OpenZeppelin | security

The Graph Protocol Arbitrum Bridge Audit

The Graph

July 27, 2022

This security assessment was prepared by **OpenZeppelin**, protecting the open economy.

Table of Contents

Summary

Type	L1/L2 Bridge
Timeline	From 2022-06-06 To 2022-07-01
Languages	Solidity

Total Issues	39 (32 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	2 (1 resolved)
Low Severity Issues	14 (11 resolved)
Notes & Additional Information	23 (20 resolved)

Scope

The Graph team asked us to review and audit four pull requests, all of which are part of the graphprotocol/contracts repository. The pull requests audited were 552, 568, 569, and 571. Within the various pull requests, the following files were considered in-scope for this audit:

- `PR #552`:
  - Commit: `359d2547deb1b2f62064da3d77a104a115b5531f`
  - Files in Scope:
    - `contracts/arbitrum/IArbToken.sol`
    - `contracts/arbitrum/IBridge.sol`
    - `contracts/arbitrum/IInbox.sol`
    - `contracts/arbitrum/IMessageProvider.sol`
    - `contracts/arbitrum/IOutbox.sol`
    - `contracts/arbitrum/ITokenGateway.sol`
    - `contracts/arbitrum/L1ArbitrumMessenger.sol`
    - `contracts/arbitrum/L2ArbitrumMessenger.sol`
    - `contracts/gateway/BridgeEscrow.sol`
    - `contracts/gateway/GraphTokenGateway.sol`
    - `contracts/gateway/L1GraphTokenGateway.sol`
    - `contracts/governance/Managed.sol`
    - `contracts/l2/gateway/L2GraphTokenGateway.sol`
    - `contracts/l2/token/L2GraphToken.sol`
    - `contracts/token/IGraphToken.sol`
- `PR #568`:
  - Commit: `40ca8886897f4a4ed3fdc11e84e30321f8edfffc`
  - Files in Scope:
    - `contracts/discovery/GNS.sol`
    - `contracts/discovery/IGNS.sol`
- `PR #569`:
  - Commit: `7d00cbf48ac3c9b74e76f1c0320d1cf0e57f0dac`
  - Files in Scope:
    - `contracts/rewards/RewardsManager.sol`

        - `contracts/rewards/RewardsManagerStorage.sol`

- `PR #571`:
    - Commit: `403be9401a693068ec380a9fbc4f24036ae8794e`
    - Files in Scope:
        - `contracts/governance/Managed.sol`
        - `contracts/l2/reservoir/L2Reservoir.sol`
        - `contracts/l2/reservoir/L2ReservoirStorage.sol`
        - `contracts/reservoir/IReservoir.sol`
        - `contracts/reservoir/L1Reservoir.sol`
        - `contracts/reservoir/L1ReservoirStorage.sol`
        - `contracts/reservoir/Reservoir.sol`
        - `contracts/reservoir/ReservoirStorage.sol`
        - `contracts/rewards/IRewardsManager.sol`
        - `contracts/rewards/RewardsManager.sol`
        - `contracts/rewards/RewardsManagerStorage.sol`
        - `contracts/staking/Staking.sol`
        - `contracts/token/IGraphToken.sol`

All other project files and directories (including tests), along with external dependencies and projects, game theory, and incentive design, were also excluded from the scope of this audit. External code and contract dependencies were assumed to work as documented.

Overview of the Changes

Below is a brief description of each pull request:

- PR #552: Implements an Arbitrum bridge for the Graph Protocol, in order to support scaling the protocol via a Layer 2 network. This changeset adds four key contracts: `L1GraphTokenGateway`, `L2GraphTokenGateway`, `BridgeEscrow`, and `L2GraphToken`. These contracts provide the foundation for transferring GRT tokens from L1 to L2 and vice versa. A detailed overview and discussion of the design decisions can be found in the Graph Improvement Proposal GIP-0031.
- PR #568: Adds a new feature to the `GNS` contract to support the transfer of curation signals to another address. The motivation for this change and design details can be found in GIP-0032.
- PR #569: Fixes a rewards calculation error by ensuring that the token supply is a snapshot taken at the corresponding block number used in the rewards formula.
- PR #571: Implements a new reward generation and distribution system to support distribution of rewards on both L1 and L2. Instead of minting rewards on demand, they are now periodically dripped into a rewards reservoir on L1, and a fraction of the dripped rewards are also sent to the L2 side. The corresponding GIP document that provides an in-depth design description is GIP-0034.

Pull request 552 lays the foundation for bridging GRT tokens from L1 and L2, and pull request 571 builds on top of that feature by changing the rewards generation model to support rewards distribution on both L1 and L2. The other pull requests 568 and 569 contain important changes unrelated to the new Layer 2 support being introduced to The Graph.

System Overview

The Graph is an open-source, decentralized protocol for indexing and querying blockchain data. The indexed data is made available to Web 3.0 applications via APIs called subgraphs.

The native token of The Graph network is the GRT token, which is used for payment, staking, and rewards distribution. The Graph ecosystem includes the following roles:

- *Indexers* provide the data to Consumers via subgraphs. Indexers are rewarded for servicing data queries and penalized if their nodes provide incorrect data.
- *Curators* provide a demand signal that indicates which subgraphs are important. Curators provide signal by staking in a subgraph and receive a share of indexing rewards.
- *Delegators* help secure the network by staking on existing Indexer nodes. In return Delegators share a portion of a node's generated fees.
- *Consumers* purchase the data. Consumers are commonly decentralized applications that pay indexers to query their subgraph APIs.

In addition to these roles, the *Governor* role performs administrative actions within the network. The Graph Council is a governance body that controls access to the Governor role via a 6-of-10 multisig. This is a powerful role that has the ability to alter the supply of GRT tokens and change key contract addresses and parameters.

