// HALBORN

Planet Mojo - MojoToken

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **February 12th, 2024 - February 16th, 2024**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	02/12/2024
0.2	Document Updates	02/15/2024
0.3	Draft Review	02/15/2024
0.4	Draft Review	02/16/2024
1.0	Remediation Plan	02/19/2024
1.1	Remediation Plan Review	02/19/2024
1.2	Remediation Plan Review	02/20/2024
1.3	Remediation Plan Edits	02/21/2024
1.4	Remediation Plan Review	02/21/2024

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

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Planet Mojo engaged Halborn to conduct a security assessment on their smart contracts beginning on February 12th, 2024 and ending on February 16th, 2024. The security assessment was scoped to the smart contract provided to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to assessment 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 assessment is to:

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

In summary, Halborn identified some security risks that were mostly addressed by the Planet Mojo team. The main ones were the following :

1. `_beforeTokenTransfer` implementation blocks any value transference to every recipient between different chains: Due to the inherent architecture of LayerZero's bridging feature, assuring that a privileged user can send tokens among chains will need the use of LayerZero tools instead of ERC20 ones.
2. Add `Zero Check` to `setAllowedTokenSender`: is a good practice to avoid setting state addresses to zero address.
3. Use custom errors: Nowadays, replacing hard-coded revert messages with `Error()` syntax is highly recommended to reduce gas waste.

1.3 SCOPE

Code repositories:

1. Planet Mojo - MojoToken

- Repository: MojoToken
- **Commit ID :** 3b5681b56aec32768848e133669b4a79a14c25fa
- Smart contract in scope:
    1. MojoToken (contracts/token/erc20/MojoToken.sol)

Out-of-scope

- LayerZero libraries and dependencies.
- Economic attacks.

Remediation Commit ID :

- 5d9daac

Out-of-scope: New features/implementations after the remediation **commit ID**.

1.4 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 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.
- 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).

2. 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.

2.1 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:

Exploitability 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:

$$E = \prod m_e$$

2.2 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)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

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

2.3 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.

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

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	1	2

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MINTING TOKENS IS BLOCKED WHILE TARGET CONTRACT IS PAUSED	Low (3.1)	RISK ACCEPTED
(HAL-02) MISSING ZERO ADDRESS CHECK	Informational (0.0)	SOLVED - 02/19/2023
(HAL-03) USE CUSTOM ERRORS	Informational (0.0)	SOLVED - 02/19/2023

FINDINGS & TECH DETAILS

4.1 (HAL-01) MINTING TOKENS IS BLOCKED WHILE TARGET CONTRACT IS PAUSED - LOW (3.1)

Description:

As described in the code walkthrough by Planet Mojo team, main purpose of pausing the token contracts is to allow allowedTokenSender to make a distribution of the Initial Supply minted within the token.
If that initial distribution includes sending tokens to a different blockchain and its implementation of the token is already paused, every send made from the main contract will revert, even if allowedTokenSender is properly set.

If an OFT on the target chain is paused, minting will be disabled; therefore, transfer will revert.

Proof of Concept:

After burning origin chain MojoTokens, the target chain receives a transfer and ERC20's function _mint is called within _credit, therefore _from ==address(0):

- **areTransfersPaused=false**: works normally
- **areTransfersPaused=true**: right side of the || is checked but, you will never surpass _from == allowedTokenSender, therefore creation of tokens in target chains, is locked for everyone.

Several tests are provided in the **MANUAL TESTING** section showing different approaches to this problem.

Code Location:

Listing 1: contracts/token/MojoToken.sol

```
78     function _beforeTokenTransfer(
79         address _from,
80         address _to,
81         uint256 _amount
82     ) internal override {
83         require(!areTransfersPaused || (allowedTokenSender !=
↳ address(0) && _from == allowedTokenSender), "Transfers are paused"
↳ );
84         super._beforeTokenTransfer(_from, _to, _amount);
85     }
```

BVSS:

AO:A/AC:L/AX:L/C:C/I:N/A:C/D:N/Y:N/R:F/S:U (3.1)

Recommendation:

If sending tokens through LayerZero while paused is a must feature, a more convenient strategy must be developed in order to attain this goal. It is recommended to use LayerZero's native messaging system.

Remediation plan:

RISK ACCEPTED: The Planet Mojo team accepted the risk of the finding, Planet Mojo stated that they will not be sending tokens through LayerZero while the contract is paused.

4.2 (HAL-02) MISSING ZERO ADDRESS CHECK - INFORMATIONAL (0.0)

Description:

Adding Zero Check to setAllowedTokenSender(address _allowedTokenSender) will optimize gas costs because it will not be checked in every transfer call.

Code Location:

Listing 2: contracts/token/MojoToken.sol

```
1     function setAllowedTokenSender(address _allowedTokenSender)
↳ external onlyOwner {
2         allowedTokenSender = _allowedTokenSender;
3     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

It is recommended to follow next steps in order to improve MojoToken functionallity:

1. Remove allowedTokenSender != address(0) comparison from _beforeTokenTransfer
2. Include a “Zero check” in setAllowedTokenSender e.g.:

Listing 3: contracts/token/MojoToken.sol

```
1     error InvalidAddress();
2
```

```
3     function setAllowedTokenSender(address _allowedTokenSender)
↳ external onlyOwner {
4         if (_allowedTokenSender == address(0)) {
5             revert InvalidAddress();
6         }
7         allowedTokenSender = _allowedTokenSender;
8     }
```

Remediation plan:

SOLVED: The Planet Mojo team solved this issue in commit ID 5d9daac by moving the zero check from _beforeTokenTransfer to setAllowedTokenSender.

4.3 (HAL-03) USE CUSTOM ERRORS - INFORMATIONAL (0.0)

Description:

In Solidity smart contract development, replacing hard-coded revert message strings with the Error() syntax is an optimization strategy that can significantly reduce gas costs. Hard-coded strings, stored on the blockchain, increase the size and cost of deploying and executing contracts.

The Error() syntax allows for the definition of reusable, parameterized custom errors, leading to a more efficient use of storage and reduced gas consumption. This approach not only optimizes gas usage during deployment and interaction with the contract but also enhances code maintainability and readability by providing clearer, context-specific error information.

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:C (0.0)

Recommendation:

It is recommended to replace hard-coded revert strings in require statements for custom errors, which can be done following the logic below.

1. Standard require statement (to be replaced):

Listing 4

```
1 require(condition, "Condition not met");
```

2. Declare the error definition to state

Listing 5

```
1 error ConditionNotMet();
```

3. As currently is not possible to use custom errors in combination with `require` statements, the standard syntax is:

Listing 6

