Sablier Lockup
Security Review

Cantina Managed review by:

0xRajeev, Lead Security Researcher
Rustyrabbit, Security Researcher

July 28, 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 Low Risk . . . 4
  - 3.1.1 Malicious hook contracts can prevent lockup stream withdrawals/cancellations causing fund loss to recipient/sender . . . 4
  - 3.1.2 User funds can be accidentally withdrawn as fees if `setNativeToken()` is not called timely . . . 4
- 3.2 Informational . . . 5
  - 3.2.1 Missing `ComptrollerManager` convenience function to query required fee . . . 5

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

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. It consists of Lockup, Merkle Airdrops, and Flow.

From Jun 19th to Jul 3rd the Cantina team conducted a review of lockup on commit hash 6f49eb84. All contracts under `src` (except `interfaces`) are in scope. The team identified a total of **3** 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	0	2
Gas Optimizations	0	0	0
Informational	1	1	0
Total	**3**	**1**	**2**

The Cantina Managed team reviewed Sablier's `lockup` holistically on commit hash 6341f0f4 and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Malicious hook contracts can prevent lockup stream withdrawals/cancellations causing fund loss to recipient/sender

Severity: Low Risk

Context: SablierLockup.sol#L148-L160, SablierLockup.sol#L349-L355, SablierLockup.sol#L630-L637

Summary: Malicious hook contracts can prevent lockup stream withdrawals and cancellations by reverting the callbacks causing fund loss to recipient/sender.

Finding Description: Lockup streams have a feature of `_allowedToHook[recipient]`, where recipient contracts are whitelisted by `SablierComptroller` to hook to Sablier when a stream is canceled or when tokens are withdrawn.

From the documentation:

> Hooks are arbitrary third-party functions that get automatically executed by the Sablier Protocol in response to cancel and withdraw events. They are similar to callback functions in web2.

> Hooks are a powerful feature that enable Sablier streams to interact with other DeFi protocols. Let's consider an example: You own a Sablier stream that expires in two years. You are interested into taking a loan against it with the intention to pay it all back after it expires. Hooks are what enable you to do that. With the help of Hooks, we can create an ecosystem of varied use cases for Sablier streams. This can range from lending, staking, credit, and more.

> It is worth noting that once a hook has been allowlisted, it can never be removed. This is to ensure stronger immutability and decentralization guarantees. Once a recipient contract is allowlisted, integrators do NOT have to trust us to keep their contract on the allowlist.

Given that hook allowlisting is irreversible, the documentation mentions a checklist of hook requirements such as non-upgradeability and required interfaces, which are expected to be "audited by a third-party security researcher".

Impact Explanation: Medium, because if an allowlisted hook is malicious/compromised (missed by audit) then streams integrated with that hook can be blocked from withdrawals and cancellations by simply reverting on their callbacks to cause fund loss to recipient/sender or other unexpected behavior depending on the nature of integration.

Likelihood Explanation: Very low, because the original recipient must have transferred stream ownership to the hooked contract and hook allowlisting is expected to be performed by protocol admin (`SablierComptroller`) after due-diligence by both the project and with consideration of an external security review.

Recommendation:

1. Ensure that the vetting of hooks is performed with great internal and external due-diligence.
2. Prepare an incident response playbook for if/when a hook is compromised.

Sablier: We agree that if the hook is malicious then it indeed would be problematic. As a response to this, we will perform extensive due diligence before registering any contract on the hook allow list. And the incident response playbook is certainly a good idea.

Cantina Managed: Acknowledged.

3.1.2 User funds can be accidentally withdrawn as fees if `setNativeToken()` is not called timely

Severity: Low Risk

Context: SablierFlow.sol#L458-L474

Summary: `setNativeToken` should be called as soon as possible either immediately at deployment or whenever the native ERC20 contract is created (e.g. a network upgrade). Otherwise, a user can create streams with the native token and user funds can then be accidentally withdrawn as fees.

Finding Description: When a chain has an ERC20 representation for its native token `setNativeToken()` allows the controller to set it as native token so that no one can create streams with that token, because doing so would mean that fee withdrawal would end up withdrawing user funds instead of only the fees.

The native token is not set in the constructor and is meant to be called during deployment (although the provided scripts do not do so). The function is also necessary when a chain would implement the ERC20 representation of its native token in a fork or upgrade.

However, if anyone creates a stream with the token between the deployment of the Flow and Lockup contracts and before `setNativeToken()` is called, then their funds will be withdrawn as fees whenever `transferFeesToComptroller()` is called.

Impact Explanation: Medium, because user funds of only the streams created with the native token would be withdrawn as fees. Those streams would be defunct until the correct balance of those streams is returned to the contract at which time the sender can cancel the streams to ensure sender and recipient each receive the correct balance.

Likelihood Explanation: Low, because this only can happen when the native token is not set in a timely manner.

Recommendation: Consider setting the native token in the constructor of `ComptrollerManager` (by querying the Comptroller) or ensure the deployment script sets the native token in the same transaction that the contract is deployed.

Additionally, consider letting the comptroller cancel streams that incidentally have been created with the native token so funds are distributed correctly.

Sablier: We will ensure that native token is set at the time of the deployment.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Missing `ComptrollerManager` convenience function to query required fee

Severity: Informational

Context: SablierMerkleBase.sol#L245-L253, SablierFlow.sol#L950-L956, SablierLockup.sol#L648-L654

Description: Withdrawals in lockup/flow and claims in airdrops require callers to pay a minimum protocol fee as determined by `comptroller.calculateFlowMinFeeWeiFor()`. However, there is no convenience function for users interacting with those contracts to directly query the required fee for successful withdrawals/claims.

Recommendation: Consider adding a `ComptrollerManager` convenience function to query required fee.

Sablier: Fixed in PR 1270, PR 454 and PR 172.

Cantina Managed: Reviewed that PR 1270, PR 454 and PR 172 add a convenience `calculateMinFeeWei()` function to lockup, flow and airdrop.