HALBORN

DAMfinance - LMCV part 3

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 27th, 2022 - November 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	10/27/2022	István Böhm
0.2	Document Updates	11/04/2022	István Böhm
0.3	Draft Review	11/04/2022	Kubilay Onur Gungor
0.4	Draft Review	11/04/2022	Gabi Urrutia
0.5	Document Updates	11/14/2022	István Böhm
1.0	Remediation Plan	11/17/2022	István Böhm
1.1	Remediation Plan Review	11/17/2022	Kubilay Onur Gungor
1.2	Remediation Plan Review	11/17/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Kubilay Onur Gungor	Halborn	Kubilay.Gungor@halborn.com
István Böhm	Halborn	Istvan.Bohm@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

DAMfinance engaged Halborn to conduct a security audit on their smart contracts beginning on October 27th, 2022 and ending on November 17th, 2022. The security assessment was scoped to the smart contracts in the TO_AUDIT folder provided in the audit3Quote branch of the DecentralizedAssetManagement/lmcv GitHub repository.

The security audit also included the new modifications of the dPrime token that allowed delayed minting in case of cross-chain token transfers. The modified contract was provided in the audit3WithBlockDelay branch of the DecentralizedAssetManagement/lmcv GitHub repository.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified security risks that were mostly addressed by the DAMfinance team.

1.3 TEST APPROACH & 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 audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walk-through
- 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)
- Local deployment (Hardhat, Remix IDE, Brownie)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped to the following smart contracts:

- lmcv/contracts/TO_AUDIT/LayerZero/IOFT.sol
- lmcv/contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol
- lmcv/contracts/TO_AUDIT/dependencies/AuthAdmin.sol
- lmcv/contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol
- lmcv/contracts/TO_AUDIT/dPrime.sol
- lmcv/contracts/TO_AUDIT/dPrimeGuardian.sol

The security audit also included the new modifications of the dPrime token:

- lmcv/contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol
- lmcv/contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol
- lmcv/contracts/TO_AUDIT/dPrime.sol

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	1	2	2	2

LIKELIHOOD

IMPACT

				(HAL-01)
	(HAL-04)		(HAL-02)	
(HAL-06)		(HAL-03)		
	(HAL-05)			
(HAL-07) (HAL-08)				

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 - UNLIMITED MINTING BY REUSING FAILED HYPERLANE MESSAGES	Critical	SOLVED - 11/11/2022
HAL-02 - MISMATCHING DATA LOCATION DURING INHERITANCE	High	SOLVED - 11/17/2022
HAL-03 - DENIAL OF SERVICE USING MINT DELAY	Medium	SOLVED - 11/15/2022
HAL-04 - INCOMPLETE GUARDIAN IMPLEMENTATION	Medium	SOLVED - 11/11/2022
HAL-05 - MISSING ZERO VALUE CHECKS	Low	SOLVED - 11/11/2022
HAL-06 - MISSING EVENTS FOR RELEVANT OPERATIONS	Low	SOLVED - 11/11/2022
HAL-07 - REVERT STRING SIZE OPTIMIZATION	Informational	ACKNOWLEDGED
HAL-08 - STATE VARIABLES MISSING IMMUTABLE MODIFIER	Informational	SOLVED - 11/16/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNLIMITED MINTING BY REUSING FAILED HYPERLANE MESSAGES - CRITICAL

Description:

It is possible to transfer dPrime tokens across chains using the dPrimeConnectorHyperlane contract. If minting the tokens on the destination chain fails, then the program stores the failed transaction in a mapping. It is possible to manually retry the failed transactions by calling the retry function on the destination chain. The function reads the failed transaction details from a mapping and mints the specified tokens for the receiver. However, the function does not update the mapping after a successfully retried transaction, so it is possible to mint as many tokens as desired by calling the retry function multiple times with the data of the failed transaction.

