Security Assessment Report

LayerZero Read

November 12, 2024

Summary

The Sec3 team (formerly Soteria) was engaged to conduct a thorough security analysis of the LayerZero Read smart contracts.

The artifact of the audit was the source code of the following contracts, excluding tests, from commit "56cc7e7" in a private repository.

CmdLib
packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLib1002.sol
packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLibBase.sol
packages/layerzero-v2/evm/messagelib/contracts/uln/interfaces/ILayerZeroReadDVN.sol
packages/layerzero-v2/evm/messagelib/contracts/interfaces/ILayerZeroReadExecutor.sol
DVN
packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/MultiSig.sol
packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/DVN.sol
OApp
packages/layerzero-v2/evm/oapp/contracts/oapp/libs/ReadCmdCodecV1.sol
packages/layerzero-v2/evm/oapp/contracts/oapp/OAppRead.sol
packages/layerzero-v2/evm/oapp/contracts/oapp/interfaces/IOAppComputer.sol

The initial audit revealed 9 issues or questions. This report provides a detailed description of the findings and their respective resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
LAYERZERO READ		
[L-01] Functions listed in "_shouldCheckHash" may not be idempotent	Low	Acknowledged
[L-02] Possible mismatches between DVNs and options	Low	Acknowledged
[L-03] MultiSig can be cross-contract or cross-chain replayed	Low	Acknowledged
[I-01] Possible values of "isBlockNum" can be more tightly restricted	Info	Acknowledged
[I-02] Consider reverting or returning offset if "_cmd" has trailing data	Info	Acknowledged
[I-03] Ensure "_evmCallRequests" is not empty	Info	Acknowledged
[I-04] Missing self-target check in DVN "execute"	Info	Acknowledged
[I-05] Opportunities for optimization	Info	Acknowledged
[Q-01] Does the endpoint ensure token transfer?	Question	Acknowledged

Findings in Detail

LAYERZERO READ

[L-01] Functions listed in "_shouldCheckHash" may not be idempotent

In the "execute" function of "DVN", if the "_shouldCheckHash" function returns "true", it will not use "usedHashes" to prevent replay attacks, based on the assumption that the two functions specified in the "_shouldCheckHash" function are idempotent.

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/DVN.sol */
176 | function execute(ExecuteParam[] calldata _params) external onlyRole(ADMIN_ROLE) {
177 |     for (uint256 i = 0; i < _params.length; ++i) {
178 |         ExecuteParam calldata param = _params[i];
199 |         // 4. should check hash
200 |         bool shouldCheckHash = _shouldCheckHash(bytes4(param.callData));
201 |         if (shouldCheckHash) {
202 |             if (usedHashes[hash]) {
203 |                 emit HashAlreadyUsed(param, hash);
204 |                 continue;
205 |             } else {
206 |                 usedHashes[hash] = true; // prevent reentry and replay attack
207 |             }
208 |         }
209 |
210 |         (bool success, bytes memory rtnData) = param.target.call(param.callData);

381 | /// to save gas, we don't check hash for some functions (where replaying won't change the state)
386 | function _shouldCheckHash(bytes4 _functionSig) internal pure returns (bool) {
387 |     // never check for these selectors to save gas
388 |     return
389 |         _functionSig != IReceiveUlnE2.verify.selector &&
                              // 0x0223536e, replaying won't change the state
390 |         _functionSig != ReadLib1002.verify.selector &&
                              // 0xab750e75, replaying won't change the state
391 |         _functionSig != ILayerZeroUltraLightNodeV2.updateHash.selector;
                              // 0x704316e5, replaying will be revert at uln
392 | }
```

However, the assumption may not hold in the following scenarios:

1. A collision may occur between Solidity function selectors, where two functions with different signatures could have the same selector.
2. "param.target" can be arbitrarily specified, and its implementation may not be idempotent.

In this context, it may be worth prioritizing robustness over gas savings.

Resolution

The team acknowledged this finding.

LAYERZERO READ

[L-02] Possible mismatches between DVNs and options

The "_assignDVNJobs" function matches "dvn" with its corresponding "options".

It loops through the combined list of required and optional DVNs from "_config". For each DVN in the loop, it checks the" dvnIds" array to find a matching index and retrieves the corresponding "options" from "optionsArray".

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLib1002.sol */
297 | function _assignDVNJobs(
298 |     ReadLibConfig memory _config,
299 |     address _sender,
300 |     bytes memory _packetHeader,
301 |     bytes calldata _cmd,
302 |     bytes memory _options
303 | ) internal returns (uint256 totalFee, uint256[] memory dvnFees) {
304 |     (bytes[] memory optionsArray, uint8[] memory dvnIds)
              = DVNOptions.groupDVNOptionsByIdx(_options);
305 |
306 |     uint8 dvnsLength = _config.requiredDVNCount + _config.optionalDVNCount;
307 |     dvnFees = new uint256[](dvnsLength);
308 |     for (uint8 i = 0; i < dvnsLength; ++i) {
309 |         address dvn = i < _config.requiredDVNCount
310 |             ? _config.requiredDVNs[i]
311 |             : _config.optionalDVNs[i - _config.requiredDVNCount];
312 |
313 |         bytes memory options = "";
314 |         for (uint256 j = 0; j < dvnIds.length; ++j) {
315 |             if (dvnIds[j] == i) {
316 |                 options = optionsArray[j];
317 |                 break;
318 |             }
319 |         }
320 |
321 |         dvnFees[i] = ILayerZeroReadDVN(dvn).assignJob(_sender, _packetHeader, _cmd, options);
322 |         if (dvnFees[i] > 0) {
323 |             fees[dvn] += dvnFees[i];
324 |             totalFee += dvnFees[i];
325 |         }
326 |     }
327 | }
```

