Sablier Staking

Security Review

Cantina Managed review by:

Cergyk, Lead Security Researcher
Cryptara, Security Researcher

October 14, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 Medium Risk . . . 4
  - 3.1.1 `LockupNFT` owner can make stream uncancellable by sending it directly to staking contract . . . 4
- 3.2 Low Risk . . . 5
  - 3.2.1 Pool admin who controls reward token can brick stakers funds by sending excessively high rewards . . . 5
  - 3.2.2 Ambiguity in Pool Start Time Initialization . . . 6
  - 3.2.3 Precision Loss in Reward Distribution Calculation . . . 6
  - 3.2.4 Premature Scaling Down . . . 7
- 3.3 Gas Optimization . . . 7
  - 3.3.1 Incorrect Reward Distribution at Pool Start Time . . . 7
- 3.4 Informational . . . 7
  - 3.4.1 Suggestion to rename `snapshotRewards` to `claimableRewardsStored` . . . 7
  - 3.4.2 Rewards accumulated during periods where stake is zero end up unclaimable . . . 8
  - 3.4.3 Unnecessary Admin Reward Snapshot . . . 9
  - 3.4.4 Use of call Instead of staticcall in Interface Validation . . . 9
  - 3.4.5 Unnatural Interface Validation Logic . . . 9

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

The Sablier Protocol is a collection of persistent, non-upgradeable smart contracts to facilitate streaming of ERC-20 tokens on Ethereum and other EVM blockchains.

From Sep 29th to Oct 2nd the Cantina team conducted a review of staking on commit hash 7db81edd. The team identified a total of **11** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	4	4	0
Gas Optimizations	1	1	0
Informational	5	4	1
Total	**11**	**10**	**1**

The Cantina Managed team reviewed Sablier's `staking` holistically on commit hash c9e9bdc8 and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Medium Risk

3.1.1 `LockupNFT` owner can make stream uncancellable by sending it directly to staking contract

Severity: Medium Risk

Context: SablierStaking.sol#L370-L406

Description: Cancellable streams allow the sender to be able to recover the unstreamed amount. When an NFT is put to staking in the `SablierStaking` contract, a hook makes it possible to update the staking balance in case the sender cancels the stream:

SablierStaking.sol#L370-L406:

```
function onSablierLockupCancel(
    uint256 streamId,
    address, /* sender */
    uint128 senderAmount,
    uint128 /* recipientAmount */
)
    external
    override
    noDelegateCall
    returns (bytes4)
{
    // Cast `msg.sender` as the Lockup NFT.
    ISablierLockupNFT msgSenderAsLockup = ISablierLockupNFT(msg.sender);

    // Get the Pool ID in which the stream ID is staked.
    uint256 poolId = _streamsLookup[msgSenderAsLockup][streamId].poolId;

    // Check: the Pool ID is not zero.
    if (poolId == 0) {
        revert Errors.SablierStaking_StreamNotStaked(msgSenderAsLockup, streamId);
    }

    // Load the storage in memory.
    address owner = _streamsLookup[msgSenderAsLockup][streamId].owner;
    uint128 streamAmountStaked = _userAccounts[owner][poolId].streamAmountStaked;

    // Checks and Effects: unstake the `senderAmount` for the user.
    _unstake({ poolId: poolId, unstakeAmount: senderAmount, maxAllowed: streamAmountStaked, user: owner });

    // Safe to use `unchecked` because `_unstake` verifies that `senderAmount` does not exceed
    ↪  `streamAmountStaked`.
    unchecked {
        // Effect: decrease the user's stream amount staked.
        _userAccounts[owner][poolId].streamAmountStaked = streamAmountStaked - senderAmount;
    }

    return ISablierLockupRecipient.onSablierLockupCancel.selector;
}
```

Unfortunately, we notice that when the NFT is owned by the staking contract, but not staked (`_streamsLookup[msgSenderAsLockup][streamId].poolId == 0`), the cancellation hook reverts:

```
// Check: the Pool ID is not zero.
if (poolId == 0) {
    //@audit hook reverts, sender is not able to cancel
    revert Errors.SablierStaking_StreamNotStaked(msgSenderAsLockup, streamId);
}
```

The stream receiver can use this to DOS the cancellation of the sender, effectively preventing him to recover the remaining funds.

Concrete scenario:

- Jan 1st, Alice sends a stream of 1_000_000 USDC to be received by Bob over the course of 100 days. Bob receives the LockupNFT associated with it.
- On day 30, Alice decides to cancel the stream and recover the remaining 700_000 USDC.

1. Alice sends her cancel transaction.
2. Bob sees Alice's transaction, and front-runs it by sending a transaction to:
   - Withdraw the streamed balance from the Lockup.
   - Send the Lockup NFT directly to the staking contract.

