

Sablier EVM Utils

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 Missing checks for `minFeeUSD <= MAX_FEE_USD` during `SablierComptroller` deployment may break invariant . . . 4
  - 3.1.2 Missing check may accidentally burn protocol fees . . . 4
  - 3.1.3 Setting an incorrect Comptroller may lead to locked funds . . . 4
  - 3.1.4 Funds may be temporarily locked due to failing oracle . . . 5
  - 3.1.5 Possible division by zero with low value native tokens can lead to claim and withdrawal reverts . . . 5
  - 3.1.6 Hardcoding 24 hours as oracle heartbeat for native tokens of all chains may lead to incorrect fees . . . 6
- 3.2 Informational . . . 7
  - 3.2.1 Using a single `FEE_MANAGEMENT_ROLE` across different protocol components is risky . . . 7
  - 3.2.2 A global `MAX_FEE_USD` may restrict flexibility and cause unexpected fees . . . 7
  - 3.2.3 Missing emit of `msg.sender` for `FEE_MANAGEMENT_ROLE` controlled functions . . . 7
  - 3.2.4 Missing emit of previous values in `SablierComptroller` custom fee setters . . . 8
  - 3.2.5 Comptroller's ability to only lower the fee can be circumvented by setting a malicious oracle . . . 8
  - 3.2.6 Native tokens are assumed to have 18 decimals for fee calculation . . . 8
  - 3.2.7 Upgradeable Comptroller does not implement UUPS proxy best practices . . . 9

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 evm-utils on commit hash 7c10048f. The particular scope consisted in the following contracts:

1. RoleAdminable.sol
2. SablierComptroller.sol
3. ComptrollerManager.sol

The team identified a total of **13** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	6	5	1
Gas Optimizations	0	0	0
Informational	7	3	4
Total	**13**	**8**	**5**

The Cantina Managed team reviewed Sablier's `evm-utils` holistically on commit hash 9a4139fe and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 Low Risk

3.1.1 Missing checks for `minFeeUSD <= MAX_FEE_USD` during `SablierComptroller` deployment may break invariant

Severity: Low Risk

Context: SablierComptroller.sol#L37-L41, SablierComptroller.sol#L52-L68, SablierComptroller.sol#L243-L247, SablierComptroller.sol#L282-L286, SablierComptroller.sol#L321-L325

Summary: Missing checks for `minFeeUSD <= MAX_FEE_USD` during `SablierComptroller` deployment may break implicit protocol invariant of fees never exceeding `MAX_FEE_USD`.

Finding Description: The protocol enforces an invariant of minimum/custom fees never exceeding `MAX_FEE_USD` (USD $100) in its setters with the `notExceedMaxFeeUSD` modifier. However, this enforcement is missing during `SablierComptroller` deployment, which allows the deployer to accidentally break the invariant.

Impact Explanation: Low, because the protocol invariant of fees never exceeding `MAX_FEE_USD` is implemented for transparency and this can be observed by users before interacting with it.

Likelihood Explanation: Low, because it is expected that protocol deployment will implicitly enforce this invariant even without the checks. The current deployment script uses a value of USD $1.

Recommendation: Consider enforcing `minFeeUSD <= MAX_FEE_USD` checks explicitly during `SablierComptroller` deployment to prevent accidentally exceeding `MAX_FEE_USD`. Counterfactual deployment concerns for constructor checks should not be an issue here.

Sablier: Fixed in PR 51.

Cantina Managed: Fix verified.

3.1.2 Missing check may accidentally burn protocol fees

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Summary: Missing zero-address check on `feeRecipient` in `_collectFees()` may accidentally burn protocol fees.

Finding Description: While the protocol implements zero-address checks in its airdrop functions using `modifier notZeroAddress`, it misses a similar check on `feeRecipient` in `SablierComptroller._collectFees()`, which is called by the admin or `FEE_COLLECTOR_ROLE` to collect/transfer all/some of the accrued protocol fees in native token to the `feeRecipient` address. If a zero-address is accidentally provided, it will burn that `feeAmount`.

Impact Explanation: Low, because this only burns the `feeAmount` for the particular call with zero `feeRecipient` address.

Likelihood Explanation: Low, because `feeRecipient` is expected to be a non-zero address and `collectFees()` is expected to be carefully triggered as such in a scheduled/automated way without much room for accidental error.

Recommendation: Consider adding a zero-address check for `feeRecipient` in `_collectFees()` to prevent any accidental burn of protocol fees.

Sablier: Fixed in PR 48.

Cantina Managed: Reviewed that PR 48 adds a zero-address check for `feeRecipient` in the refactored `transferFees()`.

