokens, leading to transaction failures or, in worst-case scenarios, permanent loss of bridged funds.

Likelihood Explanation: The issue is very likely, as all the native tokens from Arbitrum and ZkSync are bridged through these contracts.

Impact Explanation: Permanent loss of bridged funds will affect LPs and has very `HIGH` impact on the protocol.

Recommendation: Add a `receive()` function to all bridge proxy contracts (or) only the `BridgeProxy.sol` contract if it is the one to be deployed as the receiver of all bridged funds on Ethereum.

Relay Protocol: Fixed in PR 245.

Cantina Managed: Fix verified.

3.3 Medium Risk

3.3.1 Implement slippage protection in RelayPool's `updateYieldPool` function

Severity: Medium Risk

Context: RelayPool.sol#L163

Description: The `updateYieldPool()` function in the `RelayPool` contract is vulnerable to slippage attacks during both the withdrawal from the old yield pool and the deposit into the new yield pool. No Slippage Protection on Withdrawal:

- The function is called `redeem()` on the old yield pool without specifying the minimum amount of assets to receive.

  If the withdrawal occurs during unfavorable market conditions or a front-running attack is manipulating the pool, the contract could receive significantly fewer assets than expected.

  ```
  /// accepts any amount of assets from the vault
  uint256 withdrawnAssets = ERC4626(yieldPool).redeem(
      sharesOfOldPool,
      address(this),
      address(this)
    );
  ```

- No Slippage Protection on Deposit:

  The `depositAssetsInYieldPool()` function similarly lacks slippage protection:

  ```
  function depositAssetsInYieldPool(uint256 amount) internal {
    ERC20(asset).approve(yieldPool, amount);
    /// accepts any amount of shares from the vault
    ERC4626(yieldPool).deposit(amount, address(this));
    emit AssetsDepositedIntoYieldPool(amount, yieldPool);
  }
  ```

Without minimum share parameters, the contract is vulnerable to sandwich attacks during the deposit operation.

Likelihood explanation:: The `updateYieldPool()` function is rarely used; even if used, the likelihood of MEVing it is minimal.

Impact explanation:: This is a `HIGH` impact issue as it will lead to losses for LPs during pool migration.

Recommendation: Implement slippage protection in the `updateYieldPool()` as follows:

```
function updateYieldPool(
    address newPool,
    uint256 minAssetsToWithdraw,
    uint256 minSharesToReceive
) public onlyOwner {
    address oldPool = yieldPool;
    uint256 sharesOfOldPool = ERC20(yieldPool).balanceOf(address(this));

    // Redeem all the shares from the old pool with minimum output check
    uint256 withdrawnAssets = ERC4626(yieldPool).redeem(
        sharesOfOldPool,
        address(this),
        address(this)
    );

    // Verify minimum assets received
    require(withdrawnAssets >= minAssetsToWithdraw, "SLIPPAGE_TOO_HIGH_ON_WITHDRAW");

    yieldPool = newPool;

    // Calculate expected shares
    uint256 expectedShares = ERC4626(newPool).previewDeposit(withdrawnAssets);

    // Deposit all assets into the new pool
    ERC20(asset).approve(newPool, withdrawnAssets);
    uint256 receivedShares = ERC4626(newPool).deposit(withdrawnAssets, address(this));

    // Verify minimum shares received
    require(receivedShares >= minSharesToReceive, "SLIPPAGE_TOO_HIGH_ON_DEPOSIT");

    emit YieldPoolChanged(oldPool, newPool);
}
```

Alternatively, if you plan to execute this function behind a timelock, ensure it is safeguarded with PPS (price-per-share).

Relay Protocol: Fixed in PR 228.

Cantina Managed: Fix verified.

