Sablier v2 on Blast

Security Review

Cantina Managed review by:
RustyRabbit, Security Researcher

June 13, 2024

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 Use of Foundry's `ffi` cheatcode should be avoided if possible . . . 4
  - 3.1.2 UX on stream cancel might hinder or deter sender from canceling or recipient from withdrawing . . . 4
  - 3.1.3 Potential use of malicious Lockup contract when creating an airstream from the factory 5
  - 3.1.4 Merkle tree and proof depth not checked . . . 5
  - 3.1.5 Clawback not possible in case of erroneously configured `MerkleLockup` and no expiration . . . 6
  - 3.1.6 Claim indexes can appear multiple times in the Merkle tree possibly leading to stuck funds . . . 6
  - 3.1.7 Calls to untrusted contracts on Blast are incentivized to steal gas from users . . . 7
  - 3.1.8 `createMerkleLT()` can revert causing loss of funds when used for counterfactual deployment . . . 8
- 3.2 Low Risk . . . 9
  - 3.2.1 When stream sender admin is transferred, callbacks are still made to the old admin . 9
  - 3.2.2 Airstreams have to be renounced individually . . . 10
  - 3.2.3 `aggregateAmount` emitted but not validated . . . 10
- 3.3 Informational . . . 11
  - 3.3.1 Use of 2 step admin transfer is recommended . . . 11
  - 3.3.2 Blast informational findings . . . 11
  - 3.3.3 Core invariant not properly checked in base contract . . . 12

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, 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 Apr 15th to May 3rd the Cantina team conducted a review of v2-core, v2-periphery and v2-core (blast) on commit hashes a86edeee, 73831c7d and 9280d98d respectively. The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 8
- Low Risk: 3
- Gas Optimizations: 0
- Informational: 3

3 Findings

3.1 Medium Risk

3.1.1 Use of Foundry's `ffi` cheatcode should be avoided if possible

Severity: Medium Risk

Context: foundry.toml#L7-L12, Base.s.sol#L86

Description: `constructCreate2Salt()` uses foundry's `ffi` cheatcode to create a salt for deterministic deployment of the contracts based on the version of the contracts.

Use of `ffi` is considered dangerous as documented here

> It is generally advised to use this cheat code as a last resort, and to not enable it by default, as anyone who can change the tests of a project will be able to execute arbitrary commands on devices that run the tests.

Recommendation: use the `vm.readFile()` and `vm.parseJson()` to achieve the same with less risk.

Sablier: We implemented this change in both projects:

- `core` (see commit 02a3900e).
- `periphery` (see commit 502bacc6).

Cantina Managed: Fixed.

3.1.2 UX on stream cancel might hinder or deter sender from canceling or recipient from withdrawing

Severity: Medium Risk

Context: SablierV2Lockup.sol#L609-L616

Description: When the sender cancels a stream the `SablierV2Lockup` does a callback to the sender's `onLockupStreamCancled()` if it is a contract. A try/catch makes sure any revert would not block the cancel but a deliberate gas bomb by the recipient could cause the user to be prompted with a high gas cost:

```
if (recipient.code.length > 0) {
    try ISablierV2Recipient(recipient).onLockupStreamCanceled({
        streamId: streamId,
        sender: sender,
        senderAmount: senderAmount,
        recipientAmount: recipientAmount
    }) { } catch { }
}
```

The sender can just lower the transaction's gas limit. The callback will then run out of gas, the transaction will continue and the stream will be canceled. However when a less sophisticated user is prompted with the high gas cost they may not realize it and will be unable to predict if the cancel will even still work with the lower gas limit. If the user actually sends the transaction they might end up paying a high amount of ETH (or equivalent on other chains) on gas fees to get the stream canceled while a lot less would have sufficed. Theoretically the predicted gas cost might even be so high the transaction never gets included in a block due to the high gas limit.

The same issue is present on the `withdraw()` where the sender might try to hinder a recipient from withdrawing an un-cancelable stream.

Recommendation: Consider placing a limit on the gas used for `onLockupStreamCanceled()` and `onLockupStreamWithdrawn()`. This limit could be a configurable value or a percentage of the gas left although the UX of this solution should be tested.

