The Graph Timeline Aggregation Audit

OPENZEPPELIN SECURITY | SEPTEMBER 20, 2023

Security Audits

This security assessment was prepared by **OpenZeppelin**.

Table of Contents

Summary

Type
: DeFi

Timeline
: From 2023-07-31
: To 2023-08-11

Languages
: Solidity

Total Issues
: 15 (12 resolved, 2 partially resolved)

Critical Severity Issues
: 1 (1 resolved)

High Severity Issues
: 0 (0 resolved)

Medium Severity Issues
: 3 (2 resolved, 1 partially resolved)

Low Severity Issues
: 5 (3 resolved, 1 partially resolved)

Notes & Additional Information
: 6 (6 resolved)

Scope

We audited the semiotic-ai/timeline-aggregation-protocol-contracts repository at two commits on largely the same body of code. The first scope is at the d8795c7 commit covering the core functionality of the Timeline Aggregation Protocol (TAP). The second scope is at the 7f663fb

In both scopes were the following contracts:

```
src/
├── AllocationIDTracker.sol
├── Escrow.sol
├── IStaking.sol
└── TAPVerifier.sol
```

System Overview

The Timeline Aggregation Protocol (TAP) is a payment system between Indexers and gateways. It aims to aggregate the receipts per allocation to settle payments for the queries served by Indexers for each gateway. This implementation is based on the Graph Improvement Proposal GIP-0054. TAP extends the existing Scalar payment system by allowing multiple gateways, aggregated receipts, and minimized trust between the payers and payees.

Security Model and Trust Assumptions

The smart contracts interact with off-chain components of the TAP protocol, where query receipts are signed and eligible receipts are aggregated into Receipt Aggregate Vouchers (RAVs) for each allocation period. The RAVs are subsequently signed by an authorized signer from each gateway. A gateway can deposit an amount of GRT tokens specifically destined for an Indexer, which can be claimed with a signed RAV. The RAV signature conforms to EIP-712 and is verified on-chain by `TAPVerifier.sol`. Only a single RAV can be redeemed for each allocation from each sender. However, multiple vouchers can be redeemed for the same `allocationID` so long as they originate from different gateways. This is different from the current `AllocationExchange.sol` contract, which only allows Indexers to claim a voucher for an `allocationID` once globally.

Both the signing and receipts aggregation processes are off-chain, and thus out of scope for this audit. The on-chain components interact closely with the off-chain ones, as well as The Graph's

Privileged Roles

Three role types are expected to interact with the smart contracts: `senders` (gateways), `signers` and `receivers` (Indexers).

- `Signers` are authorized by `senders` to sign RAVs for receivers.
- `Senders` can revoke `signers` after a thawing period.
- `Senders` can deposit to any `receiver` and withdraw any unclaimed deposit after a grace period.

In the first scope, anyone is allowed to be a `sender`, thus being able to push any amount of GRT tokens through to The Graph's `Staking` contract via the TAP. This may have side effects on other aspects of The Graph's ecosystem. In the second scope, `senders` need to be whitelisted as `AUTHORIZED_SENDER`s and managed via a trusted `DEFAULT_ADMIN` role.

Critical Severity

Front-Running `redeem` Can Prevent Indexers From Receiving Rewards for Allocations

The `redeem` function in `Escrow.sol` enables Indexers to receive query rewards by submitting a signed Receipt Aggregate Voucher (RAV) and `allocationIDProof`. However, anyone who knows the contents of a valid `signedRAV` and `allocationIDProof` can call `redeem` regardless of whether the proof and signed RAV belong to them. This is because `redeem` only checks that the contents and signature of the passed-in `signedRAV` and `allocationIDProof` are valid, but does not check that the caller is the originator of the signatures and calldata. Additionally, the function uses the caller to determine the amount of GRT that will be sent as the query reward to the `Staking` contract.