3.3.2 Enforce minimum delay requirements for timelock in `RelayPoolFactory`

Severity: Medium Risk

Context: RelayPoolFactory.sol#L48

Description: The `RelayPoolFactory.sol` creates a timelock contract whenever a relay pool is launched through `deployPool()`. This contract serves as the owner of the relay pool (a.k.a curator). However, the absence of sanity checks on the `timelockDelay` parameter allows for the creation of relay pools with a timelock admin without delay. The timelock period is essential for safeguarding vault users against inflation attacks while enabling TVL growth.

```
function deployPool(
    ERC20 asset,
    string memory name,
    string memory symbol,
    address thirdPartyPool,
    uint timelockDelay, <--- can be zero
    uint256 initialDeposit
  ) public returns (address) {
    ....

    // clone timelock
    address timelock = Clones.clone(TIMELOCK_TEMPLATE);
    RelayPoolTimelock(timelock).initialize(
      timelockDelay, <--- will deploy a timelock with zero delay
      curator,
      curator,
      address(0) // No admin
    );

    // ...
  }
```

Recommendation: Consider bounding the `timelockDelay` value as follows:

```
+ uint256 public constant MIN_TIMELOCK_DELAY = 1 days;
+ uint256 public constant MAX_TIMELOCK_DELAY = 7 days;

  function deployPool(
      ERC20 asset,
      string memory name,
      string memory symbol,
      address thirdPartyPool,
      uint timelockDelay,
      uint256 initialDeposit
    ) public returns (address) {
      // ...

+     if(timelockDelay < MIN_TIMELOCK_DELAY || timelockDelay > MAX_TIMELOCK_DELAY) revert("INVALID TIMELOCK");
      // clone timelock
      address timelock = Clones.clone(TIMELOCK_TEMPLATE);
      RelayPoolTimelock(timelock).initialize(
        timelockDelay,
        curator,
        curator,
        address(0) // No admin
      );

      // ...
    }
```

Relay Protocol: Fixed in PR 230 (without a max).

Cantina Managed: Introducing a maximum limit would enhance the guardrails. However, it is currently the responsibility of the pool creator; establishing a pool with an extensive timelock solely impacts their capacity to modify the pool.

3.3.3 Lack of Slippage Protection in `RelayPoolNativeGateway`

Severity: Medium Risk

Context: RelayPoolNativeGateway.sol#L25, RelayPoolNativeGateway.sol#L42, RelayPoolNativeGateway.sol#L60, RelayPoolNativeGateway.sol#L88

Description: The `RelayPoolNativeGateway.sol` contract is a wrapper for interacting with RelayPool vaults that are ERC-4626 compliant and use native tokens.

However, the `RelayPoolNativeGateway.sol` wraps around existing ERC-4626 functions without slippage protections. The lack of slippage protection allows for front-running attacks where an attacker can manipulate the exchange rate just before a user's transaction is processed, leading to potential fund losses for the user.

Per the ERC-4626 security recommendations:.

> If implementors intend to support EOA account access directly, they should consider adding a function call for deposit/mint/withdraw/redeem with the means to accommodate slippage loss or unexpected deposit/withdrawal limits since they have no other means to revert the transaction if the exact output amount is not achieved.

Since `RelayPoolNativeGateway.sol` is designed to provide EOA users with ERC-4626 vaults using native ETH, it falls squarely within this recommendation.

Recommendation: Introduce a slippage parameter to the functions: `mint()`, `deposit()`, `redeem()`, and `withdraw()` to protect users from price manipulation.

```
function deposit(
  address pool,
  address receiver,
  uint256 minSharesOut  // Minimum shares to receive
) external payable returns (uint256) {
  // wrap tokens
  WETH.deposit{value: msg.value}();
  WETH.approve(pool, msg.value);

  // do the deposit
  uint256 shares = IERC4626(pool).deposit(msg.value, receiver);

  // Enforce slippage protection
  if (shares < minSharesOut) {
      revert SlippageExceeded();
  }

  return shares;
}
```

Relay Protocol: Fixed in PR 239.

Cantina Managed: Fix verified.

3.3.4 Privileged role and actions across bridging logic lead to centralization risks for users

Severity: Medium Risk

Context: RelayPool.sol#L178, RelayPool.sol#L192, RelayPool.sol#L538

Description: Several `onlyOwner` functions across `RelayPool.sol` contract logic affect critical protocol state and semantics, leading to user centralization risk. Some examples are highlighted below:

1. `addOrigin()` function can arbitrarily add any address as `proxyBridge`, thereby stealing LP funds through a malicious hyperlane message.
2. `addOrigin()` function can reset `outstandingDebt` to zero by re-adding the same origin.
3. `disableOrigin()` function allows censorship of any pending messages from Hyperlane during transit.
4. The `processFailedHandler()` function provides a failsafe from the abovementioned issue. Still, it is added with an `onlyOwner` modifier, which allows the owner to submit any data as a valid cross-chain message.
5. `transferOwnership()` could transfer the ownership of a relay pool to an EOA (or) a non-timelocked address.
6. `updateYieldPool()` allows the owner to set any arbitrary 4626 pools, which can also be malicious and steal LP funds.