The function assumes that the "dvnIds" (which map the "options" to specific DVNs) match the order of the DVNs as they appear in the combined list of "requiredDVNs" and "optionalDVNs". In other words, it assumes that "optionsArray[j]" corresponds to the options for the "dvnIds[j]"-th DVN in the combined list.

However, the function does not explicitly check whether the indices in "dvnIds" match the intended DVNs in the combined list.

Without validation, there is a risk of mismatches between DVNs and their corresponding options, which could result in job failures, incorrect fee calculations, or other errors.

Since the "send" function can only be called by the endpoint, the impact is limited. However, the endpoint must ensure that the "_options" passed in are correct, which falls outside the scope of our audit.

It would be beneficial to add a correspondence check between the "_options" and the DVNs.

Resolution

The team acknowledged this finding.

The team clarified that, from the protocol layer's perspective, if the provided fee is insufficient, the task will be rejected, causing the entire send transaction to revert. If the user does not follow the correct order, the result is simply a failed send transaction with no serious consequences. Any excess fee provided by the user will be refunded.

LAYERZERO READ

[L-03] MultiSig can be cross-contract or cross-chain replayed

Because the MultiSig contract does not implement the EIP-712 standard, signatures may be vulnerable to replay attacks across different contracts or chains.

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/MultiSig.sol */
114 | function _getEthSignedMessageHash(bytes32 _messageHash) internal pure returns (bytes32) {
115 |     return keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", _messageHash));
116 | }
```

To prevent cross-contract replay attacks, DVN uses "vid" as the verifier contract identifier.

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/DVN.sol */
144 | if (_param.vid != vid) {
145 |     revert DVN_InvalidVid(_param.vid);
146 | }
149 | bytes32 hash = hashCallData(_param.vid, _param.target, _param.callData, _param.expiration);
```

Following this approach, the "`vid`" of any DVN must be unique across all chains. If "`vid`" values are duplicated across different chains, the risk of signature replay between chains persists.

For example, signatures from "`DVN(vid=0)`" on Ethereum could be replayed on "`DVN(vid=0)`" on Arbitrum, and vice versa.

Similarly, if "`vid`" values are duplicated on the same chain, signatures could be replayed across contracts on that chain. For instance, signatures from "`DVN(vid=0)`" on Ethereum could be replayed on another "`DVN(vid=0)`" contract on Ethereum.

In fact, there are examples of DVNs with duplicate vids, as shown in the DVN addresses. For instance, the following DVNs on Ethereum share the same "`vid=101`":

- 01node(vid=101) on Ethereum
- Animoca-Blockdaemon(vid=101) on Ethereum
- BCW Group(vid=101) on Ethereum

This illustrates the risk of potential signature replay across contracts with duplicate vids on the same chain.

Consider implementing EIP-712 standard in DVN to replace the use of "`vid`" as a method to prevent cross-chain and cross-contract replay attacks.

Resolution

The team acknowledged this finding and clarified that each DVN should have distinct signers hosted by different parties.

