LayerN contracts

Security Review

Cantina Managed review by:

Optimum, Lead Security Researcher

Slowfi, Security Researcher

Carrot Smuggler, Associate Security Researcher

August 14, 2024

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

Layer N is Ethereum at internet-scale: the first highly scalable blockchain capable of enabling up to hundreds of thousands of transactions per second, sub-ms latencies, and dedicated congestion-free compute for applications.

From Jul 22nd to Aug 6th the Cantina team conducted a review of nord-contracts on commit hash 4a4625b0. The team identified a total of **16** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 6
- Gas Optimizations: 3
- Informational: 7

3 Findings

3.1 Low Risk

3.1.1 The current implementation allows null state transition

Severity: Low Risk

Context: NordOperatorFacet.sol#L49

Description: Currently the contract allows to perform `sumbitStateUpdate` and `updateSate` full transitions several times keeping the same state hash without any other verification. This is considered by the **LayerN** team the *null state transition*, as using the same hash is allowing to transit to an already reached state.

Proof of concept: The next test case succeeds proving the issue.

```
function test_updateKeepingSameStateHash() public {
    uint256 daFact = 123456789;
    eigenDAFactRegisitry.setFactState(daFact); // Created this function to simplify the test case
    FT.StateUpdateFacts memory suf = FT.StateUpdateFacts({
        prevStateHash: uint256(1),
        pendingStateHash: uint256(1),
        daFact: daFact,
        onchainUpdatesHash:
↪   uint256(97697478700445635463716678234069343688258132164933614482741759039421686466216), //hash of empty
↪   struct ou
        startingActionId: uint64(2),
        endingActionId: uint64(2)
    });
    vm.startPrank(nord.getOperator(0));
    nord.submitStateUpdate(suf);
    vm.stopPrank();
    NT.DepositUpdate[] memory deposits;
    NT.WithdrawalUpdate[] memory withdrawals;
    NT.AssetUpdate[] memory newAssets;
    NT.OnchainUpdates memory ou = NT.OnchainUpdates({
        deposits: deposits,
        withdrawals: withdrawals,
        newAssets: newAssets
    });
    vm.warp(block.timestamp + 2 days);
    vm.startPrank(nord.getOperator(0));
    nord.updateState(1, ou);
    vm.stopPrank();

    daFact = 12345678;
    eigenDAFactRegisitry.setFactState(daFact); // Created this function to simplify the test case
    suf = FT.StateUpdateFacts({
        prevStateHash: uint256(1), //same state hash as before
        pendingStateHash: uint256(1), //same state hash as before
        daFact: daFact,
        onchainUpdatesHash:
↪   uint256(97697478700445635463716678234069343688258132164933614482741759039421686466216),
        startingActionId: uint64(3),
        endingActionId: uint64(3)
    });
    vm.startPrank(nord.getOperator(0));
    nord.submitStateUpdate(suf);
    vm.stopPrank();
    vm.warp(block.timestamp + 2 days);
    vm.startPrank(nord.getOperator(0));
    nord.updateState(2, ou);
    vm.stopPrank();
}
```

Recommendation: Consider preventing the system to use the same state hash as the previous state transition. Also if possible consider any additional mechanism, on-chain or off-chain to avoid reusing a previously computed state transition hash.

LayerN: This issue is mitigated by ensuring that the previous state hash is different than the current state hash. The system also ensures that the L2 cannot repeat state hashes.

Cantina Managed: Fixed by LayerN team on commit 13046ede.

3.1.2 Withdrawal amount can be bigger than `type(uint64).max`

Severity: Low Risk

Context: NordStateUpdateLib.sol#L44

Description: The system architecture of LayerN L2 has a maximum variable size of 64 bits. Amounts coming from and going to the L2 are *quantized* to fit in this smaller storage format. So any amount coming from L2 should be less or equal than `type(uint64).max`.

The function `handleWithdrawal` from contract `NordStateUpdateLib.sol` designed to handle the users withdrawal's from L2 to L1 receives a `uint256` type variable for the amount to withdraw. Although the amounts are supposed to be on the 64 bit storage format the L1 smart contract does not enforce it and assumes the received values are correct.