As a result, the cancellation initiated by Alice reverts, and Alice is unable to recover the remaining funds.

Recommendation: It should be safe to early return instead of reverting if the NFT is not staked:

SablierStaking.sol#L388-L390:

```
  if (poolId == 0) {
-     revert Errors.SablierStaking_StreamNotStaked(msgSenderAsLockup, streamId);
+     return ISablierLockupRecipient.onSablierLockupCancel.selector;
  }
```

Sablier: Fixed in PR 48.

Cantina Managed: Fix verified.

3.2 Low Risk

3.2.1 Pool admin who controls reward token can brick stakers funds by sending excessively high rewards

Severity: Low Risk

Context: DataTypes.sol#L26-L41

Description: Even though individual reward balances or transfers are limited to `type(uint128).max` in size, ultimately the accumulation of rewards in a pool is represented by the global `uint256` index `snapshotRptDistributedScaled`:

DataTypes.sol#L26-L41:

```
struct Pool {
    // Slot 0
    address admin;
    uint40 endTime;
    uint40 startTime;
    // Slot 1
    IERC20 rewardToken;
    uint40 snapshotTime;
    // Slot 2
    IERC20 stakingToken;
    // Slot 3
    uint128 rewardAmount;
    uint128 totalStakedAmount;
    //@audit tracks the cumulated rewards per unit of staking token
    // Slot 4
    uint256 snapshotRptDistributedScaled;
}
```

This means that when a malicious pool admin controls the supply/minting of the reward token for a pool, he can use multiple rounds of rewards to brick the funds of the users staking in his pool.

Concrete scenario:

- Initial state:
  - PoolId = 1.
  - Let's consider `initialTs == block.timestamp == pool.endTime`, and only two stakers Alice and Bob have 60% and 40% of the pool stake respectively.

  Pool admin Charlie distributes `type(uint128).max` between `startTime = initialTs` and `endTime = initialTs + 1`, such that on the next block he can again configure a round (assuming ethereum mainnet 12 second slots), and send `type(uint128).max` of reward token between `startTime = initialTs + 12` and `endTime = initialTs + 13`.

As a result Bob can now claim more than `type(uint128).max` of reward token, and any call to `_snapshotRewards(1, Bob)` will revert:

SablierStaking.sol#L764-L765:

```
// Scale down the rewards earned by the user since the last snapshot.
//@audit toUint128() reverts if out of bounds
uint128 userRewardsSinceLastSnapshot = userRewardsSinceLastSnapshotScaled.scaleDown().toUint128();
```

Recommendation: Either describe the trust assumption on the rewarder in documentation, or add a `emergencyWithdraw` function which would enable to forfeit the rewards and just recover the funds.

Sablier: This could have also blocked the sender from cancelling the stream if the recipient creates a pool with a dummy token that he controls and let the calculations overflow. Fixed in PR 48.

Cantina Managed: Fixed by adding a `uint128` field `cumulativeRewardAmount`, which is used to check that any configureNextRound call does not go above `type(uint128).max` in cumulated rewards.

3.2.2 Ambiguity in Pool Start Time Initialization

Severity: Low Risk

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

Description: In the `SablierStaking` contract, the pool's `startTime` parameter is used to determine when rewards begin accruing. However, due to the unpredictable nature of block mining, it can be difficult for users to set an exact start time that matches the intended "now" moment. If a user wishes to start a pool immediately, setting `startTime` to zero could be interpreted as a request to use the current block timestamp. Without explicit handling, this may lead to confusion or require users to estimate the mining delay, potentially resulting in rewards accruing before any stake is present in the pool which will lead to lost rewards as no stake will be made yet.

Recommendation: It is recommended to treat a `startTime` value of zero as an instruction to use the current block timestamp, by adding a check such as:

```
if (startTime == 0) {
    startTime = uint40(block.timestamp);
}
```

This adjustment improves user experience by allowing immediate pool creation without requiring precise timestamp estimation. However, it is still advisable to encourage users to set a start time in the future to avoid scenarios where rewards begin accruing while the pool's total stake is zero, which can lead to unallocated or wasted rewards. Documentation and UI guidance should reinforce this best practice.

Sablier: Fixed in PR 50.

Cantina Managed: A warning will be shown on the UI regarding the lost of rewards until the first deposit for users wanting to use a "now" timestamp.

