# Introduction

The Graph asked us to review their mechanism to distribute GRT tokens under locking and vesting time conditions. We audited contracts in commit `b666f0f6c6b6b29f8b08368c962ddff8cf143e52` of the `token-distribution` repository.

At the time of writing, the audited commit is not into the master branch yet. We assume that the code used to deploy contracts will be the audited one and that the pull request associated with the audited commit will be merged.

The scope for this audit is limited to the following contracts inside the `contracts` folder:

- `GraphTokenLock.sol`.
- `GraphTokenLockWallet.sol`.
- `GraphTokenLockManager.sol`.
- `IGraphTokenLockManager.sol`.
- `MathUtils.sol`.
- `MinimalProxyFactory.sol`.
- `Ownable.sol`.

While the following are left out of scope:

- `GraphTokenDistributor.sol`.
- `GraphTokenMock.sol`.
- `Stakes.sol`.
- `StakingMock.sol`.
- All directories and files outside the `contracts` directory.

## Overview

The system aims to provide its users with a wallet containing GRT tokens. These tokens can be used to execute some actions in the protocol (like staking or unstaking them), but they can't be withdrawn by the beneficiary until some time and vesting conditions are satisfied. These wallets are created through a factory contract managed by a governance address.

The main object is the `GraphTokenLockWallet` contract, a wallet that receives GRT tokens upon creation, and that has two main functionalities:

### Locking funds under vesting and time conditions

`startTime` and `endTime` variables are used to define the timeframe in which the tokens are locked and released in slots, while the `periods` variable is used to define in how many slots the tokens must be released. All slots have equal sizes calculated as the total lock time divided by the number of periods.

The `releaseStartTime` variable also defines exactly the first available time for the release of the first slot.

Wallets can be configured as vested or not. This is represented by the use of the `revocable` variable, and if this is `true`, the owner of the contract can pull back the remaining unvested tokens. Moreover, the `vestingCliffTime` is a timeframe in which the beneficiary can't release any tokens because they are not vested yet. In the same timeframe the owner of the contract can revoke all the tokens from it.

Finally, beneficiaries can also subtract any surplus amount of tokens that they receive by interacting with The Graph protocol to their address.

### Forward calls to the protocol contracts through a list of admitted function calls

This is implemented in the fallback function present in the contract. A list of admitted functions set by the `GraphTokenLockManager` owner is used to forward or reject incoming calls into the fallback function. If the `msg.sig` of the call doesn't represent a whitelisted function call, the transaction will revert, otherwise the `_target` address will be called at the specified function with the passed `msg.data`.

In this way, users that have locked tokens in the contract can use them to operate with the system even under locking restrictions. Notice that before any valid call to the fallback function, the user must be able to first call `approveProtocol` so that target addresses of forwarded function calls, which are supposed to be the protocol's contracts, have proper allowance to move tokens out of the user's wallet.

At any time the user can also revoke such allowance.

The system uses the minimal proxy pattern to deploy copies of a `GraphTokenLockWallet` master version. The deploy of such copies is implemented using the `create2` opcode to know the address where the contract will be deployed in a more predictable way. Whenever a

new copy is created, the newly generated contract is funded by the `GraphTokenLockManager` contract.

In order for the `GraphTokenLockManager` to fund new wallets, the owner of the contract can deposit and withdraw GRT tokens.

#### Governance

There are two `owners` in the contracts:

- An `owner` for the `GraphTokenLockManager` contract which is the `msg.sender` that deployed the contract. This will be in charge of creating new wallets, set/unset authorized function calls, add/remove allowed target addresses for those functions calls (the protocol contracts), withdraw funds from the contract, and change the master copy to be used in every deploy of a wallet.
- An `owner` for the `GraphTokenLockWallet` contract which is passed as an input parameter in the call to create a new wallet. This `owner` will have the power to revoke unvested tokens if the contract is revocable and to change the `GraphTokenLockManager` address associated with the wallet.
- A `beneficiary` address which is passed as an input parameter when creating a new wallet. This account will have the power to release unlocked tokens, withdraw any surplus in the wallet, approve and revoke tokens allowance to protocol contracts and ultimately change the beneficiary itself to another address.