Recommendation: Consider checking that the received values from the prover are less or equal than `type(uint64).max`.

LayerN: Issue fixed by setting the primitive type to `uint64` on the `WithdrawalUpdate` struct.

Cantina Managed: Fixed on commit ae6cba4c.

3.1.3 Check `assetId` and `depositor` length on `handleDeposit` function

Severity: Low Risk

Context: NordStateUpdateLib.sol#L35

Description: The function `handleDeposit` from the library `NordStateUpdateLib.sol` is called when invoked through a submit state action by the operator role, this means the proover. However this function does not check that the deposited asset is a registered asset as well as the lenght of the depositor.

This is not a security problem itself to the contract, however the contract emits an event that is probably read from a bot that monitors the state transitions to allocate the funds on the L1. This may result in a non correct functionality from the bot.

Recommendation: Consider controlling the `assetId` and `depositor` fields from the `DepositUpdate` input call parameter.

LayerN: The team acknowledged this issue. This is because it has been planned to refactor the entire deposit flow. This refactor guarantees that deposits consumed by the operator are the same as what is deposited on chain, thus making this a non-issue. Therefore, the only time that deposit validation would need to happen is when the user does the deposit.

Cantina Managed: Acknowledged by the LayerN team.

3.1.4 `endingActionId` can be set to a high number, disabling further bridge updates

Severity: Low Risk

Context: NordOperatorFacet.sol#L68

Description: Operators can specify an `endingActionId`in their state updates. This is then stored in the `ns().actionId` storage location which stores an `uint64`. The issue is that if a malicious operator sets this value to be 2ˆ64-1, then the bridge can effectively be halted.

This is because further state updates to the bridge require higher values of `actionId`, which isn't possible due to the `uint64` size limit.

```
if (pendingFacts.startingActionId != ns().actionId + 1) { //revert
```

Recommendation: Limit the maximum change possible in action Id. Ensure that `endingActionId - startingActionId < 2^32`

LayerN: Fixed in commit a98c2fec.

Cantina Managed: Fixed by limiting the maximum update limit for actionIds following auditor recommendations.

3.1.5 `NordUpgradeFacet` lacks a function for atomic re-initialization, which could allow attackers to call `init` maliciously

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: In the current version of the code, `NordInit.init` is called once during the construction of `MultiFacetProxy`. In case the project team decides that a re-initialization is needed, the following operations will be invoked:

1. A new `NordInit` contract will be deployed.
2. The owner of `MultiFacetProxy` will add the `init` selector by calling `NordUpgradeFacet.addFunctions`.
3. The `init` function will be invoked.
4. The owner of `MultiFacetProxy` will remove the `init` selector by calling `NordUpgradeFacet.removeFunctions`.

The potential issue is that if operations 2-4 are not executed atomically (within the same transaction), an attacker could exploit this by front-running or back-running a call to `init` with their own malicious call before `init` is removed from the callable functions list.

Recommendation: Consider adding an owner only function to `NordUpgradeFacet` that will facilitate operations 2-4 atomically.

LayerN: Fixed in commit a98c2fec.

Cantina Managed: Fixed by implementing logic that is slightly different than the auditor's recommendation but achieves the same result.

3.1.6 `Deploy.sol`: Uninitialized parameters during deployment may cause a faulty deployment

Severity: Low Risk

Context: Deploy.sol#L132, Deploy.sol#L155

Description: We decided to incorporate the deployment script into the scope of this security review to ensure that any associated risks are adequately addressed. During our examination, we identified several uninitialized parameters that were set to zero values to facilitate the testing process. However, leaving these parameters uninitialized could potentially lead to faulty deployments and result in financial losses for users.

The uninitialized variables are as follows:

1. `eigenDAServiceManager`
2. `BOND`
3. `UNCHALLENGED_FRAUD_WINDOW`
4. `CHALLENGED_FRAUD_WINDOW`

Recommendation: Consider initializing these values before the actual deployment and for the long term consider using foundry cheatcodes like `store` to mimic a state that will be easier for testing purposes.

LayerN: Acknowledged. At the current time, we are not certain of all of these parameters and will fill them in when we are ready.

Cantina Managed: Acknowledged by the LayerN team.

