// Smart Contract Security Assessment

11.26.2024 - 11.29.2024

DLN-EVM (Upgrades)

deBridge

HALBORN

Report a Bug

DLN-EVM (Upgrades) - deBridge

Prepared by: HALBORN Last Updated 12/17/2024 Date of Engagement by: November 26th, 2024 - November 29th, 2024

Summary

100% ⓘ OF ALL REPORTED FINDINGS HAVE BEEN ADDRESSED

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

1. Introduction

deBridge engaged Halborn to conduct a security assessment on their smart contracts beginning on November 26th, 2024, and ending on November 29th, 2024. The security assessment was scoped to the smart contracts provided in the debridge-finance/dln-evm GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

TABLE OF CONTENTS

1. Introduction
2. Assessment summary
3. Test approach and methodology
4. Upgradeability assessment
5. Risk methodology
6. Scope
7. Assessment summary & findings overview
8. Findings & Tech Details

8.1 Incompatibility of externalcallexecutor contract with transparent proxy upgradeability
9. Review Notes
10. Automated Testing

2. Assessment Summary

Halborn was provided **3 days** for the engagement, and assigned one full-time security engineer to review the security of the smart contracts in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues regarding **exclusively** the upgradeability of the proposed smart contracts, including storage conflicts and compatibility with the EVM version of multiple targeted networks, as listed below:
  - Arbitrum
  - Avalanche
  - Base
  - BSC
  - Ethereum
  - Fantom
  - Linea
  - Optimism
  - Polygon

- Ensure that smart contract functionality operates as intended and the new implementation is adherent to previous business rules.

In summary, Halborn identified an improvement to reduce the likelihood and impact of risks, which was addressed by the deBridge team:

- Apply the Transparent Upgradeable Proxy pattern to the ExternalCallExecutor and ExternalCallExecutorBase contracts, in order to enhance upgradeability capabilities of the system as a whole. Alternatively, implement setter functions in contracts that are interacting with the ExternalCallExecutor.

3. Test Approach And Methodology

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose.
- Smart contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions (solgraph).
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Manual testing by custom scripts.
- Static Analysis of security for scoped contract, and imported functions (slither).
- Testnet deployment (Foundry).

4. Upgradeability Assessment

We confirm that you can safely proceed with updating the implementations for the following contracts across the networks specified above.

- **Implementation Updates**

The updates for `DlnSource`, `DlnDestination`, and `DlnExternalCallAdapter` will be performed using the following method in the `Proxy Admin` contract:

```
function upgrade(TransparentUpgradeableProxy proxy, address implementat
```

- **Proxy Admin Contract Address:** `0xa7b88a746fa457578d5abd6234471f07d895f46b`.

The proxy and implementation addresses for each contract are as follows:

1. DlnSource

- **Proxy Address:** `0xeF4fB24aD0916217251F553c0596F8Edc630EB66`
- **New Implementation Address:** `0xAFcFdA8DF6d6Ce56a69ccA70723791667C686892`

2. DlnDestination

- **Proxy Address:** `0xE7351Fd770A37282b91D153Ee690B63579D6dd7f`

- **New Implementation Address:** `0x4a811205Af19A5aD9aA16D6E4C89863dFce22180`

3. DlnExternalCallAdapter

- **Proxy Address:** `0x61eF2e01E603aEB5Cd96F9eC9AE76cc6A68f6cF9`
- **New Implementation Address:** `0xcB8E9e3CdD538559f6dA9598a4DaC4ac706970bf`

The `ExternalCallExecutor` is updated by invoking the following method in the **DlnExternalCallAdapter** contract:

```
function updateExecutor(address _newExecutor)
```

- **New Executor Address:** `0xAE0361b1C3454b297129e01046057F1D294c7974`

5. RISK METHODOLOGY

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

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

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

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

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

5.1 EXPLOITABILITY

ATTACK ORIGIN (AO):

Captures whether the attack requires compromising a specific account.

ATTACK COST (AC):

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

ATTACK COMPLEXITY (AX):

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

METRICS:

EXPLOITABILIY METRIC (M_E)	METRIC VALUE	NUMERICAL VALUE
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

5.2 IMPACT

CONFIDENTIALITY (C):

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

INTEGRITY (I):

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

AVAILABILITY (A):

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

DEPOSIT (D):

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

YIELD (Y):

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

METRICS:

IMPACT METRIC (M_I)	METRIC VALUE	NUMERICAL VALUE
Confidentiality (C)	None (I:N) Low (I:L) Medium (I:M) High (I:H) Critical (I:C)	0 0.25 0.5 0.75 1

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

Impact I is calculated using the following formula:

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

5.3 SEVERITY COEFFICIENT

REVERSIBILITY (R):

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

SCOPE (S):

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

METRICS:

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

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

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

The score is rounded up to 1 decimal places.

SEVERITY	SCORE VALUE RANGE
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

6. SCOPE

FILES AND REPOSITORY

(a) Repository: dln-evm

(b) Assessed Commit ID: a199d6f

(c) Items in scope:

- DlnSource.sol
- DlnDestination.sol
- DlnExternalCallAdapter.sol
- ExternalCallExecutor.sol

Out-of-Scope: Third-party dependencies and economic attacks.