Sablier: Thank you for the insights and we acknowledge the problem, however limiting the gas would also limit the capabilities of the hooks in supporting complex transactions. So we have decided to keep it unchanged. This also means that we would need to add an admin function for adjusting the gas passed during the hook calls, which increases the maintenance cost for us. But to lower the risks for users, we have decided to address this at the interface level by showing a "WARNING" message in case the `cancel` and `withdraw` functions try to consume gas above a certain threshold.

Cantina Managed: Acknowledged.

3.1.3 Potential use of malicious Lockup contract when creating an airstream from the factory

Severity: Medium Risk

Context: SablierV2MerkleLockupFactory.sol#L61, SablierV2MerkleLockupFactory.sol#L103, SablierV2MerkleLockupFactory.sol#L27, SablierV2MerkleLockupFactory.sol#L52, SablierV2MerkleLT.sol#L57, SablierV2MerkleLL.sol#L51

Description: `createMerkleLL()` and `createMerkleLT()` take an address for the `LockupLinear` and `LockupTranched` contracts to be used when the streams are claimed. In the constructor of both `SablierV2MerkleLL` and `SablierV2MerkleLT` it also *max approves* these contracts in order to allow it to use the funds held by the them during the claim process.

The provided address can be any contract as long as it supports the required functions not to attract any suspicion in the front-end. This can be abused by a team member of the project designated to creating the `SablierMerkleLL` or `SablierMerkleLT` even if the team applies separation of duty between them and the person sending the funds to the newly created contract. The provided contract can be a custom one deployed by the malicious team member and allow them to withdraw the funds from the Merkle lockup contract to their own address.

This could also be an issue when users provide the wrong contract address accidentally. For instance providing the address for the `LockupLinear` when creating a `MerkleLockupLT` or possibly even an older contract that doesn't take the same parameter types when creating a stream.

Recommendation: consider using (separate) whitelists of `LockupLinear` and `LockupTranched` addresses that can be used as addresses for streaming contracts when creating `MerkleLockupLL` and `MerkleLockupLT` contracts. Make sure the Merkle lockup contracts use the same parameter types for creating the streams as the whitelisted contracts when adding them to the whitelist.

Sablier: We acknowledge that this exists as a valid risk. But instead of whitelisting, we think the following can lower the risk for the users:

- Allowing clawback during the grace period will lower the risk of exploits to the users.
- Sablier UI will perform a validation check to only show the campaigns with correct Sablier contracts to both creators as well as the recipients.

Cantina Managed: Acknowledged.

3.1.4 Merkle tree and proof depth not checked

Severity: Medium Risk

Context: SablierV2MerkleLL.sol#L59-L96, SablierV2MerkleLockup.sol#L120-L139,

Description: When claiming an airstream the Merkle proof is checked against the Merkle root, but the length of the proof is not checked against the expected depth of the Merkle tree.

```
function _checkClaim(uint256 index, bytes32 leaf, bytes32[] calldata merkleProof) internal view {
    if (_claimedBitMap.get(index)) {
        revert Errors.SablierV2MerkleLockup_StreamClaimed(index);
    }

    if (!MerkleProof.verify(merkleProof, MERKLE_ROOT, leaf)) {
        revert Errors.SablierV2MerkleLockup_InvalidProof();
    }
}
```

The way this can be exploited is if a malicious member of team deploying the MerkleLockup makes one of the leaves of the Merkle tree a root hash of another tree (which can hold their own list of addresses, amounts and indexes even outside of expected index range).

This would however be visible by anyone that actually validates the Merkle tree of course as one of the leaves is not the hash of the address, amount and index, it would just be the precomputed root hash of a subtree.

Recommendation: When creating the MerkleLockup consider also storing the tree depth and validating the proof length against it during the claim process.

Sablier: Having a tree inside another tree presents a very interesting edge case. We also acknowledge that this could be a valid problem. However, we would like to leave this as it is. If there is a team member with malicious intent, he could exploit the campaign creator in several ways such as he can introduce new leaves with malicious amounts assigned to themselves. We also believe that allowing `clawback` during the grace period will lower the risk of exploits to the users.

Cantina Managed: Acknowledged.