Arbitrum Bridge

Arbitrum One is a Layer 2 Ethereum scaling solution developed by OffChain Labs. The Graph team chose Arbitrum as a potential solution to address the high fees and network congestion encountered when running The Graph protocol on Ethereum mainnet. Layer 2 solutions such as Arbitrum seek to address these issues by moving as much data storage and computation off of the Ethereum Layer 1 chain as possible, but still use the L1 chain to secure the result. Arbitrum is an *optimistic rollup* solution: Large batches of transactions are validated off-chain, and the net result is "rolled up" into an on-chain transaction. It's "optimistic" because the transactions added to the chain are assumed to be valid unless they are challenged. The Arbitrum API provides functions that allow a transaction that originates on the L1 side to be executed on the L2 side, and vice versa. For transactions from L1->L2, a *retryable ticket* is created that Arbitrum will immediately execute on the L2 side if sufficient gas was provided. For transactions from L2->L1, the transaction ticket can be executed by any user once the 7-day challenge period has passed.

PR #552 implements a token bridge that is based on the reference implementation provided by Offchain Labs. The purpose of this bridge is to transfer GRT tokens between L1 and L2. The basic operation of the bridge is as follows: GRT tokens are transferred between the L1 and L2 side by calling an `outboundTransfer` function on either the `L1GraphTokenGateway` or `L2GraphTokenGateway`. When tokens are deposited from L1 to L2, the tokens are locked

into the `BridgeEscrow` contract on the L1 side, and an equivalent amount of tokens are then minted on the L2 side. For the reverse transfer where tokens are withdrawn from L2 back to L1, the tokens are burned on the L2 side, and the equivalent amount of tokens are released from escrow on the L1 side. The `outboundTransfer` functions both invoke transaction-sending methods in the Arbitrum API: `L1GraphTokenGateway.outboundTransfer` will invoke the Arbitrum function `sendTxToL2`, and `L2GraphTokenGateway.outboundTransfer` will invoke the `sendTxToL1` function. As an additional feature, the GRT token bridge supports callhooks, allowing a whitelisted user to provide some extra data that will execute arbitrary code on the opposite side of the bridge as part of the outbound transfer transaction.

Rewards Issuance/Distribution

Currently, rewards are minted on-demand by the `RewardsManager` contract, when a staked allocation to a subgraph is closed. To simplify management of the GRT token supply when now dealing with two layers, the token bridge design assumes that every token minted on the L2 side has a pre-existing L1 counterpart, i.e. all new tokens always originate from the L1 side. This necessitated a change to the `RewardsManager` design, otherwise an L2 Rewards Manager could mint tokens that have no L1 counterpart locked in escrow.

PR #571 changes the rewards system to be compatible with the token bridge introduced in PR #552. The ability to mint rewards on-demand is removed and replaced by a new drip mechanism. Tokens are periodically minted by a new `drip` function on the L1 side, with the issuance rate and drip interval specified by governance. Using the token bridge, a portion of these newly generated rewards are sent to the L2 side, based on a rewards fraction value that is also set by governance. To store these pre-generated rewards, two new reservoir contracts were added: `L1Reservoir` and `L2Reservoir`. The callhook feature that was added in PR #552 is used to call the `receiveDrip` function which takes the additional action of moving bridged tokens into the L2 Reservoir and updating the rewards per signal rate on L2.

Security Considerations

During the audit we encountered some security concerns that are unique to bridging solutions and the Arbitrum network.

- The design incorporates a callhook whitelist on both sides of the bridge, which opens the possibility for the whitelists to get out of sync. Careful attention must be paid to the order in which the whitelists are updated to avoid rejection of a legitimate transaction.
- The atomicity of cross-layer transactions that produce a state change on both layers is not guaranteed. For example, an `outboundTransfer` call on the `L1GraphTokenGateway` consists of two parts: Locking tokens in escrow on L1, and minting an equivalent amount of tokens on L2. If the latter part of this operation fails due to invalid call data, the transfer cannot be finalized on the L2 side and the tokens will remain stuck in escrow, requiring a governance action to unlock them.
- Cross-layer synchronization of time-based data is problematic because state is not updated simultaneously across layers. There will always be some inherent delay. For example, there may be a delay in the time between when the issuance rate is updated on L1 and when the update occurs on L2, especially if there is a transaction problem that requires the drip transaction to be retried. This can result in the estimated rewards calculation on the L2 side being out of sync with the L1 calculation.
- Tokens can be minted directly on the L2 side, without passing through the token bridge. If this action is taken by governance for some reason, the token supply will be out of sync between the two sides of the bridge, and the L1 total supply will no longer be a source of truth.
- The ordering of transactions sent from L1 to L2 can be changed by Arbitrum's sequencer, which is a centralized node operated by Offchain Labs that is implicitly trusted by the Arbitrum community to be well-behaved. A re-ordering of drip transactions by a badly-behaved sequencer would require some manual intervention to retry the transaction tickets in the correct order.
- The documentation for Arbitrum One highlights the fact that the currently deployed version is considered beta software. The Offchain Labs team states that while in the beta phase, they retain the ability to upgrade contracts and pause the Arbitrum system. These centralized controls, in addition to the centralized sequencer node, present some undetermined level of risk in using a partially decentralized system to manage GRT token assets.

Additionally, the new rewards issuance design may be slow to react to changes. The implementation allows for periodic adjustments to parameters such as the issuance rate so that the protocol can dial rewards up or down as needed. However, the Graph's governance process can take anywhere from a few days to a few weeks to get the required votes and perform a transaction, so this lag could be problematic if fast adjustments are needed.