3.1.3 Setting an incorrect Comptroller may lead to locked funds

Severity: Low Risk

Context: ComptrollerManager.sol#L79-L94

Summary: Setting the comptroller to a contract unable to fulfill its responsibilities may lead to a situation where the relying contracts are unable to release funds. In extreme cases, the situation cannot be undone if the new comptroller does not provide the functionality to change the comptroller on the relying contracts.

Finding Description: When setting a new comptroller, the address is not checked to see if it provides the necessary functionality required of a comptroller.

Besides admin functions like `recover()`, `setNativeToken()`, `setNFTDescriptor()` and `lowerMinFeeUSD()` the comptroller is also used to calculate the required fee when claiming and withdrawing. If the comptroller is accidentally set to a contract that is unable to provide these functions then those calls will revert and make claiming and withdrawals impossible.

If the new comptroller is also unable to call `setComptroller()` on the relying contracts then it may be impossible to recover from the situation and funds may be permanently locked in those contracts.

Impact Explanation: High, because funds may be temporarily or indefinitely locked on the contracts.

Likelihood Explanation: Low, because this requires a an admin error.

Recommendation: Consider using ERC165 to check that the newly set contract provides the required functionality for the contract to operate properly.

Alternatively or additionally, ensure the Comptroller can transfer the 'ownership' to another contract using a 2-step process (preferably using the same `setComptroller()` but with different parameters) to finalize the change.

Sablier: Fixed in PR 53.

Cantina Managed: Reviewed that PR 53 adds a notion of `MINIMAL_INTERFACE_ID` that captures the selectors of `calculateMinFeeWeiFor()`, `convertUSDFeeToWei()`, `execute()`, and `getMinFeeUSDFor()`, because those are the minimal functions that any new comptroller should have to allow withdraw and claim to continue. This minimal Interface Id is initialized and checked for support in the new comptroller being set in `_setComptroller()`.

3.1.4 Funds may be temporarily locked due to failing oracle

Severity: Low Risk

Context: SablierComptroller.sol#L373-L385, SablierComptroller.sol#L407

Summary: Funds may be temporarily locked because claim and withdrawal functions need to query the comptroller for the fee amount in native tokens which will revert if the oracle call fails.

Finding Description: For functions that require to query the comptroller for the fee amount in native tokens, the comptroller queries the oracle for the price of the native token via its `latestRoundData()` interface. If for whatever reason this call to the oracle fails, it will bubble up to the claim and withdrawal functions thereby making withdrawals and claim (temporarily) impossible.

Impact Explanation: Medium, because claims and withdrawals are impossible until the oracle issue has been resolved or the oracle has been changed in case of permanent oracle problems. Fees could also be set to 0 to deactivate the oracle until a solution is found.

Likelihood Explanation: Low, because such oracle issues are uncommon.

Recommendation (optional): Consider using a try/catch block to detect such issues. As occasional loss of fee (due to stale oracle data) is already accepted, consider doing the same in case of general oracle failure and continue with 0 fee. Note that this also applies to the oracle's `decimal()` function.

Sablier: Fixed in PR 50 and commit 0029710.

Cantina Managed: Reviewed that PR 50 and commit #0029710 mitigate the issue as recommended.

3.1.5 Possible division by zero with low value native tokens can lead to claim and withdrawal reverts

Severity: Low Risk

Context: SablierComptroller.sol#L411-L420

Summary: Low value native tokens (in USD) terms would return a price lower than the 8 decimal representation used in calculation of the fee. Precision loss can result in a division by 0 leading to reverts in withdrawals and claims.

Finding Description: When the `oracleDecimals` is higher than 8 the returned price is reduced to 8 decimals for further use as `price8D`. The division rounds down in order to round up the fee amount. In case the value of the price is lower than 8 decimals it would however round down to 0.

The subsequent calculation of `minFeeUSD * 1e18 / price8D` would then revert because of a division by zero. This would then lead to claims and withdrawals failing.

Impact Explanation: Medium, because claims and withdrawals would revert until the fee has been set to 0 which shortcuts the fee calculation.

Likelihood Explanation: Low, because currently the lowest value L2 in terms of USD is in the range of 1e6 USD, so still above the critical value. In the future however popular L2s may have native tokens with lower value.

Recommendation: Calculate the return value as `minFeeUSD * 10 ** (10 + oracleDecimals) / price` to avoid precision loss and a possible division by zero.

Sablier: Fixed in PR 49.

Cantina Managed: Fix verified.