Consequently, a malicious user who knows a valid `signedRAV` and `allocationIDProof` can call `redeem`, which, if the user does not have a GRT balance in the `Escrow` contract, will result in zero GRT being rewarded for an `allocationID`. This also prevents future calls to `redeem` that correspond to the same `allocationID` as the ID will have been marked as used in the `AllocationIDTracker` as part of the `redeem` function's logic. This effectively

1. An Indexer calls `redeem` with their `signedRAV` and `allocationIDProof` on the Ethereum Mainnet.
2. A malicious user sees the proposed transaction in the public mempool, creates a duplicate transaction using the now public information, and pays to front-run the Indexer's transaction.
3. The malicious user's call to redeem happens first, and because they do not have any GRT balance in the `Escrow` contract, zero GRT will be awarded to the `allocationID` via the collect call. Note that both `redeem` and `collect` (in `Staking.sol`) will pass even though zero GRT is awarded to an Indexer. Additionally, this uses the `allocationID` in the `AllocationIDTracker` contract.
4. The Indexer's redeem call happens and fails because `useAllocationID` will now revert.

Consider deriving the Indexer address to pull GRT from via the `getAllocation` function in the Staking contract instead of using the `msg.sender` as the expected address in the `redeem` function. This prevents the wrong address' `EscrowAccount` 's GRT balance from being used and always ensures that regardless of who calls `redeem`, the correct address will be used when moving GRT to the `Staking` contract.

Update: *Resolved in pull request #58. The Graph's core developers stated:*

> *Thank you for finding and highlighting this critical issue. We have investigated it further and discussed a few solutions. To prevent possible front-running attacks and allow the vesting contracts to redeem, we decided to proceed with the suggested solution to use the Indexer indicated in the allocation as the receiver (i.e., obtaining the receiver address from the staking contract using `allocationID`). The associated issue can be found here.*

Medium Severity

`redeem` Lacks Slippage Protection

The `redeem` function in `Escrow.sol` contains the logic to determine the amount of GRT rewards an Indexer will receive as the minimum of the currently available escrow balance and the aggregated amount in the voucher. This allows slippage to occur against the Indexer (i.e., the Indexer may get less than what is deemed acceptable).

the Indexer, who may have otherwise expected the call to revert if the full reward amount could not be given to them.

Consider adding a parameter to the `redeem` function that allows the caller to specify an expected reward amount. If the calculated amount is not greater than or equal to the expected amount, the call should revert.

Update: *Partially resolved in pull request #61. The Graph's core developers stated:*

> *We decided to leave the code as is (option #2). This allows flexibility while maintaining a low gas cost. We have updated the inline documentation to detail the expectations of Indexers during `thaw`s and `redeem`s. Some context for the decision: Whenever a `thaw` commences and threatens to reduce the escrow account below the owed amount, the Indexer software should promptly act to close any pending RAVs with that gateway. Otherwise, the Indexer should act as if anything being thawed has already been withdrawn. Complete transparency exists regarding an account: Indexers can verify values anytime, and numerous events (like deposit, thaw, withdraw, redeem, and other account interactions) are signaled. The only methods to withdraw money from an account are either under the Indexer's control (`redeem`) or are subject to a time-lock (`withdraw`). The associated issue can be found here.*

Signed Messages Can Be Replayed

In both the `verifyProof` and `verifyAuthorizedSignerProof` functions, the `messageHash` does not include the `chainId`. Since the project will be deployed on both Ethereum and Arbitrum networks, the same signed message intended for one chain can be replayed on the other chain.

Consider adding the `chainId`, as well as the nonce or deadline to the message's content when generating proofs to ensure that signed messages are only used on the intended blockchain, and not replayable multiple times.

Update: *Resolved in pull request #56. The Graph's core developers stated:*

> *Thank you for highlighting this issue. After several discussions, we fixed this by adding a `chainID` to the `allocationID` proof (a deadline is not needed since a used*

associated issue can be found here.

Vesting Contracts Cannot Call `redeem`

Certain users participate in The Graph's protocol via a vesting contract, which enables them to be awarded GRT over a period of time and still utilize the tokens for staking, curating, and delegating. In order to prevent the awarded GRT from escaping the vesting lock before the end of the vesting period, the smart contract restricts function calls and function call targets to only those approved by The Graph. Therefore, Indexers that are vesting contracts would initially be unable to call `redeem` in the `Escrow` contract until the function signature and address are approved by The Graph. This is because the `redeem` function's logic currently calculates the amount of GRT rewards to forward to the `Staking` contract from the caller of the function.

