Sablier Lockup 2.0.0

Security Review

Cantina Managed review by:

Eric Wang, Lead Security Researcher
RustyRabbit, Security Researcher

January 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	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

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 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

Sablier is a token streaming protocol available on Ethereum, Optimism, Arbitrum, Polygon, Avalanche, and BSC. It's the first of its kind to have ever been built in crypto, tracing its origins back to 2019. Similar to how you can stream a movie on Netflix or a song on Spotify, so you can stream tokens by the second on Sablier.

From Dec 17th to Jan 6th the Cantina team conducted a review of lockup on commit hash b3ea1222. The team identified a total of **9** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	2	2	0
Gas Optimizations	0	0	0
Informational	7	5	2
Total	**9**	**7**	**2**

3 Findings

3.1 Low Risk

3.1.1 `_calculateStreamedAmount()` does not include start amount at start of stream

Severity: Low Risk

Context: SablierLockup.sol#L305-L306

Description: `_calculateStreamedAmount()` is used to calculate the streamed amount at the current time. In the current form however at the start of the stream it returns 0 instead of including the start amount if set.

This can be significant when claiming an airdrop and in the same block using `withdrawMaxAndTransfer` in the same block or transaction. The intent would be to withdraw the start amount and transfer the stream to another address but as a result the complete stream including the start amount would be transferred to the new address.

Recommendation: Replace the if statement as follows:

```
-  if (timestamps.start >= blockTimestamp)
+  if (timestamps.start > blockTimestamp)
```

This will make sure the correct calculation will occur for the stream including any unlock amount at the start.

Sablier: Fixed in PR 1131.

Cantina Managed: Verified.

3.1.2 Integer overflow allows the creation of linear streams with an invalid cliff and end time

Severity: Low Risk

Context: SablierLockup.sol#L190

Description: In the `createWithDurationsLL()` function of `SablierLockup`, the `cliffTime` of a linear stream is calculated as the sum of `timestamps.start` and `durations.cliff`, which are both of type `uint40`. In an edge case where the sum overflows to 0 due to unchecked math, e.g.,

- `timestamps.start = 1734481466`.
- `durations.cliff = type(uint40).max - 1734481466 + 1`.
- `durations.total = 1000`.

The creation of the stream will still succeed. The stream will have a duration of 1000 seconds and without a cliff. However, strictly speaking, the inputs are invalid, and the stream needs to be rejected since the cliff time is greater than the end time.

Recommendation: Consider removing the `unchecked` box when calculating the cliff and end time of the stream.

Sablier: Fixed in PR 1132.

Cantina Managed: Verified.

3.2 Informational

3.2.1 `withdrawMultiple()` not indicating failed withdrawals can be problematic for calling contracts

Severity: Informational

Context: SablierLockupBase.sol#L510-L518

Description: `withdrawMultiple()` does not return the status of each individual `withdraw()` to the caller, it only emits an event. Therfore the calling contract does not know if there are any failing withdrawals. This

includes any contract allowed to hook as calls initiated from such a contract directly will not have there hook called.

As such the calling contract has no idea about the total amount actually withdrawn and will have to resort to other tactics to determine the exact withdrawn amount and still not know which `withdraw` failed and a such still have to revert itself if that information is required for internal accounting.

Recommendation: Consider returning an array of booleans indicating the status of each individual `withdraw` to the caller.

Sablier: We have decided to prioritize minimizing gas costs for the `withdrawMultiple` function over returning an array. This function is primarily intended for end users rather than integrators. If an integrator requires a similar function with return value, we will recommend them to create their own implementation tailored to their specific needs. Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 `batch()` does not indicate return values of the underlying batch calls

Severity: Informational

Context: Batch.sol#L17-L25

Description: The `batch()` function allows users and contracts to execute a list of arbitrary functions in a single transaction. Whenever one of these underlying calls reverts it will be caught and the complete batch is reverted indicating a general error.