3.1.5 Clawback not possible in case of erroneously configured `MerkleLockup` and no expiration

Severity: Medium Risk

Context: SablierV2MerkleLockup.sol#L101, SablierV2MerkleLockup.sol#L86

Description: If for any reason the configuration of the MerkleLockup is wrong (like a bad Merkle root hash) the funds can not be recovered until the expiration date has been met. In the case that no expiration has been set, the funds cannot be recovered as `clawback()` is only allowed after expiration.

Recommendation: Consider allowing `clawback()` in erroneous configuration situations that make claiming impossible (e.g allow clawback before the first claim or initially in a certain timespan).

Sablier: We acknowledge this as a valid issue. And we are happy to accept your recommendation. `clawback` can now be called until the 7 days from the first claim which gives sufficient time to users to take actions and mitigate the risk of losing funds in case of malicious campaigns. The grace period is added irrespective of expiration. You can find the changes introduced in PR 340.

Cantina Managed: Fixed.

3.1.6 Claim indexes can appear multiple times in the Merkle tree possibly leading to stuck funds

Severity: Medium Risk

Context: SablierV2MerkleLL.sol#L71-L77, SablierV2MerkleLT.sol#L86-L95

Description: `MerkleLockup` uses a Merkle tree to distribute airstream funds The leaves contain the combination of the recipient address, the amount and an index. The index is used keep track of which leaf of the Merkle tree has already been claimed. There is no check that the same index isn't used in multiple leaves during creation of the `MerkleLockup` contract, as this would be unfeasible to validate this on-chain. This means that if this happens to be the case, one address/leaf will be unclaimable when the other instance has been claimed already.

When the aggregate amount of all the amounts in the leaves is sent to the contract the part of those funds cannot be claimed and therefor stuck in the contract until expiration or forever in case the expiration is set to 0.

Recommendation: Consider using the path of the Merkle proof to be used as a check for the index instead of encoding it in the leaf. This will ensure each index can only appear once. An example of such code can be found in Eigenlayer's Merkle library:

```
function processInclusionProofKeccak(
    bytes memory proof,
    bytes32 leaf,
    uint256 index
) internal pure returns (bytes32) {
    require(
        proof.length != 0 && proof.length % 32 == 0,
        "Merkle.processInclusionProofKeccak: proof length should be a non-zero multiple of 32"
    );
    bytes32 computedHash = leaf;
    for (uint256 i = 32; i <= proof.length; i += 32) {
        if (index % 2 == 0) {
            // if ith bit of index is 0, then computedHash is a left sibling
            assembly {
                mstore(0x00, computedHash)
                mstore(0x20, mload(add(proof, i)))
                computedHash := keccak256(0x00, 0x40)
                index := div(index, 2)
            }
        } else {
            // if ith bit of index is 1, then computedHash is a right sibling
            assembly {
                mstore(0x00, mload(add(proof, i)))
                mstore(0x20, computedHash)
                computedHash := keccak256(0x00, 0x40)
                index := div(index, 2)
            }
        }
    }
    return computedHash;
}
```

Sablier: Thank you for sharing Eigenlayer's Merkle Library and your valuable insights on this. We agree that this could be a valid issue, however we have decided to keep the current implementaion. Merkle Tree is generated using the Sablier backend which we expect users to trust. If a user submits Merkle Root directly with an IPFS hash (which we also allow through the UI), our backend performs a validation check to ensure that the Merkle tree is appropriately created

Cantina Managed: Acknowledged.

3.1.7 Calls to untrusted contracts on Blast are incentivized to steal gas from users

Severity: Medium Risk

Context: SablierV2Lockup.sol#L610-L615, SablierV2Lockup.sol#L395-L400, SablierV2Lockup.sol#L599

Description: On Blast every contract can claim the gas fees payed by the users executing their code. When calling out to untrusted contracts (e.g. callbacks to receiver and sender, but also the `transferFrom` calls on the underlying tokens) could be used by the owners of those contracts to inflate gas usage and steal gas from Sablier's users.

The callback function `onLockupStreamWithdrawn` could be used by a scam project to create an airstream with the sole intent to steal gas whenever a user withdraws, either in small increments or with maximum gas usage (gas bombs). This also applies to any novel token that does the same on token transfers which is used in the `withdraw()` and `cancel()` functions. The `onLockupStreamCanceled` could be used by a recipient as incentivized griefing attack if the stream gets canceled.

