Sablier Finance

Security Review

Cantina Managed review by:
RustyRabbit, Security Researcher

September 6, 2023

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	*Directly* exploitable security vulnerabilities that need to be fixed.
High	Security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All high issues should be addressed.
Medium	Objective in nature but are not security vulnerabilities. Should be addressed unless there is a clear reason not to.
Low	Subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
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. When determining the severity one first needs to determine whether the finding is subjective or objective. All subjective findings are considered of Minor severity.

Next it is determined whether the finding can be regarded as a security vulnerability. Some findings might be objective improvements that need to be fixed, but do not impact the project's security overall (Medium).

Finally, objective findings of security vulnerabilities are classified as either critical or major. Critical findings should be directly vulnerable and have a high likelihood of being exploited. Major findings on the other hand may require specific conditions that need to be met before the vulnerability becomes exploitable.

2 Security Review Summary

Sablier is a token streaming protocol available on Ethereum, Optimism, Arbitrum, Polygon, Ronin, 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 August 23rd to August 24th the Cantina team conducted a review of PR 160 and PR 161. The team identified a total of **5** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 3
- Gas Optimizations: 0
- Informational: 2

3 Findings

3.1 Low Risk

3.1.1 Identical functions for permit2 and Approve proxy target could lead to leakage of unused permit2 signatures.

Severity: Low Risk

Context: /src/abstracts/SablierV2ProxyTarget.sol#L175-L370

Description: Both Approve and Permit2 target contracts are based on the same `SablierV2ProxyTarget` contract using the exact same functions for creating streams. The Approve version of the target contract simply disregards the `transferData` calldata.

This opens up the risk of a user accidentally using the Approve target contract when the intent was to use the Permit2 version. If there is enough allowance the call will succeed, the provided Permit2 signature will still be valid and could be used to create another stream. This still requires a malicious or compromised envoy though, so the risk is rather low and the Permit2 expiration further mitigates the risk.

Recommendation: Consider having separate functions for Permit2 and Approve target contracts so it's more clear to the user if they need the Permit2 signature or not.

Sablier: Acknowledged. Having separate functions for Permit2 and Approve would introduce quite a bit of boilerplate. It is also worth noting that we are considering moving away from the PRBProxy architecture for Safe users.

Cantina: Acknowledged.

3.1.2 Approvals are coarse grained authorization for all target and plugin contracts

Severity: Low Risk

Context: /src/SablierV2ProxyTargetApprove.sol#L11-L35

Description: Token approvals are set for the proxy as a whole. This means if there are active token approvals any target contract or installed plugin is able to use the allowance for other purposes than the intended creation of streams. This of course depends on the functionality provided by the target and plugin contract as well as the authorizations set for the envoys.

Recommendation: On the longer term consider separating funds in per proxy vault type contract (for allowances or push funds) with more fine-grained authorization as an option to users wishing more control over which envoy or plugin can use those funds and for which purpose. We do realize however this would require invasive change to the current code. As a short term alternative a push model could mitigate the risks associated with excessive approvals.

Sablier: Acknowledged. Implementing a per-proxy vault system would be non-trivial, so we will not build it. It is also worth noting that we are considering moving away from the PRBProxy architecture for Safe users. Will follow up on the push model separately, in *"Introduction of approvals use case opens up attack surface w.r.t. long term allowances"*

Cantina: Acknowledged.

3.1.3 Introduction of approvals use case opens up attack surface w.r.t. long term allowances

Severity: Low Risk

Context: /src/SablierV2ProxyTargetApprove.sol#L11-L35

Description: PR 161 introduces the use of token approval/allowances for the proxy contract to create streams rather than use explicit Permit2 signatures. The intended use case is for Safe users to avoid having to let every cosigner need to sign the permit2 offline sig and use a Safe multi-send to batch an approve on the ERC20 token and create the stream in the same transaction.

However, some non-Safe users may be inclined to use approvals over the permit2 use case going even so far as to use infinite approvals to save gas and to ease the UX.

The risk is that remaining balances on token approvals (or max approvals) can get abused by rogue envoys (or compromised contracts of those are set as envoys) to create streams to arbitrary receivers.

This wasn't the case with permit2 signatures as each time a stream was created a new permit2 signature was needed from the owner. Additionally, any permit2 signature was short-lived due to the expiration time and typically used the exact amount needed.

Recommendation: Consider using a push model rather than a pull model as the intended use case needs a preceding (approval) transaction on the token which could be replaced by a `transfer()` transaction. Although this doesn't solve the issue at it's core, balances are more easily to query compared to approvals (watch only address in wallets) and users are less likely to actually send more funds than needed.

Sablier: This is an interesting suggestion. Am I right that the proposed flow would be something like this?

1. Have the user make a standard `transfer` to the proxy
2. Call the updated proxy target
3. Expect that the proxy has sufficient tokens to cover the `totalAmount`, otherwise, let the ERC-20 contract revert the transaction due to insufficient funding.

Separately, it is worth noting that we are considering moving away from the PRBProxy architecture for Safe users. And if we do this, there would be no risk of rogue envoys misusing the ERC-20 allowances for malicious purposes.

Cantina: Acknowledged.

3.2 Informational

3.2.1 Approve target contract test scripts sets approval to max

Severity: Informational

Context: /test/integration/Integration.t.sol#L45 /test/integration/target/TargetApprove.t.sol#L103-L111 /test/integration/target/cancel-and-create/cancelAndCreate.t.sol#L41-L68

Description: The test scripts of the Approve target contract sets the proxy's allowance to max. This simplifies testing but doesn't mimic real world conditions where the approval is most likely set each time a stream is created. More specifically the `cancelAndCreate` functions where users may have not set an approval before canceling the stream.

Recommendation: Consider adding success and fail scenarios for (in)sufficient allowances when canceling and recreating streams.

Sablier: Acknowledged. We consider the ERC-20 approval external to our system, implemented in each third-party ERC-20 contract. We don't find it helpful to perform one-time approvals before creating each stream because, in effect, we would stress-test the ERC-20 contract flow, not Sablier.

The max approval is enough proof that the ERC-20 flow can be coupled to Sablier, and that's all we care about.

Cantina: Acknowledged.

3.2.2 CancelAndRecreate functions require approval for full amount.

Severity: Informational

Context: /src/SablierV2ProxyTargetApprove.sol#L30 /src/abstracts/SablierV2ProxyTarget.sol#L297-L298

Description: The Approve target contract's `cancelAndCreate` functions first cancel the stream which ultimately send the funds back to the owner and then creates a new stream pulling funds from the owner into the proxy again to create the new stream. This requires the owner to approve the full amount of the newly created stream prior to calling the `cancelAndCreate` function which is counterintuitive and can be unnecessary in cases where the amount of the new stream matches (or is less than) the remaining part of the old one.

As the Approve target contract is mostly meant for Safe integration (batched calls) this is not really an issue but then setting an extra approval on what is meant to be net neutral is not ideal UX and might cause confusion between cosigners. Although no such tokens are known to exist some future token

might also not allow you to set an approval exceeding the current balance of the sender (which can be the case when canceling and recreating streams).

Recommendation: Consider altering the `cancelAndCreate` functions to not send the funds back to the owner before using them again to recreate the stream, only requiring an approval when the new amount exceeds the old amount.

Sablier: This is an interesting suggestion, but as you say, there is likely no significant impact today. We are happy to tolerate the slightly worse UX. I will thus mark this as "Acknowledged" and proceed to create a discussion in V2 Periphery to track this as an idea that we might choose to implement later on (proxy targets are upgradeable): v2-periphery/discussions/170

Cantina: Acknowledged.