Code Location:

Listing 1: hyperlane/dPrimeConnectorHyperlane.sol (Lines 144,145)

```
141 function retry(uint32 _origin, address _recipient, uint256 _nonce)
 ↳  external {
142     uint256 amount = failedMessages[_origin][_recipient][_nonce];
143
144     try dPrimeLike(dPrimeContract).mint(_recipient, amount) {
145         emit ReceivedTransferRemote(_origin, _recipient, amount);
146     } catch {
147         emit FailedTransferRemote(_origin, _recipient, nonce,
 ↳ amount);
148     }
149 }
```

Proof of Concept:

As proof of concept, a failed Hyperlane message was created in a local test environment. Then the same failed message was used multiple times to mint dPrime tokens for the receiver.

```
Events In This Transaction
--------------------------
└── dPrimeConnectorHyperlane (0x6fAC22cb619E83838AD7D3c346D94bFc28F1A2e3)
    └── FailedTransferRemote
        ├── origin: 6
        ├── recipient: 0xb668618740944 90f641d4927A783A2a607641D77 (user1)
        ├── nonce: 0
        └── amount: 1000000000000000000

>>> contract_dp.balanceOf(user1)
0
>>> contract_hyperlane.retry(6, user1, 0, {'from': user1})
Transaction sent: 0xe9be14a18599875fd43339aec91b5d3bcb0239acd33f369d5ba1a46ce0d73bc9
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  dPrimeConnectorHyperlane.retry confirmed   Block: 15891810   Gas used: 58349 (0.87%)

<Transaction '0xe9be14a18599875fd43339aec91b5d3bcb0239acd33f369d5ba1a46ce0d73bc9'>
>>> contract_dp.balanceOf(user1)
1000000000000000000
>>> contract_hyperlane.retry(6, user1, 0, {'from': user1})
Transaction sent: 0x8262aa9b5861f1e2b3ce2b8121bfdb6ad1a32d9584eca533de91207fcf67e26e
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 1
  dPrimeConnectorHyperlane.retry confirmed   Block: 15891811   Gas used: 43349 (0.64%)

<Transaction '0x8262aa9b5861f1e2b3ce2b8121bfdb6ad1a32d9584eca533de91207fcf67e26e'>
>>> contract_dp.balanceOf(user1)
2000000000000000000
```

It is noted that it is possible to mint as many tokens as desired using the failed message.

Risk Level:

Likelihood - 5
Impact - 5

Recommendation:

The `retry` function should delete the successfully retried messages from the `failedMessages` mapping.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit cfc13a8 by deleting the successfully retried messages from the failedMessages mapping.

3.2 (HAL-02) MISMATCHING DATA LOCATION DURING INHERITANCE - HIGH

Description:

It was identified that the dPrimeConnectorHyperlane contract was incompatible with the latest Hyperlane protocol and might not work with new versions of Hyperlane Inbox contracts, which can lead to users losing their dPrime tokens during cross-chain transfers.

The dPrimeConnectorHyperlane contract is inherited from the Hyperlane Router abstract contract. When receiving a message from another chain, the handle function is called by the corresponding Hyperlane Inbox contract. This function calls the internal _handle function that can be overridden in the derived contracts.

The following code is an example from the Hyperlane Router version 0.5.0:

Listing 2: @hyperlane-xyz/app/contracts/Router.sol (0.5.0) (Lines 85,95)

```
82     function handle(
83         uint32 _origin,
84         bytes32 _sender,
85         bytes memory _message
86     ) external virtual override onlyInbox onlyRemoteRouter(_origin
↳ , _sender) {
87         // TODO: callbacks on success/failure
88         _handle(_origin, _sender, _message);
89     }
90
91     // ============ Virtual functions ============
92     function _handle(
93         uint32 _origin,
94         bytes32 _sender,
95         bytes memory _message
96     ) internal virtual;
```

The _handle function was overridden in the dPrimeConnectorHyperlane contract:

Listing 3: hyperlane/dPrimeConnectorHyperlane.sol (Line 123)

```
120     function _handle(
121         uint32 _origin,
122         bytes32,
123         bytes memory _message
124     ) internal override alive {
```