However when the calls successful and do not revert the return information of the underlying calls is not returned to the caller. This includes the `streamId's` when creating streams, return values of getter functions but also withdrawn amounts during withdrawals (`withdrawMax()`, `withdrawMaxAndTransfer()`, `withdrawMultiple()`).

As such this limits the usefulness of the `batch()` function.

Recommendation: Consider returning the return values of the underlying batch calls as an array of bytes. Alternatively consider using OZ's `Multicall`.

Sablier: We have updated the batch function to return an array of values from the calls and to revert with the underlying error in PR 1126.

Cantina Managed: Verified.

3.2.3 Library functions can take the timestamp as parameter rather than from the block

Severity: Informational

Context: Helpers.sol#L16-L25

Description: Several of the functions in the `Helper` and `VestingMath` libraries take the current timestamp from the block to use it as start time or current time. This unnecessarily restricts the use of the library.

Recommendation: Consider passing the time as a parameter of the function and let the caller determine the time it wishes to use. This has the benefit of making the libraries pure and allows for other use cases where the timestamp is in the future or the past.

Sablier: Fixed in PR 1132.

Cantina Managed: Verified.

3.2.4 NatSpec and various improvements

Severity: Informational

Context: ISablierLockupBase.sol#L249, ISablierLockupBase.sol#L277, ISablierLockupBase.sol#L327, ISablierLockup.sol#L118, ISablierLockup.sol#L171, ISablierLockup.sol#L207, ISablierLockup.sol#L241, VestingMath.sol#L31-L36, VestingMath.sol#L103, VestingMath.sol#L180, SablierLockup.sol#L185-L187

Description:

1. VestingMath.sol#L103: Replace

```
- /// f(x) = x * sa + s + c
+ ///
+ ///        ( x * sa + s,  cliffTime > block.timestamp
+ /// f(x) = (
+ ///        ( x * sa + s + c, cliffTime <= block.timestamp
```

And add the following assumptions to the `calculateLockupLinearStreamedAmount()` function's NatSpec:

```
+ 1. unlockAmounts.start + unlockAmounts.cliff does not overflow uint128
+ 2. timestamps.start < timestamps.end and timestamps.start <= block.timestamp <= timestamps.end
+ 3. If cliffTime > 0, timestamps.start < cliffTime < timestamps.end
```

2. VestingMath.sol#L31-L36: Add the following assumptions to the `calculateLockupDynamicStreamedAmount()` function's NatSpec:

```
+ 1. The sum of all segment amounts does not overflow uint128.
+ 2. The segment timestamps are ordered chronologically.
+ 3. There are no duplicate segment timestamps.
```

   VestingMath.sol#L180: Add the same set of assumptions to the `calculateLockupTranchedStreamedAmount()` function's NatSpec (replace `segment` with `tranche`).

3. SablierLockup.sol#L185-L187: The `_createLL()` function checks that the cliff time is strictly greater than the start time (see Helpers.sol#L206-L209). Update the comment to:

```
- // is greater than or equal to the start time.
+ // is greater than the start time.
```

4. ISablierLockup.sol#L118: As the meaning of `cliffTime` has changed from "the time until the stream is suppressed" to "the time at which the unlock amount is released", clearly document the caller/integrator is now responsible for calculating the amount that will be unlocked at the `cliffTime`.

5. ISablierLockup.sol#L171: Add the requirement that `params.shape.length` must not be greater than 32 characters. Same for:
   - ISablierLockup.sol#L207.
   - ISablierLockup.sol#L241.

6. ISablierLockupBase.sol#L249: To be more precise, update the comment to:

```
/// - This function attempts to invoke a hook on the recipient, if the resolved address is on the
↪  allowlist.
```

   ISablierLockupBase.sol#L327 can also be updated to:

```
/// - This function attempts to call a hook on the recipient of the stream, unless msg.sender is the
↪  recipient or the recipient is not on the allowlist.
```

7. ISablierLockupBase.sol#L277: Change `ETH` to `native tokens` to avoid confusion as the contracts may be deployed on some other EVM chains.