Note that this issue exists on all chains, but on Blast there actually is an extra incentive in the form of Blast gas refunds.

Recommendation: On Blast consider limiting the gas used for calls to untrusted contracts or provide alternatives that do not call the callback hooks at all.

Sablier: Thank you for the insights and we acknowledge the problem, however limiting the gas would also limit the capabilities of the hooks in supporting complex transactions. So we have decided to keep it unchanged.

Cantina Managed: Acknowledged.

3.1.8 `createMerkleLT()` can revert causing loss of funds when used for counterfactual deployment

Severity: Medium Risk

Context: SablierV2MerkleLockupFactory.sol#L72-L103, SablierV2MerkleLockup.sol#L58-L62

Description: When creating a Merkle lockup from the factory `CREATE2` is used to ensure a deterministic address. If the the combined percentages of the tranches do not add up to 100% the `createMerkleLT()` reverts as creating such a Merkle lockup would create streams that do not fully stream the allotted amount to the recipient.

```
uint256 totalDuration;
for (uint256 i = 0; i < tranchesWithPercentages.length; ++i) {
    uint64 percentage = tranchesWithPercentages[i].unlockPercentage.unwrap();
    totalPercentage = totalPercentage + percentage;
}
if (totalPercentage != uUNIT) {
    revert Errors.SablierV2MerkleLockupFactory_TotalPercentageNotOneHundred(totalPercentage);
}
bytes32 salt = keccak256(
    abi.encodePacked(
        // ...
        abi.encode(tranchesWithPercentages)
    )
);
merkleLT = new SablierV2MerkleLT{ salt: salt }(baseParams, lockupTranched, tranchesWithPercentages);
```

If however this is used for a counterfactual deployment where funds are sent to the contract's off chain computed address, the funds will be lost if the percentages do not add up to 100%. Changing the percentage so they do add up to 100% would also change the salt used for the `CREATE2` which in turn leads to a different address for the contract. As no contract can thus be deployed on the precomputed address, the funds cannot be retrieved. The team indicated however this isn't a typical use case and funds are sent after the contract is created.

The same issue exists in the constructor of `SablierV2MerkleLockup` where the name is required to be less than 32 bytes. In this case however the impact is limited as merely calling `createMerkleLT()` again with the (same) name capped to 32 bytes does create the contract at the pre-computed address (provided the off chain compute code also capped the name to 32bytes).

Recommendation: Typically you should not revert when using `CREATE2`. Instead consider deploying the contract regardless and allowing to to use the `clawback` to retrieve the funds in such a case. If counterfactual deployments are a frequent use case also consider providing a view function to predict the address that uses the same checks as the actual deploy path.

Sablier: We accept your recommendation to extend clawback and allow users to retrieve the funds during the grace period. We have also decided to switch to `CREATE`, thus we will not have deterministic addresses anymore, which also means that the user can't prefund the campaign. You can see the work in PR 339 and PR 340.

Cantina Managed: Fixed.

3.2 Low Risk

3.2.1 When stream sender admin is transferred, callbacks are still made to the old admin

Severity: Low Risk

Context: SablierV2MerkleLL.sol#L82, SablierV2Lockup.sol#L265-L267, SablierV2Lockup.sol#L476-L478

Description: In the `SablierV2MerkleLT` and `SablierV2MerkleLL` contracts when users claim an airstream the sender is marked as the current admin of the contract:

```
function claim(/*...*/) {
    streamId = LOCKUP_LINEAR.createWithDurations(
        LockupLinear.CreateWithDurations({
            sender: admin,
            //...
        })
    )
}
```

If the admin is transferred to a new address any new claims will have the sender set to the new address and the old streams will still have the old admin as the sender.