However, from Hyperlane version 0.5.1, the storage location of the _message parameter was changed from memory to calldata:

Listing 4: @hyperlane-xyz/app/contracts/Router.sol (0.5.1) (Lines 85,95)

```
82     function handle(
83         uint32 _origin,
84         bytes32 _sender,
85         bytes calldata _message
86     ) external virtual override onlyInbox onlyRemoteRouter(_origin
↳ , _sender) {
87         // TODO: callbacks on success/failure
88         _handle(_origin, _sender, _message);
89     }
90
91     // ============ Virtual functions ============
92     function _handle(
93         uint32 _origin,
94         bytes32 _sender,
95         bytes calldata _message
96     ) internal virtual;
```

A bug concerning data location during inheritance was identified in Solidity on May 17, 2022. According to the Solidity team, the calldata pointer in these cases is interpreted as a memory pointer which results in reading invalid data from memory.

It is also noted that the current protocol is not configured with a fixed version of Hyperlane dependencies. In the package.json configuration

file, floating versions are used, which means that the contract might be deployed with an incompatible Hyperlane messaging protocol:

Listing 5: package.json (Lines 25,26)

```
23   "dependencies": {
24     "@chainlink/contracts": "^0.4.1",
25     "@hyperlane-xyz/app": "^0.5.0",
26     "@hyperlane-xyz/sdk": "^0.5.0",
27     "@layerzerolabs/solidity-examples": "^0.0.4",
28     "@openzeppelin/contracts": "^4.5.0",
29     "solc": "^0.8.15"
30   }
```

Proof of Concept:

As proof of concept, the dPrimeConnectorHyperlane contract was deployed with Hyperlane version 0.5.0. It was possible to mint tokens by calling the handle function:

```
>>> origin = 6
>>> amount = 10000000
>>> recepient = '0xD78FeF44993B9c1C9dEcBB5a3270663eE3861aFB'
>>> message = eth_abi.encode_abi(['address', 'uint256'], (recepient, amount)).hex()
>>> tx = contract_hyperlane.handle(origin, router, message, {'from': hyperlane})
Transaction sent: 0x186ab0d094df222430130dbea19b4c61eb73bf936cc579ba41e1b0d783ae6a0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 0
  dPrimeConnectorHyperlane.handle confirmed   Block: 15985674   Gas used: 81140 (1.21%)

>>> tx.info()
Transaction was Mined
---------------------
Tx Hash: 0x186ab0d094df222430130dbea19b4c61eb73bf936cc579ba41e1b0d783ae6a0b
From: 0x3909D73Dfc0B097EF5ef7DA039FDA347bba88E9a
To: 0x3589C2B9BdDE90D054d4E5273Ee6862957aa125D
Value: 0
Function: dPrimeConnectorHyperlane.handle
Block: 15985674
Gas Used: 81140 / 6721975 (1.2%)

Events In This Transaction
--------------------------
├── dPrime (0x5573ad23b8d8F0a327C3373a39953F640b6141c7)
│   └── Transfer
│       ├── from: 0x0000000000000000000000000000000000000000
│       ├── to: 0xD78FeF44993B9c1C9dEcBB5a3270663eE3861aFB
│       └── value: 10000000
│
└── dPrimeConnectorHyperlane (0x3589C2B9BdDE90D054d4E5273Ee6862957aa125D)
    └── ReceivedTransferRemote
        ├── origin: 6
        ├── recipient: 0xD78FeF44993B9c1C9dEcBB5a3270663eE3861aFB
        └── amount: 10000000
```

