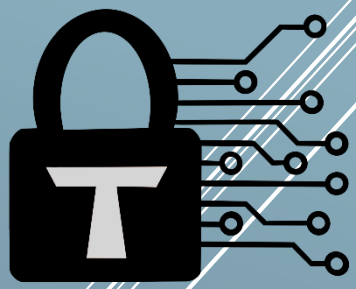
Trust Security

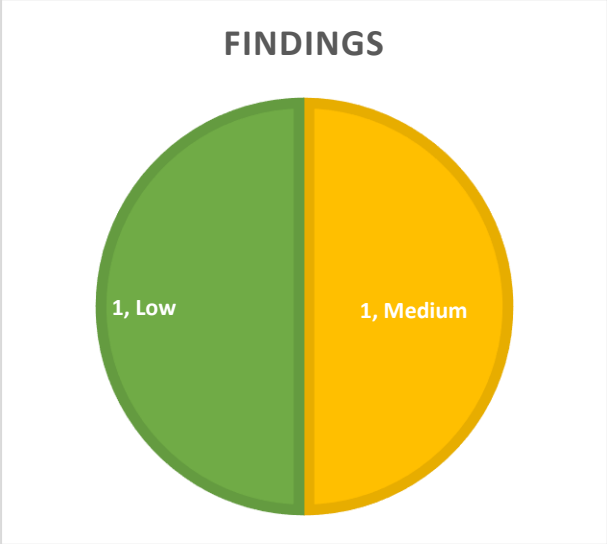


Smart Contract Audit

The Graph Operator Decentralization

16/02/23

Executive summary

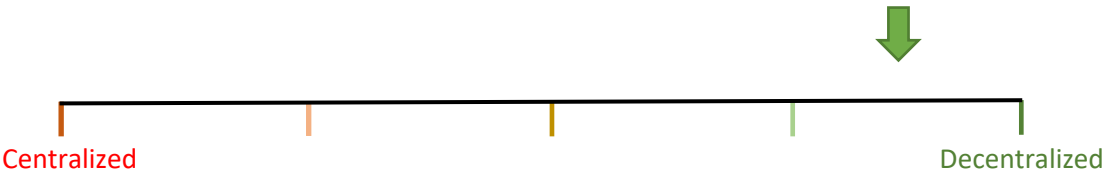


Category	Staking
Audited file count	2
LOC changed	1
LOC added	0
LOC removed	6
Auditor	Trust
Time period	13/02-16/02

Findings

Severity	Total	Fixed	Acknowledged	Disputed
High	0	-	-	-
Medium	1	-	1	-
Low	1	-	1	-

Centralization score



Signature

Digitally signed by Trust
Date: 2023-02-22 09:37:25

EXECUTIVE SUMMARY 1

DOCUMENT PROPERTIES 3

Versioning **3**

Contact **3**

INTRODUCTION 4

Scope **4**

Repository details **4**

About Trust Security **4**

Disclaimer **4**

Methodology **4**

QUALITATIVE ANALYSIS 5

SUMMARY OF CHANGES 5

FINDINGS 6

Medium severity findings **6**
TRST-M-1 Operator can spoof query fees to make net profit from pool 6

Low severity findings **7**
TRST-L-1 collected amount might be entirely burnt as protocol fees unintentionally 7

Additional recommendations **9**
Improve documentation 9
Redundant event emission 9

Centralization risks **10**
Operator – Indexer trust assumptions 10

Document properties

Versioning

Version	Date	Description
0.1	16/02/23	Client report
0.2	21/02/23	Response + Mitigation review
0.3	22/02/23	Specify commit hashes

Contact

Or Cyngiser, AKA Trust

boss@trustindistrust.com

Introduction

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

Scope

- contracts/staking/Staking.sol
- contracts/staking/StakingStorage.sol

Repository details

- **Repository URL:** https://github.com/graphprotocol/contracts
- **Audit Pull request:** https://github.com/graphprotocol/contracts/pull/749
- **Audit commit hash:** 36308c31fe49cd9fde1da5aad639ee0efd0b4d23
- **Fix pull request:** https://github.com/graphprotocol/contracts/pull/791
- **Fix commit hash:** 225be7849134073a12f4600e0253d22ed25f0fe6

About Trust Security

Trust Security has been established by top-end blockchain security researcher Trust, in order to provide high quality auditing services. Trust is the leading auditor at competitive auditing service Code4rena, reported several critical issues to Immunefi bug bounty platform and is currently a Code4rena judge.

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 pull request 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	**Excellent**	Project kept code as simple as possible, reducing attack risks
Documentation	**Good**	Project source code is documented well, but web docs mostly do not cover contract code.
Best practices	**Excellent**	Project consistently adheres to industry standards.
Centralization risks	**Good**	Some degree of trust is still needed between indexer and operator, but the protocol has stepped in the right direction.

Summary of changes

The only changes in the covered pull request are:

1. Removal of **assetHolder** setter and relevant event
2. Removal of **assetHolder** requirement on caller of *collect*()
3. Changing of **assetHolders** mapping to private

Changes (1) and (3) are self-contained and easy to assess. Change (2) exposes *collect()* call as an attack surface for unprivileged users.