Recommendation: Consider:

- Documenting the privileged role and actions for protocol user awareness.
- Privilege actions affecting critical protocol semantics should be locked behind timelocks so users can decide to exit or engage.
- Ensure curators follow the strictest opsec guidelines for privileged keys, e.g., use of reasonable multi-sig and hardware wallets.

Relay Protocol: Acknowledged.

Cantina Managed: Acknowledged.

3.4 Low Risk

3.4.1 Division by zero in `remainsToStream()` can temporarily brick core functions

Severity: Low Risk

Context: RelayPool.sol#L400-L409

Description: The `remainsToStream()` function in `RelayPool.sol` could encounter a division by zero if `block.timestamp` matches `lastAssetsCollectedAt`. This would trigger a revert within the `remainsToStream()` function, subsequently disrupting key functionalities such as `totalAssets()`, `redeem()`, and other operations that depend on these functions.

```
function remainsToStream() internal view returns (uint256) {
  if (block.timestamp > endOfStream) {
    return 0; // Nothing left to stream
  } else {
     return totalAssetsToStream - // total assets to stream
      (totalAssetsToStream * (block.timestamp - lastAssetsCollectedAt)) /
      (endOfStream - lastAssetsCollectedAt); // already streamed
  }
}
```

If `block.timestamp` equals `lastAssetsCollectedAt`, then the denominator (endOfStream - lastAssetsCollectedAt) will still have a non-zero value, but the division operation will use a numerator of 0 (since block.timestamp - lastAssetsCollectedAt would be 0). This is safe.

However, the true vulnerability occurs when `endOfStream` equals `lastAssetsCollectedAt`. In this case, the denominator (endOfStream - lastAssetsCollectedAt) would be zero, causing a division by zero error that would revert the entire transaction.

Since `remainsToStream()` is called by `totalAssets()`, a critical function used by almost all ERC4626 operations, including deposits, withdrawals, and share price calculations, this would render the contract unusable.

This issue occurs temporarily, and the contract usually functions when the `block.timestamp` moves.

Recommendation: Handle division with zero errors in all code paths to fix the temporary DoS issue.

Relay Protocol: Fixed in PR 231.

Cantina Managed: Fix verified.

3.4.2 Unhandled refund transaction failure in `bridge()` function

Severity: Low Risk

Context: RelayBridge.sol#L143

Description: The `bridge()` function in `RelayBridge.sol` contract attempts to refund excess ETH sent by users but silently ignores any failures in the refund process. The contract declares a boolean variable to capture the success status of the refund operation but never uses this value for any validation or recovery logic:

```
bool refundSuccess;
if (asset != address(0)) {
  if (msg.value > hyperlaneFee) {
    (refundSuccess, ) = msg.sender.call{value: msg.value - hyperlaneFee}(
      new bytes(0)
    );
  }
} else {
  if (msg.value > hyperlaneFee + amount) {
    (refundSuccess, ) = msg.sender.call{
      value: msg.value - hyperlaneFee - amount
    }(new bytes(0));
  }
}
```

The low-level call method sends ETH back to the original sender, a common pattern. However, the operation can fail in several scenarios:

- The receiving address is a contract that doesn't implement a receive() or fallback() function.
- The receiving contract's receive function reverts.

When the refund operation fails, the excess ETH remains trapped in the contract without notification to the user or recovery mechanism.

Recommendation: Consider validating the `refundSuccess` variable post refund as follows:

```
// ...
bool refundSuccess;
if (asset != address(0)) {
  if (msg.value > hyperlaneFee) {
    (refundSuccess, ) = msg.sender.call{value: msg.value - hyperlaneFee}(
      new bytes(0)
    );
  }
} else {
  if (msg.value > hyperlaneFee + amount) {
    (refundSuccess, ) = msg.sender.call{
      value: msg.value - hyperlaneFee - amount
    }(new bytes(0));
  }
}
if(!refundSuccess) revert("Error sending refund"); <-- validate the outcome of transaction
// ...
```

Or if failure is allowed, remove the `refundSuccess` variable.

Relay Protocol: Fixed in PR 250.

Cantina Managed: Fix verified.