Findings

Here we present our findings.

Medium Severity

M-01 Governance delays could negatively impact the rewards system

The `L1Reservoir` contract allows GRT tokens to be dripped from L1 to L2 using Arbitrum's bridge. Both the L1 and L2 reservoir act as a pool from which the indexers collect their rewards and the protocol burns the indexer's rewards on denied allocations or wrong proofs.

The `drip` function is intended to be called periodically to pre-mint an estimate of the amount needed for the upcoming interval, however the L2 reservoir could dry up if the cumulative rewards payout in L2 exceeds the amount of tokens that are sent through the bridge. Since both the issuance rate and proportion that should be sent to L2 are defined by the Governor, changing those values could increase the balance of the `L2Reservoir` contract.

The time needed for governance to pass a proposal could potentially take a few days to a few weeks according to The Graph team. Due to this delay, the balance of the `L2Reservoir` contract could be zero or close to zero until new values are set from the `L1Reservoir`. This would not only reduce the incentive of the indexers as they would not receive their reward, but would also encourage calling the `drip` function frequently to replenish the reservoirs. Furthermore, in cases where claims are denied and the rewards are intended to be burned, the transactions will also revert, preventing correct operation of the protocol.

Consider reducing the time needed to pass a new proposal and conservatively bridging extra tokens during the first rounds of dripping to prevent the depletion of the funds in L2. Furthermore, consider increasing the drip interval, to allow more time for any Governor action to take place before this checkpoint is reached.

Update: Not an issue. After reviewing the response from the Graph Team, we agreed this is not an issue. The Graph Team's response:

> *This should not really happen: the rewards payout in L2 is calculated based on the parameters that are sent through the bridge, so the balance of the bridge should be enough to cover the period until the next bridge. Changing issuanceRate or l2RewardsFraction would 1) change the amount of tokens sent to the bridge, and 2) change the values used by the reservoir to calculate the payout. So I don't see how a*

> *delay in governance changing a parameter could cause the L2Reservoir to run out of funds.*

M-02 Non-expirable permits

The `L2GraphToken` contract implements a `permit` function to allow users to approve a token allowance by validating a message signed by the holder.

However, the current implementation allows non-expirable permits by sending zero value deadlines and bypassing the expiration check. This is dangerous as the permit will not expire in case the signer changes their mind after sending the permit, requiring a manual override by frontrunning the old permit with a new one with the same nonce. Furthermore, this behavior is not documented in the inline documentation.

Consider either removing the possibility to have non-expirable permits or documenting the behavior along with implementing a function to increase the current permit's nonce.

Update: Partially fixed in commit `061ab3f6fcdad9229de5d3a1f323c473fa0ccef6` from pull request 601. More documentation has been added. However, it does not describe the procedure to cancel a non-expiring permit. The Graph's team statement for the issue:

> *Permits that don't expire are a feature we want to support. We believe it's sufficient mitigation that a user can send an overriding permit with the same nonce, rather than adding a specific function for this.*

Low Severity

L-01 DoS when changing the issuance rate

The `L1Reservoir` contract implements the functionality to drip tokens on both layers. The rate at which these tokens are minted is defined by the Governor when calling the `setIssuanceRate` function. Then, after a change in the rate, the `drip` function takes a snapshot of the rewards and changes the rate.

However, in the event of reducing the rate the `newRewardsToDistribute` value, which only depends on the rate for the upcoming interval, when added to the `mintedRewardsActual` value may be less than the tokens assigned for the previous interval. In this scenario, the calculation for the `tokensToMint` value will revert.

Even though this is mitigated by waiting until the `mintedRewardsActual` value is enough to break-even the calculation, consider either using signed integers to validate if more tokens should be minted (to allow the other operations of the `drip` function to succeed without reverting) or documenting this behavior to warn users along with adding a more explicit error message in the code.

Update: Fixed in commit `57dbbb83a13366f11296c5331b3538c083adece6` from pull request 588.

L-02 DoS when setting the L2 fraction to zero

The `L1Reservoir` contract allows dripping part of the minted tokens to Layer 2, whose proportion is based on the `l2RewardsFraction` value. To change this value, the Governor must call the `setL2RewardsFraction` function, which validates that the input is less than or equal to `1e18`.

In the scenario where the Governor sets `l2RewardsFraction` value to zero, the calculation of tokens to be sent during the next drip, before the old interval checkpoint has expired, will be of `0 - value` which will revert due to the `SafeMath.sub` method.

Even though the documentation states that when a smaller value is set for the next rewards fraction we must wait for additional time to pass, it seems that this case is not being properly

addressed in the correct conditional statement as there is one meant to catch zero L2 fractions.

Consider addressing this particular scenario to be more consistent with the rest of the code and prevent the reversion of the `drip` function when the L2 fraction changes to zero.

Update: Fixed in commit `b5967ac866aef2b0bf4094b5db76de0ed1d03446` from pull request 589. More documentation has been added along with a more detailed revert message.

L-03 Potential incorrect rewards emission when GRT token supply changes

The L1 reservoir emits reward tokens based on the `tokenSupplyCache` variable. This variable is initialized with the `totalSupply` of GRT tokens and it is only updated in the `drip` function to include new emission of reward tokens.

Any new mint or burn of the GRT token that is performed outside of the `drip` function will not be reflected back to the `tokenSupplyCache` variable, which means that the rewards emission could be based on an out-of-date total supply, causing over- or under-emitting. The Graph team has confirmed that they expect only `drip` would mint GRT tokens, however this issue will apply if GRT minting and burning is done through other channels in the future.

Consider thoroughly documenting this design decision and taking extra precautions to ensure that the GRT token supply will not change outside the reward system.

