OpenZeppelin | security

The Graph Protocol L2 Linear Rewards Minting

The Graph

November 3, 2022

This security assessment was prepared by OpenZeppelin.

Table of Contents

Summary

Type	L1/L2 Bridge
Timeline	From 2022-09-26 To 2022-10-17
Languages	Solidity

Total Issues	6 (6 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (2 resolved)
Low Severity Issues	1 (1 resolved)
Notes & Additional Information	3 (3 resolved)

Scope

We audited the graphprotocol/contracts repository at the 263355d3 commit.

In scope were the following contracts:

```
contracts/
├── gateway
│   └── L1GraphTokenGateway.sol
└── rewards
    ├── IRewardsManager.sol
    ├── RewardsManager.sol
    └── RewardsManagerStorage.sol
```

Overview of Changes

The Graph is a protocol for indexing and querying Ethereum ecosystem data in an easy-to-use, scalable format. They have recently made the decision to move their protocol from L1 on Ethereum to Arbitrum Nitro's L2 network. The Graph envisions this migration as a steady process that will allow users to gradually migrate to L2 and easily migrate back to L1 if needed or desired. The process will start with the deployment of The Graph's core contracts on Arbitrum, allowing each user to choose when to move their Graph Tokens to L2 using a new L1 gateway contract.

Coinciding with the migration, The Graph has proposed two different methods for upgrading its rewards distribution to span L1 and L2. This audit reviews one of those methods, proposed in GIP-0037, where rewards will be minted on L2 following a new linear rate of distribution. This proposal is in competition with GIP-0034, audited as part of a prior engagement, which specifies a method for dripping rewards from L1 to L2 (maintaining the exponential rate of rewards distribution). Compared to GIP-0034, we perceive the implementation in GIP-0037 easier to understand and maintain with respect to correct rewards management across networks, while separating L1 and L2 logic where possible by minimizing messages across the bridge. Although GIP-0037 is simpler, it still adds the L1 and L2 gateways as new minters to The Graph's ecosystem. The Graph's governance will choose one of these two methods for migrating the protocol to L2.

Privileged Roles

The governor role (i.e., The Graph's governance) is the primary privileged role for managing the rewards contracts. The governor has the following privileges:

- Able to and responsible for setting correct addresses for Arbitrum Bridge, Graph Token on L2, Graph Gateway on L2, and escrow address on L1.
- Able to edit the whitelisted addresses allowed to send callhook data across the bridge.
- Able to set L1 variables for the L2 mint allowance.

Operational Considerations

As mentioned, the Governor is able to set the L2 mint allowance variable for L1. This is an important responsibility and needs to be kept up-to-date in a timely manner.

In addition to ensuring the orderly update of mint allowances on L1 for L2, the Governor must also ensure a particular order of operations when upgrading the Rewards Manager contract on L1 to use the new linear rewards calculations. By ensuring that `updateAccRewardsPerSignal` is called prior to, and in the same block as, the upgrade transaction itself, this ensures that previous reward calculations up to and including the block of the upgrade transaction will be accounted for correctly using the previous exponential rewards calculation. This also prevents any rewards from being lost, since upgrading to the new linear rewards calculation will cause the accrued rewards from the previously accrued rewards block until the current block to be zero. Rewards will then begin to accrue correctly using the new calculations after the update.

Security Considerations

The Graph's new gateway contract allows users to deposit their L1 Graph Tokens (GRT) to be later redeemed on L2 by using the `outboundTransfer` function. GRT coming to L1 from L2 are handled automatically in the `finalizeInboundTransfer` function which will be triggered by Arbitrum's bridging contract and is only callable by The Graph's L2 gateway. The Graph's L1 and L2 gateway contracts are new, authorized GRT minters. To mitigate damage in the event of exploitation against the Arbitrum bridge itself, The Graph's L1 and L2 gateways are both pausable and upgradeable. Additionally, as part of the changes in scope for this audit, the L1 gateway contract contains a mint cap that limits the amount of new L1 GRT that can be minted in excess of the balance of GRT in the escrow contract. While this does not protect users whose GRT resides in the escrow contract (any tokens that reside in a bridge contract are inherently at risk of loss), it does guard the entire ecosystem against arbitrary minting that could excessively dilute the value of GRT on L1. The Graph has acknowledged these risks and set up contingency plans for many L2 disaster scenarios which can be viewed in The Graph's public Notion page.