Sablier: Fixed in PR 1131 and PR 1136.

Cantina Managed: Verified.

3.2.5 Handling empty arrays by reverting with an explicit custom error

Severity: Informational

Context: Helpers.sol#L31, Helpers.sol#L64, SablierBatchLockup.sol#L123-L125, SablierBatchLockup.sol#L338-L340

Description: An array out-of-bounds access error may be triggered in the following code:

1. In the `SablierLockup.createWithDurationsLT()` function, if the input `tranchesWithDuration` is an empty array, `Helpers.calculateTrancheTimestamps()` will cause an out-of-bounds

access error when accessing `tranchesWithTimestamps[0]`. The same issue also exists in `Helpers.calculateSegmentTimestamps()`.

2. In the `createWithTimestampsLD()` and `createWithTimestampsLT()` functions of the `SablierBatchLockup` contract, since `batch[i].segments` (or `batch[i].tranches`) can be an empty array, the calculation of its `length - 1` may overflow and cause an out-of-bounds access error.

Recommendation: Consider reverting with a custom error in the empty array cases for better error handling and debugging purposes.

Sablier: Thank you for your submission. We agree that the lack of array length validation can result in an out-of-bounds access error. However, since the transaction will ultimately revert and the user will be unable to create a stream, we have opted not to include array length validation and keep the gas cost as it is. Acknowledged.

Cantina Managed: Acknowledged.

3.2.6 Notes on emitting the `MetadataUpdate` event

Severity: Informational

Context: SablierLockupBase.sol#L361-L362, SablierLockupBase.sol#L687-L688, ISablierLockupBase.sol#L231, ISablierLockupBase.sol#L359, ISablierLockup.sol#L85, ISablierLockup.sol#L106, ISablierLockup.sol#L129, ISablierLockup.sol#L149, ISablierLockup.sol#L182, ISablierLockup.sol#L219

Description:

1. Since renouncing a stream does not update its metadata, the `MetadataUpdate` event can be omitted. For the same reason, emitting the event is not necessary when transferring the stream from one user to another.
    - SablierLockupBase.sol#L361-L362.
    - SablierLockupBase.sol#L687-L688.
2. Add a comment that a `MetadataUpdate` event will be emitted in the following functions:
    - ISablierLockup.sol#L85.
    - ISablierLockup.sol#L106.
    - ISablierLockup.sol#L129.
    - ISablierLockup.sol#L149.
    - ISablierLockup.sol#L182.
    - ISablierLockup.sol#L219.
    - ISablierLockupBase.sol#L231.
    - ISablierLockupBase.sol#L359.

Recommendation: Consider modifying the code as suggested above.

Sablier: While the recipient address per se is not included in the NFT metadata, it would be helpful to update the metadata when the NFT changes hands so that the new recipient can see the most up-to-date information on marketplaces like OpenSea. This fix is available in PR 1132.

Cantina Managed: Verified.

3.2.7 Potential integer overflow when calculating the streamed percentage of a stream

Severity: Informational

Context: LockupNFTDescriptor.sol#L201-L204

Description: In the `calculateStreamedPercentage()` function of `LockupNFTDescriptor`, an overflow is possible when calculating `streamedAmount * 10_000` at L203. Solidity will consider the constant `10_000` as of type `uint16`, so the multiplication result will be of type `uint128`. As a result, an overflow will occur if the multiplication result exceeds `type(uint128).max`, causing incorrect results of the streamed percentage.

Recommendation: Consider removing the unchecked block to avoid integer overflows.

Sablier: Another solution would be to explicitly wrap `streamedAmount` into `uint256`, like so:

```
return uint256(streamedAmount) * 10_000 / depositedAmount;
```

Fixed in PR 1131.

Cantina Managed: Verified.