Update: Fixed in commit `d5fa0a8bbd28d06abd2b3f48bf860d73ff17a7d7` from pull request 590. The supply and the value used for issuing new rewards are now decoupled.

L-04 One-time use function can be called repeatedly

The `L1Reservoir` contract has an `initialSnapshot` function that records the total supply of GRT tokens before switching over to the new drip rewards approach. It also has the ability to mint tokens to cover any pending rewards, with the amount specified by the caller. The GIP-0034 specification states that this function is intended to be called only once after deployment of the `L1Reservoir` contract.

Although calls to this function are restricted by the `onlyGovernor` modifier, there is no check in place to prevent this one-time use function from being called multiple times. If the function is called again accidentally or intentionally, it will reset the rewards snapshot

(`lastRewardsUpdateBlock` and `tokenSupplyCache`) and could also result in the minting of additional tokens outside the drip mechanism. Once the initial snapshot is recorded, `lastRewardsUpdateBlock` and `tokenSupplyCache` should only be updated by `snapshotAccumulatedRewards` as part of a call to the `drip` function, otherwise the `t0` value used to compute total rewards is changed without the corresponding issuance of rewards.

Consider adding a boolean state variable to prevent invoking the `initialSnapshot` function more than once.

Update: Fixed in commit `494df531d05b4996c0dac70be236b6d22dedbc79` from pull request 591.

L-05 L2 initial supply feature invalidates token supply assumptions

The GIP-0034 proposal states that the `L2Reservoir` contract will receive a fraction of all rewards tokens minted on the L1 side. The new rewards system is designed to use the `L2GraphToken` contract's `bridgeMint` function to create all L2 GRT tokens. However, the `L2GraphToken` contract's `initialize` function takes an `_initialSupply` parameter that enables the minting of GRT tokens on the L2 side of the bridge that have no corresponding token on the L1 side. Minting tokens via the `initialize` function would result in an incorrect calculation of the `normalizedSupply` value which currently assumes that the supply of tokens on the L2 side can be computed solely as a fraction of the total supply of GRT on the L1 side.

To prevent a miscalculation of the normalized token supply, consider removing the `_initialSupply` parameter from the `initialize` function declaration, and using a hard-coded value of 0 for the `_initialSupply` parameter required when calling `GraphTokenUpgradeable._initialize`. Also consider removing the corresponding documentation for the `_initialSupply` parameter.

Update: Fixed in commit `29702d6e87fe6e0a526fb55dd79d7f318b29d240` from pull request 592.

L-06 L2 Reservoir issuance rate might not be accurate

The `issuanceRate` variable from the `ReservoirStorage` contract in L2 is updated through the `receiveDrip` function of the `L2Reservoir` contract, which is called whenever a drip happens on L1.

However, it is possible that the `drip` function call might fail after being mined in L1 or get delayed on the bridge, producing a delay in updating the L2 reservoir `issuanceRate` variable, which in turn could cause any function dependent on that value to return an inaccurate output, for instance the `getNewRewards` function.

Considering either separating the `issuanceRate` variable updating mechanism out of the `drip` function, or thoroughly documenting the expectations regarding the accuracy of any related L2 reservoir functions/variables.

Update: Fixed in commit `2f41f81b5ccaa25400d893cae50c6a8193fedf1c` from pull request 598. More documentation has been added to warn about this possible outcome.

L-07 Lack of input validation

Some functions in the codebase lack sufficient input validation. For instance:

- The setter and whitelist functions on the `L1GraphTokenGateway` contract do not validate their inputs, allowing even zero addresses to be set.
- The setter and whitelist functions from the `L2GraphTokenGateway` contract do not validate all the input addresses.
- The destination of the tokens during an `outboundTransfer` call is not checked against the zero address, possibly burning those tokens into the zero address in L2.
- The `setL2ReservoirAddress` function in the `L1Reservoir` contract does not check if the address is zero, which could lead to tokens being burnt during a `drip` call.
- The `renounceMinter` function from the `L2GraphToken` contract allows any address to renounce their minter role, even if those addresses are not allowed, triggering the `MinterRemoved` event unnecessarily.

To avoid errors and unexpected system behavior, consider implementing `require` statements to validate all user-controlled inputs. Furthermore, consider supporting and implementing the `ERC165` standard on the known contracts to prevent setting the wrong addresses.

Update: Fixed in commit `68cc8d3e69ed244adc251eb5618be83b650a3322` from pull request 593, commit `e8544e83a09ae06cf5946a123aa51a243ac1dd49` from pull request 594, and commit `06275b755dcbf4d192887080ad4cc405bd13001e` from pull request 595. The functions to add and remove addresses from the whitelist in the `L2GraphTokenGateway` contract still accept zero addresses. However, these functions have been removed from the code in the latest pull request.

L-08 Multiple sources of truth for the allowed callhook callers

There are two `callhookWhitelist` mappings on both L1 and L2 gateways performing the same function: checking if the L1 to L2 `msg.sender` address is allowed to execute call data on bridge calls.

Although this design offers fine control over who is allowed to execute call data for both L1 and L2, it does come with certain concerns:

1. Since these two lists are maintained on different chains, messages could fail if those are out of sync.
2. Race conditions apply when removing addresses from the L1 list, although this can be mitigated if the removal is done on the L2 whitelist before updating the one in L1.
3. The design requires whitelisted addresses to be validated twice on an L1 to L2 message.

An alternative would be to keep just one whitelist mapping on L1 which will eliminate the need to sync the two lists, but it also means that the protocol will lose control over the L2's redeem process as well as the procedure to eliminate race conditions as described in the 2nd point above.

Both designs comes with risks and benefits. Consider evaluating these carefully before making a final decision on the design pattern and documenting the strategy used.

Update: Partially fixed in commit `06275b755dcbf4d192887080ad4cc405bd13001e` from pull request 595. The whitelist function was removed from the L2 side, however further documenting of the design choice would be beneficial.

L-09 L1 gateway outbound transfer could fail if `msg.value` is too big

During an outbound transfer, the `L1GraphTokenGateway` contract requires the provided `msg.value` to be the exact ticket submission cost plus L2 gas:

```
uint256 expectedEth = maxSubmissionCost + (_maxGas * _gasPriceBid);
require(msg.value == expectedEth, "WRONG_ETH_VALUE");
```

This tight restriction is unnecessary because Arbitrum bridge allows more ETH to be sent with the ticket, and it will refund the excess of ETH to the credit-back address (in this case, the original caller).

Consider relaxing the gas requirement in order to prevent rejection of outbound requests when more `msg.value` than needed is sent.

Update: Fixed in commit `395d8588d1782a3882fee54a12c820a9afa547c0` from pull request 596.

L-10 L1 accounting system pushes incorrect supply value

The `L1Reservoir` contract allows the dripping of tokens from L1 into L2. Due to the dynamic nature of the amount needed in L2, the Governor can modify the amount to send into L2 by changing its fraction.

Although the implementation takes into account changes of the fraction for the next drip, the normalized supply in L2 set during the execution of the `drip` function only takes into consideration the latest fraction without considering the history of this value and the actual tokens sent through the bridge into L2. For this reason and because this value overwrites the normalized token supply cache without any validation during the `receiveDrip` function hook call, calculations may be based on an incorrect cached balance.

Consider either accumulating the actual dripped value received from L1 instead of periodically overwriting it with a value calculated on L1, fixing the calculation in L1 to take into account past changes of the fraction, or documenting the rationale for leaving the accounting system in its current form.

Update: Fixed in commit `0b516439f7da02afc769c9487bb16afed8cbb8be` from pull request 599. The Graph Team's response:

> *I think this might be a misunderstanding. The "normalized supply" sent to L2 is not meant to track the amount of tokens sent to L2, but the base for the rewards calculation. This is explained in GIP-0034 where I introduce a q = lambda * p variable for convenience. To help avoid this confusion I've renamed the variable on the L1Reservoir: https://github.com/graphprotocol/contracts/pull/599*

L-11 Require statements with multiple conditions

Within the codebase there are `require` statements that require multiple conditions to be satisfied. For instance:

- The `require` statement on line 205 of `GNS.sol`
- The `require` statement on line 369 of `Staking.sol`

To simplify the codebase and to raise the most helpful error messages for failing `require` statements, consider having a single require statement per condition.

Update: Partially fixed in commit `09f40d160b2a65e9672ca005fa85e329bd52d3e4` from pull request 568. Only the case in the `GNS` contract has been fixed. The Graph Team's response for the issue:

> *For the one on the Staking contract, the numerator and denominator are very closely linked so we think it's fine to keep them in a single require.*

L-12 Unset RewardsManager fails silently

Within the `Staking` contract, there are several instances where the following pattern appears:

```
        IRewardsManager rewardsManager = rewardsManager();
        if (address(rewardsManager) == address(0)) {
            return;
        }
```

This code pattern can be found in the `_updateRewards`, `_distributeRewards`, and `_takeAndBurnRewards` functions. Rather than revert when the `RewardsManager` address is unavailable, these reward manipulation functions return successfully.

Using the `_closeAllocation` function as an example, depending on whether proof of indexing was presented it calls either the `_distributeRewards` or the `_takeAndBurnRewards` function, and assumes that these functions completed their respective transfer or burn operations as long as they return successfully. If the rewards

manager is not set, these functions will not perform any token operations, resulting in either a denial of rewards or a failure to reduce the token supply as intended.

To avoid silent errors involving rewards management, consider reverting when `rewardsManager` address is the zero address.

Update: Fixed in commit `f561a34aead61a1ceee0014860f892d460285e39` from pull request 600.

L-13 Token gateway does not guarantee atomicity for inbound requests with transaction data

For inbound transfers, the L2 token gateway allows execution of optional call data if the sender is whitelisted. In the opposite case when the sender is not whitelisted, it will simply ignore the extra transaction data while still minting the bridged tokens in line 238 of `L2GraphTokenGateway.sol`.

Both the L1 and L2 token gateway contain callhook whitelists, and normally these lists are expected to be in-sync, allowing a transaction with extra data to pass both whitelist validations. However, in the situation where an address is present only in the L1 whitelist and not in the corresponding L2 mapping, this could break the logic of the original transaction if it requires atomicity between its transaction data and the bridging of GRT assets.

Moreover, if the design intention is to handle asset bridging and transaction data without atomicity, then the token gateway should not revert the minting process if the transaction data fails.

Although this design makes sense for the current drip function to ensure GRT tokens are always minted on L2, it might be unsuitable for other transaction calls in the future.

Consider thoroughly documenting the design rationale and be extremely careful with the addition of new callhook endpoints in the future.

Update: Fixed in commit `06275b755dcbf4d192887080ad4cc405bd13001e` from pull request 595 and commit `8426b4c4116650dabee5085aaa636f48d2d4fadf` from pull request 646. The whitelist function was removed from the L2 side and further documenting of the design choice was added.

L-14 Not using OpenZeppelin's libraries

Within the codebase, there are places where logic already available in OpenZeppelin's contracts library is re-implemented. For instance:

- The `L2GraphToken` contract implements a minter role access control which could be incorporated by using the `AccessControl` contract.
- The `L2GraphToken` contract implements the permit functionality instead of using the `ERC20Permit` contract.

