Matter Labs

Nodle Network Smart Contracts

Security Review Report

September 27, 2024

Contents

Disclaimer

Summary

Scope

The audit focused on reviewing the smart contracts of the Nodle system, with particular attention to ensuring that they were running correctly on ZKsync.

The scope was limited to the smart contracts in the subdirectory `/src` within the https://github.com/NodleCode/rollup repository at commit `c90ba673866defc2100c544278bad0e0d048098f`.

Findings Summary

The team identified a total of 10 security issues during this security review, which were categorized based on their severity as follows:

- High: 1 issue
- Medium: 3 issues
- Low: 6 issues

In addition to the identified security issues, we made several observations regarding code quality and provided general recommendations for improvement. These observations aim to enhance overall code structure, maintainability, and adherence to security best practices.

Security Issues

1. New vesting schedule creation can be denied

Severity: High **Status:** Resolved

The `Grants` contract allows for creating time-based token vesting schedules through the `addVestingSchedule` function in `Grants.sol:58-78`. In addition, a per-beneficiary maximum is implemented in lines 207-212 to avoid an excessive number of schedules being iterated over through the codebase. Attackers can fill this maximum for any address in several ways, making legitimate vesting creation fail.

- The lack of an upper bound on the `period` parameter allows attackers to create entries with a long period and retrieve the funds later through the `cancelVestingSchedules` function.
- The `perPeriodAmount` parameter does not undergo validation; therefore, new schedules can be created without the need to provide any funds.
- The lack of an upper bound on the `start` parameter allows attackers to create entries with a distant vesting start and retrieve the funds later through the `cancelVestingSchedules` function.

Consider the following scenario where a malicious grant receiver can prevent the cancel authority from seizing vested NODL tokens from grant schedules:

1. A rogue employee (an attacker) received a grant of vested NODL tokens, which were bridged to ZKsync through the `GrantsMigrations` contract.
2. Oracles vote for the grant proposal, meeting the approval requirements.
3. Before the execution delay passes, the attacker populates their `schedules` array in the `Grants` contract by calling `addVestingSchedule()` and creating `100` schedules. They designate themselves as the `cancelAuthority` and set a distant future start date for the artificially created schedules. This enables the attacker to reclaim NODL tokens at any time without incurring losses.
4. As the `schedules` array already contains the maximum number of entries for the attacker's address, any attempt to execute the approved proposal will fail.
5. The rogue employee leaves the organization, which should provoke a vesting seizure.
6. The company's cancel authority can not seize the NODL grant from the rogue employee, as the proposal hasn't been executed in the `GrantMigrations` contract so far. As a result, the cancel authority cannot invoke `Grants.cancelVestingSchedules()` for the target grant.

7. The attacker can wait for the vesting period to end, cancel the artificially created grants from `schedules`, and execute the proposal in the `GrantsMigrations` contract, successfully obtaining the full NODL grant from the `Grants` contract.

Recommendation:

We recommend revising the grant management logic. Using mappings instead of arrays would eliminate the limitation on the maximum number of grant schedules a user can receive, allowing for more flexible and scalable handling of grants.

Status Details:

The risk was mitigated by introducing pagination instead of our recommended approach. This may lead to a decreased user experience as users need to locate the exact pages of interest to submit the start and end values.

2. Compromising deployer could lead to minting an arbitrary number of NODL tokens

Severity: Medium **Status:** Resolved

The deployer of the `NODL`, `Rewards`, `WhitelistPaymaster`, and `EnterpriseContentSign` contracts is assigned admin privileges upon deployment. If the deployer's private key is compromised, it will enable an attacker to mint an arbitrary number of NODL tokens. Since the deployer’s private key represents a single point of failure and is more vulnerable than the set of private keys used in a multi-signature account, the risk of compromise is elevated.

The severity is rated as medium because exploiting this vulnerability requires direct access to the deployer’s private key.

Recommendation:

We recommend assigning admin privileges to a designated multi-signature (multi-sig) or governance-controlled address during deployment rather than granting these privileges to the deployer. This approach reduces the risk of a single point of failure associated with the deployer’s private key.

