CANTINA

Sablier

Security Review

Cantina Solo review by:
Zach Obront, Lead Security Researcher

December 21, 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 Solo 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 Solo 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 Solo 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 provides infrastructure for money streaming and token distribution. DAOs and businesses use Sablier for vesting, payroll, airdrops, and more. Recipients can withdraw and use the streamed funds at any time, without your continued involvement.

From Nov 10th to Nov 16th the Cantina Solo team conducted a review of v2-periphery and v2-core on commit hashes 0004fd2e and 2d19596e respectively. The team identified a total of **6** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 1
- Medium Risk: 2
- Low Risk: 2
- Gas Optimizations: 0
- Informational: 1

3 Findings

3.1 High Risk

3.1.1 `MerkleStreamer` deployment can be frontrun to unlock full value of airstreams instantly

Severity: High Risk

Context: SablierV2MerkleStreamerFactory.sol#L58-L59

Description: The `MerkleStreamer` factory uses `CREATE2` to deploy `MerkleStreamers`. One reason for this decision is that campaigns can be prefunded with tokens: organizations can send tokens to the future address, and only later set up the `MerkleStreamer` to activate the airstream.

The salt is used to constrain the CREATE2 deployment to have the correct parameters for all the values it includes. However, the salt does not include `streamDurations`, `cancelable`, or `transferable`.

As a result, once a campaign has been prefunded, any user is free to create the `MerkleStreamer` by submitting the desired parameters used in the salt, but replacing the other parameters with whichever they wish. (If they are aware of the salt parameters, they do this any time. Otherwise, they will need to wait for the real transaction to observe the salt parameters and frontrun it with their preferred arguments).

As an example, a user might match all the parameters intended by the organization, but set `streamDurations = (0, 0)`, which will unlock all tokens instantly, turning the airstream into an airdrop.

Recommendation: Include all the constructor arguments for the `MerkleStreamer` in the salt, so that any address creates a completely deterministic set of arguments that must be passed to create a `MerkleStreamer` at that address.

Sablier: Fixed by following the recommendation and including all constructor arguments in the salt in PR 217.

Cantina Solo: Fixed.

3.2 Medium Risk

3.2.1 Cancel DOS attack enabled by `updateMetadata` modifier

Severity: Medium Risk

Context: SablierV2Lockup.sol#L68-L68, SablierV2Lockup.sol#L153-L153, SablierV2LockupLinear.sol#L435-L435

Description: When the sender cancels a stream, their call to `cancel()` calls the internal `_cancel()` function, which ends with a try catch block containing a callback to the recipient.

This callback creates a risk that the recipient could waste up to 63/64th of the remaining gas. This cost is imposed on the sender who is trying to cancel a stream, and might be a way to get "revenge" in the case that a stream was cancelled in a dispute.

Before the 1.1.0 changes, this attack was not an issue because the try catch block was the final action performed in the `cancel()` function. As a result, the 1/64th of gas remaining would almost always be sufficient to complete execution, and the attack would not incur substantial additional cost.

However, version 1.1.0 introduces the `updateMetadata()` modifier, which is added to the `cancel()` function. This modifier performs the function call for `cancel()`, and then emits an event notifying protocols like OpenSea that the metadata has been updated.

In this case, however, it also has the effect of adding 1433 gas after the try catch block. While this may seem like a small amount, the result is that 91,172 gas will need to be present at the start of the try catch block to ensure enough is remaining for execution after a DOS attack.

This is a large cost to put on senders to cancel (~$10 at current ETH/gas prices, but up to $100 at 2021 prices), that could make the cancellation not worth it, effectively giving recipients a way to block a cancellation.

Recommendation: Change the `updateMetadata` modifier to emit the event before the function is executed, not after.

While this may seem illogical, the event being emitted is read off-chain after the function execution either way, so it will have no difference on the desired outcome.

Sablier: Fixed by implementing the recommendation in PR 727.

Cantina Solo: Fixed.

3.2.2 Protocol fees can be increased after airstream is set up, stealing from users

Severity: Medium Risk

Context: SablierV2MerkleStreamer.sol#L87-L102

Description: Protocol fees are taken when a stream is created. In a normal situation, this allows users to observe the protocol fee for a given asset before deciding if they will create a stream.

However, in the case of airstreams, the funds are locked in and committed to be streamed in advance of the stream being created. In some cases (ie if `expiration` is set to `0`), all funds are permanently locked and can't be clawed back.

In these sitautions, Sablier would have free reign to increase the protocol fee for the given asset all the way up to the max (10%), and there would be no recourse. This in effect gives Sablier the ability to steal up to 10% of any airstream that is created.