OpenZeppelin contracts are focused on providing a safe implementation of a diverse set of common smart contract design patterns. Consider using these libraries and contracts whenever possible to reduce the attack surface created by custom implementations.

Update: Not fixed. The Graph's team statement for this issue:

> *We'd rather keep it as it is, so that the L2 GRT is as similar as possible to the code for the L1 contract (that was developed before those features were available on the OZ library)*

Notes & Additional Information

N-01 Implicit ERC20 allowance requirement

The `outboundTransfer` function from the `L1GraphTokenGateway` contract implicitly assumes the token holder has provided the contract with an allowance to spend the specified amount of underlying asset tokens. Consider documenting this in the function comments.

Update: Fixed in commit `2582b109b38f78d42e5c8e8951605732ac19f4c8` from pull request 603.

N-02 Deprecated or unused events

Within the codebase, there are cases of deprecated or unused events:

- The `OutboundTransferInitiated` and `InboundTransferFinalized` events in `ITokenGateway.sol` both include a comment indicating those events have been deprecated, but the commented out code has not been removed from the file.
- The `CallhookFailed` event from the `L2GraphTokenGateway` contract is not being used.

Consider either documenting the reason to keep these events, or removing them from the codebase to improve the code's readability.

Update: Partially Fixed in commit `bb7f2e7eabd40135d10daaad61601f2884a65b39` from pull request 604. The `CallhookFailed` event was removed. The Graph Team's statement for this issue:

> *the events on ITokenGateway are left there because we copied that file from the Arbitrum repo - I'd rather leave it as-is to minimize diff between the files*

N-03 Gas optimization on permits

The `L2GraphToken` contract manually implements support for permits. However, its implementation increments the nonce before validating the recovered address or its deadline.

In favor of reducing the gas consumption on reverted transactions, consider increasing the nonce after all validation checks have been completed.

Update: Fixed in commit `0ea0d303f85887cafbcf3920648b06914a9265c1` from pull request 605.

N-04 Gateway uses a separate pausing mechanism

Most of the contracts in the protocol use the pausing mechanism from the `Managed` contract. However, the `GraphTokenGateway` contract uses its own implementation. Although this is an intended design to allow the gateway to be paused separately, it also means it does not auto-pause `GraphTokenGateway` contracts when the system is paused from the `Managed` contract.

Consider taking extra steps to double check when using these pausing functions to avoid partially pausing the system.

Update: Not fixed. The Graph Team's statement for the issue:

> *it's a good point - we'll have to consider this so that pause guardians are aware (though I don't see any possible code changes to address this).*

N-05 General code improvements

Throughout the codebase, there are places where the readability of the code could be improved. In particular:

- The usage of underscore prefixes in parameters' names is not consistent in every function.
- In the `takeRewards` function from the `RewardsManager` contract, the `return` instruction could be placed inside the body of the first condition, after line 399.
- In the `L2GraphToken` contract, the order of functions is not based on their visibility, as an `internal` function appears before an `external` one.
- The `permit` functionality in the `L2GraphToken` contract packs the `r`, `s`, and `v` parameters when calling `ECDSA.recover` even though there is an alternate `recover` method which accepts those unpacked parameters. Furthermore, the `recover` method currently being called will unpack the parameters and call the alternate method anyway, increasing the gas cost without any need.

In favor of improving the readability of the code, consider applying the aforementioned suggestions into the code.

Update: Fixed in commit `31e07c42989bba6427678443dbb940a02b717d92` from pull request 606.

N-06 Incomplete documentation

Consider adding the following missing information to incomplete docstrings:

- In `L2GraphToken.sol`:
  - line 76: `_initialize` docstring is missing the `owner` parameter
  - line 252: `setGateway` docstring is missing the `gw` parameter
  - line 260: `setL1Address` docstring is missing the `addr` parameter

Update: Fixed in commit `14ada453964280d5ce47b6fef20457d15283981b` from pull request 607.

N-07 Inconsistent setting of drip interval

In the `L1Reservoir` contract, the `initialize` function sets the `dripInterval` value, but it does not use the `setDripInterval` function to do it. The `setDripInterval` function performs a zero check on the interval value and emits a log event, but is restricted by the `onlyGovernor` modifier, so it cannot be called directly by the `initialize` function.

To consistently perform the same steps when the drip interval is being set, and still keep the existing security restrictions in place, consider creating an internal function that accomplishes the same steps as the `setDripInterval` function, and having both `initialize` and `setDripInterval` functions call this shared function.

Update: Fixed in commit `8742cc85f2a02eb13c512fd6bdca70d4f969cff6` from pull request 608.

N-08 Inconsistent use of named return variables

Named return variables are used inconsistently. For example, while lines 300 and 266 to 268 of `L1GraphTokenGateway.sol` name their return variables, line 171 does not.

Consider removing all named return variables, explicitly declaring them as local variables, and adding the necessary return statements where appropriate. This should improve both explicitness and readability of the project.

Update: Fixed in commit `3002d3e07bb7636f8e33d0520c23255da29e1084` from pull request 609.

N-09 Incorrect or outdated documentation

We have identified several locations where the comments and/or docstrings appear to be incorrect. Consider revising the documentation in the following cases:

- In `L1GraphTokenGateway.sol`:
  - line 33: The comment says "Address of the L1 Reservoir that is the only sender allowed to send extra data", but the mapping supports multiple addresses, not just a single one.
  - line 194: The comment says that the user must provide "a msg.value of AT LEAST maxSubmissionCost", which implies that a range of values is acceptable, but in the corresponding check on line 196, `msg.value` must be exactly equal to `expectedEth`.
  - line 291: The comment states "Additional call data for the L2 transaction, which must be empty", but `outboundTransfer` can encode non-empty data if the msg.sender is whitelisted.
- In `L2GraphTokenGateway.sol`:
  - line 220: The docstring states "data param is unused because no additional data is allowed from L1", but the `_data` param is used to execute a callhook if data is present and the `_from` address is whitelisted.
- In `L1Reservoir.sol`:
  - line 84: The docstring states "The value is in fixed point at 1e18 and must be less than 1", but the check on line 88 allows the `_l2RewardsFraction` to be equal to 1 (1e18).

