The Graph L2 bridge contest Findings & Analysis Report

2022-12-21

Table of contents

- Overview
  - About C4
  - Wardens
- Summary
- Scope
- Severity Criteria
- Medium Risk Findings (4)
  - [M-01] Initialize function in `L2GraphToken.sol`, `BridgeEscrow.sol`, `L2GraphTokenGateway.sol`, `L1GraphTokenGateway.sol` can be invoked multiple times from the implementation contract
  - [M-02] If L1GraphTokenGateway's `outboundTransfer` is called by a contract, the entire `msg.value` is blackholed, whether the ticket got redeemed or not
  - [M-03] Governor can rug pull the escrow
  - [M-04] After proposed 0.8.0 upgrade kicks in, L2 `finalizeInboundTransfer` might not work
- Low Risk and Non-Critical Issues

  - Non-Critical Issues List
  - Low Risk Issues List
  - N-01 Testing all functions is best practice
  - N-02 Include `return parameters` in *NatSpec comments*
  - N-03 `0 address` check
  - N-04 Use a more recent version of Solidity
  - N-05 Require revert cause should be known
  - Context
  - N-06 Use `require` instead of `assert`
  - N-07 For modern and more readable code; update import usages
  - N-08 `Function writing` that does not comply with the `Solidity Style Guide`
  - N-09 Implement some type of version counter that will be incremented automatically for contract upgrades
  - N-10 NatSpec is Missing
  - N-11 Omissions in Events
  - N-12 Constant values such as a call to `keccak256()`, should used to immutable rather than constant
  - N-13 Floating pragma
  - N-14 Inconsistent Solidity Versions
  - N-15 For extended “using-for” usage, use the latest pragma version
  - N-16 For functions, follow Solidity standard naming conventions
  - N-17 Add to `_blacklist function_`
  - L-01 Using Vulnerable Version of Openzeppelin
  - L-02 Avoid *shadowing* `inherited state variables`
  - L-03 NatSpec comments should be increased in contracts
- Gas Optimizations
  - Gas Optimizations Summary
  - G-01 State variables can be packed into fewer storage slots

- G-02 Avoid contract existence checks by using low level calls
- G-03 State variables should be cached in stack variables rather than re-reading them from storage
- G-04 `internal` functions only called once can be inlined to save gas
- G-05 `require()` / `revert()` strings longer than 32 bytes cost extra gas
- G-06 `keccak256()` should only need to be called on a specific string literal once
- G-07 Optimize names to save gas
- G-08 Using `bool` s for storage incurs overhead
- G-09 Use a more recent version of solidity
- G-10 Use a more recent version of solidity
- G-11 Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement
- G-12 Splitting `require()` statements that use `&&` saves gas
- G-13 Don’t compare boolean expressions to boolean literals
- G-14 Stack variable used as a cheaper cache for a state variable is only used once
- G-15 `require()` or `revert()` statements that check input arguments should be at the top of the function
- G-16 Use custom errors rather than `revert()` / `require()` strings to save gas
- G-17 Functions guaranteed to revert when called by normal users can be marked `payable`

- Mitigation Review
  - Mitigation overview
  - Findings
- Disclosures

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the The Graph L2 bridge smart contract system written in Solidity. The audit contest took place between October 7—October 12 2022.

Following the C4 audit contest, warden Trust reviewed the mitigations for all identified issues; the mitigation review report is appended below the audit contest report.

Wardens

65 Wardens contributed reports to the The Graph L2 bridge contest:

1. Trust
2. ladboy233
3. csanuragjain
4. cccz
5. catchup
6. joestakey
7. d3e4
8. 0xSmartContract
9. lllllll
10. rbserver
11. delfin454000
12. mics
13. brgltd
14. 0x1f8b

15. c3phas
16. ElKu
17. __141345__
18. Bnke0x0
19. RaymondFam
20. gogo
21. mcwildy
22. oyc_109
23. fatherOfBlocks
24. 0xNazgul
25. rotcivegaf
26. ajtra
27. zzzitron
28. Chom
29. chrisdior4
30. bobirichman
31. Rahoz
32. Waze
33. nicobevi
34. bulej93
35. 0x4non
36. Josiah
37. Deivitto
38. RockingMiles (robee and pants)
39. Rolezn
40. Saintcode_
41. emrekocak
42. sakman
43. KoKo

44. carlitox477
45. RedOneN
46. aysha
47. gianganhnguyen
48. 0xdeadbeef
49. martin
50. pedr02b2
51. erictee
52. B2
53. Tomio
54. TomJ
55. saian
56. Shinchan (Sm4rty, prasantgupta52, and Rohan16)
57. ret2basic
58. medikko
59. Jujic
60. Pheonix
61. zishansami
62. gerdusx

This contest was judged by 0xean.

Mitigations reviewed by Trust.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 4 unique vulnerabilities. Of these vulnerabilities, 0 received a risk rating in the category of HIGH severity and 4 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 30 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 44 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 The Graph L2 bridge contest repository, and is composed of 23 smart contracts written in the Solidity programming language and includes 1,181 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

Medium Risk Findings (4)

[M-01] Initialize function in `L2GraphToken.sol`, `BridgeEscrow.sol`, `L2GraphTokenGateway.sol`,

`L1GraphTokenGateway.sol` can be invoked multiple times from the implementation contract

Submitted by ladboy233, also found by csanuragjain

L2GraphTokenGateway.sol#L87
L2GraphToken.sol#L48
L1GraphTokenGateway.sol#L99
BridgeEscrow.sol#L20

Initialize function in `L2GraphToken.sol`, `BridgeEscrow.sol`, `L2GraphTokenGateway.sol`, and `L1GraphTokenGateway.sol` can be invoked multiple times from the implementation contract. This means a compromised implementation can reinitialize the contract above and become the owner to complete the privilege escalation then drain the user's fund.

Usually in Upgradeable contract, an initialize function is protected by the modifier

```
initializer
```

to make sure the contract can only be initialized once.

Proof of Concept

1. The implementation contract is compromised,
2. The attacker reinitialize the BridgeEscrow contract

```
    function initialize(address _controller) external onlyImpl {
        Managed._initialize(_controller);
    }
```

the onlyGovernor modifier's result depends on the controller because

```
    function _onlyGovernor() internal view {
        require(msg.sender == controller.getGovernor(), "Caller
    }
```

3. The attacker has the governor access to the BridgeEscrow,
4. The attack can call the approve function to approve malicious contract

```
    function approveAll(address _spender) external onlyGovernor
        graphToken().approve(_spender, type(uint256).max);
    }
```

5. The attack can drain all the GRT token from the BridgeEscrow.

Recommended Mitigation Steps

We recommend the project use the modifier

```
initializer
```

to protect the initialize function from being reinitiated

```
    function initialize(address _owner) external onlyImpl initial
```

Oxean (judge) commented:

> Sponsor - please also review **#72** which is a duplicate but has a slightly different impact statement to the same underlying issue.

pcarranzav (The Graph) confirmed and commented:

> The impact from **#72** was discussed with a warden and I consider it valid and worth fixing.

> The impact from this one requires the implementation to be compromised in a way that allows calling `initialize()`. Since all implementations use `GraphUpgradeable`, I don't see how that's possible without also compromising the `GraphProxyAdmin` and therefore compromising the governor, which would mean much bigger problems.

> I still consider this risky enough that a Medium severity is reasonable, both in #72 and this one.

Trust (warden) commented:

> Exploit of this bug requires ProxyAdmin compromise, and if attacker has that then there are many ways to severely damage the proxy. It's a good observation and should be fixed, but it seems to be a best practice issue.

pcarranzav (The Graph) resolved and commented:

> Fix PRd in https://github.com/graphprotocol/contracts/pull/741

Oxean (judge) commented:

> @Trust - makes reasonable points, but I think it's still correct to award this as Medium.

Trust (warden) reviewed mitigation:

> Fixed in standalone PR. Fixed by using OpenZeppelin's initializer guard in contracts BridgeEscrow, L1GraphTokenGateway, L2GraphTokenGateway and L2GraphToken.