3.4.3 Missing fee accounting in `processFailedHandler` function

Severity: Low Risk

Context: RelayPool.sol#L538

Description: The `processFailedHandler()` function in the `RelayPool.sol` contract is intended to recover funds from failed Hyperlane messages but does not correctly account for fees. When a typical message is processed through the `handle()` function, fees are calculated and tracked via `pendingBridgeFees`. However, the `processFailedHandler()` function sends the full amount to the recipient without accounting for or collecting fees.

This issue may lead to the following outcomes issues:

- The contract's accounting is inconsistent between normal message handling and failed message handling.
- Bridge fees that should be collected are being lost when using the `processFailedHandler()`.
- This creates an economic imbalance in the protocol, potentially incentivizing intentional message failures.

Recommendation: Update the `processFailedHandler()` function to follow the same fee accounting pattern as the `handle()` function:

- Calculate fees based on the origin's `bridgeFee` parameter.
- Update `pendingBridgeFees` accordingly.
- Either collect the fees or provide a clear comment explaining why fees are intentionally not collected.
- If fees are meant to be waived in this case, add this explanation as a code comment.

Relay Protocol: Fixed in PR 249.

The approach here is actually better I think because now we require the funds from the "failed" Hyperlane message to have been received by the pool. We now take accounting into account (outstanding debt is updated), but we still have a minor issue with the fees (or what would have been the fees) which will be streaming.

Cantina Managed: The problem is partially resolved since the `processFailedHandler()` function now correctly considers the debt. However, the fees are still not processed when this function is applied, resulting in a loss of these fees for the LPs.

3.4.4 Fee streaming mechanism enables value extraction attacks

Severity: Low Risk

Context: RelayPool.sol#L282

Description: The `RelayPool.sol` contract implements a fee streaming mechanism vulnerable to value extraction attacks. The contract implements a fee streaming system that distributes bridge fees over time (default 7 days) through the following mechanism:

1. Fees are collected in the `claim()` function:

```
uint256 feeAmount = (amount * origin.bridgeFee) / 10000;
pendingBridgeFees -= feeAmount;
addToStreamingAssets(feeAmount);
```

2. These fees are gradually recognized as part of the pool's assets through the `remainsToStream()` function:

```
function remainsToStream() internal view returns (uint256) {
  if (block.timestamp > endOfStream) {
    return 0; // Nothing left to stream
  } else {
    return totalAssetsToStream - // total assets to stream
      (totalAssetsToStream * (block.timestamp - lastAssetsCollectedAt)) /
      (endOfStream - lastAssetsCollectedAt); // already streamed
  }
}
```

3. The `totalAssets()` function subtracts unstreamed assets:

```
return
  yieldPoolBalance +
  outstandingDebt -
  pendingBridgeFees -
  remainsToStream();
```

The vulnerability occurs in the following scenario:

- Fees are collected, and begin streaming.
- During the streaming period, most or all LPs withdraw their funds.
- As time passes, more fees are "streamed in" (reducing remainsToStream()).
- A new depositor enters the pool when remainsToStream() is close to zero.
- This new depositor captures nearly all the streamed fees without having contributed to generating them.

In extreme cases, if all LPs withdraw during streaming and a new user deposits afterward, they can effectively capture 100% of the streamed fees despite contributing nothing to their generation. A malicious actor can time their deposits to capture streaming fees generated by previous liquidity providers, effectively stealing value from users who have already withdrawn their funds. Furthermore, this scenario returns a `totalAssets()` value greater than zero for a supply of zero.

Recommendation:

- Consider implementing a vesting period during which new deposits gradually gain access to streaming fees rather than having immediate full access.
- Or, when users withdraw, lock a portion of fees they're entitled to in an escrow that they can claim after the streaming period ends.

Relay Protocol: Acknowledged. I am not sure this is a "vulnerability" though, but just an artifact of using a "streaming" mechanism to account for the fees.

Cantina Managed: Acknowledged.

Relay Protocol: Acknowledged.

Cantina Managed: Acknowledged.

3.5 Gas Optimization

3.5.1 Refund logic in `bridge()` function could be optimized

Severity: Gas Optimization

Context: RelayBridge.sol#L143-L156

Description: The current refund logic in the `bridge()` function of `RelayBridge.sol` contract is complex and is gas expensive. However, the function can be simplified to save on gas fees for the users as the contract is stateless and is not expected to hold any funds.