## Security assumptions

The current audit makes several assumptions which, in case of not being true, will undermine the results of the audit and may result in the addition of new risks for any party involved in the system:

- The `owner` of the `GraphTokenLockManager` contract will be a Multisig trusted by the beneficiaries.
- The `owner` of the `GraphTokenLockManager` contract will only set the `token` variable to the address of the current GRT token implementation.
- The `owner` of the `GraphTokenLockManager` contract will set the `masterCopy` variable to the address where the current `GraphTokenLockWallet` implementation is deployed.
- The `owner` of the `GraphTokenLockWallet` contract will be a Multisig trusted by the beneficiaries.

- The `owner` of the `GraphTokenLockWallet` contract will set the `manager` variable to the address of the current `GraphTokenLockManager` implementation.

# Summary

We are happy to see implementations that improve code efficiency and lower gas costs. The code is easy to read and understand with modular and compacted functions and a proper inheritance scheme.

We reviewed the code with two auditors over the course of eight days. Here we present our findings.

During the course of the review The Graph team discovered a potential issue when users were able to use unvested tokens in protocol actions and make the revocation process fail. The issue is described in #18 and addressed in PR#19.

***Update:*** *The Graph team made some fixes and comments based on our recommendations. We address below the fixes introduced in individual pull requests. Our analysis of the mitigations disregards any other changes to the codebase. Note that at the time of this writing, not all pull requests have been merged.*

# Critical Severity

None.

# High Severity

## H-01

## [H01] Addresses associated with GraphTokenLockWallet are unreliable

In the execution of the `createTokenLockWallet` function of the `GraphTokenLockManager` contract, the `_deployProxy2` public function is being used with a salt created from the initialization values and the `masterCopy` implementation address.

A malicious attacker can frontrun any transaction in the mempool calling the `createTokenLockWallet` by sending a transaction calling the `_deployProxy2` function with the same exact parameters except from the `_data` parameter, which holds the information to initialize the `GraphTokenLockWallet` contract and can be set by the attacker either to null or untrusted values. This will result in the ability to deploy contracts in the precomputed addresses which can be either initialized with nefarious variables or uninitialized.

The purpose of using `create2` instead of `create` is to have predictable addresses where contracts will be deployed and this is useful for applications that can rely on addresses of contracts which are not deployed yet. If this attack succeeds, any reliability on such addresses will be lost, since the generated `GraphTokenLockWallet` would not be a reliable contract anymore.

Consider modifying the `_deployProxy2` function visibility to `internal` so that nobody can make use of the `create2` opcode from the contract's address. Additionally, consider using the `msg.sender` value to calculate the salt, so that two transactions from different callers will produce different contracts at different addresses, protecting from such scenario.

**Update:** *Fixed in PR#11. The `_deployProxy2` function visibility is now `internal`.*

# Medium Severity

None.

# Low Severity

L-01

## [L01] Unclear inheritance design

The `GraphTokenLockWallet` contract needs to override the `initialize` function of the `GraphTokenLock` contract so that the `manager` variable is also initialized.

This is error prone and may result in future problems due to the fact that the contracts are upgradeable.

Consider converting the `GraphTokenLock` contract into an abstract one by exclusively defining the `initialize` function and implementing it in the `GraphTokenLockWallet`'s `initialize` function to have a more clear and less error-prone code. Alternatively, if for any reason the `GraphTokenLock.initialize` function is needed, consider declaring it as internal in order to avoid overriding it.

**Update:** *Fixed in PR#17 and PR#21. The `GraphTokenLock` contract is now abstract and implements an internal `_initialize`. The `GraphTokenLockWallet` only implements one `initialize` function now. A new `GraphTokenLockSimple` contract has been added too in order to deploy `GraphTokenLock` wallets that don't interact with the protocol.*

L-02

## [L02] Input parameters are not validated

The constructor function of the `GraphTokenLockManager` is setting the `_token` and `masterCopy` parameters without sanitizing the provided input values. Even if the `masterCopy` can be changed afterwards, there's no way to eventually change the `_token` address.

Additionally the `addTokenDestination` function is not checking whether the `_dst` parameter is the zero address.

Consider validating the input values in the constructor or in function calls to avoid setting incorrect values.

**Update**: *Fixed in PR#16. The* `_graphToken`, `_masterCopy` *and* `_dst` *values in the* `GraphTokenLockManager` *are now validated to be different than the zero address.*

L-03

## [L03] Lack of indexed parameters in event

There are some event definitions which are lacking of indexed parameters. Some examples of this are:

- The `MasterCopyUpdated` event of the `GraphTokenLockManager` contract.
- The `BeneficiaryChanged` event of the `GraphTokenLock` contract.
- The `ProxyCreated` event of the `MinimalProxyFactory` contract.

Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.

**Update**: *Fixed in PR#15. The* `MasterCopyUpdated` *and* `ProxyCreated` *events are now indexing their parameters. The* `BeneficiaryChanged` *event is no longer present in the codebase.*

# Notes & Additional Information

## N-01
## [N01] Using default value in variable

The `createTokenLockWallet` of the `GraphTokenLockManager` is using the `_revocable` variable to signal whether the contract is revocable or not.

The default false value will signal to the system that the created wallet is not revocable.

To avoid setting improper values and relying on default values for important settings in the system, and given the fact that default values are often returned by client applications if something went wrong, consider using an Enum type variable that treats the default value as if the variable is not being set.

**Update**: *Fixed in PR#9. The `Revocability` enum was created and the codebase have been adapted to use this variable to keep track of the possibility of a wallet being revocable. This value is enforced to be set during initialization of the `GraphTokenLock` contract.*

## N-02
## [N02] Gas optimization possible

In the `createTokenLockWallet` function of the `GraphTokenLockManager` contract, the new contract is being deployed without validating the conditions under which the `safeTransfer` call can fail.

In case the contract does not have the amount of tokens needed to fulfill the transfer, the transaction will revert after consuming a lot of gas.

Consider checking that the `GraphTokenLockManager` contract has the needed funds before deploying the new `GraphTokenLockWallet` contract.

**Update**: *Fixed in PR#12. The `GraphTokenLockManager` is now validating that it has the funds to transfer to the wallet before deploying it.*

N-03

## [N03] Using a one-step transfer for ownership and tokens recipient

In the `Ownable` contract, the `transferOwnership` function is setting a new `_owner` to the provided input address. Even if the input address is checked to not be zero, there are an infinite number of addresses that can be erroneously set.

The same happens when a beneficiary wants to change the destination of the tokens in the wallet calling the `changeBeneficiary` function of the `GraphTokenLock` contract.

Notice that both the new owner and the new beneficiary will be the only one able to change these values again, so any incorrect address set for one of them, could result in loss of ownership or impossibility to retrieve tokens from the wallet.

When doing such operations, a recommended pattern is to use a two-steps design where the transfer is first initiated and then accepted by the corresponding recipient. In this way, an incorrect value can be either overwritten by a new transaction.

Consider chosing the recommended pattern to avoid consequences of changing the `owner` or `beneficiary` variables to incorrect values.

**Update**: *Not fixed. The Graph decided to not add more complexity and cost to the operations due to the big volume of contracts they plan to deploy.*

N-04

## [N04] Not bubbling up `call` revert reasons

Both the `_deployProxy` and the `_deployProxy2` functions of the `MinimalProxyFactory` contract make use of the low-level `call` keyword to initialize the created contracts, which does not give the opportunity to retrieve the eventual revert reason.

Consider changing the `_deployProxy2` function to use the `Address.functionCall` function to improve the understanding of the failure reasons of any of these calls instead of manually handling the call's result.

**Update:** *Fixed in PR#10. Both functions make use of the `Address.functionCall()` function now.*

N-05

## [N05] Unused functions

The `_deployProxy` function of the `MinimalProxyFactory` and the `max` function of the `MathUtils` contract are defined but not used in the code base.

This increases the code complexity and the size of the bytecode to be deployed.

Consider removing all the unused functions.

**Update**: *Fixed in PR#14. The `_deployProxy` and `max` functions have been removed from the code base.*

N-06

## [N06] Wrong visibility in functions

There are some functions that are not being accessed locally but are being declared as `public` instead of `external`. Some examples are:

- The `getDeploymentAddress` function of the `MinimalProxyFactory` contract is marked as public, while it can be declared as external since it is not used inside the contract nor in contracts extending from this one.
- The `approveProtocol` and `revokeProtocol` functions of the `GraphTokenLockWallet` are declared as public but not called internally.
- The `renounceOwnership` and `transferOwnership` functions of the `Ownable` contract are all public but not internally called in the code base.

To improve clarity, readability and to possibly save some gas, consider reviewing all function visibilities in the code base and mark their visibility according to their scope and access level.

**Update**: *Fixed in PR#13. The complete set of functions mentioned in the issue have been modified to use the `external` visibility.*

# Conclusions

Only one high vulnerability was found. The token distribution contracts are simple and consistent. However, they are based on strong assumptions which should be validated by the beneficiary before making use of the wallet. Monitoring for future changes on these assumptions is also recommended for long-time distributions.