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

7. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW
0	0	0	0

INFORMATIONAL

1

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
INCOMPATIBILITY OF EXTERNALCALLEXECUTOR CONTRACT WITH TRANSPARENT PROXY UPGRADEABILITY	INFORMATIONAL	SOLVED - 12/11/2024

8. FINDINGS & TECH DETAILS

8.1 INCOMPATIBILITY OF EXTERNALCALLEXECUTOR CONTRACT WITH TRANSPARENT PROXY UPGRADEABILITY

// INFORMATIONAL

Description

During the security assessment of the smart contracts in-scope, it was identified that `ExternalCallExecutor` and its abstract parent `ExternalCallExecutorBase` are incompatible with the Transparent Proxy upgradeability pattern employed by other contracts in the system.

The `ExternalCallExecutorBase` contract utilizes a constructor for state initialization and does not inherit from upgradeable base contracts. In the Transparent Proxy pattern, the proxy contract delegates calls to the implementation (logic) contract but maintains its own storage. **Constructors in the implementation contract are not executed during proxy deployment**.

`contracts/adapters/ExternalCallExecutor.sol`

```
10 | contract ExternalCallExecutor is
11 |     ExternalCallExecutorBase,
12 |     IExternalCallExecutor
13 | {
```

`contracts/adapters/ExternalCallExecutorBase.sol`

```
37 |     constructor() {
38 |         _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
```

```
39 |     }
```

Consequently, any state variables initialized in the constructors of `ExternalCallExecutorBase` and `ExternalCallExecutor` remain uninitialized in the proxy's storage. This leads to the following issues:

- **Uninitialized State Variables:** Critical state variables may remain unset, causing the contract to behave unpredictably or fail entirely. More specifically, the `constructor` of the `ExternalCallExecutor` contract is **setting the** `DEFAULT_ADMIN_ROLE` to the contract's deployer: `msg.sender`.
- **Inability to Upgrade:** Attempting to upgrade `ExternalCallExecutor` via regular upgradeable proxy methods will not correctly initialize the contract's state, rendering the upgrade ineffective.

This incompatibility undermines the contract's functionality and the integrity of the upgrade process within the system.

Proof of Concept

In order to reproduce this issue, it is recommended to use the `Hardhat Upgrades` plugin, which can be found at the official code repository.

The following code can be utilized in this proof of concept, highlighting that upgradeability is not feasible in the `ExternalCallExecutor` contract.

- **PoC Code:**

```
import { expect } from 'chai';
import { ethers, upgrades } from 'hardhat';

describe.only('Contract Upgrade Validation - ExternalCallExecutor', func
    let oldImplementation: any;
    let newImplementation: any;
```

```
before(async function () {
    oldImplementation = await ethers.getContractFactory('ExternalCal
    newImplementation = await ethers.getContractFactory('ExternalCal
});

it('validates old implementation', async function () {
    await upgrades.validateImplementation(oldImplementation, {
        kind: 'transparent'
    });
});

it('validates new implementation', async function () {
    await upgrades.validateImplementation(newImplementation, {
        kind: 'transparent'
    });
});

it('validates storage layout compatibility', async function () {
    const errors = await upgrades.validateUpgrade(
        oldImplementation,
        newImplementation,
        {
            kind: 'transparent'
        }
    );
    expect(errors).to.be.undefined;
});

it('checks for common upgrade risks', async function () {
    try {
        await upgrades.validateImplementation(newImplementation, {
            unsafeAllow: ['delegatecall', 'selfdestruct'],
```

```
                kind: 'transparent'
            });
        } catch (e: any) {
            expect(e.message).to.not.include('Contract `ExternalCallExecu
        }
    });
});
```

- **Logs:**

```
  Contract Upgrade Validation - DlnDestination
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - DlnExternalCallAdapter
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - DlnSource
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - ExternalCallExecutor
    1) validates old implementation
    2) validates new implementation
    3) validates storage layout compatibility
    ✔ checks for common upgrade risks


  13 passing (934ms)
  3 failing

  1) Contract Upgrade Validation - ExternalCallExecutor
       validates old implementation:
     Error: Contract `contracts/adapters/ExternalCallExecutor_old.sol:ExternalCallExecutor_old` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase_old.sol:37: Contract `ExternalCallExecutorBase_old` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at validateImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/validate-impl.ts:36:20)
      at Proxy.validateImplementation (node_modules/@openzeppelin/hardhat-upgrades/src/validate-implementation.ts:16:23)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:14:9)

  2) Contract Upgrade Validation - ExternalCallExecutor
       validates new implementation:
     Error: Contract `contracts/adapters/ExternalCallExecutor.sol:ExternalCallExecutor` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase.sol:37: Contract `ExternalCallExecutorBase` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at validateImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/validate-impl.ts:36:20)
      at Proxy.validateImplementation (node_modules/@openzeppelin/hardhat-upgrades/src/validate-implementation.ts:16:23)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:20:9)

  3) Contract Upgrade Validation - ExternalCallExecutor
       validates storage layout compatibility:
     Error: Contract `contracts/adapters/ExternalCallExecutor.sol:ExternalCallExecutor` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase.sol:37: Contract `ExternalCallExecutorBase` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at Proxy.validateUpgrade (node_modules/@openzeppelin/hardhat-upgrades/src/validate-upgrade.ts:41:24)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:26:24)
```

BVSS

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

Recommendation

Considering that `ExternalCallExecutor` is already deployed as a non-upgradeable contract, the following remediation steps are recommended:

- **Refactor Contracts:**
  - Modify `ExternalCallExecutor` and `ExternalCallExecutorBase` to replace constructors with initializer functions using the `initializer` modifier.
  - Inherit from upgradeable base contracts to ensure compatibility with the Transparent Proxy pattern.