3.2 Gas Optimization

3.2.1 Using `calldata` instead of `memory` for read-only arguments in `external` functions save gas

Severity: Gas Optimization

Context: NordOperatorFacet.sol#L31, NordOperatorFacet.sol#L46, NordStateUpdateLib.sol#L19, NordStateUpdateLib.sol#L32, NordStateUpdateLib.sol#L39, NordStateUpdateLib.sol#L49

Description: The next indentified function parameters are only read from but use the `memory` keyword.

- `submitStateUpdate` function `updateFacts` param from `NordOperatorFacet`
- `updateState` function `onchainUpdates` param from `NordOperatorFacet`
- `updateState` function `onchainUpdates` param from `NordStateUpdateLib`
- `handleDeposit` function `deposit` param from `NordStateUpdateLib`
- `handleWithdrawal` function `withdrawal` param from `NordStateUpdateLib`
- `handleNewAsset` function `newAsset` param from `NordStateUpdateLib`

Recommendation: Consider replacing the `memory` keyword for `calldata` from the parameters that are going only to be read to save gas.

LayerN: Acknowledged. Do not plan to address this at this time due to the difficulties associated with forming dynamic array payloads in Solidity.

Cantina Managed: Acknowledged by the LayerN team.

3.2.2 `NordRampFacet`: calls to `validateAsset` are redundant

Severity: Gas Optimization

Context: NordRampFacet.sol#L20, NordRampFacet.sol#L28

Description: During the flows of `NordRampFacet.depositUnchecked` and `NordRampFacet.withdraw` the check that makes sure the injected `assetId` is supported in the system is being made inside the `NordRampLib` library and as such it is redundant to call it again as part of the `NordRampFacet` contract.

Recommendation: Consider removing the calls to `validateAsset`.

LayerN: Acknowledged.

Cantina Managed: Acknowledged by the LayerN team.

3.2.3 `NordOperatorFacet.updateState`: The call to `isValidStateFact` is redundant

Severity: Gas Optimization

Context: NordOperatorFacet.sol#L64

Description: `updateState` is statically calling `isValidStateFact` to make sure that the fact is valid, i.e. either proved or timed out. This check is redundant since `StateTransitionFactRegistry.finalizeOptimisticFact` already includes this check.

Recommendation: If optimizing gas efficiency is a priority, you may wish to consider removing this redundant check.

LayerN: Acknowledged. We do not plan to address this in case we want to remove the call the bridge makes to finalize the fact in the state transition fact registry.

Cantina Managed: Acknowledged by the LayerN team.

3.3 Informational

3.3.1 Use of unlicensed contracts

Severity: Informational

Context: NordOwnerFacet.sol#L1

Description: The absence of a license is denoted by the `// SPDX-License-Identifier: UNLICENSED` declaration. This indicates that the project's code is not licensed for use, distribution, or modification by others.

Operating without a license poses several security and legal risks:

1. Unclear Usage Rights: Without a license, users do not have explicit permission to use, modify, or distribute the code.
2. Lack of Liability Protection: A license typically includes disclaimers of warranties and liabilities. Without this, the protocol team could be held responsible for any damages caused by the use of the contracts.
3. Contributing Hurdles: Open-source contributors are likely to avoid projects without a clear license, reducing potential collaboration and improvement opportunities.

Recommendation: To mitigate these risks, it is recommended to choose and apply an appropriate open-source license (e.g., MIT, Apache 2.0, GPL). This will enhance security, clarify usage rights, and offer legal protection. License references.

LayerN: Acknowledged. Will fix it after deciding the adequate licensing.

Cantina Managed: Acknowledged by the LayerN team.

3.3.2 Non-standard usage of role based access control

Severity: Informational

Context: NordOwnerFacet.sol#L26

Description: In the `NordOwnerFacet.sol` contract, different roles can be assigned and un-assigned. The functions used for this purpose is protected by the `onlyOwner` modifier. This modifier checks that the caller has the `OWNER` role.

```
modifier onlyOwner() {
    AccessControlLib.checkRole(C.OWNER_ROLE, msg.sender);
    _;
}
```

Thus users with the `OWNER` role can assign and unassign the `OWNER`, `OPERATOR` and the `FRAUD_PROOF_PARTICIPANT` roles. This is a non-standard way of implementing role based access controls. In the standard implementation, every role has an associated `adminRole`. Only this `adminRole` is allowed to assign their associated role, to any address. In the current implementation, this `adminRole` is entirely unused, and users with the `OWNER` role acts as the `adminRole` for all roles.

Recommendation: Consider changing the implementation to utilise the `adminRole` functionality. This allows for more granular access control.

LayerN: Acknowledged.

Cantina Managed: Acknowledged by the LayerN team.

3.3.3 Unused `validateBlockId` modifier in `Helpers.sol` contract

Severity: Informational

Context: Helpers.sol#L30-L33

Description: The `validateBlockId` modifier is unused throughout the codebase and if the hashes are being set properly, the check is unnecessary.

Recommendation: Consider removing `validateBlockId` modifier.

LayerN: Removed in commit a98c2fec.

Cantina Managed: Removed unused code following auditor's recommendation.

3.3.4 `NordInit.init` Lacks an explicit re-initialization protection

Severity: Informational

Context: NordInit.sol#L19

Description: The repository employs a custom contract implementation to enable upgradeability using the diamond pattern. The `NordInit.init` function is used to initialize the `MultiFacetProxy` contract. During the construction of `MultiFacetProxy`, a call is made to `NordInit.init` once. If additional initialization is required, the approach involves deploying a new `NordInit` contract and invoking `NordInit.init` again.

While no issues were identified in the current code version, there remains a risk that `NordInit.init` could be inadvertently exposed through external endpoints after deployment. This exposure could potentially allow an attacker to call the function again, compromising crucial parameters such as the owner and operator.

Recommendation: We aim to implement a mitigation strategy to ensure that the init function cannot be called unless a new instance of `NordInit` has been deployed. To achieve this, consider implementing the following solution:

1. Declare a new immutable variable `bytes32 immutable INIT_KEY = keccak256(abi.encodePacked(address(this)` inside `NordInit`.
2. Declare a new mapping `mapping(bytes32 -> bool) initialized;`.
3. Upon calling the `init` function, the function will revert in case `initialized[INIT_KEY] == true` and by the end of the function `initialized[INIT_KEY]` will be set to `true`.

LayerN: Fixed in commit a98c2fec.

Cantina Managed: Fixed by implementing logic that's slightly different than the auditor's recommendation but achieves the same result.

3.3.5 `MultiFacetProxyLib.delegateCall` is unused and should be removed

Severity: Informational

Context: MultiFacetProxyLib.sol#L65

Description: `delegateCall` is one of the internal functions of `MultiFacetProxyLib` which is unused and thus should be removed.

LayerN: Fixed in commit a98c2fec.

Cantina Managed: Fixed.

3.3.6 `AccessControlLib` does not reuse the standard implementation by OpenZeppelin

Severity: Informational

Context: NordOwnerFacet.sol#L13, AccessControlLib.sol#L7

Description: The `AccessControlLib` library together with the `NordOwnerFacet` contract implement logic to restrict and manage access control over assets in the system. The logic is inspired by AccessControl.sol (the OpenZeppelin implementation) but is being "vendored" instead of used as a dependency.

Recommendation: Consider using the OpenZeppelin implementation instead.

LayerN: Acknowledged. Other prominent access control libraries do not necessarily fit with the unique proxy design of the contract, and so we opt to use our own custom implementation with an audit.

Cantina Managed: Acknowledged by the LayerN team.

3.3.7 `MultiFacetProxyLib.setFunctions` should not be externally accessible

Severity: Informational

Context: Deploy.sol#L186, NordUpgradeFacet.sol#L11

Description: The `setFunctions` function is used internally to add, remove and replace functions implemented in facets. In addition, it is accessible externally in the `NordUpgradeFacet` contract. We think it should be incapsulated to reduce complexity and enhance the flexibility of the program.

Recommendation: Consider removing `setFunctions` from `NordUpgradeFacet.sol` as well as from `Deploy.sol`.

LayerN: Fixed in commit a98c2fec.

Cantina Managed: Fixed by implementing the auditor's recommendation.