[M-02] If L1GraphTokenGateway's `outboundTransfer` is called by a contract, the entire `msg.value` is blackholed, whether the ticket got redeemed or not

Submitted by Trust

L1GraphTokenGateway.sol#L236

The outboundTransfer function in L1GraphTokenGateway is used to transfer user's Graph tokens to L2. To do that it eventually calls the standard Arbitrum Inbox's createRetryableTicket. The issue is that it passes caller's address in the `submissionRefundAddress` and `valueRefundAddress`. This behaves fine if caller is an EOA, but if it's called by a contract it will lead to loss of the submissionRefund (ETH passed to outboundTransfer() minus the total submission fee), or in the event of failed L2 ticket creation, the whole submission fee. The reason it's fine for EOA is because of the fact that ETH and Arbitrum addresses are congruent. However, the calling contract probably does not exist on L2 and even in the rare case it does, it might not have a function to move out the refund.

The docs don't suggest contracts should not use the TokenGateway, and it is fair to assume it will be used in this way. Multisigs are inherently contracts, which is one of the valid use cases. Since likelihood is high and impact is medium (loss of submission fee), I believe it to be a HIGH severity find.

Impact

If L1GraphTokenGateway's outboundTransfer is called by a contract, the entire msg.value is blackholed, whether the ticket got redeemed or not.

Proof of Concept

Alice has a multisig wallet. She sends 100 Graph tokens to L1GraphTokenGateway, and passes X ETH for submission. She receives an L1 ticket, but since the max gas was too low, the creation failed on L2 and the funds got sent to the multisig address at L2. Therefore, Alice loses X ETH.

Tools Used

- https://github.com/OffchainLabs/arbitrum/blob/master/docs/L1_L2_Messages.md
- Manual audit

Recommended Mitigation Steps

A possible fix is to add an `isContract` flag. If sender is a contract, require the flag to be true.

Another option is to add a `refundAddr` address parameter to the API.

Oxean (judge) decreased severity to Medium and commented:

> I think adding the refundAddr to the API is a good suggestion. I don't see how this qualifies as High however and think Medium severity would be more appropriate given we are talking about gas refunds.

> 2 — Med: Assets not at direct risk, but the function of the protocol or its availability could be impacted, or leak value with a hypothetical attack path with stated assumptions, but external requirements.

> This seems like a leak of value situation rather than Assets being directly lost or stolen.

pcarranzav (The Graph) acknowledged, but disagreed with severity and commented:

> The POC described by the warden is no longer valid with Arbitrum Nitro. Retryable ticket submission fee is now checked in L1, so it's impossible that the ticket creation fails in L2 (Arbitrum docs might be outdated on this).

> It is true, however, that any *refunded* value would be lost if the sender is a contract, though they could be reused in future retryable tickets by the same sender (since, if I recall correctly, the refund address will be used to pay for gas in L2). This is standard behavior for retryable tickets though, and users sending funds through the bridge should be aware of this when bridging tokens: in the worst case, the sender has specified a max submission fee, gas price and max gas, and the resulting behavior would be equivalent to *exactly* those amounts being used.

> As for the suggested mitigation, it would require adding a new/overloaded interface, since the existing API is designed to be compatible with Arbitrum's Gateway Router and is standard for Arbitrum token bridges.

> For these reasons, I think this finding might be more of a QA severity?

0xean (judge) commented:

> While there may not be a reasonable fix for this, I still think it is a Medium severity issue that integrators / callers from multi-sigs should be aware of.

pcarranzav (The Graph) commented:

> Sounds fair 👍 I'd appreciate it if the final report noted that the ticket creation failure on L2 is not an issue though.

[M-03] Governor can rug pull the escrow

Submitted by d3e4, also found by catchup, cccz, and joestakey

BridgeEscrow.sol#L28-L30

Governor can rug pull all GRT held by BridgeEscrow, which is a severe undermining of decentralization.

Proof of Concept

The governor can approve an arbitrary address to spend any amount from BridgeEscrow, so they can steal all escrowed tokens. Even if the governor is well intended, the contract can still be called out which would degrade the reputation of the protocol (e.g. see here: https://twitter.com/RugDocIO/status/1411732108029181960). This is especially an issue as the escrowed tokens are never burnt, so the users would need to trust the governor perpetually (not about stealing their L2 tokens, but about not taking a massive amount of L1 tokens for free for themselves).

This seems an unnecessary power granted to the governor and turns a decentralized bridge into a needless bottleneck of centralization.

Recommended Mitigation Steps

Restrict access to `approveAll()` to the "bridge that manages the GRT funds held by the escrow". Or, similarly to how `finalizeInboundTransfer` in the gateways is restricted to its respective counterpart, only allow spending via other protocol functions.

0xean (judge) commented [via issue #40]:

> This is the intended design to allow for multiple bridges in the future. Going to leave open for sponsor review, but may decide to close as invalid.

pcarranzav (The Graph) disputed and commented [via issue #40]:

> Yes, this is intentional to allow multiple bridges but also to allow for recovery of the funds in case of a critical issue in Arbitrum.

> Note that the Governor is not a regular EOA, but a Gnosis Safe managed by The Graph Council; this account is already able to control many protocol parameters, including adding minters to the GRT contract, so adding this control over the escrow does not change the trust assumption significantly imo.

Oxean (judge) commented [via issue #40]:

> @pcarranzav - totally understand that. I personally would happily close this issue as invalid, but Code4rena seems to have a pattern of awarding these as Medium to ensure that users are aware of the risk, even as low as it may be.

[M-04] After proposed 0.8.0 upgrade kicks in, L2 `finalizeInboundTransfer` might not work

Submitted by Trust

L2GraphTokenGateway.sol#L70

L2GraphTokenGateway uses the onlyL1Counterpart modifier to make sure finalizeInboundTransfer is only called from L1GraphTokenGateway. Its implementation is:

```
    modifier onlyL1Counterpart() {
            require(
                msg.sender == AddressAliasHelper.applyL1ToL2Alias(l1
                "ONLY_COUNTERPART_GATEWAY"
            );
            _;
        }
```

It uses applyL1ToL2Alias defined as:

```
     uint160 constant offset = uint160(0x1111000000000000000000000

         /// @notice Utility function that converts the address in
   2
   2
```

```
    /// @return l2Address L2 address as viewed in msg.sender
    function applyL1ToL2Alias(address l1Address) internal pure
        l2Address = address(uint160(l1Address) + offset);
    }
```

This behavior matches with how Arbitrum augments the sender's address to L2. The issue is that I've spoken with the team and they are planning an upgrade from Solidity 0.7.6 to 0.8.0. Their proposed changes will break this function, because under 0.8.0, this line has a ~ 1/15 chance to overflow:

```
l2Address = address(uint160(l1Address) + offset);
```

Interestingly, the sum intentionally wraps around using the uint160 type to return a correct address, but this wrapping will overflow in 0.8.0

Impact

There is a ~6.5% chance that finalizeInboundTransfer will not work.

Proof of Concept

l1Address is L1GraphTokenGateway, suppose its address is 0xF000000000000000000000000000000000000000.

Then 0xF000000000000000000000000000000000000000 + 0x1111000000000000000000000000000000001111 > UINT160_MAX , meaning overflow.

Recommended Mitigation Steps

Wrap the calculation in an unchecked block, which will make it behave correctly.

0xean (judge) commented:

> This seems out of scope. The code has been asked to be audited given a certain compiler version. Will wait for sponsor review before closing as out of scope.

pcarranzav (The Graph) confirmed and commented:

> While I agree the submission is technically out of scope, I do consider it valuable. It's unclear if we'll upgrade to 0.8 soon or not, but knowing this does prevent a pretty bad potential issue if we do. Not sure if/how it would fit in the severity scale, but I would appreciate it if you could consider this eligible for awarding.

0xean (judge) commented:

> Great, happy to leave it as a Medium risk and valid.

pcarranzav (The Graph) resolved and commented:

> (Preemptively) fixed in https://github.com/graphprotocol/contracts/pull/738

Trust (warden) reviewed mitigation:

> Fixed in future 0.8.0 upgrade branch PR. Fixed by wrapping the potential overflowing and underflowing operations in an unchecked block.

Low Risk and Non-Critical Issues

For this contest, 30 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **0xSmartContract** received the top score from the judge.

The following wardens also submitted reports: rbserver, zzzitron, Bnke0x0, RaymondFam, ladboy233, IllIllI, Chom, 0x1f8b, Trust, cccz, chrisdior4, gogo, mcwildy, oyc_109, mics, bobirichman, fatherOfBlocks, Rahoz, Waze, 0xNazgul, nicobevi, bulej93, delfin454000, rotcivegaf, 0x4non, c3phas, brgltd, ajtra, *and* Josiah.