Consider deriving the Indexer address to pull GRT from via the `getAllocation` function in the `Staking` contract instead of using the `msg.sender` as the expected address in the `redeem` function. This allows an Indexer who is also a vesting contract to use a separate caller to call the `redeem` function and still get the correct amount of GRT rewards without having to approve the `redeem` function signature for all vesting contracts.

Update: *Resolved in pull request #58. The Graph's core developers stated:*

> *This issue is fixed with the same solution as C-01, linking it to the same pull request. The associated issue can be found here.*

Low Severity

Lack of Integration Tests With Existing Contracts

TAP contracts are tested against mocks of The Graph's contracts (such as Staking), which works well for unit tests. However, having integration tests that test against the deployed contracts with varied inputs, such as using different fee-tiers, ensures there are no undesirable side effects to other parts of the ecosystem.

Update: *Acknowledged, will resolve. The Graph's core developers stated:*

> *Foundation.*

Unnecessary Event Emission

The `thaw` function in `Escrow.sol` will succeed and emit the `Thaw` event for any caller that specifies zero as the `amount` to thaw, even if the caller has not deposited any GRT into the contract. Allowing the contract to emit an arbitrary number of `Thaw` events could disrupt off-chain monitoring. Consider adding a check to ensure the passed-in `amount` to thaw is greater than zero.

Update: *Resolved in pull request #54 at commit 73efad0. The Graph's core developers stated:*

> *We added a check for the thaw request amount to be greater than 0. In the first pull request #54, we checked if `amount <= 0`, but as it is a `uint256`, we modified this to check `if == 0`, along with the thaw fixes, pointing to the last commit. The associated issue can be found here.*

`push0` OPCODE Incompatible With Arbitrum

The `push0` OPCODE (EIP-3855) is implemented in the latest Shanghai EVM version, which is the default version for the Solidity compiler `0.8.20` or higher. As of the time of writing, Arbitrum does not yet support the `push0` opcode. Thus, one should either cap the Solidity version to `0.8.19` or carefully set the EVM target to a lower version than the default one.

Update: *Resolved in pull request #55. The Graph's core developers stated:*

> *We capped the Solidity version to 0.8.18. The associated issue can be found here.*

`thawSigner` May Be Ineffective

The `thawSigner` function in `Escrow.sol` allows a sender to initiate the process of revoking authorization from a signer address. After initiating `thawSigner`, the signer can continue to sign vouchers that can be redeemed until the signer is revoked.

This process allows a sender to initiate `thawSigner` as soon as the signer is authorized. After the thaw period is over, the sender has full discretion on when to revoke the signers, which can take effect immediately. This consequently disables the redemption of any vouchers signed by the

Consider reassessing the effectiveness of `thawSigner` and r`evokeAuthorizedSigner`'s functionality, and decide whether they can be removed.

Update: *Partially resolved in pull request #59. The Graph team stated:*

> *Thank you for posting this issue. After discussing this with our partners from Edge & Node, we concluded that we do not fully align with the problem's statement. As the `thawSigner` will be called from the Indexer software, we decided to enhance its documentation to provide more meaningful details. "Thawing a signer" is designed to alert Indexers that signatures from that signer will soon be deemed invalid. Indexers without existing signed receipts or RAVs from this signer should, in essence, treat them as unauthorized. Those with existing signed documents from this signer should work towards settling their engagements. They can either initiate an RAV request using the signed receipts (with a new signer in place), redeem their most recent RAV, or request a new RAV associated with a different signer. Once a signer is thawed, they should, for all practical reasons, be viewed as revoked regardless of their revocation status. Given this context, we are confident in the current implementation's efficacy in serving as an alert mechanism, especially with the emitted event and provided grace period, which lets Indexers act accordingly. While we do not plan on altering the code based on this suggestion, we will enhance our documentation to ensure such expectations are conveyed transparently. The associated issue can be found here.*

Retrieving Escrow Account Details via Signers Can Be Misleading

The `getEscrowAccountFromSignerAddress` function accepts a `signer` as its argument and subsequently infers the `sender` address to return the amount of `EscrowAccount` deposited by the `sender` to the `receiver`. If a signer is revoked by the sender, the function will return 0 despite the fact that there could be non-zero amounts in escrow left for the `receiver` by the `sender`.