The contract was also deployed using the Hyperlane version 0.5.1. Calling the handle function with the same parameters resulted in an error and corrupted data:

```
>>> origin = 6
>>> amount = 10000000
>>> recepient = '0xD78FeF44993B9c1C9dEcBB5a3270663eE3861aFB'
>>> message = eth_abi.encode_abi(['address', 'uint256'], (recepient, amount)).hex()
>>> tx = contract_hyperlane.handle(origin, router, message, {'from': hyperlane})
Transaction sent: 0xf03e72929602011ff58af62fbdd10334154f7439b5f2fe0350ef060d434dcc9b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 1
  dPrimeConnectorHyperlane.handle confirmed (invalid JUMP at a698a6b8ae5b25fd1256b8559ec0093be7d9045490eccd5e5
61e1055ac825566/4c701545e4fc7b5072c9da2e2a881be36cda9743:3789)   Block: 15985680   Gas used: 6721975 (100.00%)

>>> tx.info()
Transaction was Mined (dPrime/invalid-address)
----------------------
Tx Hash: 0xf03e72929602011ff58af62fbdd10334154f7439b5f2fe0350ef060d434dcc9b
From: 0x3909D73Dfc0B097EF5ef7DA039FDA347bba88E9a
To: 0x4c701545e4FC7B5072c9DA2e2A881Be36cda9743
Value: 0
Function: dPrimeConnectorHyperlane.handle
Block: 15985680
Gas Used: 6721975 / 6721975 (100.0%)

Events In This Transaction
--------------------------
└── dPrimeConnectorHyperlane (0x4c701545e4FC7B5072c9DA2e2A881Be36cda9743)
    └── FailedTransferRemote
        ├── origin: 0
        ├── recipient: 0x0000000000000000000000000000000000000000
        ├── nonce: 0
        └── amount: 33
```

Note that it is not possible to recover the tokens from corrupted data using the retry function of the dPrimeConnectorHyperlane contract.

Risk Level:

Likelihood - 4
Impact - 4

Recommendation:

The dPrimeConnectorHyperlane contract should be updated to work with the latest version of Hyperlane contracts. Before deployment, DAMfinance team should ensure the dPrimeConnectorHyperlane contract is compatible with the Hyperlane Inbox contract deployed on the same chain.

It is also recommended to fix the versions of the external contracts in the package.json configuration file to versions that have been tested thoroughly with the protocol.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit c19b217 by changing the storage location of the _message parameter from memory to calldata and fixing the versions of the Hyperlane contracts in the package.json configuration file.

3.3 (HAL-03) DENIAL OF SERVICE USING MINT DELAY - MEDIUM

Description:

The new version of the dPrime token contract was extended with an additional security feature that prevents transferring funds for a certain number of blocks after a cross-chain transfer. The delay allows the DAMfinance team team to pause the contract in case the cross-chain messaging is compromised to limit the impact of such security incidents.

However, a malicious actor can also exploit this behavior to manipulate the market during volatile times or perform a griefing attack by continuously sending tokens to the targeted users. The likelihood of such attacks is increased because it is enough to send only one token to prevent the transfer of the targeted user's funds temporarily.

Code Location:

Listing 6: dPrime.sol (Line 192)

```
191     function mintAndDelay(address to, uint256 value) external auth
 ↳  {
192         transferBlockRelease[to] = block.number +
 ↳ transferBlockWait;
193         _mint(to, value);
194     }
```

Listing 7: dPrime.sol (Line 123)

```
121     function transferFrom(address from, address to, uint256 value)
 ↳  external alive returns (bool) {
122         require(to != address(0) && to != address(this), "dPrime/
 ↳ invalid-address");
123         require(block.number > transferBlockRelease[from], "dPrime
 ↳ /transfer too soon after cross-chain mint");
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Adding a configurable threshold to the mintAndDelay function is recommended. The function should be able to mint tokens below the set threshold directly.

The threshold and the length of the delay should be configured based on the specific blockchain and actual market characteristics to make it more expensive for an attacker to perform a denial of service attack while limiting the potential damage in case the cross-chain messaging of the protocol is compromised.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit e76f394 by adding a configurable threshold to the mintAndDelay function.

3.4 (HAL-04) INCOMPLETE GUARDIAN IMPLEMENTATION - MEDIUM

Description:

It was identified that in the dPrimeGuardian contract, it is not possible to add any values to the pipeAddresses mapping.
And therefore, in case of an incident, the dPrimeGuardian contract cannot be used to take back the admin privileges of any insecure connector from the dPrime token because the removeConnectorAdmin reads the required addresses from the mapping.

Code Location:

Listing 8: dPrimeGuardian.sol (Line 26)