Non-Critical Issues List

Number	Issues Details	Context
[N-01]	Testing all functions is best practice	
[N-02]	Include `return parameters` in *NatSpec comments*	7

Number	Issues Details	Context
[N-03]	`0` address check	6
[N-04]	Use a more recent version of Solidity	All contracts
[N-05]	`Require` revert cause should be known	4
[N-06]	Use `require` instead of `assert`	1
[N-07]	For modern and more readable code; update import usages	13
[N-08]	`Function writing` that does not comply with the `Solidity Style Guide`	7
[N-09]	Implement some type of version counter that will be incremented automatically for contract upgrades	2
[N-10]	NatSpec is Missing	4
[N-20]	Omissions in Events	11
[N-12]	Constant values such as a call to keccak256(), should used to immutable rather than constant	
[N-13]	Floating pragma	All contracts
[N-14]	Inconsistent Solidity Versions	
[N-15]	For extended “using-for” usage, use the latest pragma version	
[N-16]	For functions, follow Solidity standard naming conventions	2
[N-17]	Add to *blacklist function*	

Total 16 issues

Low Risk Issues List

Number	Issues Details	Context
[L-02]	Using Vulnerable Version of Openzeppelin	1

Number	Issues Details	Context
[L-03]	A protected initializer function that can be called at most once must be defined	4
[L-04]	Avoid *shadowing* `inherited state variables`	1
[L-05]	NatSpec comments should be increased in contracts	12

Total 5 issues

[N-01] Testing all functions is best practice

Description

The test coverage rate of 91%. Testing all functions is best practice in terms of security criteria.

```
-----------------------------|----------|----------|----------|-
File                         |  % Stmts | % Branch |  % Funcs |
-----------------------------|----------|----------|----------|-
  GNS.sol                    |    88.59 |    82.69 |    91.67 |
 epochs/                     |    85.19 |       50 |    90.91 |
  EpochManager.sol           |    85.19 |       50 |    90.91 |
  Managed.sol                |    94.29 |     87.5 |       95 |
  Pausable.sol               |    86.67 |       75 |      100 |
 libraries/                  |    95.24 |       50 |      100 |
  HexStrings.sol             |    93.33 |       50 |      100 |
  Staking.sol                |    96.97 |    94.74 |    93.33 |
  LibFixedMath.sol           |    71.69 |    56.58 |    47.37 |
 statechannels/              |    90.48 |    81.25 |    91.67 |
  AllocationExchange.sol     |    92.59 |       85 |     87.5 |
  GRTWithdrawHelper.sol      |    86.67 |       75 |      100 |
 upgrades/                   |    98.25 |    65.38 |    96.97 |
  GraphProxy.sol             |      100 |    71.43 |      100 |
  GraphProxyStorage.sol      |    91.67 |        0 |    85.71 |
-----------------------------|----------|----------|----------|-
All files                    |    91.71 |    81.07 |    90.79 |
-----------------------------|----------|----------|----------|-
```

[N-02] Include `return parameters` in *NatSpec comments*

Context

IRewardsManager.sol
IStaking.sol
IGraphToken.sol
GraphTokenGateway.sol#L54
IGraphProxy.sol
IEpochManager.sol
IController.sol

Description

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation. In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.

https://docs.soliditylang.org/en/v0.8.15/natspec-format.html

If Return parameters are declared, you must prefix them with `/// @return`.

Some code analysis programs do analysis by reading NatSpec details, if they can't see the `@return` tag, they do incomplete analysis.

Recommendation

Include return parameters in NatSpec comments

Recommendation Code Style:

```
/// @notice information about what a function does
/// @param pageId The id of the page to get the URI for.
/// @return Returns a page's URI if it has been minted
function tokenURI(uint256 pageId) public view virtual overric
    if (pageId == 0 || pageId > currentId) revert("NOT_MINTEI

    return string.concat(BASE_URI, pageId.toString());
}
```

[N-03] `0 address` check

Context

[BridgeEscrow.sol#L21]
[L2GraphToken.sol#L81]
[GraphProxyAdmin.sol#L69]
[GraphProxyStorage.sol#L80]
[GraphProxy.sol#L116]
[GraphTokenUpgradeable.sol#L133]

Description

Also check of the address to protect the code from 0x0 address problem just in case. This is best practice or instead of suggesting that they verify address != 0x0, you could add some good NatSpec comments explaining what is valid and what is invalid and what are the implications of accidentally using an invalid address.

Recommendation

like this;

```
if (_treasury == address(0)) revert ADDRESS_ZERO();
```

[N-04] Use a more recent version of Solidity

Context

All contracts

Recommendation

Old version of Solidity is used `0.7.6` , newer version can be used `0.8.17`

[N-05] Require revert cause should be known

Context

[GraphProxy.sol#L133]
[GraphProxyAdmin.sol#L34]

GraphProxyAdmin.sol#L47
GraphProxyAdmin.sol#L59

Description

Vulnerability related to description is not written to require, it must be written so that the user knows why the process is reverted.

Recommendation

Current Code:

```
    require(sell.order.side == Side.Sell);
```

Recommendation Code:

```
    require(sell.order.side == Side.Sell, "Sell has invalid paramete
```

[N-06] Use `require` instead of `assert`

Context

GraphProxy.sol#L47-L54

Description

Assert should not be used except for tests, `require` should be used

Prior to Solidity 0.8.0, pressing a confirm consumes the remainder of the process's available gas instead of returning it, as request()/revert() did.

Assertion() should be avoided even after solidity version 0.8.0, because its documentation states "The Assert function generates an error of type Panic(uint256). Code that works properly should never Panic, even on invalid external input. If this happens, you need to fix it in your contract. there's a mistake".

[N-07] For modern and more readable code; update import usages

Context

ICuration.sol, IGraphCurationToken.sol, BridgeEscrow.sol#L5-L7, L1GraphTokenGateway.sol#L6-L10, Managed.sol#L5-L12, L2GraphTokenGateway.sol#L6-L13, GraphTokenUpgradeable.sol#L5-L10, L2GraphToken.sol#L5-L8, IStaking.sol#L6, IGraphToken.sol#L5, GraphProxy.sol#L5-L7, GraphProxyAdmin.sol#L5-L8, GraphUpgradeable.sol#L5

Description

Our Solidity code is also cleaner in another way that might not be noticeable: the struct Point. We were importing it previously with global import but not using it. The Point struct `polluted the source code` with an unnecessary object we were not using because we did not need it. This was breaking the rule of modularity and modular programming: `only import what you need` Specific imports with curly braces allow us to apply this rule better.

Recommendation

```
import {contract1 , contract2} from "filename.sol";
```

[N-08] `Function writing` that does not comply with the `Solidity Style Guide`

Context

GraphUpgradeable.sol, GraphProxyAdmin.sol, GraphProxy.sol, Managed.sol, L2GraphTokenGateway.sol, L1GraphTokenGateway.sol, GraphTokenGateway.sol

Description

Order of Functions; ordering helps readers identify which functions they can call and to find the constructor and fallback definitions easier. But there are contracts in the project that do not comply with this.

https://docs.soliditylang.org/en/v0.8.17/style-guide.html

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private
- within a grouping, place the view and pure functions last

[N-09] Implement some type of version counter that will be incremented automatically for contract upgrades

Context

GraphProxyAdmin.sol#L77
GraphProxy.sol#L115

Description

As part of the upgradeability of Proxies , the contract can be upgraded multiple times, where it is a systematic approach to record the version of each upgrade.

```
    function upgradeTo(address _newImplementation) external ifAdmir
        _setPendingImplementation(_newImplementation);
    }
```

I suggest implementing some kind of version counter that will be incremented automatically when you upgrade the contract.

Recommendation

```
uint256 public authorizeUpgradeCounter;
```

```
    function upgradeTo(address _newImplementation) external ifAdmir
            _setPendingImplementation(_newImplementation);
           authorizeUpgradeCounter+=1;

        }
```

[N-10] NatSpec is Missing

NatSpec is missing for the following function:

Managed.sol#L43
Managed.sol#L48
Managed.sol#L52
Managed.sol#L56

[N-11] Omissions in Events

Throughout the codebase, events are generally emitted when sensitive changes are made to the contracts. However, some events are missing important parameters

The events should include the new value and old value where possible:
Events with no old value;;
L1GraphTokenGateway.sol#L114,
L1GraphTokenGateway.sol#L124,
L1GraphTokenGateway.sol#L134,
L1GraphTokenGateway.sol#L134,
L1GraphTokenGateway.sol#L134,
L1GraphTokenGateway.sol#L134,
L1GraphTokenGateway.sol#L134,
L2GraphTokenGateway.sol#L100,
L2GraphTokenGateway.sol#L110,
L2GraphTokenGateway.sol#L120,
Managed.sol#L106

[N-12] Constant values such as a call to `keccak256()` , should used to immutable rather than constant

There is a difference between constant variables and immutable variables, and they should each be used in their appropriate contexts.