Client-reported Finding

On October 31, 2022, The Graph reported a vulnerability in the codebase. Here we present the client-reported issue, followed by our findings.

Incorrect check on callhook data in L2GraphTokenGateway

When an address calls `outboundTransfer` in L1GraphTokenGateway, necessary information is encoded for the subsequent call to `finalizeInboundTransfer` on L2. The last parameter that is encoded is defined as extra callhook data. This extra callhook data can be supplied when a whitelisted address wishes to call `onTokenTransfer` on the recipient L2 address. When no extra data is supplied, the `if` statement is intended to prevent `onTokenTransfer` from being called. However, `getOutboundCalldata` will always encode non-zero length callhook data as a result of encoding two `bytes memory` variables together, regardless of whether they are both empty or not. Therefore, the `if` statement in `finalizeInboundTransfer` will always succeed. Thus, attempting to call `onTokenTransfer` prevents the L2GraphTokenGateway from operating correctly in the normal use-case of bridging GRT.

This issue was resolved by The Graph at commit `caf5ab5723353b9062878ade247a060a2b55514f` in PR #745. We recommend adding a robust test suite that integrates with Arbitrum for proper insight on test cases.

Medium Severity

M-01 Lack of validation for L1 and L2 gateway unpausing

The `L1GraphTokenGateway` and `L2GraphTokenGateway` are each initialized in a paused state. This prevents users from being able to call functions on either gateway until the contracts are unpaused by governance. However, in the comments of the `initialize` methods, there is a list of state variables that will not be set during initialization as they may not be known at that time. This is true for both the L1 gateway and the L2 gateway. It is assumed that the appropriate `set*` methods will be called by governance before both gateways are unpaused. However, in the `setPaused` function there are no checks that ensure all state variables for the respective gateway have been initialized. If either gateway is unpaused by governance before the necessary state variables have been set, incorrect behavior can occur.

For example, if the L2 gateway is unpaused before `l1GRT` is set, it would still be possible to call `outboundTransfer` as the `calculateL2TokenAddress` function will correctly return the L2 GRT token address even when `l1GRT` is `address(0)` if a user supplies the `_l1Token` address as `address(0)`. This means the transaction with the incorrectly-supplied `_l1Token` address would succeed on L2, but it would still fail on L1 due to the token address check. However, this incorrect behavior can be prevented by ensuring that all necessary state variables have been set on the respective gateway prior to `setPaused` being called.

Consider adding checks to the `setPaused` function that ensures all respective state variables are initialized for both the `L1GraphTokenGateway` and `L2GraphTokenGateway`.

Update: *Fixed as of commit `6d4a0d72c6cc6ed834a9427c0f09e6cd55571b2b` in PR #733.*

M-02 Outbound transfers with callhooks could revert

The `outboundTransfer` function creates a retryable ticket to transfer GRT to L2. The call may originate from Arbitrum's L1 router. The `parseOutboundData` function is used to recover the original `msg.sender` from the extra data for messages going through the router.

This parsed data is used in the `outboundTransfer` function as the `from`, `maxSubmissionCost`, and `extraData` parameters respectively. In the case where the `from` address is whitelisted to add callhooks and supplies callhook data, but the message originated from Arbitrum's L1 router (instead of from the `from` address), the `outboundTransfer` will revert due to a require statement that only checks `msg.sender` (rather than checking the `from` address).

Consider updating the `require` statement to check the `callhookWhitelist` using the `from` address instead of `msg.sender`.