Recommendation: Consider optimizing the function as follows:

```
function bridge(
    uint256 amount,
    address recipient,
    address l1Asset
  ) external payable returns (uint256 nonce) {
    // ...
    bool refundSuccess;
    if(address(this).balance > 0) {
      (refundSuccess, ) = msg.sender.call{value: address(this).balance}(
        new bytes(0)
      );
    }
    // ...
}
```

Relay Protocol: Acknowledged.

Cantina Managed: Acknowledged.

3.6 Informational

3.6.1 Remove unused imports

Severity: Informational

Context: ArbitrumOrbitNativeBridgeProxy.sol#L10, OPStackNativeBridgeProxy.sol#L6-L7, ZkSyncBridgeProxy.sol#L5, RelayPoolFactory.sol#L7, TokenSwap.sol#L7

Description: Multiple files under the audit scope contain import statements for files and libraries not used within the implementation. While this does not directly impact the security or functionality of the contract, it adds unnecessary complexity, potentially increases deployment gas costs, and may confuse developers during maintenance.

Recommendation: Consider removing unused file imports to improve code quality.

Relay Protocol: Fixed in PR 229.

Cantina Managed: Fix verified.

3.6.2 Use SafeERC20 library to handle ERC20 approvals & transfer

Severity: Informational

Context: BridgeProxy.sol#L61, RelayBridge.sol#L102, RelayPoolFactory.sol#L91-L92, RelayPool.sol#L496, TokenSwap.sol#L104, TokenSwap.sol#L145

Description: The `RelayBridge.sol` contract uses the `IERC20` interface for token transfers without implementing OpenZeppelin's **SafeERC20** library. This creates potential issues when interacting with non-standard ERC20 token implementations. One such example is the `RelayBridge.sol` contract is defined as follows:

```
import {IERC20} from "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// In the bridge() function:
IERC20(asset).transferFrom(msg.sender, address(this), amount);
```

Recommendation: Consider using OpenZeppelin's `SafeERC20` library.

Relay Protocol: Fixed in PR 233.

Cantina Managed: Fix verified.

3.6.3 Minor code quality issues: typos

Severity: Informational

Context: CCTPBridgeProxy.sol#L18, RelayBridge.sol#L58, RelayPoolNativeGateway.sol#L56, RelayPoolNativeGateway.sol#L56-L57, RelayPoolNativeGateway.sol#L57, RelayPoolNativeGateway.sol#L85, RelayPool.sol#L418, RelayPool.sol#L421

Description: The codebase contains minor typos and naming inconsistencies that should be addressed for better code quality and maintainability.

Recommendation: Consider fixing the typos identified.

Relay Protocol: Fixed in PR 232.

Cantina Managed: Fix verified.

3.6.4 Remove unused `SafeCast160` library

Severity: Informational

Context: TokenSwap.sol#L11-L20, TokenSwap.sol#L28

Description: The `SafeCast160` library features a casting function designed to convert uint256 to uint160 values, implemented in the `TokenSwap.sol` contract. However, this file is not used within the contract. Importing and declaring libraries that are not utilized can increase the size of the code and negatively affect code maintainability and quality.

Recommendation: Consider removing the `SafeCast160` library to improve code quality.

Relay Protocol: Fixed in PR 237.

Cantina Managed: Fix verified.

3.6.5 Redundant token approval before swap

Severity: Informational

Context: TokenSwap.sol#L96-L101

Description: The `TokenSwap.sol` contract exhibits a flawed token handling approach when working with Uniswap's Universal Router. Within the `swap()` function, the TokenSwap contract approves the Universal Router and also sends the tokens to it without properly utilizing this approval. Consequently, the `TokenSwap.sol` contract inadvertently retains residual approval granted to the Universal Router.

```
// Approve the router to spend src ERC20
TransferHelper.safeApprove(
  tokenAddress,
  UNISWAP_UNIVERSAL_ROUTER,
  tokenAmount
);

// send tokens to universal router to manipulate the token
IERC20(tokenAddress).transfer(UNISWAP_UNIVERSAL_ROUTER, tokenAmount);
```