- **Implement Upgradeable Patterns:**
  - Use OpenZeppelin's upgradeable contract libraries to facilitate secure and standard-compliant upgradeability.

- **Deploy Behind a Proxy:**
  - Deploy the new `ExternalCallExecutor` as the implementation contract behind a Transparent Upgradeable Proxy.
  - Initialize the contract state by calling the initializer function post-deployment.

- **Migrate State and Users to the New Contract:**
  - Develop scripts to transfer essential state data from the old contract to the new one.
  - Ensure data integrity during migration by verifying transferred data.

- **User Notification:**
  - Inform users and stakeholders about the deployment of the new contract.
  - Provide clear instructions on any actions they need to take, such as updating contract addresses in their applications or wallets.

- **Update System Integrations:**
  - Update all system components (APIs, services, frontends) to interact with the new contract address.
  - Ensure seamless integration to prevent service disruptions.

- **Third-Party Coordination:**
  - Communicate with third-party services that interact with `ExternalCallExecutor` to update to the new contract.

- **Finally, deprecate the Old Contract.**

Remediation

SOLVED: The **deBridge team** has solved this issue, informing that relevant setter functions are in place where the `ExternalCallExecutor` contract address needs to be changed or pointed differently.

9. REVIEW NOTES

Added custom wording to "summary" section as requested in TG channel.