```
1 if (!condition) revert ConditionNotMet();
```

More information about this topic in Official Solidity Documentation.

Remediation plan:

SOLVED: The Planet Mojo team solved this issue in commit ID 5d9daac by replacing hard-coded revert messages with `Error()` syntax.

MANUAL TESTING

After checking all Unit-Tests provided by Planet Mojo team, cross-chain transfers were checked.
All these tests are based on available OFT test in GitHub.

5.1 DEPLOYMENT TESTS AND TRANSFER OFT

Description:

It was tested in a setup having two different chains (ETH and MATIC) with MojoToken deployed and checking if a local transfer is possible in origin chain.

Test code:

Listing 7: contracts/BatchSSVTest.t.sol

```
    function setUp() public virtual override {
        console.log("OWNER ADDRESS:", address(this));
        vm.deal(userA, 1000 ether);
        vm.deal(userB, 1000 ether);

        super.setUp();
        setUpEndpoints(2, LibraryType.UltraLightNode);

        mojoETH = MojoToken(
            _deployOApp(type(MojoToken).creationCode, abi.encode(
↳ address(this), "MojoToken", "MojoToken", address(endpoints[ethEid
↳ ]), true))
        );

        mojoMATIC = MojoToken(
            _deployOApp(type(MojoToken).creationCode, abi.encode(
↳ address(this), "MojoToken", "MojoToken", address(endpoints[
↳ maticEid]), false))
        );

        mojoETH.setDelegate(address(this));
        mojoMATIC.setDelegate(address(this));
```

MANUAL TESTING

```
19
20          // config and wire the ofts
21          address[] memory ofts = new address[](2);
22          ofts[0] = address(mojoETH);
23          ofts[1] = address(mojoMATIC);
24          this.wireOApps(ofts);
25      }
26
27      function testConstructor() public {
28          assertEq(mojoETH.owner(), address(this));
29          assertEq(mojoMATIC.owner(), address(this));
30
31          assertEq(mojoETH.balanceOf(address(this)), totalSupply);
32          assertEq(mojoMATIC.balanceOf(address(this)), 0);
33
34          assertEq(mojoETH.token(), address(mojoETH));
35          assertEq(mojoMATIC.token(), address(mojoMATIC));
36      }
37
38      function testOftVersion() public {
39          bytes4 expectedId = 0x02e49c2c;
40
41          (bytes4 interfaceId, ) = mojoETH.oftVersion();
42          assertEq(interfaceId, expectedId);
43
44          (bytes4 interfaceIdMatic, ) = mojoMATIC.oftVersion();
45          assertEq(interfaceIdMatic, expectedId);
46      }
47
48      function _transferToUsers() internal {
49          mojoETH.transfer(userA, initialBalance);
50          mojoETH.transfer(userB, initialBalance);
51      }
52
53      function testTransferLocally() public {
54          _transferToUsers();
55
56          assertEq(mojoETH.balanceOf(userA), initialBalance);
57          assertEq(mojoETH.balanceOf(userB), initialBalance);
58          assertEq(mojoETH.balanceOf(address(this)), totalSupply - 2
↳  * initialBalance);
59      }
```

Results:

```
Running 3 tests for test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[PASS] testConstructor() (gas: 26032)
Traces:
  [26032] MojoTokenCrossChainTest::testConstructor()
    ├─ [2465] MojoToken::owner() [staticcall]
    │   └─ ← MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]
    ├─ [2465] MojoToken::owner() [staticcall]
    │   └─ ← MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]
    ├─ [2642] MojoToken::balanceOf(MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]) [staticcall]
    │   └─ ← 1000000000000000000000000000 [1e27]
    ├─ [2642] MojoToken::balanceOf(MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]) [staticcall]
    │   └─ ← 0
    ├─ [322] MojoToken::token() [staticcall]
    │   └─ ← MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7]
    ├─ [322] MojoToken::token() [staticcall]
    │   └─ ← MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240]
    └─ ← ()

[PASS] testOftVersion() (gas: 11256)
Traces:
  [11256] MojoTokenCrossChainTest::testOftVersion()
    ├─ [270] MojoToken::oftVersion() [staticcall]
    │   └─ ← 0x02e49c2c0000000000000000000000000000000000000000000000000000000000000000, 1
    ├─ [270] MojoToken::oftVersion() [staticcall]
    │   └─ ← 0x02e49c2c0000000000000000000000000000000000000000000000000000000000000000, 1
    └─ ← ()

[PASS] testTransferLocally() (gas: 75886)
Traces:
  [75886] MojoTokenCrossChainTest::testTransferLocally()
    ├─ [32002] MojoToken::transfer(0x0000000000000000000000000000000000000001, 10000000000000000000 [1e20])
    │   ├─ emit Transfer(from: MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], to: 0x000000000000
0000000000000000000000000001, value: 10000000000000000000 [1e20])
    │   └─ ← true
    ├─ [25202] MojoToken::transfer(0x0000000000000000000000000000000000000002, 10000000000000000000 [1e20])
    │   ├─ emit Transfer(from: MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], to: 0x000000000000
0000000000000000000000000002, value: 10000000000000000000 [1e20])
    │   └─ ← true
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000001) [staticcall]
    │   └─ ← 10000000000000000000 [1e20]
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000002) [staticcall]
    │   └─ ← 10000000000000000000 [1e20]
    ├─ [642] MojoToken::balanceOf(MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496]) [staticcall]
    │   └─ ← 999999800000000000000000 [9.999e26]
    └─ ← ()

Test result: ok. 3 passed; 0 failed; 0 skipped; finished in 6.50ms
```

5.2 SEND OFT

Description:

It was tested the function send does properly burn and mint tokens at origin and destiny.

Test code:

Listing 8: contracts/BatchSSVTest.t.sol