Proof of Concept: The following proof of concept demonstrates the following:

```
function test_tokenSwap() external {
    vm.warp(block.timestamp + 100 seconds);
    vm.startPrank(address(ourTimelock));
    relayPool.setTokenSwap(address(tokenSwap));

    deal(
        address(0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984),
        address(relayPool),
        100e18
    );
    relayPool.swapAndDeposit(
        address(0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984),
        100e18,
        3000,
        0,
        uint48(block.timestamp),
        0
    );
    console.log("approval of swap contract",
    ↪   ERC20(0x1f9840a85d5aF5bf1D1762F925BDADdC4201F984).allowance(address(tokenSwap),
    ↪   address(0x66a9893cC07D91D95644AEDD05D03f95e1dBA8Af)));
}
```

```
approval of swap contract 100000000000000000000
```

Recommendation: Remove token approval; the Universal Router uses transferred tokens, not the approved tokens. Also, remove the TransferHelper import, which will be unused after removing token approval.

Relay Protocol: Fixed in PR 237.

Cantina Managed: Fix verified.

3.6.6 Incomplete function documentation in `swap()` function

Severity: Informational

Context: TokenSwap.sol#L69

Description: The documentation for the `swap()` function in the `TokenSwap.sol` contract contains misleading information that does not align with the actual functionality implemented in the code. Specifically, the function comment states:

```
/**
 * Swap tokens to UDT and burn the tokens
 *
 * @notice The default route is token > WETH > asset.
 * If `uniswapWethPoolFeeAsset` is set to null, then we do a direct swap token > asset
 * @param deadline The deadline for the swap transaction
 * @param amountOutMinimum The minimum amount of output tokens that must be received for the transaction not to
 ↪  revert
 */
```

The comment claims that the function will "swap tokens to UDT and burn the tokens." However, there is no burning functionality present in the code. The function simply:

- Swaps input tokens to the asset tokens using Uniswap.
- Transfers the resulting asset tokens back to the pool.
- Emits a TokenSwapped event.

Recommendation: Update the function documentation to accurately reflect the actual functionality.

Relay Protocol: Fixed in PR 237.

Cantina Managed: Fix verified.

3.6.7 Minor code quality issues: naming convention

Severity: Informational

Context: RelayPool.sol#L207-L210, RelayPool.sol#L224-L227, RelayPool.sol#L305, RelayPool.sol#L314, RelayPool.sol#L400, RelayPool.sol#L419, RelayPool.sol#L462, RelayPool.sol#L517, RelayPool.sol#L525, TokenSwap.sol#L64-L66

Description: The files under scope do not consistently follow Solidity naming conventions for internal functions. Per Solidity style guides, internal functions should typically be prefixed with an underscore (_) to distinguish them clearly from public or external functions. For example:

```
function getBalance(address token) internal view returns (uint256) {
  return IERC20(token).balanceOf(address(this));
}
```

Recommendation: Ensure all internal functions follow the Solidity naming convention by adding an underscore prefix:

```
function _getBalance(address token) internal view returns (uint256) {
  return IERC20(token).balanceOf(address(this));
}
```

Relay Protocol: Fixed in PR 259 and PR 237. `beforeWithdraw()` and `afterDeposit()` are `internal` but used by solmate's ERC4626 implementation and can't be renamed.

Cantina Managed: Verified fix.

3.6.8 Add `tokenOut` parameter to `TokenSwapped` event

Severity: Informational

Context: TokenSwap.sol#L37-L42

Description: The `TokenSwapped` event emitted at the end of the swap function does not include complete information about the swap operation. Specifically, it fails to include the output token address (asset), which is crucial for off-chain tracking of swap operations. Consequently, this value must be retrieved from the pool address, potentially resulting in increased memory usage in the off-chain tracking infrastructure due to additional RPC calls. Current event definition:

```
event TokenSwapped(
  address pool,
  address tokenIn,
  uint256 amountIn,
  uint256 amountOut
);
```

Recommendation: Update the event definition to include the output token address (asset):

```
event TokenSwapped(
  address pool,
  address tokenIn,
  address tokenOut,
  uint256 amountIn,
  uint256 amountOut
);

/// in swap() function
emit TokenSwapped(pool, tokenAddress, asset, tokenAmount, amountOut);
```

Relay Protocol: Fixed in PR 237.

Cantina Managed: Fix verified.