3.2.3 Precision Loss in Reward Distribution Calculation

Severity: Low Risk

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

Description: In the `SablierStaking` contract, the `_rewardsDistributedSinceLastSnapshot` function calculates distributed rewards using integer division: `rewardsDistributed = (pool.rewardAmount * elapsedTime) / rewardsPeriod;`. This approach can result in significant precision loss, particularly when the elapsed time is short, the reward period is long, or the reward amount is small. The lack of scaling at this stage means that fractional rewards are truncated early, and this effect is compounded for tokens with high value but low decimal granularity. Relying on downstream scaling in `_rptDistributedScaled` does not recover the lost precision, leading to under-distribution of rewards and potential user dissatisfaction.

Recommendation: It is recommended to apply fixed-point scaling within the reward distribution calculation itself, renaming the function to `_rewardsDistributedSinceLastSnapshotScaled` to reflect this change. All dependent view functions should be updated to expect and handle scaled values, only scaling down for user-facing outputs. This adjustment will ensure that even for small or high-value tokens, the reward distribution remains accurate and fair, minimizing cumulative rounding errors.

Sablier: Fixed in PR 48.

Cantina Managed: Fix verified.

3.2.4 Premature Scaling Down

Severity: Low Risk

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

Description: In the `SablierStaking` contract, the `_snapshotUserRewards` function currently scales down the calculated rewards earned since the last snapshot before updating the user's `snapshotRewards`. This approach reduces the precision of the rewards at every snapshot, which can lead to cumulative rounding errors over time, especially for users who frequently stake and unstake. By scaling down at this stage, the contract loses the benefits of fixed-point arithmetic, which is designed to preserve accuracy throughout internal calculations. This design choice may result in users receiving slightly less than their fair share of rewards due to repeated truncation.

Recommendation: It is advisable to store `snapshotRewards` in its scaled (high-precision) form throughout the contract's internal logic and only perform the scale-down operation at the point of claiming rewards. This approach maintains maximum precision during all intermediate calculations and only introduces rounding at the final step, ensuring users receive the most accurate reward amounts possible. For user interfaces or external queries, a dedicated view function can be provided to return the scaled-down value for display purposes, ensuring that the UI remains user-friendly without sacrificing internal accuracy.

Sablier: Fixed in PR 51.

Cantina Managed: Fix verified.

3.3 Gas Optimization

3.3.1 Incorrect Reward Distribution at Pool Start Time

Severity: Gas Optimization

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

Description: In the `SablierStaking` contract, within the `_rewardsDistributedSinceLastSnapshot` function, the check `if (blockTimestamp < poolStartTime)` is used to determine if the current block timestamp is before the pool's start time. However, this condition does not account for the scenario where `block.timestamp` is exactly equal to `poolStartTime`. In such a case, the elapsed time is zero, and no rewards should be distributed, but the function will proceed with the calculation, potentially leading to an unnecessary computation or confusion about reward accrual at the exact start time. This could result in edge cases where users expect no rewards to be distributed at the pool's start, but the function does not explicitly enforce this, possibly leading to inconsistencies in reward calculations.

Recommendation: It is advisable to update the conditional check to `if (blockTimestamp <= poolStartTime)` to ensure that no rewards are distributed when the current block timestamp is less than or exactly equal to the pool's start time.

Sablier: Fixed in PR 49.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Suggestion to rename `snapshotRewards` to `claimableRewardsStored`

Severity: Informational

Context: DataTypes.sol#L72-L80, SablierStaking.sol#L187-L188

Description/Recommendation: The naming `snapshotRewards` in UserAccount is a bit misleading because it is being reset at each claim:

DataTypes.sol#L72-L80:

```
struct UserAccount {
    // Slot 0
    uint128 directAmountStaked;
    uint128 streamAmountStaked;
    // Slot 1
    uint128 snapshotRewards;
    // Slot 2
    uint256 snapshotRptEarnedScaled;
}
```

SablierStaking.sol#L187-L188:

```
// Effect: set the rewards to 0.
_userAccounts[msg.sender][poolId].snapshotRewards = 0;
```

It could be more meaningful to rename the variable to `claimableRewardsStored`.

Sablier: Fixed in PR 51.

Cantina Managed: Fix verified.