While it doesn't save any gas because the compiler knows that developers often make this mistake, it's still best to use the right tool for the task at hand.

Constants should be used for literal values written into the code, and immutable variables should be used for expressions, or values calculated in, or passed into the constructor.

There are 5 instances of this issue:

```
contracts/l2/token/GraphTokenUpgradeable.sol:

    bytes32 private constant DOMAIN_TYPE_HASH =
        keccak256(
            "EIP712Domain(string name,string version,uint256 cha
        );
    bytes32 private constant DOMAIN_NAME_HASH = keccak256("Graph
    bytes32 private constant DOMAIN_VERSION_HASH = keccak256("0'
    bytes32 private constant DOMAIN_SALT =
        0xe33842a7acd1d5a1d28f25a931703e5605152dc48d64dc4716efda
    bytes32 private constant PERMIT_TYPEHASH =
        keccak256(
            "Permit(address owner,address spender,uint256 value,
        );
```

[N-13] Floating pragma

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
https://swcregistry.io/docs/SWC-103

Floating Pragma List:
pragma ^0.7.6. (all contracts)

Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

[N-14] Inconsistent Solidity Versions

Description

Different Solidity compiler versions are used throughout Src repositories. The following contracts mix versions:

```
contracts/governance/IController.sol:
pragma solidity >=0.6.12 <0.8.0;

other contracts:
pragma solidity ^0.7.6;
```

Recommendation

Versions must be consistent with each other.

[N-15] For extended “using-for” usage, use the latest pragma version

Description

https://blog.soliditylang.org/2022/03/16/solidity-0.8.13-release-announcement/

Recommendation

Use solidity pragma version min. 0.8.13

[N-16] For functions, follow Solidity standard naming conventions

Context

L2GraphTokenGateway.sol#L286
L1GraphTokenGateway.sol#L290

Description

The above codes don’t follow Solidity’s standard naming convention,

internal and private functions : the mixedCase format starting with an underscore (_mixedCase starting with an underscore)

public and external functions : only mixedCase

constant variable : UPPER*CASE*WITH_UNDERSCORES (separated by uppercase and underscore)

[N-17] Add to `_blacklist function_`

Description

Cryptocurrency mixing service, Tornado Cash, has been blacklisted in the OFAC.
A lot of blockchain companies, token projects, NFT Projects have `blacklisted` all Ethereum addresses owned by Tornado Cash listed in the US Treasury Department’s sanction against the protocol.
https://home.treasury.gov/policy-issues/financial-sanctions/recent-actions/20220808
In addition, these platforms even ban accounts that have received ETH on their account with Tornadocash.

Some of these Projects;

- USDC (https://www.circle.com/en/usdc)
- Flashbots (https://www.paradigm.xyz/portfolio/flashbots)
- Aave (https://aave.com/)
- Uniswap
- Balancer
- Infura
- Alchemy

- Opensea
- dYdX

Details on the subject;
https://twitter.com/bantg/status/1556712790894706688?s=20&t=HUTDTeLikUr6Dv9JdMF7AA

https://cryptopotato.com/defi-protocol-aave-bans-justin-sun-after-he-randomly-received-0-1-eth-from-tornado-cash/

For this reason, every project in the Ethereum network must have a blacklist function, this is a good method to avoid legal problems in the future, apart from the current need.

Transactions from the project by an account funded by Tonadocash or banned by OFAC can lead to legal problems.Especially American citizens may want to get addresses to the blacklist legally, this is not an obligation

If you think that such legal prohibitions should be made directly by validators, this may not be possible:
https://www.paradigm.xyz/2022/09/base-layer-neutrality

```
 The ban on Tornado Cash makes little sense, because in the end, no
one can prevent people from using other mixer smart contracts, or
forking the existing ones. It neither hinders cybercrime, nor
privacy.
```

Here is the most beautiful and close to the project example; Manifold

Manifold Contract
https://etherscan.io/address/0xe4e4003afe3765aca8149a82fc064c0b125b9e5a#code

```
    modifier nonBlacklistRequired(address extension) {
        require(!_blacklistedExtensions.contains(extension), "E
        _;
    }
```

Recommended Mitigation Steps

Add to Blacklist function and modifier.

[L-01] Using Vulnerable Version of Openzeppelin

Context

[package.json]

Description

The package.json configuration file says that the project is using 3.4.1 — 3.4.2 of OpenZeppelin which has a vulnerability in initializers that call external contracts, There is 1 instance of this issue

Package	Affected versions	Patched versions
@openzeppelin/contracts (npm)	>=3.2.0 <4.4.1	4.4.1
@openzeppelin/contracts-upgradeable (npm)	>=3.2.0 <4.4.1	4.4.1

Recommendation

Use patched versions

Proof Of Concept

https://github.com/OpenZeppelin/openzeppelin-contracts/security/advisories/GHSA-9c22-pwxw-p6hx

[L-02] Avoid *shadowing* `inherited state variables`

Context

GraphProxy.sol#L140

Description

In `GraphProxy.sol#L140` , there is a local variable named `_pendingImplementation` , but there is a function named `_pendingImplementation()` in the inherited `GraphProxyStorage` with the same name. This use causes compilers to issue warnings, negatively affecting checking and code readability.

```
    function _acceptUpgrade() internal {
            address _pendingImplementation = _pendingImplementation(
            require(Address.isContract(_pendingImplementation), "Imp
            require(
                _pendingImplementation != address(0) && msg.sender =
                "Caller must be the pending implementation"
            );        // Codes...
```

Recommendation

Avoid using variables with the same name, including inherited in the same contract, if used, it must be specified in the NatSpec comments.

[L-03] NatSpec comments should be increased in contracts

Context

Governed.sol, Managed.sol, GraphTokenGateway.sol, IGraphCurationToken.sol, IGraphProxy.sol, IEpochManager.sol, IController.sol, IGraphToken.sol, IRewardsManager.sol, IStakingData.sol, ICuration.sol, IStaking.sol

Description

It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI). It is clearly stated in the Solidity official documentation.
In complex projects such as Defi, the interpretation of all functions and their arguments and returns is important for code readability and auditability.
https://docs.soliditylang.org/en/v0.8.15/natspec-format.html