Recommendation: An `expectedFee` parameter should be included when creating a stream. This parameter can be checked against it each time `claim()` is called to create a stream. Additionally, an override can be added to `clawback()` that allows the protocol's admin to pull back the funds in the event that Sablier's fee is increased beyond what is expected.

Sablier: Fixed in PR 220 by not allowing the protocol fee to be greater than zero. Both `claim` and `clawback` have been updated to reflect this new requirement.

Cantina Solo: Fixed.

3.3 Low Risk

3.3.1 Users can get free flashloans of tokens with callbacks

Severity: Low Risk

Context: SablierV2LockupLinear.sol#L486-L488

Description: When creating a new stream, we save the stream in storage before transferring the tokens into the protocol.

In many token standards with a callback (for example, ERC777). the callback to the sender happens before the transfer.

This creates a situation where at the moment when control flow is passed back to the sender (a) the stream exists but (b) the sender hasn't yet supplied the tokens.

This creates the opportunity for users to take free flash loans from the protocol, as follows:

- Create a cancellable stream to myself with a value equal to the full balance of the token held by Sablier.
- In the callback, cancel the stream, which will send me the full balance.
- Perform whatever actions I want with the funds.
- After the callback is complete, the function will continue and send the balance back to Sablier.

More generally, while I can't find an exploits in the current version, the pattern of passing control flow to the user when the stream exists but has not yet been funded seems risky.

Recommendation: I would not recommend breaking checks-effects-interactions to address this. However, I would recommend adding functionality to the protocol that does not allow actions to be taken on a stream during the block it was created.

This could, for example, but included in an adjusted version of the `notNull()` modifier.

Sablier: Acknowledged. We added a caveat about ERC-777 tokens in the "Assumptions" section of our security policy (see commit ba827e19).

Cantina Solo: Acknowledged.

3.3.2 `getMerkleStreamers()` will return incorrect streamers if admin role is transferred

Severity: Low Risk

Context: Adminable.sol#L34-L34, SablierV2MerkleStreamerFactory.sol#L21-L21, SablierV2MerkleStreamerFactory.sol#L28-L35

Description: The `getMerkleStreamers()` function is intended to return all the streamers owned by a given admin. It will be used on the front end to display the relevant streamers when a user's wallet is connected.

The admin role for streamers uses the Adminable.sol contract, which makes the role transferrable.

On the other hand, the `_merkleStreamers` mapping in the factory saves a newly created streamer to the `initialAdmin`'s array, and does not change it upon transfer.

As a result, any streamers that are transferred will continue to show up on the `initialAdmin`'s front end, and will not show up for the new admin.

Recommendation: The easiest option would be to disallow transferring of the admin role for streamers. This may be the cleaner option, as the admin is passed as the `sender` to all the created streams, which would cause a weird experience if it was transferred and airstreams claimed and different times had different senders.

The alternative option is to implement logic to update the `_merkleStreamers` mapping when the admin role is transferred. While this would be a little expensive gas-wise (because of the need to swap and pop from the old admin's array), an admin transfer is a rare enough event that this extra gas cost would be acceptable.

Sablier: Fixed by removing the `_merkleStreamers` mapping and the `getMerkleStreamers` getter in PR 218.

Cantina Solo: Fixed.

3.4 Informational

3.4.1 Cancelable airstreams using bots create risk for private key theft

Severity: Informational

Context: SablierV2MerkleStreamerLL.sol#L89-L98

Description: When airstreams are created, the `CANCELABLE` flag can be set to true or false, which will trickle through to the settings of all the individual streams created by the airstream.

If the flag is true, the `admin` of the airstream can cancel streams at any time, and they are personally returned any funds that haven't been streamed yet.

One example of when this might be used is for streams that require reaching certain KPIs. As described by the Sablier team, this might include automated bots monitoring the user's engagement and cancelling airstreams if they do not meet certain criteria.

It is important to note that whatever private key is given the ability to perform this action will, in effect, be holding the complete value of the entire airstream. This is because they have the ability to cancel all streams and personally receive all the refunds.

There is not presently a way to divide up the roles so that a bot can operate the cancellations but not receive the funds. Only the sender themselves can cancel streams, and they will receive the funds personally.

As a result, the expected use case would require putting private keys in the cloud that could control millions of dollars of airdrop tokens, which would provide a juicy target for private key theft.

Recommendation: I would recommend creating an `AirstreamAdmin.sol` contract. Users could deploy proxies of this contract to serve as the admin for their airstreams. This contract could allow users to register bots who can call `cancel()` on their behalf, but an immutable address will be set to receive the funds, minimizing the risk of deploying private keys to the cloud.