3.4.2 Rewards accumulated during periods where stake is zero end up unclaimable

Severity: Informational

Context: SablierStaking.sol#L660-L667, SablierStaking.sol#L728-L737

Description: When `pool.totalStakedAmount == 0`, the reward calculation in `_rewardsDistributedSinceLastSnapshot` always returns zero:

SablierStaking.sol#L660-L667:

```
function _rewardsDistributedSinceLastSnapshot(uint256 poolId) private view returns (uint256 rewardsDistributed)
↪  {
    // Read the pool from storage using storage pointer.
    Pool storage pool = _pools[poolId];

    // If the total staked amount is 0, return 0.
    if (pool.totalStakedAmount == 0) {
        //@audit early return when pool.totalStakedAmount == 0
        return 0;
    }
```

But since the `pool.snapshotTime` is always updated:

SablierStaking.sol#L728-L737:

```
function _snapshotGlobalRewards(uint256 poolId) private returns (uint256 rptDistributedScaled) {
    // Get the latest value of the cumulative rewards distributed per ERC20 token.
    rptDistributedScaled = _rptDistributedScaled(poolId);

    // Effect: snapshot the rewards distributed per ERC20 token.
    _pools[poolId].snapshotRptDistributedScaled = rptDistributedScaled;

    // Effect: update the snapshot time.
    _pools[poolId].snapshotTime = uint40(block.timestamp);
}
```

This means that the rewards allocated for the period are unrecoverable.

> It is unlikely that the pool goes back to having zero stake after first deposit, but this is something to be mindful of when creating a new pool. If the pool remains empty after the startTime has been reached, rewards accrued are lost.

Recommendation: Document this behavior, especially for the pool creation case.

Sablier: Acknowledged. We will not take any action. Since this would be an extremely rare scenario, the cost of handling it can not be justified. We will document it in the UI during the pool creation flow as well as encourage pool admin to stake 1 wei as a stopgap solution.

Cantina Managed: Acknowledged.

3.4.3 Unnecessary Admin Reward Snapshot

Severity: Informational

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

Description: In the `SablierStaking` contract, the `configureNextRound` function performs a snapshot of both global and admin rewards if the admin has a nonzero stake. However, this snapshot may be redundant. The global snapshot is already sufficient to record the state of rewards at the end of a round, and since the admin's stake is not being modified during this operation, their rewards can be accurately computed the next time they interact (such as when depositing or claiming). The monotonically increasing nature of the global snapshot, combined with the restriction that a new round cannot start until the previous one ends, ensures that the admin's rewards are always up to date without needing a separate snapshot at this point. Including an admin-specific snapshot here introduces unnecessary complexity and may confuse future maintainers about the intended reward accounting logic.

Recommendation: It is recommended to remove the user-specific snapshot for the admin in `configureNextRound` and rely solely on the global snapshot. The admin's rewards will be correctly updated the next time they interact with the contract, maintaining accurate accounting without redundant state changes. If there is a specific edge case that requires the admin snapshot, it should be clearly documented. It is also recommended to thoroughly test that this change does not introduce any edge cases and that admin staking values remain consistent and correct after the modification, which should be as there is no difference between an admin stake and an normal user stake.

Sablier: Fixed in PR 48.

Cantina Managed: Fix verified.

3.4.4 Use of call Instead of staticcall in Interface Validation

Severity: Informational

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

Description: In the `SablierStaking` contract, the `_hasRequiredInterface` function currently uses low-level `call` to check for the existence of required view functions in the lockup contract. While these functions are intended to be read-only, using `call` does not enforce this at the EVM level, and a malicious or misconfigured contract could potentially alter state during the check. This could lead to unintended side effects or security issues, especially if the lockup contract is not fully trusted or if future upgrades introduce state-changing logic in these functions.

Recommendation: It is advisable to replace the use of `call` with `staticcall` when verifying the presence of view or pure functions in the lockup contract. `staticcall` enforces read-only execution at the EVM level, ensuring that no state changes can occur during the interface check.

Sablier: Fixed in PR 48.

Cantina Managed: Fix verified.

3.4.5 Unnatural Interface Validation Logic

Severity: Informational

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

Description: Within the `SablierStaking` contract, the `_hasRequiredInterface` function is responsible for verifying that a given lockup contract implements the required methods from the `ISablierLockupNFT` interface. The current implementation loops through all five selectors, but the logic is somewhat convoluted: it checks the first three selectors (which are mandatory) and then, for the last two (`getUnderlyingToken` and `getAsset`), it uses a break and a conditional revert. This approach, while functional, is not intuitive and may reduce code clarity and maintainability. The logic could be more clearly separated between the mandatory selectors and the optional/fallback ones, making the intent and error handling more explicit for future auditors and developers.

Recommendation: It is recommended to refactor the `_hasRequiredInterface` function by first looping through the three mandatory selectors and reverting immediately if any are missing. For the last two selectors, which serve as alternatives, a separate check can be performed: attempt to call `getUnderlyingToken`, and if it fails, then attempt `getAsset`, reverting only if both are missing. This can be achieved with a conditional structure or a try/catch block if using Solidity 0.8.x and above. This refactoring will improve code readability and make the interface validation logic more natural and easier to audit or extend in the future.

Sablier: Fixed in PR 48.

Cantina Managed: Fix verified.