Recommendation

NatSpec comments should be increased in Contracts

```
    Recommendation  Code:
```

~~~
```js
 /**
     * @notice Sets approval for specific withdrawer addresses
     * @dev This function updates the amount of the withdrawAppr
     * @param withdrawer_ Withdrawer address from state variable
     * @param token_ instance of ERC20
     * @param amount_ User amount
     * @param permissioned Control of admin for access
     */
    function setApprovalFor(
        address withdrawer_,
        ERC20 token_,
        uint256 amount_
    ) external permissioned {
        withdrawApproval[withdrawer_][token_] = amount_;
        emit ApprovedForWithdrawal(withdrawer_, token_, amount_)
    }
~~~

pcarranzav (The Graph) commented:

> Good QA report, I can see why it was selected for the audit report.

> [re: N-04] I'll leave my usual note on versions that I've left in other issues:

- 0.7.6 is used for compatibility with existing / deployed contracts and interfaces
- ^0.7.6 is safe (imo) because it would only pick up a newer patch for 0.7 if it ever existed
- some interfaces may use solidity version ranges to make them easier to include in third-party code
- OpenZeppelin version chosen for compatibility with solidity 0.7

pcarranzav (The Graph) commented re: N-06:

> Since it's been noted in several QA reports, I wanted to comment on the use of `assert()` as well. Newer OZ versions have removed it, but the code that uses `assert()` in our case is similar to: https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v3.4.2/contracts/proxy/TransparentUpgradeableProxy.sol#L34

> Its purpose is to validate at deployment time that the admin/implementation slot has been computed correctly (or, alternatively, that the EVM is computing hashes as expected 😅) and I believe is a legitimate use of the expression.

pcarranzav (The Graph) linked to a PR:

> C4 QA fixes for #699

0xean (judge) commented:

> [N-17] - should be informational only

> I do not think this type of opinion should be presented as fact by wardens.

> *For this reason, every project in the Ethereum network must have a blacklist function, this is a good method to avoid legal problems in the future, apart from the current need.*

Gas Optimizations

For this contest, 44 reports were submitted by wardens detailing gas optimizations. The report highlighted below by IllIllI received the top score from the judge.

The following wardens also submitted reports: Deivitto, Bnke0x0, RockingMiles, Rolezn, gogo, Saintcode_, emrekocak, mcwildy, sakman, KoKo, oyc_109, carlitox477, RedOneN, aysha, gianganhnguyen, ElKu, fatherOfBlocks, 0xdeadbeef, RaymondFam, martin, pedr02b2, erictee, __141345__, 0x1f8b, catchup, rbserver, B2, Tomio, TomJ, saian, Shinchan, ret2basic, 0xSmartContract, 0xNazgul, medikko, Jujic, delfin454000, c3phas, rotcivegaf, ajtra, Pheonix, zishansami, *and* gerdusx.

Gas Optimizations Summary

	Issue	Instances	Total Gas Saved
[G-01]	State variables can be packed into fewer storage slots	1	-

	Issue	Instances	Total Gas Saved
[G-02]	Avoid contract existence checks by using low level calls	3	300
[G-03]	State variables should be cached in stack variables rather than re-reading them from storage	10	970
[G-04]	`internal` functions only called once can be inlined to save gas	3	60
[G-05]	`require()` / `revert()` strings longer than 32 bytes cost extra gas	6	-
[G-06]	`keccak256()` should only need to be called on a specific string literal once	6	252
[G-07]	Optimize names to save gas	19	418
[G-08]	Using `bool` s for storage incurs overhead	4	68400
[G-09]	Use a more recent version of solidity	20	-
[G-10]	Use a more recent version of solidity	3	-
[G-11]	Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement	3	18
[G-12]	Splitting `require()` statements that use `&&` saves gas	3	9
[G-13]	Don’t compare boolean expressions to boolean literals	1	9
[G-14]	Stack variable used as a cheaper cache for a state variable is only used once	4	12
[G-15]	`require()` or `revert()` statements that check input arguments should be at the top of the function	1	-
[G-16]	Use custom errors rather than `revert()` / `require()` strings to save gas	60	-
[G-17]	Functions guaranteed to revert when called by normal users can be marked `payable`	32	672

Total: 179 instances over 17 issues with **7120 gas** saved

Gas totals use lower bounds of ranges and count two iterations of each `for` -loop. All values above are runtime, not deployment, values; deployment values are listed in the individual issue descriptions

[G‑01] State variables can be packed into fewer storage slots

If variables occupying the same slot are both written the same function or by the constructor, avoids a separate Gsset (**20000 gas**). Reads of the variables can also be cheaper

There is 1 instance of this issue:

```
File: contracts/governance/Pausable.sol

/// @audit Variable ordering with 3 slots instead of the current
///           uint256(32):lastPausePartialTime, uint256(32):last
8:        bool internal _partialPaused;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Pausable.sol#L8

[G‑02] Avoid contract existence checks by using low level calls

Prior to 0.8.10 the compiler inserted extra code, including `EXTCODESIZE` (**100 gas**), to check for contract existence for external function calls. In more recent solidity versions, the compiler will not insert these checks if the external call has a return value. Similar behavior can be achieved in earlier versions by using low-level calls, since low level calls never check for contract existence

There are 3 instances of this issue:

```
File: contracts/gateway/BridgeEscrow.sol

/// @audit approve()
29:            graphToken().approve(_spender, type(uint256).max);
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/BridgeEscrow.sol#L29

```
File: contracts/gateway/L1GraphTokenGateway.sol

/// @audit bridge()
77:             IBridge bridge = IInbox(inbox).bridge();

/// @audit l2ToL1Sender()
81:             address l2ToL1Sender = IOutbox(bridge.activeOutbo
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L77

[G-03] State variables should be cached in stack variables rather than re-reading them from storage

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (**100 gas**) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses.

There are 10 instances of this issue:

```
File: contracts/governance/Governed.sol

/// @audit governor on line 62
65:             emit NewOwnership(oldGovernor, governor);

/// @audit pendingGovernor on line 44
46:             emit NewPendingOwnership(oldPendingGovernor, pendi

/// @audit pendingGovernor on line 55
55:                 pendingGovernor != address(0) && msg.sender ==

/// @audit pendingGovernor on line 55
60:             address oldPendingGovernor = pendingGovernor;
```

```
/// @audit pendingGovernor on line 60
62:            governor = pendingGovernor;

/// @audit pendingGovernor on line 63
66:            emit NewPendingOwnership(oldPendingGovernor, pendi
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L65

```
File: contracts/governance/Pausable.sol

/// @audit _partialPaused on line 30
31:            if (_partialPaused) {

/// @audit _partialPaused on line 31
34:            emit PartialPauseChanged(_partialPaused);

/// @audit _paused on line 44
45:            if (_paused) {

/// @audit _paused on line 45
48:            emit PauseChanged(_paused);
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Pausable.sol#L31

[G-04] `internal` functions only called once can be inlined to save gas

Not inlining costs **20 to 40 gas** because of two extra `JUMP` instructions and additional stack operations needed for function calls.

There are 3 instances of this issue:

```
File: contracts/governance/Managed.sol
```

```
43        function _notPartialPaused() internal view {
44:           require(!controller.paused(), "Paused");

52        function _onlyGovernor() internal view {
53:           require(msg.sender == controller.getGovernor(), "C

56        function _onlyController() internal view {
57:           require(msg.sender == address(controller), "Caller
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L43-L44

[G-05] `require()` / `revert()` strings longer than 32 bytes cost extra gas

Each extra memory word of bytes past the original 32 incurs an MSTORE which costs **3 gas**

There are 6 instances of this issue:

```
File: contracts/gateway/GraphTokenGateway.sol

19            require(
20                msg.sender == controller.getGovernor() || msg.
21                "Only Governor or Guardian can call"
22:           );
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/GraphTokenGateway.sol#L19-L22

```
File: contracts/governance/Managed.sol

53:           require(msg.sender == controller.getGovernor(), "C
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/govern ance/Managed.sol#L53

```
File: contracts/upgrades/GraphProxy.sol

105:            require(_newAdmin != address(0), "Cannot change t

141:            require(Address.isContract(_pendingImplementation)

142             require(
143                 _pendingImplementation != address(0) && msg.se
144                 "Caller must be the pending implementation"
145:            );
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxy.sol#L105

```
File: contracts/upgrades/GraphUpgradeable.sol

32:             require(msg.sender == _implementation(), "Caller m
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphUpgradeable.sol#L32

[G-06] `keccak256()` should only need to be called on a specific string literal once

It should be saved to an immutable variable, and the variable used instead. If the hash is being used as a part of a function selector, the cast to `bytes4` should also only be done once

There are 6 instances of this issue:

```
File: contracts/governance/Managed.sol

114:            return ICuration(_resolveContract(keccak256("Curat
122:            return IEpochManager(_resolveContract(keccak256("E
130:            return IRewardsManager(_resolveContract(keccak256(
138:            return IStaking(_resolveContract(keccak256("Stakin
146:            return IGraphToken(_resolveContract(keccak256("Gra
154:            return ITokenGateway(_resolveContract(keccak256("G
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L114

[G-07] Optimize names to save gas

`public` / `external` function names and `public` member variable names can be optimized to save gas. See this link for an example of how it works. Below are the interfaces/abstract contracts that can be optimized so that the most frequently-called functions use the least amount of gas possible during method lookup. Method IDs that have two leading zero bytes can save **128 gas** each during deployment, and renaming functions to have lower method IDs will save **22 gas** per call, per sorted position shifted

There are 19 instances of this issue:

```
File: contracts/curation/ICuration.sol

/// @audit setDefaultReserveRatio(), setMinimumCurationDeposit()
7:    interface ICuration {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/curation/ICuration.sol#L7

```
File: contracts/curation/IGraphCurationToken.sol

/// @audit initialize(), burnFrom(), mint()
```

```
7:    interface IGraphCurationToken is IERC20Upgradeable {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/curation/IGraphCurationToken.sol#L7

```
File: contracts/epochs/IEpochManager.sol

/// @audit setEpochLength(), runEpoch(), isCurrentEpochRun(), bl
5:    interface IEpochManager {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/epochs/IEpochManager.sol#L5

```
File: contracts/gateway/BridgeEscrow.sol

/// @audit initialize(), approveAll(), revokeAll()
15:   contract BridgeEscrow is GraphUpgradeable, Managed {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/BridgeEscrow.sol#L15

```
File: contracts/gateway/GraphTokenGateway.sol

/// @audit setPauseGuardian(), setPaused(), paused()
14:   abstract contract GraphTokenGateway is GraphUpgradeable, I
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/GraphTokenGateway.sol#L14

```
File: contracts/gateway/ICallhookReceiver.sol

/// @audit onTokenTransfer()
```

```
11:   interface ICallhookReceiver {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/ICallhookReceiver.sol#L11

```
File: contracts/gateway/L1GraphTokenGateway.sol

/// @audit initialize(), setArbitrumAddresses(), setL2TokenAddre
21:   contract L1GraphTokenGateway is GraphTokenGateway, L1Arbit
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L21

```
File: contracts/governance/IController.sol

/// @audit getGovernor(), setContractProxy(), unsetContractProx
5:    interface IController {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/IController.sol#L5

```
File: contracts/governance/Managed.sol

/// @audit syncAllContracts()
23:   contract Managed {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L23

```
File: contracts/l2/gateway/L2GraphTokenGateway.sol

/// @audit initialize(), setL2Router(), setL1TokenAddress(), set
```

```
23:   contract L2GraphTokenGateway is GraphTokenGateway, L2Arbit
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/gateway/L2GraphTokenGateway.sol#L23

```
File: contracts/l2/token/GraphTokenUpgradeable.sol

/// @audit addMinter(), removeMinter(), renounceMinter(), mint()
28:   contract GraphTokenUpgradeable is GraphUpgradeable, Goverr
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/GraphTokenUpgradeable.sol#L28

```
File: contracts/l2/token/L2GraphToken.sol

/// @audit initialize(), setGateway(), setL1Address()
15:   contract L2GraphToken is GraphTokenUpgradeable, IArbToken
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/L2GraphToken.sol#L15

```
File: contracts/rewards/IRewardsManager.sol

/// @audit setIssuanceRate(), setMinimumSubgraphSignal(), setSuk
5:    interface IRewardsManager {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/rewards/IRewardsManager.sol#L5

```
File: contracts/staking/IStaking.sol

/// @audit setMinimumIndexerStake(), setThawingPeriod(), setCura
```

```
8:    interface IStaking is IStakingData {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/staking/IStaking.sol#L8

```
File: contracts/token/IGraphToken.sol

/// @audit burn(), burnFrom(), mint(), addMinter(), removeMinte
7:    interface IGraphToken is IERC20 {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/token/IGraphToken.sol#L7

```
File: contracts/upgrades/GraphProxyAdmin.sol

/// @audit getProxyImplementation(), getProxyPendingImplementati
17:   contract GraphProxyAdmin is Governed {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxyAdmin.sol#L17

```
File: contracts/upgrades/GraphProxy.sol

/// @audit implementation(), pendingImplementation(), setAdmin()
16:   contract GraphProxy is GraphProxyStorage {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxy.sol#L16

```
File: contracts/upgrades/GraphUpgradeable.sol

/// @audit acceptProxy(), acceptProxyAndCall()
```

```
11:    contract GraphUpgradeable {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphUpgradeable.sol#L11

```
File: contracts/upgrades/IGraphProxy.sol

/// @audit setAdmin(), implementation(), pendingImplementation()
5:    interface IGraphProxy {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/IGraphProxy.sol#L5

[G-08] Using `bool` s for storage incurs overhead

```
    // Booleans are more expensive than uint256 or any type that
    // word because each write operation emits an extra SLOAD tc
    // slot's contents, replace the bits taken up by the boolear
    // back. This is the compiler's defense against contract upç
    // pointer aliasing, and it cannot be disabled.
```

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/58f635312aa21f947cae5f8578638a85aa2519f5/contracts/security/ReentrancyGuard.sol#L23-L27 Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess ([100 gas]()) for the extra SLOAD, and to avoid Gsset (**20000 gas**) when changing from `false` to `true` , after having been `true` in the past

There are 4 instances of this issue:

```
File: contracts/gateway/L1GraphTokenGateway.sol

35:        mapping(address => bool) public callhookWhitelist;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L35

```
File: contracts/governance/Pausable.sol

8:        bool internal _partialPaused;

10:       bool internal _paused;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Pausable.sol#L8

```
File: contracts/l2/token/GraphTokenUpgradeable.sol

51:       mapping(address => bool) private _minters;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/GraphTokenUpgradeable.sol#L51

[G-09] Use a more recent version of solidity

Use a solidity version of at least 0.8.0 to get overflow protection without `SafeMath`
Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining
Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads
Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings
Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There are 20 instances of this issue:

```
File: contracts/curation/ICuration.sol
```

```
3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/curation/ICuration.sol#L3

```
File: contracts/curation/IGraphCurationToken.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/curation/IGraphCurationToken.sol#L3

```
File: contracts/epochs/IEpochManager.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/epochs/IEpochManager.sol#L3

```
File: contracts/gateway/BridgeEscrow.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/BridgeEscrow.sol#L3

```
File: contracts/gateway/GraphTokenGateway.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/GraphTokenGateway.sol#L3

```
File: contracts/gateway/ICallhookReceiver.sol

9:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/ICallhookReceiver.sol#L9

```
File: contracts/gateway/L1GraphTokenGateway.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L3

```
File: contracts/governance/Governed.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L3

```
File: contracts/governance/Managed.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L3

```
File: contracts/governance/Pausable.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Pausable.sol#L3

```
File: contracts/l2/gateway/L2GraphTokenGateway.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/gateway/L2GraphTokenGateway.sol#L3

```
File: contracts/l2/token/GraphTokenUpgradeable.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/GraphTokenUpgradeable.sol#L3

```
File: contracts/l2/token/L2GraphToken.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/L2GraphToken.sol#L3

```
File: contracts/rewards/IRewardsManager.sol
```

```
3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/rewards/IRewardsManager.sol#L3

```
File: contracts/token/IGraphToken.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/token/IGraphToken.sol#L3

```
File: contracts/upgrades/GraphProxyAdmin.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxyAdmin.sol#L3

```
File: contracts/upgrades/GraphProxy.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxy.sol#L3

```
File: contracts/upgrades/GraphProxyStorage.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxyStorage.sol#L3

```
File: contracts/upgrades/GraphUpgradeable.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphUpgradeable.sol#L3

```
File: contracts/upgrades/IGraphProxy.sol

3:    pragma solidity ^0.7.6;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/IGraphProxy.sol#L3

[G-10] Use a more recent version of solidity

Use a solidity version of at least 0.8.2 to get simple compiler automatic inlining
Use a solidity version of at least 0.8.3 to get better struct packing and cheaper multiple storage reads
Use a solidity version of at least 0.8.4 to get custom errors, which are cheaper at deployment than `revert()/require()` strings
Use a solidity version of at least 0.8.10 to have external calls skip contract existence checks if the external call has a return value

There are 3 instances of this issue:

```
File: contracts/governance/IController.sol

3:    pragma solidity >=0.6.12 <0.8.0;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/IController.sol#L3

```
File: contracts/staking/IStakingData.sol

3:    pragma solidity >=0.6.12 <0.8.0;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/staking/IStakingData.sol#L3

```
File: contracts/staking/IStaking.sol

3:    pragma solidity >=0.6.12 <0.8.0;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/staking/IStaking.sol#L3

[G-11] Using `> 0` costs more gas than `!= 0` when used on a `uint` in a `require()` statement

This change saves 6 gas per instance. The optimization works until solidity version 0.8.13 where there is a regression in gas costs.

There are 3 instances of this issue:

```
File: contracts/gateway/L1GraphTokenGateway.sol

201:            require(_amount > 0, "INVALID_ZERO_AMOUNT");

217:                    require(maxSubmissionCost > 0, "NO_SUBMISS
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gatewa y/L1GraphTokenGateway.sol#L201

```
File: contracts/l2/gateway/L2GraphTokenGateway.sol

146:            require(_amount > 0, "INVALID_ZERO_AMOUNT");
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/gateway/L2GraphTokenGateway.sol#L146

[G-12] Splitting `require()` statements that use `&&` saves gas

See this issue which describes the fact that there is a larger deployment gas cost, but with enough runtime calls, the change ends up being cheaper by **3 gas**

There are 3 instances of this issue:

```
File: contracts/gateway/L1GraphTokenGateway.sol

142:            require(_escrow != address(0) && Address.isContrac
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L142

```
File: contracts/governance/Governed.sol

54              require(
55                  pendingGovernor != address(0) && msg.sender ==
56                  "Caller must be pending governor"
57:             );
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L54-L57

```
File: contracts/upgrades/GraphProxy.sol

142            require(
143                _pendingImplementation != address(0) && msg.se
144                "Caller must be the pending implementation"
145:           );
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxy.sol#L142-L145

[G-13] Don't compare boolean expressions to boolean literals

`if (<x> == true)` => `if (<x>)`, `if (<x> == false)` => `if (!<x>)`

There is 1 instance of this issue:

```
File: contracts/gateway/L1GraphTokenGateway.sol

214:                     extraData.length == 0 || callhookWhite
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L214

[G-14] Stack variable used as a cheaper cache for a state variable is only used once

If the variable is only accessed once, it's cheaper to use the state variable directly that one time, and save the **3 gas** the extra stack assignment would spend

There are 4 instances of this issue:

```
File: contracts/governance/Governed.sol

43:            address oldPendingGovernor = pendingGovernor;
```

```
59:            address oldGovernor = governor;

60:            address oldPendingGovernor = pendingGovernor;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L43

```
File: contracts/governance/Pausable.sol

56:            address oldPauseGuardian = pauseGuardian;
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Pausable.sol#L56

[G-15] `require()` or `revert()` statements that check input arguments should be at the top of the function

Checks that involve constants should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before wasting a Gcoldsload (**2100 gas***) in a function that may ultimately revert in the unhappy case.

There is 1 instance of this issue:

```
File: contracts/gateway/L1GraphTokenGateway.sol

/// @audit expensive op on line 199
201:            require(_amount > 0, "INVALID_ZERO_AMOUNT");
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L201

[G-16] Use custom errors rather than `revert()` / `require()` strings to save gas

Custom errors are available from solidity version 0.8.4. Custom errors save ~50 gas each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas

There are 60 instances of this issue:

```
File: contracts/gateway/GraphTokenGateway.sol

19              require(
20                  msg.sender == controller.getGovernor() || msg.
21                  "Only Governor or Guardian can call"
22:             );

31:             require(_newPauseGuardian != address(0), "PauseGua

40:             require(!_paused, "Paused (contract)");
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/GraphTokenGateway.sol#L19-L22

```
File: contracts/gateway/L1GraphTokenGateway.sol

74:             require(inbox != address(0), "INBOX_NOT_SET");

78:             require(msg.sender == address(bridge), "NOT_FROM_B

82:             require(l2ToL1Sender == l2Counterpart, "ONLY_COUNT

110:            require(_inbox != address(0), "INVALID_INBOX");

111:            require(_l1Router != address(0), "INVALID_L1_ROUTE

122:            require(_l2GRT != address(0), "INVALID_L2_GRT");

132:            require(_l2Counterpart != address(0), "INVALID_L2_

142:            require(_escrow != address(0) && Address.isContrac
153:            require(_newWhitelisted != address(0), "INVALID_AI

154:            require(!callhookWhitelist[_newWhitelisted], "ALRE

165:            require(_notWhitelisted != address(0), "INVALID_AI

166:            require(callhookWhitelist[_notWhitelisted], "NOT_W

200:            require(_l1Token == address(token), "TOKEN_NOT_GRT

201:            require(_amount > 0, "INVALID_ZERO_AMOUNT");

202:            require(_to != address(0), "INVALID_DESTINATION");

213                     require(
214                         extraData.length == 0 || callhookWhite
215                         "CALL_HOOK_DATA_NOT_ALLOWED"
216:                    );

217:                    require(maxSubmissionCost > 0, "NO_SUBMISS

224:                        require(msg.value >= expectedEth, "WRO

271:            require(_l1Token == address(token), "TOKEN_NOT_GRT

275:            require(_amount <= escrowBalance, "BRIDGE_OUT_OF_F
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L74

```
File: contracts/governance/Governed.sol

24:             require(msg.sender == governor, "Only Governor car

41:             require(_newGovernor != address(0), "Governor must

54              require(
55                  pendingGovernor != address(0) && msg.sender ==
56                  "Caller must be pending governor"
57:             );
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L24

```
File: contracts/governance/Managed.sol

44:         require(!controller.paused(), "Paused");

45:         require(!controller.partialPaused(), "Partial-paus

49:         require(!controller.paused(), "Paused");

53:         require(msg.sender == controller.getGovernor(), "C

57:         require(msg.sender == address(controller), "Caller

104:        require(_controller != address(0), "Controller mus
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L44

```
File: contracts/l2/gateway/L2GraphTokenGateway.sol

69          require(
70              msg.sender == AddressAliasHelper.applyL1ToL2Al
71              "ONLY_COUNTERPART_GATEWAY"
72:         );

98:         require(_l2Router != address(0), "INVALID_L2_ROUTE

108:        require(_l1GRT != address(0), "INVALID_L1_GRT");

118:        require(_l1Counterpart != address(0), "INVALID_L1_

145:        require(_l1Token == l1GRT, "TOKEN_NOT_GRT");

146:        require(_amount > 0, "INVALID_ZERO_AMOUNT");

147:        require(msg.value == 0, "INVALID_NONZERO_VALUE");

148:        require(_to != address(0), "INVALID_DESTINATION");
```

```
153:            require(outboundCalldata.extraData.length == 0, "

233:            require(_l1Token == l1GRT, "TOKEN_NOT_GRT");

234:            require(msg.value == 0, "INVALID_NONZERO_VALUE");
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/gateway/L2GraphTokenGateway.sol#L69-L72

```
File: contracts/l2/token/GraphTokenUpgradeable.sol

60:             require(isMinter(msg.sender), "Only minter can ca

94:             require(_owner == recoveredAddress, "GRT: invalid

95:             require(_deadline == 0 || block.timestamp <= _dea

106:            require(_account != address(0), "INVALID_MINTER");

115:            require(_minters[_account], "NOT_A_MINTER");

123:            require(_minters[msg.sender], "NOT_A_MINTER");
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/GraphTokenUpgradeable.sol#L60

```
File: contracts/l2/token/L2GraphToken.sol

36:             require(msg.sender == gateway, "NOT_GATEWAY");

49:             require(_owner != address(0), "Owner must be set")

60:             require(_gw != address(0), "INVALID_GATEWAY");

70:             require(_addr != address(0), "INVALID_L1_ADDRESS")
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/L2GraphToken.sol#L36

```
File: contracts/upgrades/GraphProxy.sol

105:            require(_newAdmin != address(0), "Cannot change th

141:            require(Address.isContract(_pendingImplementation)

142             require(
143                 _pendingImplementation != address(0) && msg.se
144                 "Caller must be the pending implementation"
145:            );

157:            require(msg.sender != _admin(), "Cannot fallback t
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxy.sol#L105

```
File: contracts/upgrades/GraphProxyStorage.sol

62:             require(msg.sender == _admin(), "Caller must be ad
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxyStorage.sol#L62

```
File: contracts/upgrades/GraphUpgradeable.sol

24:             require(msg.sender == _proxy.admin(), "Caller must

32:             require(msg.sender == _implementation(), "Caller m
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphUpgradeable.sol#L24

[G-17] Functions guaranteed to revert when called by normal users can be marked `payable`

If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function. Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided. The extra opcodes avoided are `CALLVALUE` (2), `DUP1` (3), `ISZERO` (3), `PUSH2` (3), `JUMPI` (10), `PUSH1` (3), `DUP1` (3), `REVERT` (0), `JUMPDEST` (1), `POP` (2), which costs an average of about **21 gas per call** to the function, in addition to the extra deployment cost

There are 32 instances of this issue:

```
File: contracts/gateway/BridgeEscrow.sol

20:        function initialize(address _controller) external only

28:        function approveAll(address _spender) external onlyGov

36:        function revokeAll(address _spender) external onlyGove
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/BridgeEscrow.sol#L20

```
File: contracts/gateway/GraphTokenGateway.sol

30:        function setPauseGuardian(address _newPauseGuardian) e

47:        function setPaused(bool _newPaused) external onlyGover
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/GraphTokenGateway.sol#L30

```
File: contracts/gateway/L1GraphTokenGateway.sol
```

```
99:       function initialize(address _controller) external onl
109:      function setArbitrumAddresses(address _inbox, address
121:      function setL2TokenAddress(address _l2GRT) external o
131:      function setL2CounterpartAddress(address _l2Counterpa
141:      function setEscrowAddress(address _escrow) external o
152:      function addToCallhookWhitelist(address _newWhitelist
164:      function removeFromCallhookWhitelist(address _notWhit
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/gateway/L1GraphTokenGateway.sol#L99