This would also provide a solution regarding the `getMerkleStreamers()` function, as we could make the admin on the streamer contract itself immutable, and move the logic into `AirstreamAdmin.sol`.

Sablier: Due to business and time budget constraints, we will just acknowledge this issue. We want to understand the demand for airstreams before optimizing toward this use case.

Cantina Solo: Acknowledged.

4 Additional Comments

The Cantina Solo security researcher reviewed Sablier's v2-core and v2-periphery changes on commit hashes a4bf69cf7024006b9a324eef433f20b74597eaaf and 8350d10b28314475951b17651f30c2ede33d7722, and determined that all issues were resolved and no new issues were identified.

4.1 On-chain contracts

The reviewed codebase has been verified to correspond to the following on-chain contracts:

Ethereum Mainnet

Contract	Address
`SablierV2LockupLinear`	0xAFb979d9afAd1aD27C5eFf4E27226E3AB9e5dCC9
`SablierV2LockupDynamic`	0x7CC7e125d83A581ff438608490Cc0f7bDff79127
`SablierV2NFTDescriptor`	0x23eD5DA55AF4286c0dE55fAcb414dEE2e317F4CB
`SablierV2Comptroller`	0xC3Be6BffAeab7B297c03383B4254aa3Af2b9a5BA

Table 1: `v2-core` contracts on Ethereum Mainnet

Contract	Address
`SablierV2Batch`	0xEa07DdBBeA804E7fe66b958329F8Fa5cDA95Bd55
`SablierV2MerkleStreamerFactory`	0x1A272b596b10f02931480BC7a3617db4a8d154E3

Table 2: `v2-periphery` contracts on Ethereum Mainnet

Arbitrum One

Contract	Address
`SablierV2LockupLinear`	0xFDD9d122B451F549f48c4942c6fa6646D849e8C1
`SablierV2LockupDynamic`	0xf390cE6f54e4dc7C5A5f7f8689062b7591F7111d
`SablierV2NFTDescriptor`	0x2fb103fC853b2F5022a840091ab1cDf5172E7cfa
`SablierV2Comptroller`	0x17Ec73692F0aDf7E7C554822FBEAACB4BE781762

Table 3: `v2-core` contracts on Arbitrum One

Contract	Address
`SablierV2Batch`	0xAFd1434296e29a0711E24014656158055F00784c
`SablierV2MerkleStreamerFactory`	0x237400eF5a41886a75B0e036228221Df075b3B80

Table 4: `v2-periphery` contracts on Arbitrum One

Base

Contract	Address
SablierV2LockupLinear	0xFCF737582d167c7D20A336532eb8BCcA8CF8e350
SablierV2LockupDynamic	0x461E13056a3a3265CEF4c593F01b2e960755dE91
SablierV2NFTDescriptor	0x67e0a126b695DBA35128860cd61926B90C420Ceb
SablierV2Comptroller	0x7Faaedd40B1385C118cA7432952D9DC6b5CbC49e

Table 5: v2-core contracts on Base

Contract	Address
SablierV2Batch	0x94E596EEd73b4e3171c067f05A87AB0268cA993c
SablierV2MerkleStreamerFactory	0x5545c8E7c3E1F74aDc98e518F2E8D23A002C4412

Table 6: v2-periphery contracts on Base

BNB Smart Chain

Contract	Address
SablierV2LockupLinear	0x14c35E126d75234a90c9fb185BF8ad3eDB6A90D2
SablierV2LockupDynamic	0xf900c5E3aA95B59Cc976e6bc9c0998618729a5fa
SablierV2NFTDescriptor	0xEcAfcF09c23057210cB6470eB5D0FD8Bafd1755F
SablierV2Comptroller	0x33511f69A784Fd958E6713aCaC7c9dCF1A5578E8

Table 7: v2-core contracts on BNB Smart Chain

Contract	Address
SablierV2Batch	0x2E30a2ae6565Db78C06C28dE937F668597c80a1c
SablierV2MerkleStreamerFactory	0x434D73465aAc4125d204A6637eB6C579d8D69f48

Table 8: v2-periphery contracts on BNB Smart Chain

Gnosis

Contract	Address
SablierV2LockupLinear	0xce49854a647a1723e8Fb7CC3D190CAB29A44aB48
SablierV2LockupDynamic	0x1DF83C7682080B0f0c26a20C6C9CB8623e0Df24E
SablierV2NFTDescriptor	0x01dbFE22205d8B109959e2Be02d0095379309eed
SablierV2Comptroller	0x73962c44c0fB4cC5e4545FB91732a5c5e87F55C2