```
25 function removeConnectorAdmin(bytes32 pipeName) external auth {
26     dPrimeLike(dPrimeContract).deny(pipeAddresses[pipeName]);
27     emit HaltedPipe(pipeName);
28 }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

Modify the contract to allow the removeConnectorAdmin function to be used.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit cfc13a8 by adding the setPipeAddress function to the dPrimeGuardian contract.

3.5 (HAL-05) MISSING EVENTS FOR RELEVANT OPERATIONS - LOW

Description:

The connector and guardian contracts are inherited from the AuthAdmin contract. It was identified that in this contract, the cage function did not emit any events. As a result, blockchain monitoring systems might not be able to timely detect suspicious behaviors related to the cage function.

Note that the Cage event declared in the dPrimeConnectorHyperlane contract is not used and could be removed from the contract.

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Adding events for all important operations is recommended to help monitor the contracts and detect suspicious behavior. A monitoring system that tracks relevant events would allow the timely detection of compromised system components.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit cfc13a8 by emitting the event in the cage function of the AuthAdmin contract and removing the redundant Cage event from the dPrimeConnectorHyperlane contract.

3.6 (HAL-06) MISSING ZERO VALUE CHECKS - LOW

Description:

It was identified that within the code, several functions were lacking zero address or zero value validation on important parameters. Setting invalid parameters in the examples below might result in loss of funds, waste of gas, lose of administrative controls, or reverting during important operations:

Missing zero address checks:

LayerZero/dPrimeConnectorLZ.sol:
- Line 24 constructor is missing zero-address checks for _lzEndpoint, _dPrimeContract.

hyperlane/dPrimeConnectorHyperlane.sol:
- Line 71 initialize is missing zero-address checks for _abacusConnectionManager, _interchainGasPaymaster, dPrimeContract.
- Line 95 transferRemote is missing zero-address check for _recipient.

dPrimeGuardian.sol
- Line 16: constructor is missing zero-address check for _dPrimeContract.

It is also recommended to validate the following parameters and variables:

hyperlane/dPrimeConnectorHyperlane.sol:
- Line 95 transferRemote is missing zero value checks for _destination, _amount.
- Line 142 retry is missing zero-value check for amount.

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended to review the above list of functions and add checks where applicable.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commit cfc13a8 by adding checks where appropriate.

3.7 (HAL-07) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL

Description:

Shortening the revert strings to fit within 32 bytes will decrease deployment time gas and reduce runtime gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional mstore, along with additional overhead to calculate memory offset, etc.

Recommendation:

Shorten the revert strings to fit within 32 bytes. That will affect gas optimization.

Remediation Plan:

ACKNOWLEDGED: The DAMfinance team acknowledged this finding.

3.8 (HAL-08) STATE VARIABLES MISSING IMMUTABLE MODIFIER - INFORMATIONAL

Description:

State variables can be declared as constant or immutable. In both cases, the variables cannot be modified after the contract has been constructed. For constant variables, the value has to be fixed at compile-time, while for immutable, it can still be assigned at construction time.

The following state variables are missing the immutable modifier:

dPrimeGuardian.sol
- Line 16: address public dPrimeContract;

dPrimeConnectorLZ.sol
- Line 19: address public dPrimeContract;

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

It is recommended to add the immutable modifier to the state variables mentioned.

Remediation Plan:

SOLVED: The DAMfinance team solved the issue in commits e1a2d28 and cfc13a8 by adding the immutable modifier to the state variables.

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the scoped contracts. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats, Slither was run on the all-scoped 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.

Slither results:

LayerZero/dPrimeConnectorLZ.sol

```
Reentrancy in dPrimeConnectorLZ._creditTo(uint16,address,uint256) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#45-48):
        External calls:
        - dPrimeLike(dPrimeContract).mint(_toAddress,_amount) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#46)
        Event emitted after the call(s):
        - MintLayerZero(_toAddress, _amount) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#47)
Reentrancy in dPrimeConnectorLZ._debitFrom(address,uint16,bytes,uint256) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#36-43):
        External calls:
        - require(bool,string)(dPrimeLike(dPrimeContract).decreaseAllowanceAdmin(_from,spender,_amount),dPrimeConnectorLZ/Must have proper allowance) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#39)
        - dPrimeLike(dPrimeContract).burn(_from,_amount) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#41)
        Event emitted after the call(s):
        - BurnLayerZero(_from, amount) (contracts/TO_AUDIT/LayerZero/dPrimeConnectorLZ.sol#42)
Reentrancy in OFTCore._send(address,uint16,bytes,uint256,address,address,bytes) (node_modules/@layerzerolabs/solidity-examples/contracts/token/oft/OFTCore.sol#53-62):
        External calls:
        - _lzSend(_dstChainId,lzPayload,_refundAddress,_zroPaymentAddress,_adapterParams,msg.value) (node_modules/@layerzerolabs/solidity-examples/contracts/token/oft/OFTCore.sol#59)
                - lzEndpoint.send{value: _nativeFee}(_dstChainId,trustedRemote,_payload,_refundAddress,_zroPaymentAddress,_adapterParams) (node_modules/@layerzerolabs/solidity-examples/contracts/lzApp/LzApp.sol#48)
        Event emitted after the call(s):
        - SendToChain(_dstChainId,_from,_toAddress,_amount) (node_modules/@layerzerolabs/solidity-examples/contracts/token/oft/OFTCore.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

hyperlane/dPrimeConnectorHyperlane.sol

```
Reentrancy in dPrimeConnectorHyperlane._handle(uint32,bytes32,bytes) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#115-133):
        External calls:
        - dPrimeLike(dPrimeContract).mint(recipient,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#126-131)
        State variables written after the call(s):
        - failedMessages[_origin][recipient][nonce] = amount (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#129)
        - nonce ++ (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#132)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2

Reentrancy in dPrimeConnectorHyperlane._handle(uint32,bytes32,bytes) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#115-133):
        External calls:
        - dPrimeLike(dPrimeContract).mint(recipient,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#126-131)
        Event emitted after the call(s):
        - FailedTransferRemote(_origin,recipient,nonce,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#130)
        - ReceivedTransferRemote(_origin,recipient,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#127)
Reentrancy in dPrimeConnectorHyperlane.retry(uint32,address,uint256) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#141-149):
        External calls:
        - dPrimeLike(dPrimeContract).mint(_recipient,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#144-148)
        Event emitted after the call(s):
        - FailedTransferRemote(_origin,_recipient,nonce,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#147)
        - ReceivedTransferRemote(_origin,_recipient,amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#145)
Reentrancy in dPrimeConnectorHyperlane.transferRemote(uint32,address,uint256) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#95-107):
        External calls:
        - dPrimeLike(dPrimeContract).burn(msg.sender,_amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#100)
        - _dispatchWithGas(_destination,abi.encode(_recipient,_amount),msg.value) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#101-105)
                - _outbox.dispatch(_destinationDomain,_router,_msg) (node_modules/@hyperlane-xyz/app/contracts/Router.sol#192)
                - interchainGasPaymaster.payGasFor{value: _gasPayment}(address(_outbox),_leafIndex,_destinationDomain) (node_modules/@hyperlane-xyz/app/contracts/Router.sol#167-171)
        External calls sending eth:
        - _dispatchWithGas(_destination,abi.encode(_recipient,_amount),msg.value) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#101-105)
                - interchainGasPaymaster.payGasFor{value: _gasPayment}(address(_outbox),_leafIndex,_destinationDomain) (node_modules/@hyperlane-xyz/app/contracts/Router.sol#167-171)
        Event emitted after the call(s):
        - SentTransferRemote(_destination,_recipient,_amount) (contracts/TO_AUDIT/hyperlane/dPrimeConnectorHyperlane.sol#106)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