Update: *Acknowledged. This is by design, and not an issue. The Graph's statement:*

> *Callhooks are not meant to be called through the Arbitrum Router but directly to the L1GraphTokenGateway, so checking msg.sender instead of from is intentional.*

Low Severity

L-01 Lack of validation of contract addresses

The following functions could benefit from validating that address parameters are contracts:

- `addToCallhookWhitelist` could potentially limit the whitelist to only contract addresses.
- `setArbitrumAddresses` also does not validate that the `_inbox` and `_l1Router` addresses are valid contracts.

Consider validating that these addresses are contracts.

Update: *Fixed as of commit* `16dc44e47157510c330b97eebedcbf50eb7ec2b5` *in PR #734.*

Notes & Additional Information

N-01 Redundant gas check

The Arbitrum upgrade from Classic to Nitro added several improvements, which include collecting the retryable ticket creation cost as part of the L1 transaction. This ensures that enough gas has been passed by a caller in order to successfully create a retryable ticket on L2. Before Nitro, a situation could occur where the L1 transaction succeeded but the L2 retryable ticket failed to be created. This renders the gas check in `outboundTransfer` redundant.

Consider removing the gas check from `outboundTransfer` in `L1GraphTokenGateway`.

Update: *Fixed as of commit* `2dfacb6322627d58a883803035f7b3b0c57f22ea` *in PR #735.* `maxSubmissionCost` *is still checked to ensure it is non-zero.*

N-02 Inequality should be strict

The `updateL2MintAllowance` and `setL2MintAllowanceParametersManual` functions validate that the `_updateBlockNum` and `_lastL2MintAllowanceUpdateBlock` are not in the future, respectively, by comparing the parameter to the current `block.number`. In each instance, the inequality is not strict (i.e., `<=`), while the spec in GIP-0037 and documentation state that this update must happen after the update on L2. Therefore, it is not feasible to ever expect the update to happen on L2 with the same `block.number` as L1.

Consider using a strict inequality to better validate that these function parameters use a historical block number.

Update: *Fixed as of commit* `dbdda2f563c3cc65491f2f3125e93fdbe2a44384` *in PR #736.*

N-03 Deprecated function

The `_pow` function in `RewardsManager` was originally used in `getNewRewardsPerSignal` to calculate `accRewardsPerSignal` when the calculation

was exponential. Now that the rewards equation is being changed to a linear equation, `_pow` can be deprecated and removed from the contract.

Consider removing the `_pow` function from `RewardsManager` as it is no longer used.

Update: *Fixed as of commit `18452596702d20883407650f4a42681da7e3f026` in PR #737.*

Conclusions

Two medium severity issues were found. Some recommendations were proposed to follow best practices and reduce the potential attack surface. We also recommend implementing monitoring and/or alerting functionality (see Appendix).

Appendix

Monitoring Recommendations

While audits help in identifying code-level issues in the current implementation and potentially the code deployed in production; we encourage The Graph team to consider incorporating monitoring activities in the production environment. Ongoing monitoring of deployed contracts helps in identifying potential threats and issues impacting the production environment. Hence, with the goal of providing a complete security assessment, we want to raise several actions addressing trust assumptions and out-of-scope components that can benefit from on-chain monitoring:

- **Repeated minting on L1 from L2, which could indicate a bridge exploit** - This involves checking if the proof of a transaction is a valid leaf in the Merkle Tree and has not been spent before. This requires checking when `executeTransaction` is called on the L1 `Outbox` contract, and verifying correctness by calling `calculateMerkleRoot` and `isSpent` with the passed-in input of `executeTransaction`. It is advisable to store past proofs and see if `executeTransaction` is being called on the same proof more than once.
- **Retryable tickets failing on L2, to ensure smooth operations and proactively investigate issues** - This involves checking the `sequenceNum` emitted by the `RedeemScheduled` event on retryable tickets that are submitted, looking for relevant `retryTxHash` and `to` addresses within the same block as the `RedeemScheduled` event.
- **Timely updates of the L2 mint allowance variables on L1, to maintain rewards accounting** - This can be validated by checking the public `lastL2MintAllowanceUpdateBlock` and `l2MintAllowancePerBlock` on the L1 gateway, and comparing with the `issuancePerBlock` variable in the Rewards Manager on L2.