The *collect()* call validates correctly that the operation is performed on a non-empty allocation. It pulls tokens from the sender and distributes them to three groups. A portion is burnt as protocol tax, another portion is sent to the curator and the rest is registered as fees for the allocation. The function takes into account the **AllocationState.Closed** state, and directly inserts the fees to the rebate pool as it was already set up on *_closeAllocation()*. All invariants concerning the fees registered in the allocation and the rebate pool hold. For these reasons, our analysis concludes change (2) does not introduce complexity.

Findings

Medium severity findings

TRST-M-1 Operator can spoof query fees to make net profit from pool

- **Category:** Incentives issue
- **Source:** Staking.sol
- **Status:** Open

Description

Operators call *collect()* to pay query fees to the indexer. The fees are accumulated for all allocations that end in the same epoch in a **Rebates.Pool** structure, later to be split per indexer using the Cobb-Douglas production function.

This fee structure *should* be resistant to query fee donations that:

1. Lower another indexer's total rebate amount
2. Result in a net positive for the donator (query fee MEV)

However, stress testing of the function found that guarantee 2 is not held. Operators, which are transitioning to be a decentralized role, can fake queries and increase their portion of the pool. This does not directly harm other indexers, because their share of the pool increases as well. However, the share of the rebate pool that stays in the protocol decreases.

An example is provided below. The rewards per indexer are calculated as follows:

$$reward(i) = totalRewards * \left(\frac{fee_i}{totalFees}\right)^{\alpha} * \left(\frac{stake_i}{totalStake}\right)^{1-\alpha}$$

Suppose $\alpha = 0.4523$, and the layout of pool participants is as follows:

Participant	Fee	Stake
Participant 1	511	515
Participant 2	794	1

The calculated rewards are:

$$reward(1) = 1305 * \left(\frac{511}{1305}\right)^{0.4523} * \left(\frac{515}{516}\right)^{1-0.4523} \cong 853$$

$$reward(2) = 1305 * \left(\frac{794}{1305}\right)^{0.4523} * \left(\frac{1}{516}\right)^{1-0.4523} \cong 34$$

Fees of the rebate pool not rewarded:

$$remaining = 1305 - 853 - 34 = 418$$

At this point, participant 1 donates 720. The new layout is:

Participant	Fee	Stake
Participant 1	1231	515
Participant 2	794	1

The calculated rewards are:

$$reward(1) = 2025 * \left(\frac{1231}{1305}\right)^{0.4523} * \left(\frac{515}{516}\right)^{1-0.4523} \cong 1615$$

$$reward(2) = 2025 * \left(\frac{794}{1305}\right)^{0.4523} * \left(\frac{1}{516}\right)^{1-0.4523} \cong 43$$

Fees of the rebate pool not rewarded:

$$remaining = 2025 - 1615 - 43 = 367$$

Calculating pool loss (of profits) from donation:

$$loss = 418 - 367 = 51$$

Profit was split between the participants:

$$profit(1) = 1615 - 720 - 853 = 42$$

$$profit(2) = 43 - 34 = 9$$

Attackers can donate query fees at any point before the allocation is finalized, which is when a certain number of epochs have passed since it was closed. This means in the final block there will be incentives for large amounts of MEV activity of the different participants, to the loss of the protocol.

A python script that fuzzes the Cobb-Douglas formula has been provided separately.

Recommended mitigation

Consider making use of a different awarding formula, which would disincentivize forging of query activities.

Team response

Acknowledged. There is ongoing economic research on alternatives to Cobb-Douglas, but for now we think this is an acceptable consequence of decentralizing gateways.

Low severity findings

TRST-L-1 collected amount might be entirely burnt as protocol fees unintentionally

- **Category:** Time-sensitive operations
- **Source:** Staking.sol

- **Status:** Open

Description

When *collect()* is called after allocation closed, entire pulled amount is consumed as protocol tax. The transition from closed state to finalized state is instantaneous at a specific epoch. Therefore, it may occur that money sent for curation and indexer query fees is consumed entirely as tax. This happens when the time between sending of TX and its execution is larger than the remaining dispute window.

Recommended mitigation

Consider adding an optional parameter in the *collect()* API, **allowMaxTax**. If users are willing for the fee to be completely burned, they may set it to true.

Team response

Acknowledged. The proposed fix would require updating the interface with the existing **AllocationExchange**, which is not upgradable, so instead we will document this risk in the function's notice so that callers ensure they call the function well before the end of the dispute window.

Additional recommendations

Improve documentation

Documentation of the *collect()* function states:

```
/**
 * @dev Collect query fees from state channels and assign them to an
allocation.
 * Funds received are only accepted from a valid sender.
 * To avoid reverting on the withdrawal from channel flow this
function will:
 * 1) Accept calls with zero tokens.
 * 2) Accept calls after an allocation passed the dispute period, in
that case, all
 *    the received tokens are burned.
 * @param _tokens Amount of tokens to collect
 * @param _allocationID Allocation where the tokens will be assigned
 */
```

Note that the highlighted text is no longer relevant, now that the operator is decentralized. It should be omitted.

Redundant event emission

In *collect()*, the event AllocationCollected is emitted outside the main if block. It is recommended that it shall be placed inside the if block, as when **queryFees** is zero, the function doesn't change state and therefore shouldn’t emit an event.

Centralization risks

Operator – Indexer trust assumptions

The fee collection mechanism in Graph Staking is still somewhat trusted. For example, if operator does not maintain sufficient GRT balance, the indexer would not be able to trigger collection from the operator. This could be seen as outside the scope of the Staking protocol; however it is Graph's responsibility to create an incentive structure that enables establishment of relations across the different Graph roles.