AUTOMATED TESTING

dPrime.sol

```
dPrime.permit(address,address,uint256,uint256,uint8,bytes32,bytes32) (contracts/TO_AUDIT/dPrime.sol#210-234) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(block.timestamp <= deadline,dPrime/permit-expired) (contracts/TO_AUDIT/dPrime.sol#211)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

dPrimeGuardian.sol

```
dPrimeGuardian.pipeAddresses (contracts/TO_AUDIT/dPrimeGuardian.sol#15) is never initialized. It is used in:
        - dPrimeGuardian.removeConnectorAdmin(bytes32) (contracts/TO_AUDIT/dPrimeGuardian.sol#25-28)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-state-variables

dPrimeGuardian.constructor(address)._dPrimeContract (contracts/TO_AUDIT/dPrimeGuardian.sol#21) lacks a zero-check on :
                - dPrimeContract = _dPrimeContract (contracts/TO_AUDIT/dPrimeGuardian.sol#22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

Reentrancy in dPrimeGuardian.cageDPrime() (contracts/TO_AUDIT/dPrimeGuardian.sol#30-33):
        External calls:
        - dPrimeLike(dPrimeContract).cage(0) (contracts/TO_AUDIT/dPrimeGuardian.sol#31)
        Event emitted after the call(s):
        - CagedDPrime() (contracts/TO_AUDIT/dPrimeGuardian.sol#32)
Reentrancy in dPrimeGuardian.removeConnectorAdmin(bytes32) (contracts/TO_AUDIT/dPrimeGuardian.sol#25-28):
        External calls:
        - dPrimeLike(dPrimeContract).deny(pipeAddresses[pipeName]) (contracts/TO_AUDIT/dPrimeGuardian.sol#26)
        Event emitted after the call(s):
        - HaltedPipe(pipeName) (contracts/TO_AUDIT/dPrimeGuardian.sol#27)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```

- No major issues were found by Slither.
- The reentrancy vulnerabilities are all false positives.

4.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues, and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on all the contracts and sent the compiled results to the analyzers to locate any vulnerabilities.

MythX results:

dependencies/AuthAdmin.sol

Report for contracts/lmcv/dependencies/AuthAdmin.sol
https://dashboard.mythx.io/#/console/analyses/501d6c07-8baa-4de7-9a89-eb226a919c63

Line	SWC Title	Severity	Short Description
3	(SWC-103) Floating Pragma	Low	A floating pragma is set.

hyperlane/dPrimeConnectorHyperlane.sol

Report for contracts/lmcv/hyperlane/dPrimeConnectorHyperlane.sol
https://dashboard.mythx.io/#/console/analyses/3dbc097e-6402-480b-8d93-57ab8decf4bb

Line	SWC Title	Severity	Short Description
19	(SWC-123) Requirement Violation	Low	Requirement violation.
22	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
100	(SWC-123) Requirement Violation	Low	Requirement violation.
126	(SWC-104) Unchecked Call Return Value	Medium	Unchecked return value from external call.
129	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
129	(SWC-107) Reentrancy	Low	Write to persistent state following external call
130	(SWC-107) Reentrancy	Low	Read of persistent state following external call
132	(SWC-107) Reentrancy	Low	Read of persistent state following external call.
144	(SWC-104) Unchecked Call Return Value	Medium	Unchecked return value from external call.
147	(SWC-107) Reentrancy	Low	Read of persistent state following external call.

dPrimeGuardian.sol

Report for dPrimeGuardian.sol
https://dashboard.mythx.io/#/console/analyses/75b50e57-566c-49de-b737-a0556750b908

Line	SWC Title	Severity	Short Description
26	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.
31	(SWC-107) Reentrancy	Low	A call to a user-supplied address is executed.

- No major issues were found by MythX.
- The reentrancy vulnerabilities and requirement violations are all false positives.

THANK YOU FOR CHOOSING

// HALBORN