3.1.6 Hardcoding 24 hours as oracle heartbeat for native tokens of all chains may lead to incorrect fees

Severity: Low Risk

Context: SablierComptroller.sol#L398-L404

Summary: Hardcoding 24 hours as the oracle heartbeat for native token price feed on a chain, irrespective of the chain, may lead to accepting stale feeds on chains with lower heartbeats and therefore calculating incorrect fees.

Finding Description: `_calculateMinFeeWei()` uses Chainlink oracle price feed for converting the fees in USD to the chain's native token value. It calls `AggregatorV3Interface(oracle).latestRoundData()` and implements checks on the returned `price` and `updatedAt`. For staleness, it checks `block.timestamp - updatedAt > 24 hours`, which ignores the price and returns 0 if the oracle hasn't been updated in the last 24 hours.

While this check is correct for the target deployment chains of Scroll/Arbitrum/Gnosis/Linea/others where the heartbeat is 24 hours, it is much higher than the heartbeat values for other target deployment chains e.g. 3600s on Ethereum Mainnet and 1200s on Base

Therefore for chains with heartbeats lower than 24 hours, even if there is any oracle issue that delays price feed for the native token beyond the heartbeat, such stale and potentially incorrect values are accepted to calculate protocol fee for user actions.

Impact Explanation: Low, if we assume that the stale values will not deviate a lot from the actual value.

Likelihood Explanation: Low, if we assume that Chainlink oracles will typically not suffer delays/outages.

Recommendation: Consider:

1. Using the Chainlink-specified heartbeat value for the specific chain during deployment.
2. Using a privileged setter to modify this variable if/when this heartbeat value is changed by Chainlink.

Sablier: We have decided to continue using the hardcoded value of 24 hours for the heartbeat. If we set it to the chainlink specified heartbeat and if the chainlink does not update the price in time, we end up losing revenue as the function would return 0. Therefore, we would like to prioritise revenue over some discrepancies in the fee due to the stale price.

Cantina Managed: Acknowledged.

3.2 Informational

3.2.1 Using a single `FEE_MANAGEMENT_ROLE` across different protocol components is risky

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Protocol uses `FEE_MANAGEMENT_ROLE` as the role with the authority to update fee-related functionality across the Sablier contracts of Lockup, Flow and Airdrops. This role is authorized to call `disableAirdropsCustomFeeUSD()`, `disableFlowCustomFeeUSD()`, `disableLockupCustomFeeUSD()`, `setAirdropsCustomFeeUSD()`, `setAirdropsMinFeeUSD()`, `setFlowCustomFeeUSD()`, `setFlowMinFeeUSD()`, `setLockupCustomFeeUSD()` and `setLockupMinFeeUSD()`.

Using the same role across different protocol components is risky because if an account with this role is compromised then it can affect fee-related functionality across all the components.

Recommendation: Consider applying the "Privilege Separation" security design principle, which recommends to divide a system into components with distinct privilege levels, isolating high-privilege operations from less-privileged ones. For example, we can assign different roles to lockup, flow and airdrop components or different roles to set and disable fees.

Sablier: The single `FEE_MANAGEMENT_ROLE` across protocols is designed intentionally considering the specific and limited use case for this role.

Cantina Managed: Acknowledged.

3.2.2 A global `MAX_FEE_USD` may restrict flexibility and cause unexpected fees

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `SablierComptroller` enforces a global constant of `MAX_FEE_USD = 100e8` (USD $100) as the the maximum USD fee that can be set for claiming an airdrop or withdrawing from a stream/flow. This global `MAX_FEE_USD` may restrict flexibility for high-value airdrops/streams where it could be reasonable for the protocol to charge a higher fee to its clients.

While the protocol plans to set the fee between USD 1-2 initially, this global constant technically allows them to increase it to USD $100 across their new airdrops and new/existing streams, which can cause unexpected high fees thereafter for airdrop/stream recipients.

Recommendation: Consider:

1. Defining this constant value per airdrop/flow/lockup instead of a protocol-wide constant. This will allow greater flexibility without sacrificing the transparency/predictability of fee upper-bound for users.
2. Putting the fee setters behind a reasonable timelock so that impact of any custom fee hikes can be avoided to a certain extent by lockup/flow recipients.

Sablier: A global `MAX_FEE_USD` offers peace of mind to external integrators that fee in the contract will not exceed a certain value. Its sort of a business decision to design it this way.

Cantina Managed: Acknowledged.

3.2.3 Missing emit of `msg.sender` for `FEE_MANAGEMENT_ROLE` controlled functions

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: `FEE_MANAGEMENT_ROLE` controls all the fee management setter functions in `SablierComptroller`. While these functions emit events to log the changes, they do not log the `msg.sender` assuming that there is a single account with `FEE_MANAGEMENT_ROLE`. However, there may be more than one in which case this triggering account is missed in the event emits.