As the sender of a stream is unchangeable, this creates a number of potential issues:

1) As only the sender of a stream can cancel or renounce it, the new admin cannot cancel or renounce the old streams and the old admin cannot cancel or renounce the new streams:

```
function cancel(uint256 streamId) internal {
    if (!_isCallerStreamSender(streamId)) {
        revert Errors.SablierV2Lockup_Unauthorized(streamId, msg.sender);
    }
}
function _isCallerStreamSender(uint256 streamId) internal view returns (bool) {
    return msg.sender == _streams[streamId].sender;
}
```

2) Whenever the recipient withdraws from the stream the `onLockupStreamWithdrawn()` callbacks happen to the sender at the time of the claim, again making a difference before or after the change of admin/ownership of the Lockup.

```
function withdraw(/*...*/)
    try ISablierV2Sender(sender).onLockupStreamWithdrawn({
        // ...
    }) { } catch { }
}
```

Recommendation: Consider setting the Merkle lockup contract as sender and forward the callbacks to the current owner. Also make sure the `MerkleLockup` then provides admin functions to cancel and renounce streams. Lastly if streams are canceled funds would be sent back to the `MerkleLockup` contract and thus must be forwarded or retrievable by the admin.

Sablier: We acknowledge that this could be a problem when admins are normal contracts. But we have decided to leave this as it is because most of the admins who create and fund campaigns are Gnosis Safes, in which they can use different proxy targets to forward the callbacks.

Cantina Managed: Acknowledged.

3.2.2 Airstreams have to be renounced individually

Severity: Low Risk

Context: SablierV2MerkleLT.sol#L104, SablierV2MerkleLL.sol#L86

Description: When airstreams are claimed the cancelability is set based on the immutable `CANCELABLE` specified during the creation of the MerkleLockup contract. If the sender wants to renounce cancelability they are required to do so on each stream individually which can be costly.

Recommendation: Consider making the cancelability configurable at the Merkle lockup contract so that when the owner wants to renounce all streams the new streams can be renounced before they are claimed and don't need individual transactions.

Similarly if airstreams can be set to cancelable at the start it makes sense to allow the owner to actually cancel all future streams (by canceling the Merkle lockup itself) instead of having to cancel all the new streams individually. Old streams will still have to be canceled individually.

Sablier: We agree that this could be a lot of effort for users. However, we have decided to leave it unchanged because of the following reasons:

- We believe that creators should not be allowed to change the configuration after the campaign has been created. Allowing this could lead to a situation where some users have cancelable streams while others have non-cancelable and could be unfair for them. If a campaign has been accidentally created with the incorrect configuration, the user can clawback funds and re-create another.
- The mental overload of having different streams from the same campaign is problematic.

Cantina Managed: Acknowledged.

3.2.3 `aggregateAmount` emitted but not validated

Severity: Low Risk

Context: SablierV2MerkleLockupFactory.sol#L19-L56, SablierV2MerkleLockupFactory.sol#L59-L116

Description: In the `createMerkleLT()` and `createMerkleLT()` the `aggregateAmount` (and `recipientCount`) specified as an input parameter is not used or validate except to be emitted in the `CreateMerkleLT` and `CreateMerkleLL` events. This is used in the frontend to show the amount to be sent to the Merkle lockup contract.

Mostly people tend to trust any event data as they think it is being validated on chain. If this event is used to actually determine the amount to send to the contract an attacker could front run the transaction and increase the `aggregateAmount` as a griefing attack (possibly combined with trying to short the token) especially when the expiration is set to 0 and the funds cannot be clawed back.

Recommendation: Consider leaving out the `aggregateAmount` and `recipientCount` of the API and event as it provides no on chain benefit.

Sablier: We acknowledge that this could be a problem but `aggregateAmount` is critical for the UI to determine the funding amount. This issue can be mitigated as a consequence of allowing `clawback` until 7 days from the first claim, as implemented in PR 340.

Cantina Managed: Fixed by mitigation.

3.3 Informational

3.3.1 Use of 2 step admin transfer is recommended

Severity: Informational

Context: SablierV2MerkleLockup.sol#L18, Adminable.sol#L34-L40

Description: Most contracts inherit their admin access control from `Adminable` which uses a one step transfer. Current best practices advise to use a 2-step transfer especially when less sophisticated users are involved as could be the case with `SablierMerkleLockup`. It's worth noting here that the `admin` of `SablierV2MerkleLockup` is also the sender of the streams, which users less familiar with the code might overlook.

Recommendation: Consider using a 2 step transfer for admin functionality where appropriate.

Sablier: We acknowledge this information and we have decided to not add 2-step transfer for admin functionality.

Cantina Managed: Acknowledged.

