CANTINA

LI.FI Contracts Commit c9e624

Security Review

Cantina Managed review by:

Kankodu, Security Researcher
Víctor Martínez, Security Researcher

January 10, 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

LI.FI is a cross-chain bridge aggregation protocol that supports any-2-any swaps by aggregating bridges and connecting them to DEX aggregators.

From Dec 19th to Dec 22nd the Cantina team conducted a review of contracts on commit hash c9e62461. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	1	0	1
Gas Optimizations	1	1	0
Informational	5	5	0
Total	**8**	**7**	**1**

3 Findings

3.1 Medium Risk

3.1.1 Griefing attack possible by frontrunning the `callDiamondWithEIP2612Signature` function call

Severity: Medium Risk

Context: Permit2Proxy.sol#L75-L93

Description: The `callDiamondWithEIP2612Signature` function performs two distinct actions:

1. Calls `ERC20.permit` to submit the signature provided by `msg.sender`, ensuring the contract has sufficient allowance to pull funds from the user.
2. Pull the funds from msg.sender to execute `diamondCalldata` on a specific diamond.

However, there is a potential frontrunning attack where an attacker can ensure that the `ERC20.permit` call fails, causing the entire transaction to revert.

1. A user signs a permit and sends a `callDiamondWithEIP2612Signature` transaction to the network.
2. An attacker frontruns this transaction and sends their own transaction to directly call the `ERC20.permit` function, using the permit information copied from the user's transaction:
    - The attacker's transaction succeeds, setting the allowance and marking the permit signature as used.
3. When the user's transaction is eventually included, it fails:
    - The `ERC20.permit` call made by the `Permit2Proxy` fails because the signature has already been used. This causes the entire transaction to fail.

While the attacker cannot profit from this attack, they can prevent users from successfully executing `callDiamondWithEIP2612Signature`.

Proof of Concept: Add the test below in `test/solidity/Periphery/Permit2Proxy.t.sol`

```
```solidity
function test_permit_frontrun_attack()
    public
    assertBalanceChange(ADDRESS_USDC, PERMIT2_USER, 0)
    returns (TestDataEIP2612 memory)
{
    vm.startPrank(PERMIT2_USER);

    bytes32 domainSeparator = ERC20Permit(ADDRESS_USDC).DOMAIN_SEPARATOR();

    TestDataEIP2612 memory testdata =
        _getTestDataEIP2612SignedByPERMIT2_USER(ADDRESS_USDC, domainSeparator, block.timestamp + 1000);

    //user calls Permit2Proxy with signature
    //when the user call is in transit, the attacker copies the permit data and calls the permit function
    ↪  directly on the ERC20 contract
    // permit2Proxy.callDiamondWithEIP2612Signature(
    //     ADDRESS_USDC,
    //     defaultUSDCAmount,
    //     testdata.deadline,
    //     testdata.v,
    //     testdata.r,
    //     testdata.s,
    //     testdata.diamondCalldata
    // );
    vm.stopPrank();

    vm.startPrank(address(0xA));
    //attacker calls ERC20.permit directly
    ERC20Permit(ADDRESS_USDC).permit(
        PERMIT2_USER, //victim address
        address(permit2Proxy),
        defaultUSDCAmount,
        testdata.deadline,
        testdata.v,
        testdata.r,
        testdata.s
```
```

```
    );
    vm.stopPrank();

    vm.startPrank(PERMIT2_USER);

    vm.expectRevert("EIP2612: invalid signature");
    //user's transaction finally gets included and it fails because the signature has already been used
    permit2Proxy.callDiamondWithEIP2612Signature(
        ADDRESS_USDC,
        defaultUSDCAmount,
        testdata.deadline,
        testdata.v,
        testdata.r,
        testdata.s,
        testdata.diamondCalldata
    );
    vm.stopPrank();
    return testdata;
}
```

Recommendation: Place the `ERC20.permit` call inside a `try` block. in the case it fails and allowance is enough, continue the execution.

LI.FI Fixed in commit bdf16c01.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Missing minimum and maximum value checks for Gas.zip deposits

Severity: Low Risk

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

Description: As per the Gas.zip docs it is recommended to limit the minimum and maximum amount deposited into the router: "Do not send less than $0.25 USD or more than $50.00 USD PER CHAIN to the deposit address". However, neither `GasZipFacet` nor `GasZipPeriphery` enforce these checks.

Recommendation: Consider either enforcing these checks at contract or at frontend level to follow Gas.zip recommendations.

LI.FI We already enforce this from our backend.

Cantina Managed: Acknowledged.

3.3 Gas Optimization

3.3.1 Reduce unnecessary approval each time

Severity: Gas Optimization

Context: GasZipPeriphery.sol#L59-L63

Description: If `LibAsset.maxApproveERC20` is called with `amount = type(uint256).max` and a token's approval logic does not treat an allowance of `type(uint256).max` as infinite, the allowance will be reduced to less than `type(uint256).max` after each transaction. This would require re-approval for every subsequent transaction.

