CANTINA

Telcoin: TelX

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Cryptara, Security Researcher

November 12, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Telcoin creates low-cost, high-quality financial products for every mobile phone user in the world.

From Oct 21st to Oct 24th the Cantina team conducted a review of telcoin-laboratories-contracts on commit hash b537324a. The team identified a total of **12** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	2	2	0
Low Risk	2	1	1
Gas Optimizations	2	2	0
Informational	5	4	1
Total	**12**	**10**	**2**

2.1 Scope

The security review had the following components in scope for telcoin-laboratories-contracts on commit hash b537324a:

```
contracts/telx/core
├── PositionRegistry.sol
├── TELxIncentiveHook.sol
└── TELxSubscriber.sol
```

3 Findings

3.1 High Risk

3.1.1 Subscriptions can be permanently filled

Severity: High Risk

Context: PositionRegistry.sol#L303-L304

Description: `PositionRegistry.handleSubscribe` accepts any caller whose owner has no current subscriptions and inserts that owner into the global subscribed list after a length check:

```
if (ownerSubscriptions.length == 0) {
    require(subscribed.length < MAX_SUBSCRIBED, "PositionRegistry: Max subscribed
    ↪   reached");
    subscribedIndexes[tokenOwner] = subscribed.length;
    subscribed.push(tokenOwner);
}
```

The registry only frees a slot inside `_removeSubscription` when `subscriptions[tokenOwner]` becomes empty. An attacker can add liquidity, call subscribe and immediately withdraw everything without triggering unsubscribe. Their `subscriptions[tokenOwner]` array retains the `tokenId` and keeps the owner marked as active even though liquidity is zero, so `_removeSubscription` never executes. Repeating this across 1000 fresh addresses fills the entire subscribed array. Every new LP that calls `handleSubscribe` now reverts on Max subscribed reached, permanently denying service.

Therefore, because of this, the TELx incentive program can be bricked by anyone at negligible cost while recovering their deposits.

Proof of concept: *(See gist 899ec1bd).*

Recommendation: Require meaningful liquidity before adding an owner to subscribed and drop entries when liquidity hits zero. For example, reject `handleSubscribe` when `params.liquidity < PREDEFINED_MIN_LIQ_AMOUNT` and regularly trim empty positions so `_removeSubscription` executes and frees capacity.

Telcoin Association: Fixed in commit 2fd0fcb4.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 `TELxRewardsCalculator.ts` script issues

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The published spec states that an off-chain rewards script consumes `PositionRegistry` data to weight payouts, but the current implementation of this `TELxRewardsCalculator.ts` script fails in several ways. First, it assumes every `ModifyLiquidity` log can be resolved to an owner via `ownerOf`. When Uniswap v4 burns a position, Solmate's ERC721 clears storage before emitting the log, so the lookup reverts with `NOT_MINTED`. The script does not catch that exception, meaning any LP that burns (or mint-burns intra-block) collapses the entire run and blocks distribution for the period:

```
const lp = await client.readContract({
  address: positionManager,
  abi: POSITION_MANAGER_ABI,
  functionName: "ownerOf",
  args: [tokenId],
  blockNumber: log.blockNumber,
});
```

Secondly, `updatePositions` records every modify event even when `liquidityDelta` is zero and adds it to `liquidityModifications`:

```
const change: LiquidityChange = {
  blockNumber: log.blockNumber,
  newLiquidityAmount: currentLiquidity + toBigInt(liquidityDelta!),
  owner: lp,
};
```

Later, `processFees` iterates those redundant entries, fetching fee growth twice per sub-period and re-running reward math. An attacker can spam zero-liquidity "pokes" to multiply off-chain RPC calls and CPU time, delaying or crashing reward cycles.

Finally, despite `PositionRegistry.configureWeights` exposing `JIT_WEIGHT`, `ACTIVE_WEIGHT` and lifetime thresholds, the calculator never reads them. It simply totals raw fee growth and distributes rewards pro rata:

```
const { token0Fees, token1Fees } = calculateFees(...);
const totalFees = Array.from(lpData.values())
  .map((data) => data.totalFeesCommonDenominator!)
  .reduce((a, b) => a + b, 0n);
```

Because these weights are ignored, operators cannot tune incentives against short-lived JIT liquidity and the program pays the wrong participants after the script survives the earlier crashes. Combined, these flaws let adversaries halt payouts, exhaust off-chain infrastructure and bypass the intended weighting scheme.

Recommendation: Harden the calculator to match the specification: wrap `ownerOf` calls in try/catch and treat reverts as burned positions so the run continues. Discard zero-delta modify events (or skip sub-periods where start equals end) before appending to `liquidityModifications` to bound RPC work. Finally fetch the registry's configured weights, classifying liquidity intervals accordingly before calculating rewards, or else remove the unused configuration and document that unweighted payouts are intentional.

Telcoin Association: Fixed in PR 51.

Cantina Managed: Fix verified.