3. Missing check for `rewardPercentage` in the constructor

Severity: Medium **Status:** Resolved

The `constructor` of the `Rewards` contract checks that `batchSubmitterRewardPercentage` is less than `100` in `Rewards.sol:176`. However, this is incorrect because `batchSubmitterRewardPercentage` is a storage variable, and its default value is `0` at this point. The check should actually validate the `rewardPercentage` parameter passed to the constructor instead to ensure it is less than 100. This could enable a batch submitter to receive more NODL rewards than anticipated.

Recommendation:

We recommend including a check in the constructor of the `Rewards` contract to ensure that the `rewardPercentage` remains below `100`.

4. NFT minting can be blocked

Severity: Medium **Status:** Acknowledged

Users cannot mint NFTs once the `individualHolders` variable exceeds the `maxHolders` value, which is enforced in `MigrationNFT.sol:151-153`. To mint a level-1 NFT, users will need to bridge a relatively small amount of NODL tokens compared to other levels. Given that the `individualHolders` variable is shared among all NFT levels, malicious actors could exploit this by bridging small amounts of NODL tokens to multiple addresses until reaching the `maxHolders` limit and minting level-1 NFTs for them. As a result, it will disrupt the minting process for legitimate users.

Recommendation:

We recommend revising the NFT minting process to allow unlimited NFT minting for lower levels.

5. Limited precision for `batchSubmitterRewardPercentage`

Severity: Low **Status:** Resolved

In `Rewards.sol:230`, the `batchSubmitterRewardPercentage` value only supports whole numbers ranging from `0` to `99`. This prevents the use of fractional percentages (e.g., `1.5%` or `0.25%`) when rewarding batch submitters via the `mintBatchReward` function, which may not provide the necessary precision for certain reward structures and limit finer granularity in reward calculations.

Recommendation:

We recommended using basis points (BPS or `0.01%`, `1/10_000`) instead of `1/100` to measure `batchSubmitterRewardPercentage`.

6. The absence of expiration for ECDSA signatures can lead to quota manipulation

Severity: Low **Status:** Acknowledged

In `Rewards.sol:362` and `Rewards.sol:374`, the `digestReward` and `digestBatchReward` functions of the `Rewards` contract do not incorporate an expiration timestamp for the ECDSA-signature verification routine (e.g., `validBeforeTimestamp`). Therefore, the signature is irrevocable and non-expirable until the sequence number is incremented.

This situation could be exploited in a griefing attack scenario because individual rewards share the quota amount with batched rewards. Since the signatures do not expire, individual users are not required to call `mintReward` immediately; they can postpone it for later. When the quota is almost used up for a period, malicious users can collude and call `mintReward` with an aged signature just before the call to `mintBatchReward`, which will revert because the quota is exhausted. This scenario may discourage users from utilizing the batch reward mechanism.

Recommendation:

We recommend adding an expiration timestamp for the ECDSA signature.

7. `period` storage variable cannot be changed

Severity: Low **Status:** Resolved

In `Rewards.sol:182`, the value of the `period` storage variable in the `Rewards` contract cannot be modified once it is set in the constructor. If the `period` variable is set to an incorrect value or requires a different value due to business needs, the `Rewards` contract has to be redeployed.

Recommendation:

We recommend adding a setter method similar to the existing `setQuota` function, allowing the `DEFAULT_ADMIN_ROLE` user to adjust the `period`.

8. Critical events are not observable

Severity: Low **Status:** Resolved

The following state-changing functions are not emitting events, making it difficult to track important contract actions.

1. The `setBatchSubmitterRewardPercentage` function of the `Rewards` contract in `Rewards.sol:252-256`.
2. The `addWhitelistedContracts`, `removeWhitelistedContracts`, `addWhitelistedUsers`, and `removeWhitelistedUsers` functions in `WhitelistPaymaster.sol:21-49`.
3. The `withdraw` function of the `BasePaymaster` contract in `BasePaymaster.sol:83-88`.

Recommendation:

We recommend emitting events in the aforementioned cases.

9. Missing validation steps could lead to erroneous states

Severity: Low **Status:** Partially Resolved

Several `constructor` functions do not perform adequate parameter validation. If incorrect values are provided during deployment, it could lead to the contracts becoming unusable or causing issues with token vesting, potentially preventing users from claiming vested tokens.

- `MigrationNFT.sol:53`: The `maxHolders` parameter can be set to an arbitrary value. If set to zero or a very low value, the contract will be unusable.