10. AUTOMATED TESTING

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

```
INFO:Detectors:
DlnExternalCallAdapter._execute(bytes32,address,uint256,bytes,address) (contracts/adapters/DlnExternalCallAdapter.sol#322-390) sends eth to arbitrary user
        Dangerous calls:
        - (executionStatus,callResult) = currentExecutor.onEtherReceived{value: _tokenAmount}(_orderId,dataEnvelope.fallbackAddress,dataEnvelope.payload) (contracts/adapters/DlnExternalCallAdapter.sol#361-363)
DlnExternalCallAdapter._safeTransferETH(address,uint256) (contracts/adapters/DlnExternalCallAdapter.sol#575-578) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = to.call{value: value}(new bytes(0)) (contracts/adapters/DlnExternalCallAdapter.sol#576)
DlnExternalCallAdapter_old._execute(bytes32,address,uint256,bytes,address) (contracts/adapters/DlnExternalCallAdapter_old.sol#235-293) sends eth to arbitrary user
        Dangerous calls:
        - (executionStatus,callResult) = currentExecutor.onEtherReceived{value: _tokenAmount}(_orderId,dataEnvelope.fallbackAddress,dataEnvelope.payload) (contracts/adapters/DlnExternalCallAdapter_old.sol#267-271
)
DlnExternalCallAdapter_old._safeTransferETH(address,uint256) (contracts/adapters/DlnExternalCallAdapter_old.sol#397-400) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = to.call{value: value}(new bytes(0)) (contracts/adapters/DlnExternalCallAdapter_old.sol#398)
ExternalCallExecutorBase._safeTransferETH(address,uint256) (contracts/adapters/ExternalCallExecutorBase.sol#126-129) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = to.call{value: value}(new bytes(0)) (contracts/adapters/ExternalCallExecutorBase.sol#127)
ExternalCallExecutorBase_old._safeTransferETH(address,uint256) (contracts/adapters/ExternalCallExecutorBase_old.sol#126-129) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = to.call{value: value}(new bytes(0)) (contracts/adapters/ExternalCallExecutorBase_old.sol#127)
MockWeth._safeTransferETH(address,uint256) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/mock/MockWeth.sol#53-56) sends eth to arbitrary user
        Dangerous calls:
        - (success,None) = to.call{value: value}(new bytes(0)) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/mock/MockWeth.sol#54)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations
INFO:Detectors:
DlnDestination.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination.sol#373-451) calls abi.encodePacked() with multiple dynamic arguments:
        - instructionsData = abi.encodePacked(instructionsData,EncodeSolanaDlnMessage.encodeClaimUnlockInstruction(getChainId(),solanaSrcProgramId,_beneficiary,giveTokenAddress,orderId,_claimUnlockInstructionRewa
rd)) (contracts/DLN/DlnDestination.sol#426-436)
DlnDestination.sendSolanaUnlock(DlnOrderLib.Order,bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination.sol#472-509) calls abi.encodePacked() with multiple dynamic arguments:
        - instructionsData = abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_beneficiary,giveTokenAddress,_initWalletIfNeededInstructionReward),EncodeSolanaDlnMessage.encodeClaimUnloc
kInstruction(getChainId(),BytesLib.toBytes32(dlnSourceAddresses[giveChainId],0),_beneficiary,giveTokenAddress,orderId,_claimUnlockInstructionReward)) (contracts/DLN/DlnDestination.sol#485-499)
DlnDestination.sendSolanaOrderCancel(DlnOrderLib.Order,bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination.sol#584-638) calls abi.encodePacked() with multiple dynamic arguments:
        - claimCancelMethod = abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_cancelBeneficiary,_orderGiveTokenAddress,_initWalletIfNeededInstructionReward),EncodeSolanaDlnMessage.enc
odeClaimCancelInstruction(getChainId(),BytesLib.toBytes32(solanaSrcProgramId,0),_cancelBeneficiary,_orderGiveTokenAddress,orderId,_claimCancelInstructionReward)) (contracts/DLN/DlnDestination.sol#614-628)
DlnDestination_old.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination_old.sol#367-445) calls abi.encodePacked() with multiple dynamic arguments:
        - instructionsData = abi.encodePacked(instructionsData,EncodeSolanaDlnMessage.encodeClaimUnlockInstruction(getChainId(),solanaSrcProgramId,_beneficiary,giveTokenAddress,orderId,_claimUnlockInstructionRewa
rd)) (contracts/DLN/DlnDestination_old.sol#420-430)
DlnDestination_old.sendSolanaUnlock(DlnOrderLib.Order,bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination_old.sol#466-503) calls abi.encodePacked() with multiple dynamic arguments:
        - instructionsData = abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_beneficiary,giveTokenAddress,_initWalletIfNeededInstructionReward),EncodeSolanaDlnMessage.encodeClaimUnloc
kInstruction(getChainId(),BytesLib.toBytes32(dlnSourceAddresses[giveChainId],0),_beneficiary,giveTokenAddress,orderId,_claimUnlockInstructionReward)) (contracts/DLN/DlnDestination_old.sol#479-493)
DlnDestination_old.sendSolanaOrderCancel(DlnOrderLib.Order,bytes32,uint256,uint64,uint64) (contracts/DLN/DlnDestination_old.sol#578-632) calls abi.encodePacked() with multiple dynamic arguments:
        - claimCancelMethod = abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_cancelBeneficiary,_orderGiveTokenAddress,_initWalletIfNeededInstructionReward),EncodeSolanaDlnMessage.enc
odeClaimCancelInstruction(getChainId(),BytesLib.toBytes32(solanaSrcProgramId,0),_cancelBeneficiary,_orderGiveTokenAddress,orderId,_claimCancelInstructionReward)) (contracts/DLN/DlnDestination_old.sol#608-622)
MockDlnDestination.encodeSolanaClaim(uint256,bytes32,bytes32,bytes32,bytes32,uint64,uint64) (contracts/mock/MockDlnDestination.sol#49-74) calls abi.encodePacked() with multiple dynamic arguments:
        - abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_actionBeneficiary,_orderGiveTokenAddress,_reward1),EncodeSolanaDlnMessage.encodeClaimUnlockInstruction(_takeChainId,_srcProgr
amId,_actionBeneficiary,_orderGiveTokenAddress,_orderId,_reward2)) (contracts/mock/MockDlnDestination.sol#59-73)
```

```
MockDlnDestination.encodeSolanaCancel(uint256,bytes32,bytes32,bytes32,bytes32,uint64,uint64) (contracts/mock/MockDlnDestination.sol#76-101) calls abi.encodePacked() with multiple dynamic arguments:
        - abi.encodePacked(EncodeSolanaDlnMessage.encodeInitWalletIfNeededInstruction(_actionBeneficiary,_orderGiveTokenAddress,_reward1),EncodeSolanaDlnMessage.encodeClaimCancelInstruction(_takeChainId,_srcProgr
amId,_actionBeneficiary,_orderGiveTokenAddress,_orderId,_reward2)) (contracts/mock/MockDlnDestination.sol#86-100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#abi-encodePacked-collision
INFO:Detectors:
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) has bitwise-xor operator ^ instead of the exponentiation operator **:
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-exponentiation
INFO:Detectors:
Reentrancy in DlnDestination._fulfillOrder(bytes,DlnOrderLib.Order,uint256,bytes32,address,address) (contracts/DLN/DlnDestination.sol#185-257):
        External calls:
        - _safeTransferETH(tokenReceiver,takerAmount) (contracts/DLN/DlnDestination.sol#222)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
        - takeTokenAddress.executePermit(_permitEnvelope) (contracts/DLN/DlnDestination.sol#226)
        - _safeTransferFrom(takeTokenAddress,msg.sender,tokenReceiver,takerAmount) (contracts/DLN/DlnDestination.sol#228-233)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - token.safeTransferFrom(_from,_to,_amount) (contracts/DLN/DlnBase.sol#123)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        External calls sending eth:
        - _safeTransferETH(tokenReceiver,takerAmount) (contracts/DLN/DlnDestination.sol#222)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
        - _safeTransferFrom(takeTokenAddress,msg.sender,tokenReceiver,takerAmount) (contracts/DLN/DlnDestination.sol#228-233)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - orderState.status = OrderTakeStatus.Fulfilled (contracts/DLN/DlnDestination.sol#237)
        DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35) can be used in cross function reentrancies:
        - DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35)
        - orderState.takerAddress = _unlockAuthority (contracts/DLN/DlnDestination.sol#238)
        DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35) can be used in cross function reentrancies:
        - DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35)
        - orderState.giveChainId = uint32(_order.giveChainId) (contracts/DLN/DlnDestination.sol#241)
        DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35) can be used in cross function reentrancies:
        - DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35)
        - orderState.bigGiveChainId = _order.giveChainId (contracts/DLN/DlnDestination.sol#244)
        DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35) can be used in cross function reentrancies:
        - DlnDestination.takeOrders (contracts/DLN/DlnDestination.sol#35)
Reentrancy in DlnDestination_old._fulfillOrder(bytes,DlnOrderLib.Order,uint256,bytes32,address,address) (contracts/DLN/DlnDestination_old.sol#180-251):
        External calls:
        - _safeTransferETH(tokenReceiver,takerAmount) (contracts/DLN/DlnDestination_old.sol#217)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase_old.sol#154)
        - _executePermit(takeTokenAddress,_permitEnvelope) (contracts/DLN/DlnDestination_old.sol#221)
                - IERC20Permit(_tokenAddress).permit(msg.sender,address(this),permitAmount,deadline,v,r,s) (contracts/DLN/DlnBase_old.sol#121-129)
        - _safeTransferFrom(takeTokenAddress,msg.sender,tokenReceiver,takerAmount) (contracts/DLN/DlnDestination_old.sol#222-227)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - token.safeTransferFrom(_from,_to,_amount) (contracts/DLN/DlnBase_old.sol#142)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        External calls sending eth:
        - _safeTransferETH(tokenReceiver,takerAmount) (contracts/DLN/DlnDestination_old.sol#217)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase_old.sol#154)
        - _safeTransferFrom(takeTokenAddress,msg.sender,tokenReceiver,takerAmount) (contracts/DLN/DlnDestination_old.sol#222-227)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - orderState.status = OrderTakeStatus.Fulfilled (contracts/DLN/DlnDestination_old.sol#231)
        DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34) can be used in cross function reentrancies:
        - DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34)
        - orderState.takerAddress = _unlockAuthority (contracts/DLN/DlnDestination_old.sol#232)
        DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34) can be used in cross function reentrancies:
        - DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34)
        - orderState.giveChainId = uint32(_order.giveChainId) (contracts/DLN/DlnDestination_old.sol#235)
        DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34) can be used in cross function reentrancies:
        - DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34)
        - orderState.bigGiveChainId = _order.giveChainId (contracts/DLN/DlnDestination_old.sol#238)
        DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34) can be used in cross function reentrancies:
        - DlnDestination_old.takeOrders (contracts/DLN/DlnDestination_old.sol#34)
Reentrancy in DeBridgeGate._send(bytes,address,uint256,uint256,bool) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#635-781):
        External calls:
        - _validateToken(_tokenAddress) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#646)
```

```
        DlnSource.giveOrders (contracts/DLN/DlnSource.sol#30) can be used in cross function reentrancies:
        - DlnSource.giveOrders (contracts/DLN/DlnSource.sol#30)
Reentrancy in DlnSource_old.patchOrderGive(DlnOrderLib.Order,uint256,bytes) (contracts/DLN/DlnSource_old.sol#397-428):
        External calls:
        - _executePermit(giveTokenAddress,_permitEnvelope) (contracts/DLN/DlnSource_old.sol#415)
                - IERC20Permit(_tokenAddress).permit(msg.sender,address(this),permitAmount,deadline,v,r,s) (contracts/DLN/DlnBase_old.sol#121-129)
        - _safeTransferFrom(giveTokenAddress,msg.sender,address(this),_addGiveAmount) (contracts/DLN/DlnSource_old.sol#416-421)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - token.safeTransferFrom(_from,_to,_amount) (contracts/DLN/DlnBase_old.sol#142)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        External calls sending eth:
        - _safeTransferFrom(giveTokenAddress,msg.sender,address(this),_addGiveAmount) (contracts/DLN/DlnSource_old.sol#416-421)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - orderState.percentFee += percentFee.toUint208() (contracts/DLN/DlnSource_old.sol#425)
        DlnSource_old.giveOrders (contracts/DLN/DlnSource_old.sol#28) can be used in cross function reentrancies:
        - DlnSource_old.giveOrders (contracts/DLN/DlnSource_old.sol#28)
Reentrancy in DlnSource.withdrawFee(address[],address) (contracts/DLN/DlnSource.sol#494-506):
        External calls:
        - _safeTransferEthOrToken(token,_beneficiary,feeAmount) (contracts/DLN/DlnSource.sol#502)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
                - IERC20Upgradeable(tokenAddress).safeTransfer(to,value) (contracts/DLN/DlnBase.sol#152)
        External calls sending eth:
        - _safeTransferEthOrToken(token,_beneficiary,feeAmount) (contracts/DLN/DlnSource.sol#502)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - collectedFee[token] = 0 (contracts/DLN/DlnSource.sol#503)
        DlnSource.collectedFee (contracts/DLN/DlnSource.sol#39) can be used in cross function reentrancies:
        - DlnSource.collectedFee (contracts/DLN/DlnSource.sol#39)
Reentrancy in DlnSource_old.withdrawFee(address[],address) (contracts/DLN/DlnSource_old.sol#487-499):
        External calls:
        - _safeTransferEthOrToken(token,_beneficiary,feeAmount) (contracts/DLN/DlnSource_old.sol#495)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase_old.sol#154)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
                - IERC20Upgradeable(tokenAddress).safeTransfer(to,value) (contracts/DLN/DlnBase_old.sol#171)
        External calls sending eth:
        - _safeTransferEthOrToken(token,_beneficiary,feeAmount) (contracts/DLN/DlnSource_old.sol#495)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase_old.sol#154)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - collectedFee[token] = 0 (contracts/DLN/DlnSource_old.sol#496)
        DlnSource_old.collectedFee (contracts/DLN/DlnSource_old.sol#37) can be used in cross function reentrancies:
        - DlnSource_old.collectedFee (contracts/DLN/DlnSource_old.sol#37)
Reentrancy in DlnSource.withdrawUnclaimedAffiliateFees(address[],address) (contracts/DLN/DlnSource.sol#448-468):
        External calls:
        - _safeTransferEthOrToken(token,_beneficiary,amount) (contracts/DLN/DlnSource.sol#464)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts-upgradeable/token/ERC20/utils/SafeERC20Upgradeable.sol#110)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
                - IERC20Upgradeable(tokenAddress).safeTransfer(to,value) (contracts/DLN/DlnBase.sol#152)
        External calls sending eth:
        - _safeTransferEthOrToken(token,_beneficiary,amount) (contracts/DLN/DlnSource.sol#464)
                - (success,None) = to.call{value: value}(new bytes(0)) (contracts/DLN/DlnBase.sol#135)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol#135)
        State variables written after the call(s):
        - unclaimedAffiliateETHFees[msg.sender] = 0 (contracts/DLN/DlnSource.sol#458)
        DlnSource.unclaimedAffiliateETHFees (contracts/DLN/DlnSource.sol#51) can be used in cross function reentrancies:
        - DlnSource.unclaimedAffiliateETHFees (contracts/DLN/DlnSource.sol#51)
        - unclaimedERC20AffiliateFees[token][msg.sender] = 0 (contracts/DLN/DlnSource.sol#461)
        DlnSource.unclaimedERC20AffiliateFees (contracts/DLN/DlnSource.sol#56) can be used in cross function reentrancies:
        - DlnSource.unclaimedERC20AffiliateFees (contracts/DLN/DlnSource.sol#56)
Reentrancy in DlnSource_old.withdrawUnclaimedAffiliateFees(address[],address) (contracts/DLN/DlnSource_old.sol#441-461):
        External calls:
```

```
INFO:Detectors:
BytesLib.concatStorage(bytes,bytes) (contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (contracts/libraries/BytesLib.sol#223)
BytesLib.concatStorage(bytes,bytes) (contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (contracts/libraries/BytesLib.sol#189)
BytesLib.concatStorage(bytes,bytes) (contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(_preBytes,fslot_concatStorage_asm_0 + mload(uint256)(_postBytes + 0x20) / 0x100 ** 32 - mlength_concatStorage_asm_0 * 0x100 ** 32 - newlength_concatStorage_asm_0 + mlength_concat
Storage_asm_0 * 2) (contracts/libraries/BytesLib.sol#115-140)
BytesLib.equalStorage(bytes,bytes) (contracts/libraries/BytesLib.sol#450-520) performs a multiplication on the result of a division:
        - fslot_equalStorage_asm_0 = fslot_equalStorage_asm_0 / 0x100 * 0x100 (contracts/libraries/BytesLib.sol#477)
BytesLib.concatStorage(bytes,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (node_modules/@debridge-finance/debridge-contracts-v1/contrac
ts/libraries/BytesLib.sol#223)
BytesLib.concatStorage(bytes,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(sc_concatStorage_asm_0,mload(uint256)(mc_concatStorage_asm_0) / mask_concatStorage_asm_0 * mask_concatStorage_asm_0) (node_modules/@debridge-finance/debridge-contracts-v1/contrac
ts/libraries/BytesLib.sol#189)
BytesLib.concatStorage(bytes,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#91-226) performs a multiplication on the result of a division:
        - sstore(uint256,uint256)(_preBytes,fslot_concatStorage_asm_0 + mload(uint256)(_postBytes + 0x20) / 0x100 ** 32 - mlength_concatStorage_asm_0 * 0x100 ** 32 - newlength_concatStorage_asm_0 + mlength_concat
Storage_asm_0 * 2) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#115-140)
BytesLib.equalStorage(bytes,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#439-509) performs a multiplication on the result of a division:
        - fslot_equalStorage_asm_0 = fslot_equalStorage_asm_0 / 0x100 * 0x100 (node_modules/@debridge-finance/debridge-contracts-v1/contracts/libraries/BytesLib.sol#466)
DeBridgeGate._normalizeTokenAmount(address,uint256) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#987-1000) performs a multiplication on the result of a division:
        - _amount = _amount / multiplier * multiplier (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#997)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse = (3 * denominator) ^ 2 (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#117)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#121)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#122)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#123)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#124)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#125)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - denominator = denominator / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#102)
        - inverse *= 2 - denominator * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#126)
MathUpgradeable.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#55-135) performs a multiplication on the result of a division:
        - prod0 = prod0 / twos (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#105)
        - result = prod0 * inverse (node_modules/@openzeppelin/contracts-upgradeable/utils/math/MathUpgradeable.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
DeBridgeGate.send(address,uint256,uint256,bytes,bytes,bool,uint32,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#229-279) uses a dangerous strict equality:
        - autoParams.data.length > 0 && autoParams.fallbackAddress.length == 0 (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#261)
SignatureVerifier.submit(bytes32,bytes,uint8) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/SignatureVerifier.sol#58-114) uses a dangerous strict equality:
        - currentBlock == uint40(block.number) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/SignatureVerifier.sol#100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Contract locking ether found:
        Contract MockCallProxy (contracts/mock/MockCallProxy.sol#4-28) has payable functions:
         - MockCallProxy.bypassCall(bytes,uint256,address,bytes) (contracts/mock/MockCallProxy.sol#8-26)
        But does not have a function to withdraw the ether
Contract locking ether found:
        Contract MockToken (node_modules/@debridge-finance/debridge-contracts-v1/contracts/mock/MockToken.sol#8-34) has payable functions:
         - MockToken.fallback() (node_modules/@debridge-finance/debridge-contracts-v1/contracts/mock/MockToken.sol#19)
         - MockToken.receive() (node_modules/@debridge-finance/debridge-contracts-v1/contracts/mock/MockToken.sol#21)
        But does not have a function to withdraw the ether
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#contracts-that-lock-ether
INFO:Detectors:
Reentrancy in DeBridgeGate.deployNewAsset(bytes,uint256,string,string,uint8,bytes) (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#343-364):
```

```
INFO:Detectors:
WidoCallExecutor.onERC20Received(bytes32,address,uint256,address,bytes) (contracts/adapters/examples/WidoCallExecutor.sol#50-74) ignores return value by widoRouter.call(_payload) (contracts/adapters/examples/Wido
CallExecutor.sol#65)
MockReceiver.receiveETH(address) (contracts/mock/MockReceiver.sol#13-16) ignores return value by _receiver.call{value: msg.value}(new bytes(0)) (contracts/mock/MockReceiver.sol#14)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-low-level-calls
INFO:Detectors:
DlnDestination_old.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).giveChainId (contracts/DLN/DlnDestination_old.sol#379) is a local variable never initialized
DlnDestination._encodeAutoParamsTo(bytes,uint256,bytes).autoParams (contracts/DLN/DlnDestination.sol#774) is a local variable never initialized
DlnExternalCallAdapter_old._execute(bytes32,address,uint256,bytes,address).executionStatus (contracts/adapters/DlnExternalCallAdapter_old.sol#243) is a local variable never initialized
DlnSource_old.claimBatchUnlock(bytes32[],address).distinctGiveToken (contracts/DLN/DlnSource_old.sol#285) is a local variable never initialized
DlnDestination.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).giveChainId (contracts/DLN/DlnDestination.sol#385) is a local variable never initialized
DeBridgeGate.sendMessage(uint256,bytes,bytes,uint256,uint32).autoParams (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#210) is a local variable never initialized
DlnDestination.sendBatchEvmUnlock(bytes32[],address,uint256).giveChainId (contracts/DLN/DlnDestination.sol#325) is a local variable never initialized
DlnSource_old.claimBatchUnlock(bytes32[],address).distinctGiveTokenAmount (contracts/DLN/DlnSource_old.sol#286) is a local variable never initialized
DlnDestination_old.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).giveTokenAddress (contracts/DLN/DlnDestination_old.sol#380) is a local variable never initialized
DlnSource.claimBatchUnlock(bytes32[],address).distinctGiveTokenAmount (contracts/DLN/DlnSource.sol#293) is a local variable never initialized
DlnDestination.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).giveTokenAddress (contracts/DLN/DlnDestination.sol#386) is a local variable never initialized
DlnDestination_old.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).solanaSrcProgramId (contracts/DLN/DlnDestination_old.sol#381) is a local variable never initialized
DlnDestination.sendBatchSolanaUnlock(DlnOrderLib.Order[],bytes32,uint256,uint64,uint64).solanaSrcProgramId (contracts/DLN/DlnDestination.sol#387) is a local variable never initialized
DlnDestination_old.sendBatchEvmUnlock(bytes32[],address,uint256).giveChainId (contracts/DLN/DlnDestination_old.sol#319) is a local variable never initialized
DlnExternalCallAdapter_old._execute(bytes32,address,uint256,bytes,address).callResult (contracts/adapters/DlnExternalCallAdapter_old.sol#244) is a local variable never initialized
DeBridgeGate.claim(bytes32,uint256,uint256,address,uint256,bytes,bytes).autoParams (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#293) is a local variable never initial
ized
DeBridgeGate._send(bytes,address,uint256,uint256,bool).assetsFixedFee (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#721) is a local variable never initialized
DlnExternalCallAdapter._execute(bytes32,address,uint256,bytes,address).executionStatus (contracts/adapters/DlnExternalCallAdapter.sol#330) is a local variable never initialized
DeBridgeGate.send(address,uint256,uint256,bytes,bytes,bool,uint32,bytes).autoParams (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/DeBridgeGate.sol#252) is a local variable never initia
lized
SignatureVerifier.submit(bytes32,bytes,uint8).currentRequiredOraclesCount (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/SignatureVerifier.sol#68) is a local variable never initialized
DlnDestination_old._encodeAutoParamsTo(bytes,uint256,bytes).autoParams (contracts/DLN/DlnDestination_old.sol#757) is a local variable never initialized
DlnSource._createSaltedOrder(DlnOrderLib.OrderCreation,uint64,bytes,uint32,bytes,bytes).affiliateAmount (contracts/DLN/DlnSource.sol#228) is a local variable never initialized
DlnSource_old._createSaltedOrder(DlnOrderLib.OrderCreation,uint64,bytes,uint32,bytes,bytes).affiliateAmount (contracts/DLN/DlnSource_old.sol#221) is a local variable never initialized
SignatureVerifier.submit(bytes32,bytes,uint8).confirmations (node_modules/@debridge-finance/debridge-contracts-v1/contracts/transfers/SignatureVerifier.sol#70) is a local variable never initialized
DlnSource.claimBatchUnlock(bytes32[],address).distinctGiveToken (contracts/DLN/DlnSource.sol#292) is a local variable never initialized
DlnExternalCallAdapter._execute(bytes32,address,uint256,bytes,address).callResult (contracts/adapters/DlnExternalCallAdapter.sol#331) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables
INFO:Detectors:
ERC1967Upgrade._upgradeToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#65-74) ignores return value by Address.functionDelegateCall(newImplementation,data) (nod
e_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#72)
ERC1967Upgrade._upgradeBeaconToAndCall(address,bytes,bool) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#174-184) ignores return value by Address.functionDelegateCall(IBeacon(newBeacon).i
mplementation(),data) (node_modules/@openzeppelin/contracts/proxy/ERC1967/ERC1967Upgrade.sol#182)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Slither:. analyzed (90 contracts with 53 detectors), 81 result(s) found
```

All issues identified by Slither were proved to be false positives, out-of-scope or have been added to the issue list in this report.

Furthermore, Halborn performed individual tests for upgradeability using the Hardhat Upgrades plugin. The results are the following:

```
  Contract Upgrade Validation - DlnDestination
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - DlnExternalCallAdapter
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - DlnSource
    ✔ validates old implementation
    ✔ validates new implementation
    ✔ validates storage layout compatibility
    ✔ checks for common upgrade risks

  Contract Upgrade Validation - ExternalCallExecutor
    1) validates old implementation
    2) validates new implementation
    3) validates storage layout compatibility
    ✔ checks for common upgrade risks


  13 passing (934ms)
  3 failing

  1) Contract Upgrade Validation - ExternalCallExecutor
       validates old implementation:
     Error: Contract `contracts/adapters/ExternalCallExecutor_old.sol:ExternalCallExecutor_old` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase_old.sol:37: Contract `ExternalCallExecutorBase_old` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at validateImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/validate-impl.ts:36:20)
      at Proxy.validateImplementation (node_modules/@openzeppelin/hardhat-upgrades/src/validate-implementation.ts:16:23)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:14:9)

  2) Contract Upgrade Validation - ExternalCallExecutor
       validates new implementation:
     Error: Contract `contracts/adapters/ExternalCallExecutor.sol:ExternalCallExecutor` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase.sol:37: Contract `ExternalCallExecutorBase` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at validateImpl (node_modules/@openzeppelin/hardhat-upgrades/src/utils/validate-impl.ts:36:20)
      at Proxy.validateImplementation (node_modules/@openzeppelin/hardhat-upgrades/src/validate-implementation.ts:16:23)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:20:9)

  3) Contract Upgrade Validation - ExternalCallExecutor
       validates storage layout compatibility:
     Error: Contract `contracts/adapters/ExternalCallExecutor.sol:ExternalCallExecutor` is not upgrade safe

contracts/adapters/ExternalCallExecutorBase.sol:37: Contract `ExternalCallExecutorBase` has a constructor
    Define an initializer instead
    https://zpl.in/upgrades/error-001
      at assertUpgradeSafe (node_modules/@openzeppelin/upgrades-core/src/validate/query.ts:19:11)
      at Proxy.validateUpgrade (node_modules/@openzeppelin/hardhat-upgrades/src/validate-upgrade.ts:41:24)
      at async Context.<anonymous> (test/UpgradeTest_ExternalCallExecutor.test.ts:26:24)
```

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