3.2.2 Checkpoint Data Inconsistency Between OpenZeppelin Checkpoints and Custom Fee Tracking

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: The `_writeCheckpoint` function in the `PositionRegistry` contract contains an inconsistency in how it handles multiple checkpoints within the same block. The function uses OpenZeppelin's `Checkpoints.Trace224` for `liquidityModifications` tracking, which automatically updates existing checkpoint values when multiple updates occur in the same block rather than creating new entries contracts/utils/structs/Checkpoints.sol#L354-L359 However, the custom `feeGrowthCheckpoints` mapping and `CheckpointMetadata.totalCheckpoints` counter do not follow this same behavior, leading to data corruption and incorrect accounting.

When `pos.liquidityModifications.push(checkpointBlock, newLiquidity)` is called multiple times within the same block, the OpenZeppelin Checkpoints library internally checks if a checkpoint already exists for that block and updates/replaces the existing entry rather than creating a new one or updating it. Also the `feeGrowthCheckpoints[checkpointBlock]` mapping assignment overwrites the previous fee growth data for that block, and `metadata.totalCheckpoints++` increments every time regardless of whether a new checkpoint was actually created in the `liquidityModifications` array.

This creates several issues: first, the `totalCheckpoints` count becomes inflated and no longer accurately reflects the actual number of checkpoints stored in `liquidityModifications`; second, intermediate fee growth data within the same block gets permanently lost due to mapping overwrites; third, off-chain systems consuming the `Checkpoint` events will receive incorrect `checkpointIndex` values that don't correspond to actual array positions in the `liquidityModifications` structure. And finally the `liquidityModifications` checkpoint will also be replaced and not updated.

Recommendation: Implement consistent checkpoint handling by adopting the OpenZeppelin Checkpoints pattern for both liquidity and fee growth data. Always make sure to get the old values and take them into account when storing or pushing to the position/metadata. Consider checking the return value or comparing lengths before and after the push operation to determine if `totalCheckpoints` should be incremented.

Telcoin Association: Fixed in commit cd07ee30.

Cantina Managed: Fix verified.

3.3 Low Risk

3.3.1 JIT liquidity can drain TELx rewards

Severity: Low Risk

Context: TELxIncentiveHook.sol#L106-L108

Description: `TELxIncentiveHook` forwards the raw Uniswap v4 fee report straight into `registry.addOrUpdatePosition`, so any `feesAccrued.amount0()` and `feesAccrued.amount1()` values become part of the TELx checkpoint without validation. An attacker can take a flash loan to become nearly the entire liquidity share of a low-liquidity pool, call `modifyLiquidity` to add that liquidity, execute a large `donate()` and exit within the same unlock. Because the donation is distributed pro-rata at the instant it lands, their just-in-time liquidity absorbs roughly 99.999% of the fabricated fees; the rest is a tiny leak to existing LPs. Off chain, `TELxRewardsCalculator.processFees` records that spike in `periodFeesCurrency0/periodFeesCurrency1` and `populateTotalFeesCommonDenominator` plus `calculateRewardDistribution` convert it into a dominant TELx payout share. The attacker walks away with the scheduled reward pool each period while only sacrificing the dust donated to others.

Recommendation: Consider taking this behaviour into account in the off-chain rewards distribution script.

Telcoin Association: Acknowledged.

Cantina Managed: Acknowledged.

3.3.2 Improper Validation Order in Position Subscription Logic

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: The `handleSubscribe` function contains a validation order flaw that prevents the proper enforcement of the `UNTRACKED` flag for positions not created via the PositionManager. When `addOrUpdatePosition` encounters a position that wasn't created through the PositionManager, it calls `_setUntracked(tokenId)` which sets `positions[tokenId].owner = UNTRACKED`, but crucially does not initialize the `poolId` field, leaving it as the default zero value.

In `handleSubscribe`, the validation checks occur in the wrong order. The function first validates `validPool(pos.poolId)` before checking `pos.owner != UNTRACKED`. Since positions marked as `UNTRACKED` have uninitialized `poolId` values (which default to zero), the `validPool` check fails first and the transaction reverts with ”PositionRegistry: Invalid pool” rather than the intended ”PositionRegistry: Only positions created via PositionManager can be subscribed” message.

Recommendation: Reorder the validation checks in `handleSubscribe` to verify the `UNTRACKED` status before validating the pool or initialise the position information via `_updatePosition` and `isNew == false`, that allows setting the owner and poolId, instead of manually calling `_setUntracked` that will only set the owner.

Telcoin Association: Fixed in commit 2bcb6c7b.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Redundant Public Getter Function

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: The `PositionRegistry` contract implements a `getSubscribed()` function that simply returns the `subscribed` array, which is already declared as a public state variable. Solidity automatically generates a public getter function for public arrays, making the explicit `getSubscribed()` function redundant.

Recommendation: Consider either removing the explicit `getSubscribed()` function to rely solely on the automatically generated public getter, or alternatively change the `subscribed` array visibility to `internal` or `private` and keep the explicit function for interface compliance.

Telcoin Association: Fixed in commit bd35e748.

Cantina Managed: Fix verified.