3.6.9 Variable `hyperlaneMailbox` could be made `immutable`

Severity: Informational

Context: RelayBridgeFactory.sol#L8

Description: In the `RelayBridgeFactory.sol` contract, the `hyperlaneMailbox` is defined as a standard state variable instead of using the `immutable` keyword, despite being initialized only in the constructor.

```
address public hyperlaneMailbox;
```

Recommendation: Update the variable declaration to use the immutable keyword:

```
address public immutable hyperlaneMailbox;
```

Relay Protocol: Fixed in PR 238.

Cantina Managed: Fix verified.

3.6.10 Variables `asset`, `bridgeProxy` and `hyperlaneMailbox` could be made `immutable`

Severity: Informational

Context: RelayBridge.sol#L24-L26

Description: The `RelayBridge.sol` contract sets multiple variables in the constructor but does not declare them as `immutable`, even though they are never intended to change after initialization:

```
address public asset;
BridgeProxy public bridgeProxy;
address public hyperlaneMailbox;

constructor(
  address _asset,
  BridgeProxy _bridgeProxy,
  address _hyperlaneMailbox
) {
  asset = _asset;
  bridgeProxy = _bridgeProxy;
  hyperlaneMailbox = _hyperlaneMailbox;
}
```

Recommendation: Declare the variables mentioned above as `immutable` to ensure they cannot be changed after contract deployment:

```
address public immutable asset;
BridgeProxy public immutable bridgeProxy;
address public immutable hyperlaneMailbox;
```

Relay Protocol: Fixed in PR 248.

Cantina Managed: Fix verified.

3.6.11 Hardcoded values should be replaced with named constants

Severity: Informational

Context: CCTPBridgeProxy.sol#L53, OPStackNativeBridgeProxy.sol#L34, OPStackNativeBridgeProxy.sol#L50, RelayBridge.sol#L100, RelayPool.sol#L366

Description: The files under audit scope contain hardcoded values that should be replaced with named constants for better readability and maintainability and to prevent potential errors during future modifications.

Recommendation: Consider replacing the hardcoded values with named constants as follows:

```
uint32 public constant ETH_DST_DOMAIN = 0;

MESSENGER.depositForBurn(
    amount,
    ETH_DST_DOMAIN, // mainnet domain is zero
    targetAddressBytes32,
    USDC
);
```

Relay Protocol: Fixed (partially) in PR 259. We are not sure `address(0)` and `10000` are better constants.

Cantina Managed: Verified the partial fix. Yes, `10000` as max fee value constant will serve some information about the hardcoded value.

3.6.12 Remove unused `OUTBOX` declarations in `ArbitrumOrbitNativeBridgeProxy`

Severity: Informational

Context: ArbitrumOrbitNativeBridgeProxy.sol#L9, ArbitrumOrbitNativeBridgeProxy.sol#L21, ArbitrumOrbitNativeBridgeProxy.sol#L29, ArbitrumOrbitNativeBridgeProxy.sol#L35

Description: The `ArbitrumOrbitNativeBridgeProxy.sol` contract includes declarations and imports for the `OUTBOX` contract; however, it remains unused, which may negatively impact code quality and readability.

Recommendation: Remove unused `OUTBOX` declarations and imports.

Relay Protocol: Fixed in PR 242.

Cantina Managed: Fix verified.

3.6.13 Remove unused `TRANSMITTER` declarations in `CCTPBridgeProxy`

Severity: Informational

Context: CCTPBridgeProxy.sol#L5, CCTPBridgeProxy.sol#L13, CCTPBridgeProxy.sol#L25, CCTPBridgeProxy.sol#L32

Description: The `CCTPBridgeProxy.sol` contract includes declarations and imports for the TRANSMITTER contract; however, it remains unused, which may negatively impact code quality and readability.

Recommendation: Remove unused TRANSMITTER declarations and imports.

Relay Protocol: Fixed in PR 242.

Cantina Managed: Fix verified.

3.6.14 Removed unused `PORTAL_PROXY` declarations in `OPStackNativeBridgeProxy`

Severity: Informational

Context: OPStackNativeBridgeProxy.sol#L14, OPStackNativeBridgeProxy.sol#L17, OPStackNativeBridge-Proxy.sol#L22

Description: The `OPStackNativeBridgeProxy.sol` contract includes declarations and imports for the PORTAL_PROXY contract; however, it remains unused, which may negatively impact code quality and readability.

Recommendation: Remove unused PORTAL_PROXY declarations and imports.

Relay Protocol: Fixed in PR 242.

Cantina Managed: Fix verified.