- `Bridgebase.sol:75`: The `delay` state variable can be set to an arbitrary value. If this value is set to zero or a very low value, the purpose of the delay feature will be rendered useless as there will not be enough time to react between a proposal being passed and its execution.
- `GrantsMigration.sol:121-130`: The `_createProposal` function does not validate that the `schedules` array is not empty or below a maximum. A very large array could lead to out-of-gas exceptions for loops, while an empty array will pointlessly waste gas. In addition, the `amount` parameter is not validated to be equal to or greater than the total of all the schedules to be created, which would prevent users from claiming their full vesting.

Recommendation:

We recommend adding thorough validation to each of the instances outlined above.

Status Details:

The issue has been marked as “Partially Resolved” because the `delay` variable in the Bridgebase.sol contract did not follow the recommended changes. Although the Nodle team has stated that allowing it to be zero is intended, turning this feature off creates potential risks.

10. Potential forced token transfer to users

Severity: Low **Status:** Acknowledged

Once the proposal has sufficient votes and the safety delay has passed, any user can invoke the `withdraw` method of `NODLMigration.sol:51` transferring tokens to the target user’s address, even if the caller is not the intended recipient. However, this could potentially go against the user’s desires, who may not want that transfer at that moment, for example due to tax implications or high token prices.

Recommendation:

We recommend adding an option for users to opt in or defer the withdrawal process, providing more control over when they receive the tokens.

Observations

We classify as “observations” any comment or recommendation about code quality or security best practices that are unlikely to pose an immediate and exploitable risk to the assets within the security review’s scope. These informational findings are shared to provide a more comprehensive view of the codebase and ensure the audit's completeness.

1. The Check-Effects-Interaction pattern is not followed in `Grants.sol:57-78`: the token transfer is `performed` before `adding` the schedule to the array. As this contract is designed to be used with the ERC-20 NODL token, there is no impact on the NODL protocol. However, if a different token standard that implements callbacks is ever used in the future, the vulnerability will become exploitable. Status: **Resolved**.
2. There are typos that need correction. Specifically, `Rewards.sol:304-307` and `Rewards.sol:167`. We recommend integrating a spell-checking tool in the CI/CD pipeline to detect such typos automatically. Status: **Resolved**.
3. The `voted` mapping maps “oracles to proposals to the vote status”. However, the rest of the codebase uses a different order of mapping “proposals to oracles to the vote status”. We recommend changing the order to “proposals to oracles to the vote status” to match the rest of the codebase. Status: **Resolved**.
4. The codebase uses custom errors in reverts. However, `BridgeBase.sol:65-66` uses `require` statements. We recommend replacing `require` statements with custom errors for consistency with the rest of the codebase. Status: **Resolved**.
5. The `addWhitelistedContracts` function in `WhitelistPaymaster:21-25` is not consistent with the structure of other functions in the same contract. To enhance consistency and readability, we recommend refactoring the code by moving the logic from the `_setContractWhitelist` function directly into the `addWhitelistedContracts` function. This will align its implementation with the rest of the contract and simplify the overall structure. Status: **Resolved**.
6. In the BridgeBase.sol:85-104, the _createVote and _recordVote functions share the same logic, with the only difference being the events emitted at the end. We recommend abstracting the common logic into a separate function. Status: **Resolved**.
7. The `bridge` method calls the _mustNotHaveVotedYet unnecessarily in NODLMigration.sol:40, as this check is already performed within the _recordVote function. Status: **Resolved**.
8. The `NODLMigration` contract does not implement a quota for minting tokens. If the threshold number of oracles is compromised, an arbitrary amount of NODL tokens can be minted. Status: **Acknowledged**.
9. The implementation returns the same URI from `levelToTokenURI[level - 1]` for different token IDs within the same level. This means that all NFTs of the same level share the same `tokenURI` (metadata), which might not be ideal in certain use cases. If each token in a collection is intended to have unique metadata (e.g., for artistic

collections), this approach will not work as intended. We recommend modifying the logic to generate a unique URI for each `tokenId` to ensure proper uniqueness for each NFT. Status: **Acknowledged**.

10. Calling `_mustNotHaveVotedYet()` in `src/bridge/BridgeBase.sol:86` is unnecessary since the proposal doesn't exist yet. Status: **Resolved**.