3.4.2 Gas-Intensive Linear Search in Token Subscription Check

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: The `isTokenSubscribed` function performs a linear search through the entire `subscriptions` array for a given owner to determine if a specific `tokenId` is subscribed. With `MAX_SUBSCRIPTIONS` set to 100, this function can iterate through up to 100 array elements in the worst-case scenario, consuming substantial gas especially when called multiple times within a single transaction or when used in complex contract interactions.

Recommendation: Implement a dedicated mapping such as `mapping(uint256 => bool) public tokenSubscrib` to provide O(1) constant-time lookup for token subscription status. Update this mapping to `true` when tokens are subscribed in `handleSubscribe` and set to `false` during unsubscription in `_removeSubscription`. This approach maintains the existing `subscriptions` array for enumeration purposes while providing efficient subscription status checks.

Telcoin Association: Fixed in commit dd18e2a5.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Unused swap event

Severity: Informational

Context: TELxIncentiveHook.sol#L22-L25

Description: `TELxIncentiveHook` declares `SwapOccurredWithTick` with a comment claiming it fires on every swap:

```
event SwapOccurredWithTick(
    PoolId indexed poolId,
    address indexed trader,
    int256 amount0,
    int256 amount1,
    int24 currentTick
);
```

No code path actually emits this event, so the comment is inaccurate and downstream tools cannot rely on it. The referenced `TELxRewardsCalculator.ts` script reads on-chain fee growth directly and never listens for this event, meaning it serves no functional purpose.

Recommendation: Drop the unused declaration (and its comment) or emit it where swaps occur if off-chain range validation truly needs it.

Telcoin Association: Fixed in commit 742a7eed.

Cantina Managed: Fix verified.

3.5.2 `erc20Rescue` function can drain rewards

Severity: Informational

Context: PositionRegistry.sol#L448-L451

Description: `PositionRegistry` escrows TEL for outstanding `unclaimedRewards`, yet the `erc20Rescue` helper lets any `SUPPORT_ROLE` operator transfer arbitrary ERC20 balances without checking pending obligations:

```
function erc20Rescue(IERC20 token, address destination, uint256 amount)
    external
    onlyRole(SUPPORT_ROLE)
{
    token.safeTransfer(destination, amount);
}
```

Because the function neither restricts token to non-reward assets nor enforces that the post-transfer balance still covers the sum of recorded claims, a compromised support wallet can empty the TEL escrow while `unclaimedRewards` continues to report debts. Subsequent `claim()` calls then revert due to insufficient funds, freezing payouts.

Recommendation: Restrict `erc20Rescue` so it can not touch the TEL rewards escrow, for example, reverting if the token is the reward token(TEL).

Telcoin Association: Acknowledged.

Cantina Managed: Acknowledged.

3.5.3 Unused Import Statements

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `PositionRegistry` contract imports `SafeCast` from OpenZeppelin and `FullMath` from Uniswap V4 core libraries but does not utilize either of these imports anywhere in the contract code.

Recommendation: Remove the unused import statements for `SafeCast` and `FullMath` to improve code clarity.

Telcoin Association: Fixed in commit 895da39e.

Cantina Managed: Fix verified.

3.5.4 Checkpoint Block Number Truncation

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `PositionRegistry` contract contains an issue where block numbers are unsafely cast to `uint32` in the `_writeCheckpoint` function, creating a potential overflow condition that could corrupt checkpoint data. The contract casts `block.number` to `uint32` in the `_writeCheckpoint` function parameter, limiting the maximum representable block number to approximately 4.29 billion ($2^{32} - 1$).

While Ethereum mainnet is currently around block 23 million, this truncation creates several serious issues. First, faster blockchains or testnets with accelerated block times could reach the overflow threshold much sooner than anticipated. Second, the OpenZeppelin Checkpoints library internally uses `uint48` for checkpoint keys, providing a much larger range that can accommodate block numbers up to 281 trillion, but the contract's premature truncation to `uint32` wastes this extended capacity and creates an artificial limitation.

Recommendation: Change the block number parameter type from `uint32` to `uint48` to align with the OpenZeppelin Checkpoints library's internal key format and eliminate the overflow risk. Update the `CheckpointMetadata` struct to use `uint48` for `firstCheckpoint` and `lastCheckpoint` fields, and modify all related function parameters and variables that handle block numbers.

Telcoin Association: Fixed in commit 8db6e1ad.

Cantina Managed: Fix verified.

3.5.5 Inconsistent Position Manager Access Control Implementation

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The TELx protocol contains an inconsistency in the implementation of position manager access control across its core contracts. The `TELxIncentiveHook` and `TELxSubscriber` contracts both implement `onlyPositionManager` modifiers, but use different error handling mechanisms and parameter structures, creating inconsistent user experience and gas efficiency patterns within the same protocol.

Recommendation: Standardize the position manager access control implementation across all TELx contracts by adopting the more efficient custom error approach used in `TELxSubscriber`. Update `TELxIncentiveHook` to define a custom error `OnlyPositionManager()` and modify the `onlyPositionManager` modifier to use `revert OnlyPositionManager()` instead of the string-based `require` statement.

Telcoin Association: Fixed in commit 30555bac.

Cantina Managed: Fix verified.