3.6.15 Misleading comment about fund availability during L2 failures

Severity: Informational

Context: RelayBridge.sol#L99

Description: The comment in the `bridge()` function stating, "If the L2 is halted/reforged, the funds will remain in this contract" is misleading. It incorrectly suggests that funds are safe during L2 network failures, but the current implementation does not ensure this.

```
// Get the funds. If the L2 is halted/reorged, the funds will remain in this contract
```

Recommendation: Consider removing the inaccurate comment.

Relay Protocol: Fixed in PR 259.

Cantina Managed: Fix verified.

3.6.16 `STANDARD_BRIDGE` missing token approval is incompatible with non-`OptimismMintableERC20` tokens

Severity: Informational

Context: OPStackNativeBridgeProxy.sol#L45

Description: The `OPStackNativeBridgeProxy.sol` contract attempts to bridge ERC20 tokens without first approving the `STANDARD_BRIDGE` to spend tokens on its behalf.

When handling non-ETH tokens in the bridge function, the contract calls `bridgeERC20To()` on the `STANDARD_BRIDGE`, which expects token approvals for non-OptimismMintableERC20 tokens.

Hence, all attempts to bridge non-OptimismMintable ERC20 tokens will fail due to the missing approval.

Recommendation: Implement the token approval before calling `bridgeERC20To()` if the protocol intends to support non-`OptimismMintableERC20` tokens:

```
IERC20(currency).approve(STANDARD_BRIDGE, amount);
L2StandardBridge(STANDARD_BRIDGE).bridgeERC20To(
  currency,
  l1Token,
  L1_BRIDGE_PROXY,
  amount,
  200000,
  data
);
```

Relay Protocol: Acknowledged.

Cantina Managed: Acknowledged.

3.6.17 Lack of `l1Asset` parameter validation in `bridge()` function

Severity: Informational

Context: CCTPBridgeProxy.sol#L38, OPStackNativeBridgeProxy.sol#L27, ZkSyncBridgeProxy.sol#L26, RelayBridge.sol#L137

Description: The `bridge()` function takes an `l1Asset` parameter and sends it to the inherited `proxyBridge` for further validation. There is an inconsistency because this parameter is validated only in the `ArbitrumOrbitNativeBridgeProxy.sol`, while other proxy bridges like `CCTPBridgeProxy`, `OPStackNativeBridgeProxy`, and `ZkSyncBridgeProxy` do not validate it and disregard it. Nevertheless, the `l1Asset` parameter is still included in the event emission in the calling contract (RelayBridge), which may result in inconsistent event data.

```
// In OPStackNativeBridgeProxy
function bridge(
  address currency,
  address /* l1Asset */, // Parameter is ignored but commented out
  uint256 amount,
  bytes calldata data
) external payable override {
  // ...
  address l1Token = IOptimismMintableERC20(currency).remoteToken();
  // l1Asset parameter is ignored in favor of l1Token
  // ...
}

// In RelayBridge
event BridgeInitiated(
  uint256 indexed nonce,
  address indexed sender,
  address recipient,
  address asset,
  address l1Asset, // This uses the ignored parameter
  uint256 amount,
  BridgeProxy bridgeProxy
);
```

Recommendation: Consider validating the `l1Asset` parameter in the other proxy bridges to be consistent, if not possible avoid depending on the parameter for off-chain validation.

Relay Protocol: Indeed, each bridge behaves differently. Sometimes, the native bridge expects us to submit the address of the L1 asset, and other times, it does not. Yet we want an identical signature for all calls, which is why we include it, even if it is sometimes ignored.

Fixed for `OPStackNativeBridgeProxy.sol` in PR 258.

Cantina Managed: The `l1Asset` is now validated for `OPStackNativeBridgeProxy.sol`, and the fix has been verified. However, the issue still remains in other bridge proxy contracts (`CCTPBridgeProxy` and `ZkSyncBridgeProxy`).

3.6.18 Increase test coverage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Key contracts under review lack complete test coverage. Adequate testing and regular reporting are essential to ensure the codebase functions properly. Insufficient coverage can result in unexpected issues and regressions from changes in the underlying smart contract.

Recommendation: Add to test coverage, ensuring all execution paths are covered. See also Paul R Berg's BTT thread and talk.

Relay Protocol: Acknowledged.

Cantina Managed: Fix Acknowledged.