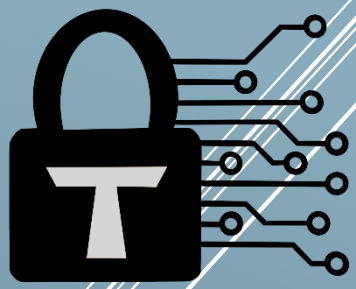
Trust Security



Smart Contract Audit

The Graph – PR #1279

15/02/26

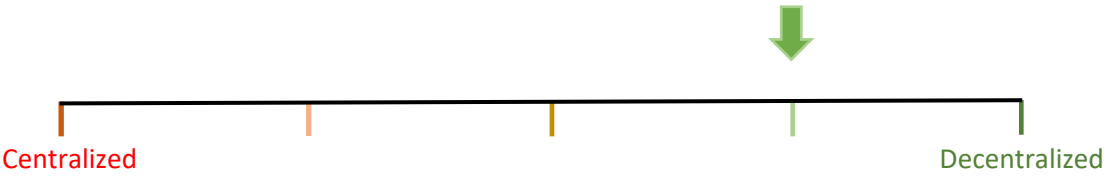
Executive summary

Category	Indexing
Auditor	Trust
Time period	02/02/26 – 15/02/26

Findings

Severity	Total	Open	Fixed	Acknowledged
High	2	-	2	-
Medium	0	-	-	-
Low	1	-	1	-

Centralization score



Signature

Digitally signed by Trust
Date: 2026.02.17 12:32:00 +02'00'

Document properties

Versioning

Version	Date	Description
0.1	15/02/26	Client report
0,2	17/02/26	Mitigation review

Contact

Trust

trust@trust-security.xyz

Introduction

Trust Security has conducted an audit at the customer's request. The audit is focused on uncovering security issues and additional bugs contained in the code defined in scope. Some additional recommendations have also been given when appropriate.

Scope

All non-test files changed between the trusted and target hashes below.

Repository details

- **Repository URL:** https://github.com/graphprotocol/contracts
- **Trusted commit hash:** b838a8e00f19f2c107da7ee6e98b05723282a7d7
- **Target commit hash:** 16dbd7373a57e5964ebd6c6643a7d95bd688857d
- **Mitigation commit hash:** fdec3bdad083087c657fa3d5cc3e224e6405ba23

Note that some issues have been identified and reported prior to the final target commit hash above.

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Since its inception it has safeguarded over 30 clients through private services and over 30 additional projects through bug bounty submissions.

About the Auditors

Trust has established a dominating presence in the smart contract security ecosystem since 2022. He is a resident on the Immunefi, Sherlock and C4 leaderboards and is now focused in auditing and managing audit teams under Trust Security. When taking time off auditing & bug hunting, he enjoys sharing knowledge and experience with aspiring auditors through X or the Trust Security blog.

Disclaimer

Smart contracts are an experimental technology with many known and unknown risks. Trust Security assumes no responsibility for any misbehavior, bugs or exploits affecting the audited code or any part of the deployment phase.

Furthermore, it is known to all parties that changes to the audited code, including fixes of issues highlighted in this report, may introduce new issues and require further auditing.

Methodology

In general, the primary methodology used is manual auditing. The entire in-scope code has been deeply looked at and considered from different adversarial perspectives. Any additional dependencies on external code have also been reviewed.

Qualitative analysis

Metric	Rating	Comments
Code complexity	Moderate	Rewarding logic is moderately complex across different accrual layers and qualification logic handling.
Documentation	Excellent	Project is very well documented.
Best practices	Excellent	Project consistently adheres to industry standards.
Centralization risks	Good	Project does not introduce significant unnecessary centralization risks.

Findings

High severity findings

TRST-H-1 Rewards are lost due to desynchronization between update callback functions

- **Category:** Accounting issues
- **Source:** RewardsManager.sol
- **Status:** Fixed

Description

In the RewardsManager, calling *onSubgraphSignalUpdate()* advances the signal snapshot (**accRewardsPerSignalSnapshot**) and accumulates rewards into **accRewardsForSubgraph**, but does not update **accRewardsForSubgraphSnapshot** or distribute rewards to **accRewardsPerAllocatedToken**. When *onSubgraphAllocationUpdate()* later runs, it calculates new rewards solely from the signal delta — which was already zeroed — and skips the undistributed rewards sitting in the delta between **accRewardsForSubgraphSnapshot** and **accRewardsForSubgraph**. Those rewards are permanently bricked.

Recommended mitigation

Streamline the accrual logic between the different *update()* functions.

Team response

Fixed.

Mitigation review

The different update functions have largely been merged. Any prior **accRewardsForSubgraph** delta is respected, and new deltas are not introduced through correct rewarding logic.

TRST-H-2 Mishandling of non-claimable rewards in onSubgraphSignalUpdate() causes loss of rewards

- **Category:** Accounting issues
- **Source:** RewardsManager.sol
- **Status:** Fixed

Description

In *onSubgraphSignalUpdate()*, it uses the view function *getAccRewardsForSubgraph()* to calculate the new accumulated rewards. This view function conditionally excludes rewards for non-claimable subgraphs (returning **accRewardsForSubgraph + 0** instead of **accRewardsForSubgraph + newRewards**). The signal snapshot is then advanced, permanently zeroing the delta. Unlike *onSubgraphAllocationUpdate()*, which calls *_reclaimRewards()* for

non-claimable conditions, *onSubgraphSignalUpdate()* has no reclaim path — the rewards simply vanish.