Table 9: v2-core contracts on Gnosis

Contract	Address
SablierV2Batch	0xBd9DDbC55B85FF6Dc0b76E9EFdCd2547Ab482501
SablierV2MerkleStreamerFactory	0x777F66477FF83aBabADf39a3F22A8CC3AEE43765

Table 10: v2-periphery contracts on Gnosis

Optimism

Contract	Address
`SablierV2LockupLinear`	0x4b45090152a5731b5bc71b5baF71E60e05B33867
`SablierV2LockupDynamic`	0xd6920c1094eABC4b71f3dC411A1566f64f4c206e
`SablierV2NFTDescriptor`	0xF5050c04425E639C647F5ED632218b16ce96694d
`SablierV2Comptroller`	0x1EECb6e6EaE6a1eD1CCB4323F3a146A7C5443A10

Table 11: `v2-core` contracts on Optimism

Contract	Address
`SablierV2Batch`	0x8145429538dDBdDc4099B2bAfd24DD8958fa03b8
`SablierV2MerkleStreamerFactory`	0x044EC80FbeC40f0eE7E7b3856828170971796C19

Table 12: `v2-periphery` contracts on Optimism

Polygon

Contract	Address
`SablierV2LockupLinear`	0x5f0e1dea4A635976ef51eC2a2ED41490d1eBa003
`SablierV2LockupDynamic`	0xB194c7278C627D52E440316b74C5F24FC70c1565
`SablierV2NFTDescriptor`	0x8683da9DF8c5c3528e8251a5764EC7DAc7264795
`SablierV2Comptroller`	0x9761692EDf10F5F2A69f0150e2fd50dcecf05F2E

Table 13: `v2-core` contracts on Polygon

Contract	Address
`SablierV2Batch`	0x5865C73789C4496665eDE1CAF018dc52ac248598
`SablierV2MerkleStreamerFactory`	0xF4906225e783fb8977410BDBFb960caBed6C2EF4

Table 14: `v2-periphery` contracts on Polygon

Scroll

Contract	Address
`SablierV2LockupLinear`	0x57e14AB4DAd920548899d86B54AD47Ea27F00987
`SablierV2LockupDynamic`	0xAaff2D11f9e7Cd2A9cDC674931fAC0358a165995
`SablierV2NFTDescriptor`	0xB71440B85172332E8B768e85EdBfdb34CB457c1c
`SablierV2Comptroller`	0x859708495E3B3c61Bbe19e6E3E1F41dE3A5C5C5b

Table 15: `v2-core` contracts on Scroll

Contract	Address
`SablierV2Batch`	0xD18faa233E02d41EDFFdb64f20281dE0592FA3b5
`SablierV2MerkleStreamerFactory`	0xb3ade5463000E6c0D376e7d7570f372eBf98BDAf

Table 16: `v2-periphery` contracts on Scroll

Arbitrum Sepolia

Contract	Address
`SablierV2LockupLinear`	0x483bdd560dE53DC20f72dC66ACdB622C5075de34
`SablierV2LockupDynamic`	0x8c8102b92B1f31cC304A085D490796f4DfdF7aF3
`SablierV2NFTDescriptor`	0x593050f0360518C3A4F11c32Eb936146e1096FD1
`SablierV2Comptroller`	0xA6A0cfA3442053fbB516D55205A749Ef2D33aed9

Table 17: `v2-core` contracts on Arbitrum Sepolia testnet

Contract	Address
`SablierV2Batch`	0x72D921E579aB7FC5D19CD398B6be24d626Ccb6e7
`SablierV2MerkleStreamerFactory`	0xcc87b1A4de285832f226BD585bd54a2184D32105

Table 18: `v2-periphery` contracts on Arbitrum Sepolia testnet

Sepolia

Contract	Address
`SablierV2LockupLinear`	0x7a43F8a888fa15e68C103E18b0439Eb1e98E4301
`SablierV2LockupDynamic`	0xc9940AD8F43aAD8e8f33A4D5dbBf0a8F7FF4429A
`SablierV2NFTDescriptor`	0xE8fFEbA8963CD9302ffD39c704dc2c027128D36F
`SablierV2Comptroller`	0x2006d43E65e66C5FF20254836E63947FA8bAaD68

Table 19: `v2-core` contracts on Sepolia testnet

Contract	Address
`SablierV2Batch`	0xd2569DC4A58dfE85d807Dffb976dbC0a3bf0B0Fb
`SablierV2MerkleStreamerFactory`	0xBacC1d151A78eeD71D504f701c25E8739DC0262D

Table 20: `v2-periphery` contracts on Sepolia testnet