LAYERZERO READ

[I-01] Possible values of "isBlockNum" can be more tightly restricted

In the "decodeEVMCallRequestV1" and "decodeEVMCallComputeV1" functions, the "isBlockNum" flag is set to "true" only if the input value is exactly 1.

```
/* packages/layerzero-v2/evm/oapp/contracts/oapp/libs/ReadCmdCodecV1.sol */
097 | function decodeEVMCallRequestV1(
098 |     bytes calldata _cmd,
099 |     uint256 _offset,
100 |     uint16 _appRequestLabel
101 | ) internal pure returns (EVMCallRequestV1 memory request, uint256 newOffset) {
109 |     request.isBlockNum = uint8(_cmd[newOffset]) == 1;

122 | function decodeEVMCallComputeV1(
123 |     bytes calldata _cmd,
124 |     uint256 _offset
125 | ) internal pure returns (EVMCallComputeV1 memory compute, uint256 newOffset) {
138 |     compute.isBlockNum = uint8(_cmd[newOffset]) == 1;
```

This may not align with user expectations. Users might mistakenly assume that any non-zero value (e.g., 123) would result in "true".

To avoid confusion, it's recommended to restrict the "isBlockNum" input to only accept values of 0 or 1 for clearer validation and user experience.

Resolution

The team acknowledged this finding.

LAYERZERO READ

[I-02] Consider reverting or returning offset if "_cmd" has trailing data

The "decode" function currently does not reject "_cmd" if trailing data remains after decoding.

```
/* packages/layerzero-v2/evm/oapp/contracts/oapp/libs/ReadCmdCodecV1.sol */
040 | function decode(
041 |     bytes calldata _cmd
042 | )
043 |     internal
044 |     pure
045 |     returns (...)
046 | {
047 |     uint256 offset = 0;
048 |     uint16 cmdVersion = uint16(bytes2(_cmd[offset:offset + 2]));
049 |     offset += 2;
050 |     if (cmdVersion != CMD_VERSION) revert InvalidVersion();
051 |
052 |     appCmdLabel = uint16(bytes2(_cmd[offset:offset + 2]));
053 |     offset += 2;
054 |
055 |     (evmCallRequests, offset) = decodeRequestsV1(_cmd, offset);
056 |
057 |     // decode the compute if it exists
058 |     if (offset < _cmd.length) {
059 |         (compute, ) = decodeEVMCallComputeV1(_cmd, offset);
060 |     }
061 | }
```

Consider reverting or returning the offset after parsing to prevent client encoding errors.

Resolution

The team acknowledged this finding.

LAYERZERO READ

[I-03] Ensure "_evmCallRequests" is not empty

The "encode" function in "CmdCodecV1" currently does not verify whether "_evmCallRequests" is empty.

```
/* packages/layerzero-v2/evm/oapp/contracts/oapp/libs/ReadCmdCodecV1.sol */
166 | function encode(
167 |     uint16 _appCmdLabel,
168 |     EVMCallRequestV1[] memory _evmCallRequests,
169 |     EVMCallComputeV1 memory _evmCallCompute
170 | ) internal pure returns (bytes memory) {
171 |     bytes memory cmd = encode(_appCmdLabel, _evmCallRequests);
172 |     if (_evmCallCompute.targetEid != 0) {
173 |         // if eid is 0, it means no compute
174 |         cmd = appendEVMCallComputeV1(cmd, _evmCallCompute);
175 |     }
176 |     return cmd;
177 | }
```

While there are constraints in the "LzReadCounter", this validation should also be implemented within the "CmdCodecV1".

```
/* packages/layerzero-v2/evm/oapp/contracts/oapp/examples/LzReadCounter.sol */
073 | function buildCmd(
074 |     uint16 appLabel,
075 |     EvmReadRequest[] memory _readRequests,
076 |     ComputeSetting memory _computeSetting
077 | ) public view returns (bytes memory) {
078 |     require(_readRequests.length > 0, "LzReadCounter: empty requests");
```

Resolution

The team acknowledged this finding.

LAYERZERO READ

[I-04] Missing self-target check in DVN "execute"

The "execute" function does not restrict "params.target" from being "address(this)".