Recommendation: Instead of using `type(uint256).max`, pass `_swapData.fromAmount` to avoid the need for repeated approvals for every transaction.

LI.FI Fixed in commit 32bdfa12.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Use functions provided in standard libraries for `safeApprove`

Severity: Informational

Context: LibAsset.sol#L70-L71, ReceiverAcrossV3.sol#L108-L109

Description: The `Solady.SafeTransferLib` library provides a safeApproveWithRetry function that calls `approve` on a token with the desired amount. If the call fails due to a check that prevents setting a non-zero amount when the current allowance is also non-zero, it first resets the approval to zero and then calls `approve` with the desired amount.

This optimization is useful because most tokens allow allowances to be overridden directly.

However, in `ReceiverAcrossV3._swapAndCompleteBridgeTokens`, this functionality is not being utilized. Instead, the approval is reset to zero every time, which is unnecessary. The standard function provided by the library should be used.

Similarly, in `LibAsset.maxApproveERC20`, the approval logic could be improved. `SafeERC20` provides a `forceApprove` function for this exact purpose, and it should be adopted.

Recommendation: Use the functions provided by standard libraries as recommended to simplify and optimize the approval logic.

LI.FI Fixed in commit 80c68b14.

Cantina Managed: Fix verified.

3.4.2 Remove unused inherited contracts

Severity: Informational

Context: GasZipPeriphery.sol#L20-L26

Description: In `GasZipPeriphery`, `ILiFi`, `SwapperV2`, `ReentrancyGuard` and `Validatable` are inherited but never used. `ReentrancyGuard` adds additional storage, increasing the contract deployment cost. Similarly, the logic in `Validatable` and `SwapperV2` is not utilized.

Recommendation: Consider removing the inherited contracts mentioned above if they are unnecessary, or use them where required.

LI.FI Fixed in commit 80c68b14.

Cantina Managed: Fix verified.

3.4.3 Avoid duplicated code whenever possible to reduce inconsistencies

Severity: Informational

Context: GasZipPeriphery.sol#L100-L103, ReceiverAcrossV3.sol#L78-L82

Description:

- `ReceiverAcrossV3.pullToken`:

  The `pullToken` function replicates the functionality of `WithdrawablePeriphery.withdrawToken`. To avoid code duplication, this contract should inherit `WithdrawablePeriphery`.

  This also helps avoid discrepancies, such as `WithdrawablePeriphery.withdrawToken` emitting the `TokensWithdrawn` event, while `pullToken` does not.

- `GasZipPeriphery.depositToGasZipNative`:

```
uint256 remainingNativeBalance = address(this).balance;
if (remainingNativeBalance > 0) {
    msg.sender.safeTransferETH(remainingNativeBalance);
}
```

  This logic can be replaced by the `SwapperV2.refundExcessNative` modifier to avoid duplicating the code for refunding native ETH.

Recommendation: Remove duplicated code as suggested to improve maintainability and reduce potential inconsistencies.

LI.FI Withdrawable Periphery was added in PR 909.

Cantina Managed: Fix verified.

3.4.4 Inconsistent variable naming

Severity: Informational

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

Description:

- `DeBridgeDlnFacet` and `GasZipFacet` contract use a custom value to identify non evm addresses `0x11f111f111f111F111f111f111F111f111f111F1`. However the naming is not consistent, `NON_EVM_ADDRESS` in `DeBridgeDlnFacet` while `NON_EVM_RECEIVER_IDENTIFIER` in `GasZipFacet`.
- In function `getDeBridgeChainId` the variable name `chainId` is misleading, as it actually represents the DeBridge chain ID.

Recommendation:

- Consider changing `GasZipFacet` `NON_EVM_RECEIVER_IDENTIFIER` variable name to `NON_EVM_ADDRESS` to maintain consistency with the rest of the facets.
- Rename `chainId` to `_deBridgeChainId` or something more descriptive.

LI.FI Fixed in commit 80c68b14.

Cantina Managed: Fix verified.

3.4.5 Suggestions to improve Readability

Severity: Informational

Context: GasZipFacet.sol#L133, GasZipPeriphery.sol#L113, Permit2Proxy.sol#L204

Description:

- Use Named Constants:

  In `GasZipFacet.getDestinationChainsValue`, if `chainIds.length` exceeds 32, the function fails with a `TooManyChainIds` error. Readability would be improved by using a named constant, such as `MAX_CHAINID_LENGTH_ALLOWED`, instead of the raw value `32`.

  The same suggestion applies to `GasZipPeriphery.getDestinationChainsValue`.

- Use Proper Variable Names:

  In `Permit2Proxy.getPermit2MsgHash`, the result of the `_getTokenPermissionsHash` function is named `permit`. This is misleading, as `permit` is commonly used for variables of type `ISignatureTransfer.PermitTransferFrom`.

  The variable should be renamed to `tokenPermissionsHash` to accurately reflect its purpose and avoid unnecessary confusion.

Recommendation: Implement the suggested changes to improve readability and maintainability.

LI.FI Fixed in commit ac43bb2e.

Cantina Managed: Fix verified.