OneSig PR 121
Security Review

Cantina Managed review by:

0xIcingdeath, Lead Security Researcher

Chris Smith, Lead Security Researcher

Anurag Jain, Security Researcher

May 20, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

LayerZero is a technology that enables applications to move data across blockchains, uniquely supporting censorship-resistant messages and permissionless development through immutable smart contracts.

From Apr 4th to Apr 7th the Cantina team conducted a review of OneSig[PR-121] on commit hash d44bc555. The team identified a total of **6** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	1	0	1
Gas Optimizations	3	0	3
Informational	2	2	0
Total	**6**	**2**	**4**

3 Findings

3.1 Low Risk

3.1.1 `executeTransaction` treats EOAs and Contracts the same when calling

Severity: Low Risk

Context: OneSig.sol#L166-L192

Description: After validating the transaction, `executeTransaction` calls to the `to` addresses with `value` and `calldata`. The primary purpose of this flow is to execute calls on smart contracts with the appropriate `data`. The function correctly checks the returned `bool success` value and reverts if the call does not return success.

Another expected flow for this sequence is that a `call` goes to an EOA and sends `value` (sending value to smart contracts is also possible). In this case, any `data` value is still used even though there will be no execution because the EOA does not have `data` to execute. Ethereum in this case still returns `true` for `success`. Since this is an expected flow, everything is operating normally so far.

There is the potential for misconfiguration and silent errors however. Since the `OneSig` system is meant to be used in a multichain environment, a call sequence on such as the one below may succeed on Ethereum, but fail on an L2 without reverting.

```
calls = [
  {to: address(contract-1), data: "setup()", value: 0},
  {to: address(contract-2), data: "receiveFee()", value 1},
  {to: address(EOA-3), data:"no-call", value 2}
]
```

On Ethereum, the system has been deployed and the two contracts are setup. However, on the L2 `contract-2` has yet to be deployed and is effectively still an EOA. Because of this the return on the call will be `success` but the intended code for `receiveFee` will not have been run.

This is preventable using off-chain systems and recoverable if the L2 contract can still be deployed at that address and the `receiveFee` data can be "recreated".

However, it is a potential concern and "hassle" because the intention of the transaction was not correctly executed and yet it did not revert.

This interaction can become even more complicated when the Ethereum Pectra upgrade launches. After that, EIP-7702 will be launched allowing EOAs to delegate code execution. In the call sequence above, if `EOA-3` does that without a payable function, then the call sequence **will revert**. Correcting this would require either EOA-3 undelegating or sending the ETH to another EOA.

Recommendation: It would be possible to more strictly process the calls to 1) smart contracts, 2) EOAs, 3) (eventually) EOAs with delegation. This would add code and complexity, so the team should evaluate whether that level of in-contract security is warranted.

At a minimum, ensuring these edge cases are documented and handled wherever feasible off-chain is encouraged to prevent hidden problems or locked funds.

LayerZero: This will be documented well for the contract and off-chain system, but this is not necessary and too complicated to change in the contract.

Cantina Managed: Agreed, Documenting these edge cases is the most important aspect.

3.2 Gas Optimization

3.2.1 Gas improvement with `executeTransaction` → `verifyTransactionProof` → `encodeLeaf` flow

Severity: Gas Optimization

Context: OneSig.sol#L176

Description: `verifyTransaction` currently `SLOADs` `nonce` just to pass it to `encodeLeaf`, then `executeTransaction` `SLOADs` `nonce` against to store `n` and update `nonce++`.

Recommendation: This could be streamlined by:

```
uint256 n = nonce++;

verifyTransactionProof(_merkleRoot, _transaction, n);

// ...

function verifyTransactionProof(bytes32 _merkleRoot, Transaction calldata _transaction, uint256 _nonce) public
↪  view {
        bytes32 leaf = encodeLeaf(_nonce, _transaction.calls);
```

If the external signature of `verifyTransactionProof` needs to stay the same, the pattern used in `MultiSig` for `verifySignatures` and `verifyNSignatures` could be followed with a public interface keeping the current signature and an internal one that receives the `_nonce` param.

LayerZero: We won't change at this point in this version, but could be included in a later version of this code.

Cantina Managed: Acknowledged.

3.2.2 Math optimization in `verifyNSignatures`

Severity: Gas Optimization

Context: MultiSig.sol#L188

Description: It is not necessary to perform two `MUL` operations for each signature.

Recommendation: This line could be:

```
uint256 sigStart = i * SIGNATURE_LENGTH;
bytes calldata signature = _signatures[sigStart:sigStart + SIGNATURE_LENGTH];
```

LayerZero: Since the code has been audited several times and the benefits are small, we won't change at this point, but we can consider for possible future versions.

Cantina Managed: Acknowledged.

3.2.3 `getSigners` is not called internally, could be `external`

Severity: Gas Optimization

Context: MultiSig.sol#L204

Description: `getSigners` is not currently called internally, so could save some gas by setting to `external` instead of `public`.

Recommendation:

```
function getSigners() external view returns (address[] memory)
```

LayerZero: This could be useful for future extensions on this contract so won't change.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 `verifySignatures` fails even when signature above threshold are provided

Severity: Informational

Context: MultiSig.sol#L181

Description: The threshold check in `verifySignatures` is overly restrictive. For instance, if `threshold = 3` and an admin submits a transaction that has been signed by 5 valid signers, this transaction would revert even though it "meets the threshold".

Recommendation: The check could be slightly relaxed by changing it to:

```
if (_signatures.length < _threshold * SIGNATURE_LENGTH || _signatures.length % SIGNATURE_LENGTH != 0) revert
↪  SignatureError();
```

LayerZero: Fixed in MultiSig.sol#L172.

Cantina Managed: The fix appropriately allows more than `threshold` signatures to be submitted and still ensures all signatures are valid.

3.3.2 Use `VERSION` instead of string in `OneSig.sol`

Severity: Informational

Context: OneSig.sol#L51

Description: The `OneSig` contract's `DOMAIN_SEPARATOR` uses the string ("0.0.1") for the `version` instead of the constant defined in the contract. This could lead to an error in a future version of the contract where only the constant is updated and not the string.

Recommendation: Use:

```
bytes32 private constant DOMAIN_SEPARATOR =

    keccak256(

        abi.encode(

            EIP712DOMAIN_TYPE_HASH,

            keccak256(bytes("OneSig")), // this contract name

            keccak256(bytes(VERSION)), // version

            1, // Ethereum mainnet chainId

            address(0xdEaD) // verifyingContract

        )

    );
```

LayerZero: Fixed in commit a5b5b1ce.

Cantina Managed: Fix verified.