The issue is currently gated behind the unrelated TRST-H-1 issue, however as it describes a second, independent root cause which causes permanent loss of funds, it is provided separately.

Recommended mitigation

Ensure that rewards are reclaimed in any flow where they are not claimed – avoid unaccounted consumption of rewards.

Team response

Fixed.

Mitigation review

The issue has been addressed as part of the refactor of *onSubgraphSignalUpdate()*, ensuring rewards are reclaimed correctly.

Low severity findings

TRST-L-1 Changes to the minimum eligible signal could affect retroactive dispatch of rewards

- **Category:** Logical flaws
- **Source:** RewardsManager.sol
- **Status:** Fixed

Description

In the RewardsManager, admin changes to **minimumSubgraphSignal** could make previously ineligible subgraphs eligible or vice-versa. This behavior is similar to changes through *setDenied()* however differently to denial status, minimum signal changes do not ensure affected subgraphs are updated. Consequently, the change could apply retroactively for the non-updated period, possibly leading to subgraphs receiving rewards for a period they were not eligible, or loss of rewards for an eligible period.

It is prohibitive to enforce on-chain updating of all subgraphs in the setter function since there is no reasonable limit to number of active subgraphs. Therefore prevention should focus on preemptive updates for subgraphs with impactful changes.

Recommended mitigation

Add documentation and clear process flows for minimum signal changes. When minimum signal goes up, it is up to subgraph admins to update the subgraph state prior to scheduled changes. Conversely, when minimum signal goes down, Governance should ensure major ineligible subgraphs are updated, to avoid them retroactively diluting rewards.

Team response

Fixed.

Mitigation review

Clear documentation has been introduced, through following the described process no reward mishandling would occur.

Additional recommendations

TRST-R-1 Make _resizeAllocation() logic more defensive

In AllocationManager, users can resize their allocations, in which case the following logic unconditionally accrues any accrued per-token rewards:

```
// Update the allocation
_allocations[_allocationId].tokens = _tokens;
_allocations[_allocationId].accRewardsPerAllocatedToken = accRewardsPerAllocatedToken;
_allocations[_allocationId].accRewardsPending += _graphRewardsManager().calcRewards(
    oldTokens,
    accRewardsPerAllocatedTokenPending
);
```

The logic does not incorporate the various condition checks imposed at *_presentPoi()* and RewardManager's *takeRewards()*. There is no impact under the assumption that any accrued pending rewards would not be materializable by the user, since they would have to go through the gated logic later. However it is still not advised to allow users to accrue pending when their allocation should not qualify for rewards, to secure the contract against unknown vectors and any future changes in rewarding logic. At least when the allocation is stale or not eligible, consider reclaiming the rewards at this stage.

TRST-R-2 Improve accuracy of documentation

The following snippet describes behavior of which rewards are accounted in *getRewards()*.

```
/**
 * @inheritdoc IRewardsManager
 * @dev Returns claimable rewards based on current accumulator state.
 * Reflects deterministic exclusions (denied, below minimum signal, no
allocations) but NOT indexer eligibility.
 * Indexer eligibility is checked at claim time and can change independently
of reward accrual.
 */
```

However, in case of exclusion due to minimum signal, the calculation below in *getAccRewardsPerAllocatedToken()* would include the delta between the snapshot and **accRewardsForSubgraph** which will not be distributed.

```
// getAccRewardsForSubgraph already handles claimability: excludes new
rewards when not claimable
uint256 accRewardsForSubgraph =
getAccRewardsForSubgraph(_subgraphDeploymentID);
uint256 newRewardsForSubgraph = MathUtils.diffOrZero(
    accRewardsForSubgraph,
    subgraph.accRewardsForSubgraphSnapshot
);
```

Consider synchronizing the docs to line up with observed behavior.

Secondly, observe the documentation of *reclaimRewards()*:

```
/**
 * @inheritdoc IRewardsManager
 * @dev bytes32(0) as a reason is reserved as a no-op and will not be
reclaimed.
 */
function reclaimRewards(bytes32 reason, address allocationID) external
override returns (uint256) {
```

However, when a default reclaim address is set, a bytes32(0) reason **will** lead to reclaims.

TRST-R-3 Remove redundant check

In *_updateSubgraphRewards()*, the following check handles reward distribution:

```
if (undistributedRewards != 0 && subgraphAllocatedTokens != 0) {
    newAccRewardsPerAllocatedToken = accRewardsPerAllocatedToken.add(

undistributedRewards.mul(FIXED_POINT_SCALING_FACTOR).div(subgraphAllocatedTo
kens)
    );
    subgraph.accRewardsPerAllocatedToken = newAccRewardsPerAllocatedToken;
}
```

It should be observed that if **undistributedRewards != 0**, then it is guaranteed that **subgraphAllocatedTokens != 0**, because **RewardsCondition** has to be **NONE** and is not **NO_ALLOCATION**. Moreover, in the invalid case when it **is** zero, the undistributed rewards should be reclaimed.