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/dvn/DVN.sol */
176 | function execute(ExecuteParam[] calldata _params) external onlyRole(ADMIN_ROLE) {
177 |     for (uint256 i = 0; i < _params.length; ++i) {
178 |         ExecuteParam calldata param = _params[i];
179 |         // 1. skip if invalid vid
180 |         if (param.vid != vid) {
181 |             continue;
182 |         }
184 |         // 2. skip if expired
185 |         if (param.expiration <= block.timestamp) {
186 |             continue;
187 |         }
189 |         // generate and validate hash
190 |         bytes32 hash = hashCallData(param.vid, param.target, param.callData, param.expiration);
```

This can result in all signatures with "target == address(this)" being verified by both the "execute" function and the "quorumChangeAdmin" function simultaneously.

A potential attack scenario is as follows:

1. A user generates a signature with "target == address(DVN)" and prepares to call the "execute" function.
2. An attacker detects the transaction in the mempool and obtains the parameters.
3. The attacker uses the same parameters to call the "quorumChangeAdmin" function.
4. "usedHashes[hash]" is set to "true".
5. The user's original transaction becomes invalid due to the "HashAlreadyUsed" error.

Consider adding the following constraints in the "execute" function:

```
if (_param.target == address(this)) {
    revert DVN_InvalidTarget(_param.target);
}
```

Resolution

The team acknowledged this finding.

LAYERZERO READ

[I-05] Opportunities for optimization

1. Gas optimization - "i++" in the loop

```
/* packages/layerzero-v2/evm/oapp/contracts/oapp/libs/ReadCmdCodecV1.sol */
184 | for (uint256 i = 0; i < _evmCallRequests.length; i++) {
```

could be changed to:

```
for (uint256 i = 0; i < _evmCallRequests.length; ) {
    ...
    unchecked {
        ++i;
    }
}
```

2. Gas optimization - change "/ 2" to ">> 1"

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLibBase.sol */
030 | uint8 private constant MAX_COUNT = (type(uint8).max - 1) / 2;
```

could be changed to:

```
uint8 private constant MAX_COUNT = (type(uint8).max - 1) >> 1;
```

3. Avoid amplified rounding errors

```
/* packages/layerzero-v2/evm/messagelib/contracts/PriceFeed.sol */
295 | uint256 gasForL1CallData = ((_callDataSize * ARBITRUM_COMPRESSION_PERCENT) / 100) *
296 |     _arbitrumPriceExt.gasPerL1CallDataByte;
```

could be changed to:

```
uint256 gasForL1CallData = _callDataSize * ARBITRUM_COMPRESSION_PERCENT *
                           _arbitrumPriceExt.gasPerL1CallDataByte / 100;
```

Resolution

The team acknowledged this finding.

LAYERZERO READ

[Q-01] Does the endpoint ensure token transfer?

The "_assignDVNJobs" and "_payExecutor" functions only increase stored "fees" amount, without an actual transfer.

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLib1002.sol */
505 | function _payExecutor(
506 |     address _executor,
507 |     address _sender,
508 |     bytes memory _executorOptions
509 | ) internal returns (uint256 executorFee) {
510 |     executorFee = ILayerZeroReadExecutor(_executor).assignJob(_sender, _executorOptions);
511 |     if (executorFee > 0) {
512 |         fees[_executor] += executorFee;
513 |     }
514 |     emit ExecutorFeePaid(_executor, executorFee);
515 | }
```

But the "withdrawFee" function transfers token out from the contract itself.

```
/* packages/layerzero-v2/evm/messagelib/contracts/uln/readlib/ReadLib1002.sol */
169 | function withdrawFee(address _to, uint256 _amount) external {
170 |     uint256 fee = fees[msg.sender];
171 |     if (_amount > fee) revert LZ_CL_InvalidAmount(_amount, fee);
172 |     unchecked {
173 |         fees[msg.sender] = fee - _amount;
174 |     }
175 |
176 |     // transfers native if nativeToken == address(0x0)
177 |     address nativeToken = ILayerZeroEndpointV2(endpoint).nativeToken();
178 |     Transfer.nativeOrToken(nativeToken, _to, _amount);
179 |     emit NativeFeeWithdrawn(msg.sender, _to, _amount);
180 | }
```

So does the endpoint ensure token transfer with the correct amount when calling "send"?

Resolution

The team acknowledged this finding and clarified that the endpoint will transfer the fee to the message library.

Appendix: Methodology and Scope of Work

Assisted by the Sec3 Scanner developed in-house, the manual audit particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a Sec3 (the "Company") and LayerZero Labs Ltd. (the "Client''). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

The Sec3 audit team comprises a group of computer science professors, researchers, and industry veterans with extensive experience in smart contract security, program analysis, testing, and formal verification. We are also building automated security tools that incorporate static analysis, penetration testing, and formal verification.

At Sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.