Additionally, in the following locations, the documentation appears to need updating:

- In `RewardsManager.sol`:
  - line 307: The docstring for `updateAccRewardsPerSignal` has not been updated to reflect the fact that `tokenSupplySnapshot` is modified when the function is called.

Update: Fixed in commit `5c1f8773be384304a603b9591dd3e88f4273b6b1` from pull request 610.

N-10 Initialization inconsistency

Several of the contracts in the codebase are meant to follow the initialization upgradeability pattern, using setter functions to define most of the parameters in the protocol.

Some initializers do not set anything except the controller's address, and there are other cases where only a subset of the protocol's parameters are being set during initialization.

Although the reason could be not knowing the values at the deployment stage, in favor of improving the code's readability, consider documenting the reason why some protocol parameters are not set by initializer functions.

Update: Fixed in commit `bd236162e1b136895ee7c721235817984f488b25` from pull request 611.

N-11 Getter function inconsistency

The `Managed` contract implements an interface to interact with the controller but also provides local caching for other protocols' contract addresses.

In the pull requests 552 and 571, the contract added the `Reservoir` and the `GraphTokenGateway` to the sync list. However, the corresponding getter functions were not implemented to easily query their addresses, resulting in the getter functionality being inlined in the following locations:

- `L1Reservoir.sol`, line 249
- `L2Reservoir.sol`, line 28
- `RewardsManager.sol`, line 440

In order to improve the consistency of the code, consider adding the getter functions to all the synced contracts in the `Managed` contract.

Update: Fixed in commit `0f5fc9bd113202b181e56606d5852d6650a81fda` from pull request 612.

N-12 Re-implementation of Arbitrum library

The `L2GraphTokenGateway` contract re-implements logic intended to prevent cross-chain exploits from Arbitrum's `AddressAliasHelper` library, as this library is written in Solidity `^0.6.11`.

However, this behavior is not consistent with the rest of dependencies copied from the Arbitrum project, which have had their Solidity version updated to the one used by the project.

To improve the code's readability and consistency, consider copying the library and updating its Solidity version to `^0.7.6`, as was done with the rest of the files.

Update: Fixed in commit `46a8444b1574addfcd84c74316f8a6d17fdc3554` from pull request 613.

N-13 Multiple contracts per file

For clarity, consider splitting up the following files and placing each contract in its own separate file with a matching filename:

- `L2GraphToken.sol`: Contains the `GraphTokenUpgradeable` and `L2GraphToken` contracts; can be decomposed into `GraphTokenUpgradeable.sol` and `L2GraphToken.sol`
- `IReservoir.sol`: Contains the `IReservoir` interface and `IL2Reservoir` interface; can be decomposed into `IReservoir.sol` and `IL2Reservoir.sol`

Despite the above guidance, in the case of `RewardsManagerStorage.sol`, placing multiple contract versions within the same file improves clarity because it groups all the storage variables together in one place, so no changes are suggested for this file.

Update: Fixed in commit `505a2040378d3f20c59a922c118c6df712631602` from pull request 616.

N-14 Naming suggestions

We believe some constants and variables could benefit from renaming. These are our suggestions:

- The `TOKEN_DECIMALS` constant is assigned a value of `1e18`, but the name suggests a value of `18`. Consider renaming this constant to `TOKEN_PRECISION`.
- The `outboundTransfer` function in `L2GraphTokenGateway` contains a local variable named `s` that is type `OutboundCalldata`. It is best practice to avoid single-letter variable names that do not convey any information about the value stored. Consider changing this from `s` to a more descriptive name such as `callhookData` or `outboundData`.

Update: Fixed in commit `b7b2a6c6b583031c2418ec6bfa5735e0a883d405` from pull request 617.

N-15 Lack of documentation for potential out-of-order calls to `drip`

The `L1Reservoir` contract allows dripping tokens from L1 to L2 by using Arbitrum's bridge. For each drip, a nonce is attached which has to match with the `nextDripNonce` value expected by the L2 Reservoir.

However, in the case where the Arbitrum sequencer does not correctly sort the `drip` transactions as it should and the 2nd `drip` is executed before the 1st `drip` call, then the 2nd transaction will fail as it does not match the current nonce and only the 1st transaction will succeed shortly after. Because funds have been sent from the `L1Reservoir` contract to the escrow (in L1) and because the `L1Reservoir` cannot replay the bridge, these funds will get stuck.

Although it is possible to solve this problem by manually retrying the 2nd transaction (Arbitrum allows users to retry a transaction in L2 if a wrong sorting scenario appears) consider documenting the procedure to prevent assets from becoming soft-stuck.

Update: Fixed in commit `1f798d89f43a4a8b0177a5beb8db3dccf3a39cba` from pull request 618.

N-16 Typographical errors

Consider addressing the following typographical errors:

- In `IReservoir.sol`:
  - line 8: "on each layers" should be "on each layer"
- In `ITokenGateway.sol`:
  - line 69: "but not deploy" should be "but not deployed"
- In `L1GraphTokenGateway.sol`:
  - line 94: "sets" should be "Sets"
  - line 192: "required for" should be "as required for"
- In `L1Reservoir.sol`:
  - lines 128-130: "L2RewardsFraction" should be "I2RewardsFraction"
- In `L2GraphToken.sol`:
  - line 18: "Permit()" should be "permit()"
  - line 105: "version" should be "recovery identifier" or "recovery id"
- In `L2GraphTokenGateway.sol`:
  - line 15: "Sending" should be "Sends"
- In `RewardsManager.sol`:
  - line 307: "save" should be "saves the"

Update: Fixed in commit `012e3d2d3637bb9a719886bb4e1df6488058debd` from pull request 619.