Recommendation: Consider emitting `msg.sender` for `FEE_MANAGEMENT_ROLE` controlled functions.

Sablier: Fixed in PR 47.

Cantina Managed: Fix verified.

3.2.4 Missing emit of previous values in `SablierComptroller` custom fee setters

Severity: Informational

Context: SablierComptroller.sol#L221-L240, SablierComptroller.sol#L259-L279, SablierComptroller.sol#L299-L318

Description: `setAirdropsCustomFeeUSD()`, `setFlowCustomFeeUSD()` and `setLockupCustomFeeUSD()` log the `sender` for whom the custom fee is being set and the new `customFeeUSD` value. However, they do not log the previous `customFeeUSD` value that is being overwritten. This deviates from other event emits which log both the old and new values.

Recommendation: Consider emitting the previous values in `SablierComptroller` custom fee setters for consistency and better logging.

Sablier: Fixed in PR 52.

Cantina Managed: Fix verified.

3.2.5 Comptroller's ability to only lower the fee can be circumvented by setting a malicious oracle

Severity: Informational

Context: SablierMerkleBase.sol#L148-L166

Summary: The comptroller is only allowed to lower the fee, but by maliciously setting an oracle that returns a lower price for the native token it could still increase the required minimum fee.

Finding Description: To provide trust to the airdrop projects, once the airdrop is created, the protocol only allows the comptroller to lower the fee expressed in USD, not increase it.

However when claiming, the comptroller is queried to convert the fee amount from USD to the chain's native token through an oracle set in the comptroller. The comptroller is authorized to set the oracle used for this conversion. This indirectly allows the comptroller to set an oracle that increases the fee amount expressed in native token.

Impact Explanation: High, because technically the fee could be set so high that fee imposed is unreasonably high compared to the user's claim. In the case of airdrops without expiration the funds could be economically locked in the contract.

Likelihood Explanation: Very low, because this would require malicious intent from the Sablier team which would harm their reputation.

Recommendation: The possible reputation damage may be enough for projects to not need any other mechanism to safeguard from malicious intent. As being able to change the oracle is also required to safeguard from oracle failure, a balance must be struck between providing trust between both parties and fault tolerance of the protocol.

Sablier: While it's technically possible, it's highly unlikely since these functions are only called by the admin. We will exercise the maximum caution when changing oracles.

Cantina Managed: Acknowledged.

3.2.6 Native tokens are assumed to have 18 decimals for fee calculation

Severity: Informational

Context: SablierComptroller.sol#L419-L420

Description: `_calculateMinFeeWei()`, which takes the fee in USD to calculate the equivalent fee in the native token, assumes that the native token has 18 decimals in its calculation. While this is correct for all currently targeted blockchains (Ethereum Mainnet, Arbitrum One, Avalanche, Base, BNB Smart Chain, Gnosis Chain, Linea, Optimism, Polygon and Scroll from the deployment scripts) for the protocol and is also assumed in Solidity for EVM chains (for Ether units), if this changes for any future supported blockchain then this calculation will be incorrect if used as-is.

Recommendation: Consider:

1. Making this a configurable deployment option.
2. Checking this assumption mandatorily before supporting any new blockchain.

Sablier: We acknowledge it but do not want to take any action. Until there exists a chain that does not use 18 decimals, we would like to continue using 18 decimals standard. In the unlikely event, that we find a chain operating with a different decimal format, we will adjust the contract accordingly and deploy a customized version.

Cantina Managed: Acknowledged.

3.2.7 Upgradeable Comptroller does not implement UUPS proxy best practices

Description: PR 54 implements an upgradeable Comptroller using UUPS proxy pattern, but does not implement UUPS proxy best practices.

Recommendation: Consider:

1. Using `openzeppelin/openzeppelin-contracts-upgradeable/contracts/proxy/utils/UUPSUpgradeable.sol` instead of `openzeppelin/contracts/proxy/utils/UUPSUpgradeable.sol`.
2. Using upgradeable version of `ERC165.sol`.
3. Removing `_initialize(initialAirdropMinFeeUSD, initialFlowMinFeeUSD, initialLockupMinFeeUSD, initialOracle);` from the constructor.
4. Adding missing calls to `__UUPSUpgradeable_init()` and `__ERC165_init()` in `initialize()`.
5. Renaming `_gap` to `__gap` and moving it from `RoleAdminable.sol` to `SablierComptroller.sol` with appropriate changes to the number of its slots.

Sablier: Fixed in PR 57.

Cantina Managed: Fix verified.