```
    function testSendOft() public {
        uint256 tokensToSend = 1 ether;
        _transferToUsers();
        bytes memory options = OptionsBuilder.newOptions().
↳ addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam,
↳ false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
        assertEq(mojoMATIC.balanceOf(userB), 0);

        vm.prank(userA);
        mojoETH.send{ value: fee.nativeFee }(sendParam, fee,
↳ payable(address(this)));
        verifyPackets(maticEid, addressToBytes32(address(mojoMATIC
↳ )));

        assertEq(mojoETH.balanceOf(userA), initialBalance -
↳ tokensToSend);
        assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
    }
```

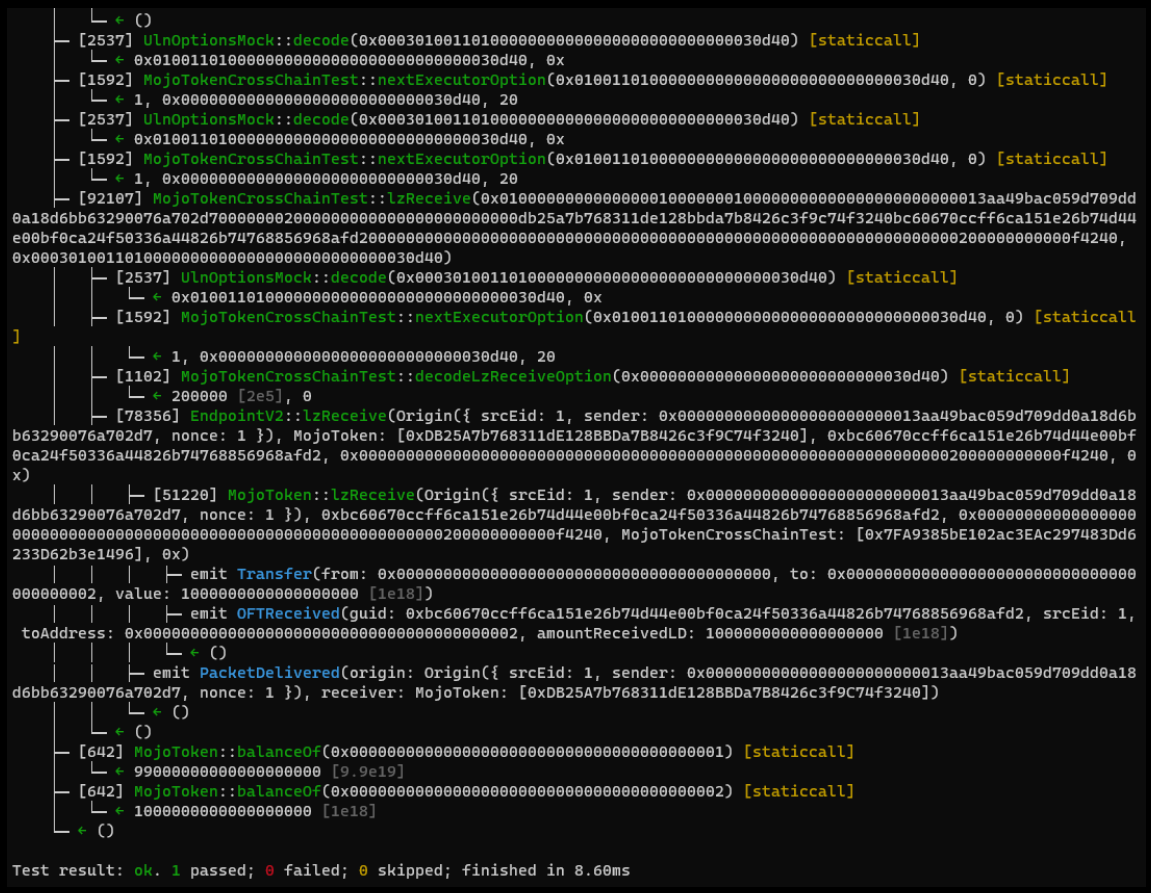
Results:

```
Running 1 test for test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[PASS] testSendOft() (gas: 793854)
Traces:
  [793854] MojoTokenCrossChainTest::testSendOft()
    ├─ [32002] MojoToken::transfer(0x0000000000000000000000000000000000000001, 10000000000000000000 [1e20])
    │   ├─ emit Transfer(from: MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], to: 0x000000000000000000000000000000000000000000000000000000000000001, value: 10000000000000000000 [1e20])
    │   └─ ← true
    ├─ [25202] MojoToken::transfer(0x0000000000000000000000000000000000000002, 10000000000000000000 [1e20])
    │   ├─ emit Transfer(from: MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], to: 0x000000000000000000000000000000000000000000000000000000000000002, value: 10000000000000000000 [1e20])
    │   └─ ← true
    ├─ [111444] MojoToken::quoteSend(SendParam({ dstEid: 2, to: 0x0000000000000000000000000000000000000000000000000000000000000002, amountLD: 100000000000000000 [1e18], minAmountLD: 100000000000000000 [1e18], extraOptions: 0x0003010011010000000000000000000000000030d40, composeMsg: 0x, oftCmd: 0x }), false) [staticcall]
    │   ├─ [97411] EndpointV2::quote(MessagingParams({ dstEid: 2, receiver: 0x000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, message: 0x0000000000000000000000000000000000000000000000000000000000000020000000000000f4240, options: 0x000301001101000000000000000000000000030d40, payInLzToken: false }), MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7]) [staticcall]
    │   │   ├─ [83637] SendUln302Mock::quote(Packet({ nonce: 1, srcEid: 1, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, dstEid: 2, receiver: 0x000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, message: 0x0000000000000000000000000000000000000000000000000000000000000000000020000000000000f4240 }), 0x0003010011010000000000000000000000000030d40, false) [staticcall]
    │   │   │   ├─ [30184] DVN::getFee(2, 100, MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7], 0x) [staticcall]
    │   │   │   │   ├─ [14543] DVNFeeLib::getFee(FeeParams({ priceFeed: 0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9, dstEid: 2, confirmations: 100, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, quorum: 1, defaultMultiplierBps: 12000 [1.2e4] }), DstConfig({ gas: 5000, multiplierBps: 10000 [1e4], floorMarginUSD: 10000000000 [1e10] }), 0x) [staticcall]
    │   │   │   │   │   ├─ [8488] PriceFeed::estimateFeeByEid(2, 452, 5000) [staticcall]
    │   │   │   │   │   │   └─ ← 5452, 10000000000000000000 [1e20], 10000000000000000000 [1e20], 0
    │   │   │   │   │   └─ ← 5452
    │   │   │   │   └─ ← 5452
    │   │   │   ├─ [25215] Executor::getFee(2, MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7], 40, 0x010011010000000000000000000000030d40) [staticcall]
    │   │   │   │   ├─ [7569] ExecutorFeeLibMock::getFee(FeeParams({ priceFeed: 0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9, dstEid: 2, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, calldataSize: 40, defaultMultiplierBps: 12000 [1.2e4] }), DstConfig({ baseGas: 5000, multiplierBps: 10000 [1e4], floorMarginUSD: 10000000000 [1e10], nativeCap: 100000000000000000 [1e17] }), 0x010011010000000000000000000000030d40) [staticcall]
    │   │   │   │   │   ├─ [2488] PriceFeed::estimateFeeByEid(2, 40, 205000 [2.05e5]) [staticcall]
    │   │   │   │   │   │   └─ ← 205040 [2.05e5], 10000000000000000000 [1e20], 10000000000000000000 [1e20], 0
    │   │   │   │   │   └─ ← 205040 [2.05e5]
    │   │   │   │   └─ ← 205040 [2.05e5]
    │   │   │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    │   │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000001) [staticcall]
    │   └─ ← 10000000000000000000 [1e20]
    ├─ [2642] MojoToken::balanceOf(0x0000000000000000000000000000000000000002) [staticcall]
    │   └─ ← 0
```


[0] VM::prank(0x0000000000000000000000000000000000000001)
[364671] MojoToken::send{value: 210492}(SendParam({ dstEid: 2, to: 0x0000000000000000000000000000000000000000000000000000000000000002, amountLD: 100000000000000000 [1e18], minAmountLD: 100000000000000000 [1e18], extraOptions: 0x0003010011010000000000000000000000000030d40, composeMsg: 0x, oftCmd: 0x }), MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 }), MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
emit Transfer(from: 0x0000000000000000000000000000000000000001, to: 0x0000000000000000000000000000000000000000, value: 100000000000000000 [1e18])
[341157] EndpointV2::send{value: 210492}(MessagingParams({ dstEid: 2, receiver: 0x000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, message: 0x0000000000000000000000000000000000000000000000000000000000000020000000000000f4240, options: 0x0003010011010000000000000000000000000030d40, payInLzToken: false }), MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
[294781] SendUln302Mock::send(Packet({ nonce: 1, srcEid: 1, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, dstEid: 2, receiver: 0x000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, message: 0x0000000000000000000000000000000000000000000000000000000000000000000200000000000f4240 }), 0x0003010011010000000000000000000000000030d40, false)
[12906] Executor::assignJob(2, MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7], 40, 0x010011010000000000000000000000000030d40)
[7609] ExecutorFeeLibMock::getFeeOnSend(FeeParams({ priceFeed: 0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9, dstEid: 2, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, calldataSize: 40, defaultMultiplierBps: 12000 [1.2e4] }), DstConfig({ baseGas: 5000, multiplierBps: 10000 [1e4], floorMarginUSD: 10000000000 [1e10], nativeCap: 1000000000000000000 [1e17] }), 0x010011010000000000000000000000000030d40)
[2437] PriceFeed::estimateFeeOnSend(2, 40, 205000 [2.05e5])
← 205040 [2.05e5], 10000000000000000000 [1e20], 10000000000000000000 [1e20], 0
← 205040 [2.05e5]
← 205040 [2.05e5]
emit ExecutorFeePaid(executor: Executor: [0x1d1499e622D69689cdf9004d05Ec547d650Ff211], fee: 205040 [2.05e5])
[12301] DVN::assignJob(AssignJobParam({ dstEid: 2, packetHeader: 0x0100000000000000010000000100000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, payloadHash: 0x094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8480e5cf0347c3eb5, confirmations: 100, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7 }), 0x)
[6015] DVNFeeLib::getFeeOnSend(FeeParams({ priceFeed: 0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576A9, dstEid: 2, confirmations: 100, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, quorum: 1, defaultMultiplierBps: 12000 [1.2e4] }), DstConfig({ gas: 5000, multiplierBps: 10000 [1e4], floorMarginUSD: 10000000000 [1e10] }), 0x)
[2437] PriceFeed::estimateFeeOnSend(2, 452, 5000)
← 5452, 10000000000000000000 [1e20], 10000000000000000000 [1e20], 0
← 5452
← 5452
emit DVNFeePaid(requiredDVNs: [0x03A6a84cD762D9707A21605b548aaaB891562aAb], optionalDVNs: [], fees: [5452])
[202301] MojoTokenCrossChainTest::schedulePacket(0x0100000000000000010000000100000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000020000000000000f4240, 0x0003010011010000000000000000000000000030d40)
← ()
← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 }), 0x0100000000000000010000000100000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000

```
000000000000002000000000f4240
    │   │   ├─ emit PacketSent(encodedPayload: 0x01000000000000000100000001000000000000000000000000000013aa49bac059d709dd0a1
8d6bb63290076a702d70000002000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44e00
bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000000000000000002000000000f4240, opt
ions: 0x0003010011010000000000000000000000000030d40, sendLibrary: SendUln302Mock: [0xc7183455a4C133Ae270771860664b6B7ec
320bB1])
    │   │   ├─ [55] SendUln302Mock::receive{value: 210492}()
    │   │   │   └─ ← ()
    │   │   └─ ← MessagingReceipt({ guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, nonce: 1,
fee: MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 }) })
    │   ├─ emit OFTSent(guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, dstEid: 2, fromAddress
: 0x0000000000000000000000000000000000000001, amountSentLD: 100000000000000000 [1e18], amountReceivedLD: 10000000000000
00000 [1e18])
    │   └─ ← MessagingReceipt({ guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, nonce: 1, fee:
 MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 }) }), OFTReceipt({ amountSentLD: 100000000000000000 [1e18],
 amountReceivedLD: 100000000000000000 [1e18] })
    ├─ [875] MojoTokenCrossChainTest::assertGuid(0x01000000000000001000000010000000000000000000000013aa49bac059d709dd0
a18d6bb63290076a702d70000002000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44e
00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000000000000000002000000000f4240, 0
xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2) [staticcall]
    │   └─ ← ()
    ├─ [159096] MojoTokenCrossChainTest::validatePacket(0x01000000000000001000000010000000000000000000000013aa49bac059
d709dd0a18d6bb63290076a702d70000002000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26
b74d44e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000000000000000002000000000f
4240)
    │   ├─ [5098] EndpointV2::getReceiveLibrary(MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 1) [staticcall]
    │   │   └─ ← ReceiveUln302: [0x212224D2F2d262cd093eE13240ca4873fcCBbA3C], true
    │   ├─ [270] ReceiveUln302::version() [staticcall]
    │   │   └─ ← 3, 0, 2
    │   ├─ [12473] ReceiveUln302::getConfig(1, MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 2) [staticcall]
    │   │   └─ ← 0x0000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000064000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000c0
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001
0000000000000000000000000000000000000000000000000000000000000001000000000000000000000000d16d567549a2a2a2005aeacf7fb193851603dd7000000
0000000000000000000000000000000000000000000000000000000000000000
    │   ├─ [1209] DVN::hashCallData(2, ReceiveUln302: [0x212224D2F2d262cd093eE13240ca4873fcCBbA3C], 0x0223536e0000000000
0000000000000000000000000000000000000000000000000000000060094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8480e5cf0347c3eb500
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000064
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000510100000000000000010000000100000000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d70000002000000000000000000
00000000db25a7b768311de128bbda7b8426c3f9c74f32400000000000000000000000000000000, 1001) [staticcall]
    │   │   └─ ← 0x9215655dea10e13d784edf42baa5b95cfbfc3a849bfd7a2d817f07fe160a268b
    │   ├─ [0] VM::sign(1, 0x6a8b2ddaa443b655771da9b2495b94b075b27ad2610d841c157783385b5adebd) [staticcall]
    │   │   └─ ← 27, 0xb3125536dff25ece74ddf0072ff8a84e85dea23f492fc6c1e6f9a5b0d8e6afcf, 0x2dc48195690fde7395b5107a51a41
02ffe4a0de5a6c347a81dc82e4d42d4bfe8
    │   ├─ [41637] DVN::execute([ExecuteParam({ vid: 2, target: 0x212224D2F2d262cd093eE13240ca4873fcCBbA3C, callData: 0x
0223536e0000000000000000000000000000000000000000000000000000000000000060094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8
480e5cf0347c3eb500000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000064
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000510100000000000000010000000100000000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d7000000
0200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f32400000000000000000000000000000000, expiration: 1001, sign
atures: 0xb3125536dff25ece74ddf0072ff8a84e85dea23f492fc6c1e6f9a5b0d8e6afcf2dc48195690fde7395b5107a51a4102ffe4a0de5a6c347
```

```
a81dc82e4d42d4bfe81b })])
    │   │   ├─ [3000] PRECOMPILES::ecrecover(0x6a8b2ddaa443b655771da9b2495b94b075b27ad2610d841c157783385b5adebd, 27, 809
96391271431638076744877452922489647982301008334005724437885799946374262735, 207012745643186589266436180891547648841255384621265835762355342202229209661 20) [staticcall]
    │   │   │   └─ ← 0x00000000000000000000000007e5f4552091a69125d5dfcb7b8c2659029395bdf
    │   │   ├─ [26506] ReceiveUln302::verify(0x010000000000000001000000010000000000000000000000013aa49bac059d709dd0a18d
6bb63290076a702d70000000200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, 0x094e3837f93343ff9b55eb4a174
3debd8599c4e32b3429c8480e5cf0347c3eb5, 100)
    │   │   │   ├─ emit PayloadVerified(dvn: DVN: [0xD16d567549A2a2a2005aEACf7fB193851603dd70], header: 0x01000000000000
0001000000010000000000000000000013aa49bac059d709dd0a18d6bb63290076a702d70000000200000000000000000000db25a7b76831
1de128bbda7b8426c3f9c74f3240, confirmations: 100, proofHash: 0x094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8480e5cf03
47c3eb5)
    │   │   │   │   └─ ← ()
    │   │   │   └─ ← ()
    │   │   ├─ [1197] DVN::hashCallData(2, ReceiveUln302: [0x212224D2F2d262cd093eE13240ca4873fcCBbA3C], 0x0894edf10000000000
0000000000000000000000000000000000000000000000000040094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8480e5cf0347c3eb500
0000000000000000000000000000000000000000000000000000000000005101000000000000001000000010000000000000000000000013aa49ba
c059d709dd0a18d6bb63290076a702d70000000200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240000000000000000000
00000000000000, 1001) [staticcall]
    │   │   │   └─ ← 0xfe88e19990b703f3f561cf202c6e4b343a8907425f1d68739b35b628fb1c0e2d
    │   │   ├─ [0] VM::sign(1, 0x152122db4b5e892ed00c7dabe796ae451f10a4eaa4daaf8ab1e3a66f8159709b) [staticcall]
    │   │   │   └─ ← 28, 0x435759a34eb8023f7af1bfccfca2442dcb76b74dcc1d3ddd9423a781f64323f1, 0x36878a5cd57535ab267c9e8238881
672c5b4a001ee2c16aff17ba98a9cb01195
    │   ├─ [71671] DVN::execute([ExecuteParam({ vid: 2, target: 0x212224D2F2d262cd093eE13240ca4873fcCBbA3C, callData: 0x
0894edf10000000000000000000000000000000000000000000000000000000000040094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8
480e5cf0347c3eb5000000000000000000000000000000000000000000000000000000000000510100000000000000100000001000000000000000
000000000013aa49bac059d709dd0a18d6bb63290076a702d70000000200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f32
4000000000000000000000000000000, expiration: 1001, signatures: 0x435759a34eb8023f7af1bfccfca2442dcb76b74dcc1d3ddd9423a7
81f64323f136878a5cd57535ab267c9e8238881672c5b4a001ee2c16aff17ba98a9cb011951c })])
    │   │   ├─ [3000] PRECOMPILES::ecrecover(0x152122db4b5e892ed00c7dabe796ae451f10a4eaa4daaf8ab1e3a66f8159709b, 28, 304
592952079007941355501539980477328650840819225352384120497985586806529156 97, 24664373121032152708063538873965717900827976
311316803589610752002679017353 17) [staticcall]
    │   │   │   └─ ← 0x00000000000000000000000007e5f4552091a69125d5dfcb7b8c2659029395bdf
    │   │   ├─ [40136] ReceiveUln302::commitVerification(0x010000000000000001000000010000000000000000000000013aa49bac05
9d709dd0a18d6bb63290076a702d70000000200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240, 0x094e3837f93343f
f9b55eb4a1743debd8599c4e32b3429c8480e5cf0347c3eb5)
    │   │   │   ├─ [33188] EndpointV2::verify(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a
18d6bb63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0x094e3837f93343ff9b55eb4a17
43debd8599c4e32b3429c8480e5cf0347c3eb5)
    │   │   │   │   ├─ [2867] MojoToken::allowInitializePath(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa4
9bac059d709dd0a18d6bb63290076a702d7, nonce: 1 })) [staticcall]
    │   │   │   │   │   └─ ← true
    │   │   │   │   ├─ emit PacketVerified(origin: Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d70
9dd0a18d6bb63290076a702d7, nonce: 1 }), receiver: MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], payloadHash:
0x094e3837f93343ff9b55eb4a1743debd8599c4e32b3429c8480e5cf0347c3eb5)
    │   │   │   │   └─ ← ()
    │   │   │   └─ ← ()
    │   │   └─ ← ()
```



```
    │   └─ ← ()
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000030d40, 20
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000030d40, 20
    ├─ [92107] MojoTokenCrossChainTest::lzReceive(0x010000000000000010000000100000000000000000000000013aa49bac059d709dd
0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44
e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000200000000000f4240,
0x0003010011010000000000000000000000000030d40)
    │   ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 0x0100110100000000000000000000000030d40, 0x
    │   ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000030d40, 0) [staticcall
]
    │   │   └─ ← 1, 0x00000000000000000000000030d40, 20
    │   ├─ [1102] MojoTokenCrossChainTest::decodeLzReceiveOption(0x00000000000000000000000030d40) [staticcall]
    │   │   └─ ← 200000 [2e5], 0
    │   ├─ [78356] EndpointV2::lzReceive(Origin({ srcEid: 1, sender: 0x000000000000000000000013aa49bac059d709dd0a18d6b
b63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0xbc60670ccff6ca151e26b74d44e00bf
0ca24f50336a44826b74768856968afd2, 0x000000000000000000000000000000000000000000000000000000000000000200000000000f4240, 0
x)
    │   │   ├─ [51220] MojoToken::lzReceive(Origin({ srcEid: 1, sender: 0x000000000000000000000013aa49bac059d709dd0a18
d6bb63290076a702d7, nonce: 1 }), 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, 0x0000000000000000
0000000000000000000000000000000000000000000000000200000000000f4240, MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6
233D62b3e1496], 0x)
    │   │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x0000000000000000000000000000000000
000002, value: 100000000000000000 [1e18])
    │   │   │   ├─ emit OFTReceived(guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, srcEid: 1,
 toAddress: 0x0000000000000000000000000000000000000002, amountReceivedLD: 100000000000000000 [1e18])
    │   │   │   └─ ← ()
    │   │   ├─ emit PacketDelivered(origin: Origin({ srcEid: 1, sender: 0x000000000000000000000013aa49bac059d709dd0a18
d6bb63290076a702d7, nonce: 1 }), receiver: MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240])
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000001) [staticcall]
    │   └─ ← 9900000000000000000 [9.9e19]
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000002) [staticcall]
    │   └─ ← 100000000000000000 [1e18]
    └─ ← ()

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 8.60ms
```

5.3 SEND OFT FROM NOT ALLOWED PAUSED TO UNPAUSED CONTRACT

Description:

With this test, it is confirmed that a non-allowed address is incapable of sending MojoTokens to the target chain when the origin contract is paused.

Test code:

Listing 9: contracts/BatchSSVTest.t.sol

```
    function testSendOftFromNotAllowedPausedContractToUnpausedContract() public {
        uint256 tokensToSend = 1 ether;

        _transferToUsers();

        assertEq(mojoETH.allowedTokenSender(), address(0));

        mojoETH.pauseTransfers();
        assertEq(mojoETH.areTransfersPaused(), true);

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam, false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
        assertEq(mojoMATIC.balanceOf(userB), 0);
```

```
25
26          vm.prank(userA);
27          mojoETH.send{ value: fee.nativeFee }(sendParam, fee,
↳ payable(address(this)));
28
29          verifyPackets(maticEid, addressToBytes32(address(mojoMATIC
↳ )));
30
31          assertEq(mojoETH.balanceOf(userA), initialBalance -
↳ tokensToSend);
32          assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
33      }
```

Results:

```
    │   │   │   ├─ [25215] Executor::getFee(2, MojoToken: [0x13aa49bAc059d709dd0a18D6bb63290076a702D7], 40, 0x0100110100
0000000000000000000000000030d40) [staticcall]
    │   │   │   │   ├─ [7569] ExecutorFeeLibMock::getFee(FeeParams({ priceFeed: 0x5991A2dF15A8F6A256D3Ec51E99254Cd3fb576
A9, dstEid: 2, sender: 0x13aa49bAc059d709dd0a18D6bb63290076a702D7, calldataSize: 40, defaultMultiplierBps: 12000 [1.2e4]
 }), DstConfig({ baseGas: 5000, multiplierBps: 10000 [1e4], floorMarginUSD: 10000000000 [1e10], nativeCap: 1000000000000
00000 [1e17] }), 0x0100110100000000000000000000000000030d40) [staticcall]
    │   │   │   │   │   │   ├─ [2488] PriceFeed::estimateFeeByEid(2, 40, 205000 [2.05e5]) [staticcall]
    │   │   │   │   │   │   │   └─ ← 205040 [2.05e5], 10000000000000000000 [1e20], 10000000000000000000 [1e20], 0
    │   │   │   │   │   │   └─ ← 205040 [2.05e5]
    │   │   │   │   │   └─ ← 205040 [2.05e5]
    │   │   │   │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    │   │   │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    │   │   └─ ← MessagingFee({ nativeFee: 210492 [2.104e5], lzTokenFee: 0 })
    │   ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000001) [staticcall]
    │   │   └─ ← 10000000000000000000 [1e20]
    │   ├─ [2642] MojoToken::balanceOf(0x0000000000000000000000000000000000000002) [staticcall]
    │   │   └─ ← 0
    │   ├─ [0] VM::prank(0x0000000000000000000000000000000000000001)
    │   │   └─ ← ()
    │   ├─ [1841] MojoToken::send{value: 210492}(SendParam({ dstEid: 2, to: 0x0000000000000000000000000000000000000000000000000000
00000000000002, amountLD: 100000000000000000 [1e18], minAmountLD: 100000000000000000 [1e18], extraOptions: 0x00030
100110100000000000000000000000030d40, composeMsg: 0x, oftCmd: 0x }), MessagingFee({ nativeFee: 210492 [2.104e5], lzTo
kenFee: 0 }), MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
    │   │   └─ ← revert: Transfers are paused
    │   └─ ← revert: Transfers are paused

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 6.95ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[FAIL. Reason: revert: Transfers are paused] testSendOftFromNotAllowedPausedContractToUnpausedContract() (gas: 217029)
```

5.4 SEND OFT FROM ALLOWED AND PAUSED TO UN-PAUSED CONTRACT

Description:

With this test, it is confirmed that an allowed address can send MojoTokens to the target chain even when the origin contract is paused.

Test code:

Listing 10: contracts/BatchSSVTest.t.sol

```
    function testSendOftFromAllowedPausedContractToUnpausedContract() public {
        uint256 tokensToSend = 1 ether;

        _transferToUsers();

        mojoETH.setAllowedTokenSender(userA);
        assertEq(mojoETH.allowedTokenSender(), userA);

        mojoETH.pauseTransfers();
        assertEq(mojoETH.areTransfersPaused(), true);

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam, false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
        assertEq(mojoMATIC.balanceOf(userB), 0);

```

```
27          vm.prank(userA);
28          mojoETH.send{ value: fee.nativeFee }(sendParam, fee,
↳ payable(address(this)));
29
30          verifyPackets(maticEid, addressToBytes32(address(mojoMATIC
↳ )));
31
32          assertEq(mojoETH.balanceOf(userA), initialBalance -
↳ tokensToSend);
33          assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
34      }
```

Results:

```
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [92107] MojoTokenCrossChainTest::lzReceive(0x010000000000000001000000010000000000000000000000000013aa49bac059d709dd
0a18d6bb63290076a702d7000000020000000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44
e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000200000000000f4240,
0x0003010011010000000000000000000000000030d40)
    │   ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    │   ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall
]
    │   │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    │   ├─ [1102] MojoTokenCrossChainTest::decodeLzReceiveOption(0x00000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 200000 [2e5], 0
    │   ├─ [78356] EndpointV2::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18d6b
b63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0xbc60670ccff6ca151e26b74d44e00bf
0ca24f50336a44826b74768856968afd2, 0x000000000000000000000000000000000000000000000000000000000000000200000000000f4240, 0
x)
    │   │   ├─ [51220] MojoToken::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18
d6bb63290076a702d7, nonce: 1 }), 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, 0x00000000000000000000
0000000000000000000000000000000000000000000000000000200000000000f4240, MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd6
233D62b3e1496], 0x)
    │   │   │   ├─ emit Transfer(from: 0x0000000000000000000000000000000000000000, to: 0x0000000000000000000000000000000000
000002, value: 100000000000000000 [1e18])
    │   │   │   ├─ emit OFTReceived(guid: 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, srcEid: 1,
 toAddress: 0x0000000000000000000000000000000000000002, amountReceivedLD: 100000000000000000 [1e18])
    │   │   │   └─ ← ()
    │   │   ├─ emit PacketDelivered(origin: Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18
d6bb63290076a702d7, nonce: 1 }), receiver: MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240])
    │   │   └─ ← ()
    │   └─ ← ()
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000001) [staticcall]
    │   └─ ← 9900000000000000000 [9.9e19]
    ├─ [642] MojoToken::balanceOf(0x0000000000000000000000000000000000000002) [staticcall]
    │   └─ ← 100000000000000000 [1e18]
    └─ ← ()

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 8.21ms
```

5.5 SEND OFT FROM UN-PAUSED TO PAUSED AND ALLOWED CONTRACT

Description:

With this test, it is confirmed that an allowed address cannot receive MojoTokens on the target chain while the origin contract is paused.

Test code:

Listing 11: contracts/BatchSSVTest.t.sol

```
    function testSendOftToPausedContractAllowedTokenSenderIsChosen() public {
        uint256 tokensToSend = 1 ether;
        address chosenOne = userA; // endpoints[1]

        _transferToUsers();

        mojoMATIC.setAllowedTokenSender(chosenOne);
        assertEq(mojoMATIC.allowedTokenSender(), chosenOne);

        mojoMATIC.pauseTransfers();
        assertEq(mojoMATIC.areTransfersPaused(), true);

        bytes memory options = OptionsBuilder.newOptions().addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam, false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
        assertEq(mojoMATIC.balanceOf(userB), 0);
```

```
        vm.prank(userA);
        mojoETH.send{ value: fee.nativeFee }(sendParam, fee, payable(address(this)));

        // vm.expectRevert();
        verifyPackets(maticEid, addressToBytes32(address(mojoMATIC)));

        assertEq(mojoETH.balanceOf(userA), initialBalance - tokensToSend);
        assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
    }
```

Results:

```
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x01001101000000000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x01001101000000000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x000000000000000000000000000030d40, 20
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x01001101000000000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x01001101000000000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x000000000000000000000000000030d40, 20
    ├─ [41757] MojoTokenCrossChainTest::lzReceive(0x0100000000000000001000000010000000000000000000000013aa49bac059d709dd
0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44
e00bf0ca24f50336a44826b74768856968afd200000000000000000000000000000000000000000000000000000000000000200000000000f4240,
0x0003010011010000000000000000000000000000030d40)
    │   ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 0x01001101000000000000000000000000000000030d40, 0x
    │   ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x01001101000000000000000000000000000000030d40, 0) [staticcall
]
    │   │   └─ ← 1, 0x000000000000000000000000000030d40, 20
    │   ├─ [1102] MojoTokenCrossChainTest::decodeLzReceiveOption(0x000000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 200000 [2e5], 0
    │   ├─ [28014] EndpointV2::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000000013aa49bac059d709dd0a18d6b
b63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0xbc60670ccff6ca151e26b74d44e00bf
0ca24f50336a44826b74768856968afd2, 0x000000000000000000000000000000000000000000000000000000000000200000000000f4240, 0
x)
    │   │   ├─ [3044] MojoToken::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000000013aa49bac059d709dd0a18d
6bb63290076a702d7, nonce: 1 }), 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, 0x000000000000000000
000000000000000000000000000000000000000000200000000000f4240, MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd62
33D62b3e1496], 0x)
    │   │   │   └─ ← revert: Transfers are paused
    │   │   └─ ← revert: Transfers are paused
    │   └─ ← revert: Transfers are paused
    └─ ← revert: Transfers are paused

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 8.54ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[FAIL. Reason: revert: Transfers are paused] testSendOftToPausedContractAllowedTokenSenderIsChosen() (gas: 814844)

Encountered a total of 1 failing tests, 0 tests succeeded
```

5.6 SEND OFT FROM UN-PAUSED TO PAUSED AND OWNER ALLOWED CONTRACT

Description:

With this test, it is confirmed that an allowed owner address cannot receive MojoTokens on the target chain while the target contract is paused.

Test code:

Listing 12: contracts/BatchSSVTest.t.sol

```
    function testSendOftToPausedContractAllowedTokenSenderIsOwner
↳ () public {
        uint256 tokensToSend = 1 ether;
        address chosenOne = address(this); // endpoints[1]

        _transferToUsers();

        mojoMATIC.setAllowedTokenSender(chosenOne);
        assertEq(mojoMATIC.allowedTokenSender(), chosenOne);

        mojoMATIC.pauseTransfers();
        assertEq(mojoMATIC.areTransfersPaused(), true);

        bytes memory options = OptionsBuilder.newOptions().
↳ addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam,
↳ false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
```

```
26          assertEq(mojoMATIC.balanceOf(userB), 0);
27
28          vm.prank(userA);
29          mojoETH.send{ value: fee.nativeFee }(sendParam, fee,
↳ payable(address(this)));
30
31          // vm.expectRevert();
32          verifyPackets(maticEid, addressToBytes32(address(mojoMATIC
↳ )));
33
34          assertEq(mojoETH.balanceOf(userA), initialBalance -
↳ tokensToSend);
35          assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
36      }
```

Results:

```
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [41757] MojoTokenCrossChainTest::lzReceive(0x010000000000000001000000010000000000000000000000000013aa49bac059d709dd
0a18d6bb63290076a702d7000000020000000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44
e00bf0ca24f50336a44826b74768856968afd2000000000000000000000000000000000000000000000000000000000000000200000000000f4240,
0x0003010011010000000000000000000000000030d40)
    │   ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    │   ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall
]
    │   │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    │   ├─ [1102] MojoTokenCrossChainTest::decodeLzReceiveOption(0x00000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 200000 [2e5], 0
    │   ├─ [28014] EndpointV2::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18d6b
b63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0xbc60670ccff6ca151e26b74d44e00bf
0ca24f50336a44826b74768856968afd2, 0x00000000000000000000000000000000000000000000000000000000000000200000000000f4240, 0
x)
    │   │   ├─ [3044] MojoToken::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18d
6bb63290076a702d7, nonce: 1 }), 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, 0x000000000000000000
0000000000000000000000000000000000000000000000200000000000f4240, MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd62
33D62b3e1496], 0x)
    │   │   │   └─ ← revert: Transfers are paused
    │   │   └─ ← revert: Transfers are paused
    │   └─ ← revert: Transfers are paused
    └─ ← revert: Transfers are paused

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 8.87ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[FAIL. Reason: revert: Transfers are paused] testSendOftToPausedContractAllowedTokenSenderIsOwner() (gas: 814744)
```

5.7 SEND OFT FROM UN-PAUSED TO PAUSED AND UNSET ALLOWANCE CONTRACT

Description:

With this test, it is confirmed that with `allowedTokenSender==0` nobody cannot receive `MojoTokens` on the target chain while the target contract is paused.

Test code:

Listing 13: contracts/BatchSSVTest.t.sol

```
    function testSendOftToPausedContractZeroAllowedTokenSender()
↳ public {
        uint256 tokensToSend = 1 ether;

        _transferToUsers();

        mojoMATIC.pauseTransfers();
        assertEq(mojoMATIC.areTransfersPaused(), true);

        assertEq(mojoMATIC.allowedTokenSender(), address(0));

        bytes memory options = OptionsBuilder.newOptions().
↳ addExecutorLzReceiveOption(200000, 0);
        SendParam memory sendParam = SendParam(
            maticEid,
            addressToBytes32(userB),
            tokensToSend,
            tokensToSend,
            options,
            "",
            ""
        );
        MessagingFee memory fee = mojoETH.quoteSend(sendParam,
↳ false);

        assertEq(mojoETH.balanceOf(userA), initialBalance);
        assertEq(mojoMATIC.balanceOf(userB), 0);

```

```
26          vm.prank(userA);
27          mojoETH.send{ value: fee.nativeFee }(sendParam, fee,
↳ payable(address(this)));
28          // vm.expectRevert();
29          verifyPackets(maticEid, addressToBytes32(address(mojoMATIC
↳ )));
30
31          assertEq(mojoETH.balanceOf(userA), initialBalance -
↳ tokensToSend);
32          assertEq(mojoMATIC.balanceOf(userB), tokensToSend);
33      }
```

Results:

```
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall]
    │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    ├─ [41608] MojoTokenCrossChainTest::lzReceive(0x0100000000000000010000001000000000000000000000000013aa49bac059d709dd
0a18d6bb63290076a702d70000000200000000000000000000000db25a7b768311de128bbda7b8426c3f9c74f3240bc60670ccff6ca151e26b74d44
e00bf0ca24f50336a44826b74768856968afd200000000000000000000000000000000000000000000000000000000000200000000000f4240,
0x0003010011010000000000000000000000000000030d40)
    │   ├─ [2537] UlnOptionsMock::decode(0x0003010011010000000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 0x0100110100000000000000000000000000030d40, 0x
    │   ├─ [1592] MojoTokenCrossChainTest::nextExecutorOption(0x0100110100000000000000000000000000030d40, 0) [staticcall
]
    │   │   └─ ← 1, 0x00000000000000000000000000030d40, 20
    │   ├─ [1102] MojoTokenCrossChainTest::decodeLzReceiveOption(0x00000000000000000000000000030d40) [staticcall]
    │   │   └─ ← 200000 [2e5], 0
    │   ├─ [27865] EndpointV2::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18d6b
b63290076a702d7, nonce: 1 }), MojoToken: [0xDB25A7b768311dE128BBDa7B8426c3f9C74f3240], 0xbc60670ccff6ca151e26b74d44e00bf
0ca24f50336a44826b74768856968afd2, 0x0000000000000000000000000000000000000000000000000000000000200000000000f4240, 0
x)
    │   │   ├─ [2895] MojoToken::lzReceive(Origin({ srcEid: 1, sender: 0x0000000000000000000000013aa49bac059d709dd0a18d
6bb63290076a702d7, nonce: 1 }), 0xbc60670ccff6ca151e26b74d44e00bf0ca24f50336a44826b74768856968afd2, 0x00000000000000000
0000000000000000000000000000000000000000000000200000000000f4240, MojoTokenCrossChainTest: [0x7FA9385bE102ac3EAc297483Dd62
33D62b3e1496], 0x)
    │   │   │   └─ ← revert: Transfers are paused
    │   │   └─ ← revert: Transfers are paused
    │   └─ ← revert: Transfers are paused
    └─ ← revert: Transfers are paused

Test result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 8.77ms

Ran 1 test suites: 0 tests passed, 1 failed, 0 skipped (1 total tests)

Failing tests:
Encountered 1 failing test in test/MojoTokenCrossChainTest.t.sol:MojoTokenCrossChainTest
[FAIL. Reason: revert: Transfers are paused] testSendOftToPausedContractZeroAllowedTokenSender() (gas: 813293)

Encountered a total of 1 failing tests, 0 tests succeeded
```

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

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.
The security team assessed all findings identified by the Slither software, however, findings with severity Information and Optimization are not included in the below results for the sake of report readability.

Results:

Slither results for MojoToken.sol	
Finding	**Impact**
MojoToken.constructor(address,string,string,address,bool)._owner (src/MojoToken.sol#21) shadows: - Ownable._owner (lib/openzeppelin/contracts/contracts/access/Ownable.sol#21) (state variable)	Low
MojoToken.constructor(address,string,string,address,bool)._symbol (src/MojoToken.sol#23) shadows: - ERC20._symbol (lib/openzeppelin/contracts/contracts/token/ERC20/ERC20.sol#43) (state variable)	Low
MojoToken.constructor(address,string,string,address,bool)._name (src/MojoToken.sol#22) shadows: - ERC20._name (lib/openzeppelin/contracts/contracts/token/ERC20/ERC20.sol#42) (state variable)	Low
MojoToken.setAllowedTokenSender(address)._allowedTokenSender (src/MojoToken.sol#56) lacks a zero-check on : - allowedTokenSender = _allowedTokenSender (src/MojoToken.sol#57)	Low
End of table for MojoToken.sol	

Findings obtained as a result of the Slither scan were reviewed. The majority of Slither findings were determined false-positives.

THANK YOU FOR CHOOSING

// HALBORN