N-17 Unnecessary inheritance

The `GraphTokenUpgradeable` contract inherits directly from two OpenZeppelin contracts: `ERC20Upgradeable` and the extension contract `ERC20BurnableUpgradeable`. However, `ERC20BurnableUpgradeable` also inherits from `ERC20Upgradeable`, making it unnecessary to include the latter in the inheritance list.

Consider removing `ERC20Upgradeable` from the list of base contracts for `GraphTokenUpgradable`.

Update: Fixed in commit `1e171dbfe62fa61ee52b512473138dc56abaadd8` from pull request 620.

N-18 Unnecessary private visibility on MAX_UINT256 constant

The `BridgeEscrow` contract declares `MAX_UINT256` as private on line 16:

```
uint256 private constant MAX_UINT256 = 2**256 - 1;
```

In the `Reservoir` contract this happens again on line 22:

```
    uint256 private constant MAX_UINT256 = 2**256 - 1;
    uint256 internal constant TOKEN_DECIMALS = 1e18;
    uint256 internal constant MIN_ISSUANCE_RATE = 1e18;
```

The value of `MAX_UINT256` is universal so there's no benefit to limiting the inheritance of this constant by restricting the visibility to `private`. Consider changing `private` in these instances to `internal`.

Update: Fixed in commit `a11501f819f24d8642b74886d596d3e3ee6e7489` from pull request 622.

N-19 Unnecessary code

Consider making the following changes to eliminate unnecessary or redundant code:

- In `L1GraphTokenGateway.sol`:
  - `exitNum` is decoded from the input calldata, but it is always encoded as 0 when `L2GraphTokenGateway` generates the outbound calldata. Due to this invariance of the `exitNum` value, consider eliminating the decoding step and hard coding an `exitNum` value of 0 in the `WithdrawalFinalized` event emission.
- In `L2GraphTokenGateway.sol`:
  - The `calculateL2TokenAddress` function is called using the `this` keyword, but the function is already declared `public`, so `this` is not required in order to call it.
  - The call to `outboundTransfer` contains unnecessary uint256() casts.
  - Within the `finalizeOutboundTransfer` function, the second condition in the callhook data check can be shortened from `callhookWhitelist[_from] == true` to `callhookWhitelist[_from]`.

Update: Fixed in commit `65829afc20a4357f67e48e0860cc7df0d4ea84e8` from pull request 621.

N-20 Unneeded maximum value calculation

Within the scope of the audited code, there are instances where the maximum value for the `uint256` variable type is calculated as `2**256 - 1`:

- On line 16 from `BridgeEscrow.sol`
- On line 22 from `Reservoir.sol`

However, as of Solidity v0.6.8, the max and min values for every integer type `T` can be accessed directly via the syntax `type(T).max` and `type(T).min`.

Use of this syntax can make the code more clear and readable to stakeholders, and can reduce bugs in implementation. Consider accessing the extreme values of integer types directly using the `type(T).max`/`type(T).min` syntax.

Update: Fixed in commit `a11501f819f24d8642b74886d596d3e3ee6e7489` from pull request 622.

N-21 Unused imports

To improve readability and avoid confusion, consider removing the following unused imports:

- In the `L1Reservoir` contract, the IReservoir interface
- In the `ReservoirStorage` contract, the IReservoir interface
- In the `RewardsManagerStorage` contract, the IRewardsManager interface

Update: Fixed in commit `960cc04584945e206ff61eb885f5eef9d412395d` from pull request 623. The Graph team explained one of the imports was already removed in another fix, and the IRewardsManager import is actually used for the Subgraph type.

N-22 Unused return values

Within the scope of the audited code, there are some cases of unused return values:

- The `updateAccRewardsPerSignal` function in the `RewardsManager` contract returns the accumulated rewards per signal. However, none of the locations where this function is called make use of the value.

- The `transferSignal` function in the `GNS` contract always returns the value `true`. This return value is also not documented in the function's docstring.

In favor of improving the code's readability, consider removing these unused return values, or documenting the reason to have them in the first place.

Update: Partially fixed. The Graph Team's statement for this issue:

> *updateAccRewardsPerSignal is public and permissionless, so I'd rather keep its return value as it is rather than breaking backwards compatibility.*

N-23 Whitelist add/remove functions don't check if address already exists

The `addToCallhookWhitelist` and `removeFromCallhookWhitelist` functions in both the `L1GraphTokenGateway` and `L2GraphTokenGateway` contracts do not first check to see if the target address is already present before adding or removing it. An attempt to add an existing address will result in emitting the `AddedToCallhookWhitelist` event, even though nothing was added. Likewise, an attempt to remove an address that is not on the callhook whitelist will result in emitting the `RemovedFromCallhookWhitelist`, even though nothing was removed.

To avoid emitting events when no state change has occurred, consider implementing logic within `addToCallhookWhitelist` and `removeFromCallhookWhitelist` that checks for the presence of the address being added or removed before emitting an event.

Update: Fixed in commit `3a6c3b5f8bd005d72036341bf8314106b36d26f0` from pull request 624.

Conclusions

No critical or high severity issue were found. Potential outcomes from the proposed design were highlighted. Recommendations have been given to ensure production readiness of the code, improve quality, and minimize errors.