3.3.2 Blast informational findings

Severity: Informational

Context: SablierV2MerkleLockupFactory.sol#L19

Description: Some informational findings regarding Blast deployment:

- The `SablierV2MerkleLockupFactory` does not inherit from `SablierV2Blast` which is needed to be able to claim Blast's gas refunds. (`configureYieldAnGas()` function to set the `Governor`). This also applies to `SablierV2BatchLockup` and `SablierV2NFTDescriptor` although the gas usage here will probably minimal.
- WETH and USDB (and probably every future rebasing token on Blast) are set to AUTO yield by default. When a new asset is used for the first time in each of the Lockup contracts (LL, LD and LT) the yield up to the point the yield mode is changed to claimable will be stuck in the contract. Make sure you monitor for new rebasing tokens being used for the first time and after reviewing set the yield to claimable.
- `claimRebasingAssetYield()`.
- Yield on assets held in the `SablierV2LockupLinear`, `SablierV2LockupDynamic` and `SablierV2LockupTranched` can only be claimed by Sablier as there is no way to determine the percentages each stream contributed to the yield. For some tokens wrapped non-rebasing versions exist (e.g. `NrEth` and `NrUSDB`) These allow users to keep the yield themselves and will probably be preferred.

Recommendation:

- Consider inheriting from `SablierV2Blast` in `SablierV2MerkleLockupFactory` so that gas refunds can be claimed.
- Make sure you monitor for new rebasing tokens being used for the first time and after reviewing set the yield to claimable.

Sablier: Thanks for the information. We agree and have decided to inherit `SablierV2Blast` in both `SablierV2MerkleLockupFactory` and `SablierV2Batch` contracts. You can see it in PR 341.

Cantina Managed: Fixed.

3.3.3 Core invariant not properly checked in base contract

Severity: Informational

Context: SablierV2Lockup.sol#L500-L510, SablierV2Lockup.sol#L543, SablierV2Lockup.sol#L368-L371, SablierV2Lockup.sol#L640-L646

Description: One of the most important invariants of any stream is that it should never be possible to withdraw more than what has been deposited minus the refunded amount.

In `SablierV2Lockup::_withdraw()` there is no explicit check for this, it implicitly leaves that up to other functions:

- `_calculateStreamedAmount()` of the child contracts calculates the amount streamed at the current timestamp.
- `_streamedAmountOf()` then limits that amount to `deposited - refunded` if the stream has been canceled or `withdrawn` if it's marked as depleted.
- `_withdrawableAmountOf()` then subtracts the already `withdrawn` amount.
- `withdraw()` then checks if the requested amount is less than that.

So in essence it's the child contract's `_calculateStreamedAmount` that determines how much can be withdrawn from the stream at any time.

Note that at no point in `SablierV2Lockup` is there a check that the amount returned from `_calculateStreamedAmount()` does not exceed the deposited amount. This is done in each of the child contracts. The extra safety check in`_withdraw()` even allows the `withdrawn` amount to exceed the `deposited - refunded`.

```
if (amounts.withdrawn >= amounts.deposited - amounts.refunded) {
    _streams[streamId].isDepleted = true;
    // stream cannot be canceled.
    _streams[streamId].isCancelable = false;
}
```

Although it's a good approach to set the `isDepleted` to `true` regardless, it should not be possible to withdraw the specified amount in such a case.

In conclusion the crucial invariant check `withdrawn + refunded <= deposited` is spread out over different functions of different contracts, which makes it hard to ensure it's being met at all times by all types of streams.

Additionally although in the current code `SablierV2Lockup::_withdraw()` is only used from `SablierV2Lockup` it is marked as `internal` meaning it could be called from (future versions of) the child contracts which may not perform the invariant check correctly as they don't deal with refunds and withdrawals.

Recommendation: Consider adding an explicit invariant check in `SablierV2Lockup::_withdrawn()` that `withdrawn + refunded` can never be greater then `deposited` regardless of the child contract's implementation or the supporting calculation functions.

Also consider marking both `_withdraw()` as `private` to ensure other important checks are not accidentally circumvented in future versions of inheriting contracts.

Sablier: Thank you for looking at it and sharing your recommendations. We acknowledge your suggestion and would like to include it in the future versions of the protocol when we add a new child contract.

Cantina Managed: Acknowledged.