Consider reverting with an error message indicating that the signer is no longer authorized.

Update: *Resolved in pull request #53. The Graph's core developers stated:*

> *The associated issue can be found here.*

Notes & Additional Information

Unused Import

In `TAPVerifier.sol`, the import `Address` is unused and could be removed.

Consider removing unused imports to improve the overall clarity and readability of the codebase.

Update: *Resolved in pull request #47. The Graph's core developers stated:*

> *We removed the `Address` import. The associated issue can be found here.*

Sender Is Unable to Decrease the Thaw Amount or Cancel the Thaw

The `thaw` function in `Escrow.sol` can only increase the value to be withdrawn from an `EscrowAccount`. Additionally, the only way to reset the `amountThawing` for an `EscrowAccount` to zero is to call `withdraw`, which will pull the requested GRT from the account. Therefore, there is no way to cancel a `thaw` action. This can lead to a poor user experience for depositors and accounts. Consider creating an additional function, `cancelThaw`, that reduces the `amountThawing` value for an account to zero.

Update: *Resolved in pull request #52. The Graph's core developers stated:*

> *If the requested amount is zero, any thawing in progress will be cancelled. If the requested amount is greater than zero, any thawing in progress will be cancelled and a new thawing request will be initiated. The associated issue can be found here.*

Typographical Errors

The following typographical errors were identified:

Update: *Resolved in pull request #51. The Graph's core developers stated:*

> *The associated issue can be found here.*

`unchecked` Arithmetic Blocks

The following arithmetic in `Escrow.sol` occurs in `unchecked` blocks:

- Line 263
- Line 387

There does not appear to be an underflow risk in the logic as the amount deducted is always less than or equal to the escrow balance. Inserting mathematical operations inside an `unchecked` block will reduce the amount of gas used in a transaction. However, in case of unpredictable scenarios (e.g., unknown compiler issues), having an underflow in the `balance` variable would be disastrous. We recommend a safer approach by removing the `unchecked` block to allow the compiler to insert overflow and underflow guards

Update: *Resolved in pull request #50. The Graph's core developers stated:*

> *We agree it is not worth the risk to save the gas costs. The associated issue can be found here.*

Signature Does Not Fully Comply With EIP-712

The signature for a Receipt Aggregate Voucher (RAV) is intended to be compliant with EIP-712. However, there is a discrepancy in the `typeHash`. Per EIP-712, the `typeHash` includes the `encodeType` for a struct, which is encoded as:

> The type of a struct is encoded as name ‖ "(" ‖ member$_1$ ‖ "," ‖ member$_2$ ‖ "," ‖ … ‖ member$_n$ ")" where each member is written as type ‖ " " ‖ name.

However, the name of the encoded struct in the `typeHash` within `TAPVerifier.sol` is `ReceiptAggregateVoucher`, whereas the name of the corresponding struct defined in the contract is `ReceiptAggregationVoucher` ("Aggregate" vs. "Aggregation"). The name of the

Consider renaming the struct to be identical to the name used in the `typeHash`.

Update: *Resolved in pull request #49. The Graph's core developers stated:*

> *The `ReceiptAggregationVoucher` struct was renamed to `ReceiptAggregateVoucher`. The associated issue can be found here.*

Incorrect Documentation

- Line 50 of `AllocationIDTracker.sol`: The word 'collateral' should be changed to 'escrow' as it is referring to the renamed `Escrow` contract.
- Line 71 of `AllocationIDTracker.sol`: The word 'collateral' should be changed to 'escrow' as it is referring to the renamed `Escrow` contract.
- Line 29 of Escrow.sol: 'Block number' should be changed to 'Timestamp' as `thawEndTimestamp` represents a timestamp.
- Line 34 of Escrow.sol: 'Block number' should be changed to 'Timestamp' as `thawEndTimestamp` represents a timestamp.

Consider addressing these instances of incorrect documentation.

Update: *Resolved in pull request #48. The Graph's core developers stated:*

> *Good catch, fixed. The associated issue can be found here.*

Conclusion

One critical-severity and a few medium-severity issues were identified during this audit. These issues stem from the increased surface area of external interactions that the TAP contracts allow. Additional suggestions were made to increase operational similarities between the TAP contracts and the current `AllocationExchange` contract.

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits