OpenZeppelin | security

Compound Streamer Audit

June 19, 2025

Table of Contents

Summary

Type	DeFi
Timeline	From 2025-06-11 To 2025-06-13
Total Issues	13 (8 resolved, 3 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	1 (0 resolved, 1 partially resolved)
Low Severity Issues	7 (4 resolved, 1 partially resolved)
Notes & Additional Information	5 (4 resolved, 1 partially resolved)

Scope

OpenZeppelin audited the woof-software/compound-streamer repository at commit a230960.

```
contracts
├── Streamer.sol
├── StreamerFactory.sol
└── interfaces
    ├── AggregatorV3Interface.sol
    ├── IStreamer.sol
    └── IStreamerFactory.sol
```

System Overview

The contracts under review implement a flexible, oracle-powered asset streaming mechanism designed to facilitate value transfer with reduced exposure to token price volatility. Originally developed to support Compound's funding arrangements, this infrastructure is now being proposed for broader use across the ecosystem through a publicly accessible deployment factory.

The `Streamer` contract enables the continuous distribution of a streaming asset, such as COMP or WETH, over a predefined duration, based on a target value denominated in a native asset like USDC or USD. The system relies on two price feeds, one for the native asset and one for the streaming asset, to dynamically determine how much of the streaming asset should be released over time to match the intended native asset value. This dual-asset architecture ensures that recipients receive a consistent value regardless of market fluctuations in the assets' price. For example, if the goal is to stream $100,000 worth of COMP over six months, the actual amount of COMP delivered will be adjusted according to its market price at each claim.

The `StreamerFactory` contract acts as a deployment mechanism for `Streamer` instances, allowing anyone to create new stream contracts in a standardized and secure way. This supports scalability while maintaining consistency in deployment logic.

Trust Assumptions and Privileged Roles

This review assumes that the price feeds used to quote the native and streaming assets are reliable and consistently return accurate data.

The only privileged role in the system is the **Stream Creator**, which has the ability to:

- initialize the stream after depositing the required initial balance
- terminate the stream early, subject to a minimum notice period

- sweep all streaming asset tokens from the Streamer before initialization or after the stream ends
- rescue ERC-20 tokens (other than the streaming asset) from the Streamer's balance

The stream recipient should be aware of these privileged actions available to the Stream Creator.

Medium Severity

M-01 Global State in Create2 Salt May Break Counterfactual Interactions

The `CREATE2` opcode enables counterfactual contract interactions by allowing contracts to be deployed at deterministic addresses, as outlined in EIP-1014. The `StreamerFactory` contract is designed to leverage `CREATE2` to deploy new `Streamer` instances at predictable addresses.

The `deployStreamer` function derives a `uniqueSalt` that is passed to `CREATE2`, enabling multiple deployments by the same `msg.sender` using identical constructor arguments. However, this salt incorporates a global `counter` that increments with each new `Streamer` deployment, regardless of who initiates it. Since the counter is shared across all users, any deployment can alter the expected outcome of a future `CREATE2` address computation. As a result, deterministic address calculation for counterfactual use cases becomes unreliable. For example, if a Compound proposal deploys a new `Streamer` instance and transfers funds to the precomputed deployment address, those funds will be lost if someone front-runs the deployment and increments the global `counter`, invalidating the original address computation.

Consider allowing the caller to specify a custom salt via a function argument. This would restore the ability to compute addresses deterministically and enable safe counterfactual interactions as intended.

***Update**: Partially resolved in commit `b0185a0` by introducing a per-deployer counter. This prevents users from interfering with each other, but there may still be a risk if a single user submits multiple deployments that are executed out of order. The WOOF team stated:*

> *During development, we decided to use a common counter instead of allowing the caller to pass custom salt in order to simplify the interaction with the Factory for users and DAO.*

Low Severity

L-01 Maximum Number of Decimals Is Not Checked

The documentation for the `Streamer` constructor states that token and price feed decimals should fall within the range of 6 to 18 to ensure accurate calculations. However, only a minimum decimals constraint is currently enforced in the code.

Consider also enforcing a maximum decimals constraint for both tokens and price feeds to align the implementation with the documented assumptions.

***Update**: Acknowledged, not resolved. The WOOF team stated:*

> *The comment is left as a suggestion for a maximum decimals allowed. Since a wide spread of decimals could have affected the calculation, we added a suggestion on the allowed decimals. However, we didn't explicitly disabled too high decimals values since they would revert the deployment due to "panic: arithmetic underflow or overflow (0x11)" in the streaming amount validation. Large decimals will also revert during initialization when a calculation of balance is performed.*
>
> *Thus, explicit validation was omitted in the constructor. Also, one of the reasons why we didn't add an explicit validation is that most tokens already do not exceed 18 decimals.*

L-02 Missing Event Emission When Stream Asset Balance Is Insufficient

The `claim` function of the `Streamer` contract transfers the accrued amount of the streaming asset to the recipient. If the contract lacks sufficient balance due to depreciation of the asset, it transfers the remaining balance instead. The stream owner must then replenish the funds to resume the stream. However, no event is emitted when the contract balance is insufficient to cover the full accrued amount. This omission makes such situations harder to detect and may result in extended periods where the recipient is unable to claim their full entitlement.

To improve observability and facilitate monitoring, consider emitting an event whenever the streaming asset balance is insufficient to fulfill the entire claim.

***Update**: Resolved in commit cae45d2.*

L-03 Incomplete `IStreamer` Interface

The `IStreamer` interface currently omits two `external` functions defined in the `Streamer` contract: `terminateStream` and `rescueToken`.

Consider adding the two aforementioned functions to the `IStreamer` interface to ensure consistency and completeness.

***Update**: Resolved in commit d6fcbe0.*

L-04 Misleading Documentation

The `slippage` state variable of the `Streamer` contract is used ([1], [2]) to reduce the streaming asset price in order to account for price fluctuations. However, its documentation currently states that it represents a flat percentage **added** to the asset price during calculations, which does not accurately reflect its actual behavior.

In order to improve clarity and avoid confusion, consider updating the documentation to accurately describe how slippage is applied.

***Update**: Resolved in commit 2d5b40f.*

L-05 Uninitialized Stream Allows Erroneous Owed Calculation

Within `Streamer.sol`, the `getNativeAssetAmountOwed` `public` `view` function does not ensure that the stream is initialized. If it is called before initialization (or if `terminateStream` is invoked prior to initialization), the `startTimestamp` remains at its default value of 0. This results in an overestimation of the elapsed time (`block.timestamp - 0`) and consequently an incorrect calculation of the accrued native asset amount, which may lead to severe miscalculation of the owed amount. This problem does not arise when `getNativeAssetAmountOwed` is called from the `claim` function as the `claim` function already has a check for a non-initialized stream.

Consider returning 0 from the `getNativeAssetAmountOwed` function when the contract is not initialized.

***Update**: Resolved in commit bb33188.*

L-06 Possible Precision Loss Due to Division Before Multiplication

Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

In `Streamer.sol`, multiple instances of division leading to precision loss were identified:

- In the `calculateStreamingAssetAmount` function, the `scaleAmount` function is called three times and `scaleAmount` performs division if `fromDecimals` is less than `toDecimals`. Another intermediate division happens in line 289. Moreover, in line 298, the `nativeAssetAmountInUSD` is divided by `10 ** streamingAssetDecimals` but is immediately multiplied by the same number in the next line. All these factors combined can lead to a loss of precision in the `calculateStreamingAssetAmount` function.
- In the `calculateNativeAssetAmount` function, `scaleAmount` is used two times before the final multiplication. More intermediate division happens in line 323 and line 331.

Performing multiplication before division is generally better to avoid loss of precision. As such, consider multiplying before dividing.

***Update**: Partially Resolved by using 18 decimals for all intermediate calculations in commit `4ae9764`.*

L-07 Floating Pragma

Pragma directives should be fixed to clearly identify the Solidity version with which the contracts will be compiled.

Throughout the codebase, multiple instances of floating pragma directives were identified:

- `Streamer.sol` has the `solidity ^0.8.29` floating pragma directive.
- `StreamerFactory.sol` has the `solidity ^0.8.29` floating pragma directive.
- `AggregatorV3Interface.sol` has the `solidity ^0.8.29` floating pragma directive.
- `IStreamer.sol` has the `solidity ^0.8.29` floating pragma directive.

- `IStreamerFactory.sol` has the `solidity ^0.8.29` floating pragma directive.

Consider using fixed pragma directives.

***Update**: Acknowledged, not resolved. The WOOF team stated:*

> *We will deploy the smart contract with the 0.8.29 version with which the contract has been tested.*

Notes & Additional Information

N-01 Tokens with Multiple Addresses May Be Withdrawn at Any Time

The `rescueToken` function of the `Streamer` contract allows the stream creator to transfer any ERC-20 token, except the streaming asset, from the contract to the `returnAddress`. However, if the streaming asset is a token with multiple addresses, the creator could use `rescueToken` to withdraw the token via an alternative address at any time, including before the stream ends.

Consider documenting the aforementioned behavior to caution users against selecting tokens with multiple addresses as the streaming asset, as it may undermine the intended guarantees of the stream.

***Update**: Resolved in commits 8740d29 and 0376483.*

N-02 Code Clarity

Throughout the codebase, multiple opportunities to improve code clarity were identified:

- The `initialize` function should use the `onlyStreamCreator` modifier instead of re-implementing its logic.
- The `streamEnd` value could be exposed via a `public` function to support external querying and reduce duplication.

- The `StreamState` enum variants are somewhat misleading. For example, the `state` variable is set to `ONGOING` when the stream starts but may remain unchanged even after the stream ends unless it is explicitly terminated early. The `TERMINATED` variant is used when the stream creator shortens the stream duration. More descriptive names like `STARTED` and `SHORTENED` may improve readability. Additionally, `state` could be made `internal` and accompanied by a `public` getter that introduces a derived `FINISHED` state when the stream has ended.
- The constructor of the `Streamer` contract and the `deployStreamer` function take many parameters, which could be grouped into structs to reduce the chance of ordering errors and enable more readable, named argument usage.

Consider implementing the above-listed improvements to enhance clarity, reduce duplication, and improve overall code quality.

***Update**: Partially resolved in commits 0376483 and 04cf54f. All recommendations were incorporated, except for the use of structs as constructor arguments.*

N-03 Redundant Code

Within the `deployStreamer` function of the `StreamerFactory` contract, two instances of redundant code were identified:

- Including `constructorParams` in the `CREATE2` salt is unnecessary, as they are already part of `bytecodeWithParams`.
- Explicitly precomputing the `newContract` address and checking its code length is redundant, as `Create2.deploy` already returns the deployed address and reverts if the contract already exists.

Consider removing any redundant logic to improve the readability and maintainability of the codebase.

***Update**: Resolved in commit 660002c.*

N-04 Parameter Naming Mismatch between Implementation and Interface

Within `StreamerFactory.sol`, in line 34, the function `deployStreamer` defines a parameter called `_sweepCooldown` while the `IStreamerFactory` interface defines the corresponding parameter as `finishCooldown`. This naming inconsistency can lead to

confusion when integrating or interacting with the contract because the parameter's intended use may be misinterpreted.

Consider harmonizing the parameter names in both the implementation and the interface so that the documentation, intended behavior, and actual code are aligned.

***Update**: Resolved in commit 177ecb6.*

N-05 Lack of Security Contact

Providing a specific security contact (such as an email or ENS name) within a smart contract significantly simplifies the process for individuals to communicate if they identify a vulnerability in the code. This practice is quite beneficial as it permits the code owners to dictate the communication channel for vulnerability disclosure, eliminating the risk of miscommunication or failure to report due to a lack of knowledge on how to do so. In addition, if the contract incorporates third-party libraries and a bug surfaces in those, it becomes easier for their maintainers to contact the appropriate person about the problem and provide mitigation instructions.

Throughout the codebase, multiple instances of contracts not having a security contact were identified:

- The `Streamer` contract
- The `StreamerFactory` contract

Consider adding a NatSpec comment containing a security contact above each contract definition. Using the `@custom:security-contact` convention is recommended as it has been adopted by the OpenZeppelin Wizard and the ethereum-lists.

***Update**: Resolved in commit 1e3d694.*

Conclusion

The Compound Streamer system enables users to stream stable values in volatile tokens. For example, using this system, anyone can stream, say 500 USD a month to a recipient in COMP tokens. The price change of COMP would not affect the value streamed.

One medium-severity issue was identified during the review, which prevents users from reliably precomputing the deployment addresses of new `Streamer` instances. Additionally, several recommendations were made to improve code readability and clarity. Overall, the codebase was found to be well-written and secure.