```
File: contracts/governance/Governed.sol

40:       function transferOwnership(address _newGovernor) exte
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Governed.sol#L40

```
File: contracts/governance/Managed.sol

95:       function setController(address _controller) external
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/governance/Managed.sol#L95

```
File: contracts/l2/gateway/L2GraphTokenGateway.sol

87:       function initialize(address _controller) external onl
```

```
97:        function setL2Router(address _l2Router) external onlyG

107:       function setL1TokenAddress(address _l1GRT) external or

117:       function setL1CounterpartAddress(address _l1Counterpar
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/gateway/L2GraphTokenGateway.sol#L87

```
File: contracts/l2/token/GraphTokenUpgradeable.sol

105:       function addMinter(address _account) external onlyGove

114:       function removeMinter(address _account) external onlyG

132:       function mint(address _to, uint256 _amount) external c
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/GraphTokenUpgradeable.sol#L105

```
File: contracts/l2/token/L2GraphToken.sol

48:        function initialize(address _owner) external onlyImpl

59:        function setGateway(address _gw) external onlyGovernor

69:        function setL1Address(address _addr) external onlyGove

80:        function bridgeMint(address _account, uint256 _amount)

90:        function bridgeBurn(address _account, uint256 _amount)
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/l2/token/L2GraphToken.sol#L48

```
File: contracts/upgrades/GraphProxyAdmin.sol

68:       function changeProxyAdmin(IGraphProxy _proxy, address

77:       function upgrade(IGraphProxy _proxy, address _implemer

86:       function acceptProxy(GraphUpgradeable _implementation,

96        function acceptProxyAndCall(
97            GraphUpgradeable _implementation,
98            IGraphProxy _proxy,
99            bytes calldata _data
100:      ) external onlyGovernor {
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphProxyAdmin.sol#L68

```
File: contracts/upgrades/GraphUpgradeable.sol

50:       function acceptProxy(IGraphProxy _proxy) external onl

59        function acceptProxyAndCall(IGraphProxy _proxy, bytes
60            external
61:           onlyProxyAdmin(_proxy)
```

https://github.com/code-423n4/2022-10-thegraph/blob/48e7c7cf641847e07ba07e01176cb17ba8ad6432/contracts/upgrades/GraphUpgradeable.sol#L50

tmigone (The Graph) commented:

> We've found this submission to be of high quality.

Mitigation Review

Mitigation review by Trust

Project repository: https://github.com/graphprotocol/contracts

Review commit: caf5ab5723353b9062878ade247a060a2b55514f

Mitigation overview

The C4 Graph contest identified a total of 4 Medium issues and several QA and gas optimization suggestions. The following is the list of issues addressed:

- Contract unupgradeability issue - Fixed in a standalone PR as it was initially classified as medium severity but ultimately marked as QA issue. Fixed by inserting a `__gap` state variable in GraphTokenGateway and GraphTokenUpgradeable. Therefore, those 2 contracts are now upgradeable.
- Insufficient validation of l2Counterpart in L1GraphTokenGateway issue - Fixed in a standalone PR as it was also initially classified as Medium severity but ultimately marked as QA issue. Fixed by requiring L2Counterpart to not be zero in every instance of L2Counterpart validation.
- Contract re-initialization issue - Fixed in standalone PR. Fixed by using OpenZeppelin's initializer guard in contracts BridgeEscrow, L1GraphTokenGateway, L2GraphTokenGateway and L2GraphToken.
- Arbitrum address 0.8.0 overflow issue - Fixed in future 0.8.0 upgrade branch PR. Fixed by wrapping the potential overflowing and underflowing operations in an unchecked block.
- Non-empty _data leading to callbacks issue, was identified by The Graph team and fixed in PR. Fixed by not serializing the data with unused bytes.
- Multiple QA issues, including #198/202: " `L1GraphTokenGateway` should not allow `l1Router` as `callhookWhitelist` " - Fixed in PR
- Multiple gas optimization issues - Fixed in PR

The following is a list of issues acknowledged by The Graph:

- Governance centralization issue - Governor privileged address can approve any address to spend BridgeEscrow locked funds.
- Leak of gas issue - Contract calls to outboundTransfer() may lose the gas refunds sent on L2.

The fixes applied by The Graph have been stacked on top of each other in the git commit tree. As such, the Mitigation Review was done on the last PR in the fix tree, which covers the result of the aggregation of the fixes.

Findings

Findings are labeled with the following notation `M.S-N`, representing `Mitigation.Severity-Number`.

No Medium or High severity issues have been uncovered in the Mitigation Review. This can be attributed to the combination of high coding and code review standards of the team, and to the fact that little to no logic has been introduced to any of the contracts.

M.L-01 Several contracts remain non-upgradeabe

The PR addressed upgradeability of two contracts - GraphTokenGateway and GraphTokenUpgradeable.

However, the following inherited contracts have not been addressed:

1. Managed.sol
2. Pausable.sol
3. Governed.sol

Managed.sol has a 10 slot gap, which is non-standard and may not be sufficient for desired future functionality. The other two contracts have no slot gap. Without introducing `__gap` slots in the above contracts, they will not be effectively upgradeable once deployed.

Contracts that derive from the above contracts are (among others):

1. GraphTokenGateway
2. BridgeEscrow
3. L1GraphTokenGateway
4. L2GraphTokenGateway

It is important to note that derived contracts *can* be upgraded, but state variables in the base contracts cannot be introduced.

The Graph has acknowledged the finding and reasoned that the above contracts are already used by existing Graph facilities. Additionally, those contracts have specific functionality and it is reasonable that they won't need to be upgraded.

M.L-02 Lack of safety checks in `_checksBeforeUnpause`

The PR contains some refactoring done after an additional audit by Open Zeppelin. Now, every derived GraphTokenGateway implements a `_checksBeforeUnpause` function. Below is how it is defined for L1GraphTokenGateway.sol:

```
    function _checksBeforeUnpause() internal view override {
        require(inbox != address(0), "INBOX_NOT_SET");
        require(l1Router != address(0), "ROUTER_NOT_SET");
        require(l2Counterpart != address(0), "L2_COUNTERPART_NOT_SET
        require(escrow != address(0), "ESCROW_NOT_SET");
    }
```

Note that there is no check for the important l2GRT state variable. It can be set at any time using the setL2TokenAddress function:

```
    function setL2TokenAddress(address _l2GRT) external onlyGovernor
        require(_l2GRT != address(0), "INVALID_L2_GRT");
        l2GRT = _l2GRT;
        emit L2TokenAddressSet(_l2GRT);
    }
```

We advise adding the following check: `require(l2GRT != address(0), "L2GRT_NOT_SET");`

M.G-01 Check sequence can be improved

Outbound transfer checks previously looked like this:

```
    IGraphToken token = graphToken();
    require(_l1Token == address(token), "TOKEN_NOT_GRT");
    require(_amount != 0, "INVALID_ZERO_AMOUNT");
    require(_to != address(0), "INVALID_DESTINATION");
```

The Graph implemented a gas optimization suggestion and re-ordered the checks to the code below:

```
    IGraphToken token = graphToken();
    require(_amount != 0, "INVALID_ZERO_AMOUNT");
    require(_l1Token == address(token), "TOKEN_NOT_GRT");
    require(_to != address(0), "INVALID_DESTINATION");
```

However, it is recommended that the expensive "token" variable lookup would be delayed until deemed necessary, after the `\_amount` and `\_to` checks:

```
    require(_to != address(0), "INVALID_DESTINATION");
    require(_amount != 0, "INVALID_ZERO_AMOUNT");
    IGraphToken token = graphToken();
    require(_l1Token == address(token), "TOKEN_NOT_GRT");
```

M.G-02 Split compound require statements

The Graph implemented gas optimizations for `require()` statements which use the && operator. For example

```
        require(_escrow != address(0) && Address.isContract(_escrow)
```

was changed to:

```
        require(_escrow != address(0), "INVALID_ESCROW");
        require(Address.isContract(_escrow), "MUST_BE_CONTRACT");
```

We would advise the team to fix an additional compound require statement in Governed.sol's `acceptOwnership()`:

```
        require(
                oldPendingGovernor != address(0) && msg.sender == ol
                "Caller must be pending governor"
        );
```

Change to:

```
        require(oldPendingGovernor != address(0), "INVALID_PENDING_(
        require(msg.sender == oldPendingGovernor, "INVALID_SENDER")
```

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top