Sablier Flow Audit

31/10/2024

Contents

Protocol Audit Report

Version 1.0

Prepared by: CasinoCompiler

Lead Auditors: CC

Protocol Summary

Flow is a debt tracking protocol that tracks tokens owed between two parties, enabling open-ended payment streaming.

A Flow stream is characterized by its **rate per second (rps)**. **Rate per second** is defined as the rate at which tokens are increasing by the second to the recipient. The relationship between the amount owed and time elapsed is linear.

The Flow protocol can be used in several areas of everyday finance, such as payroll, distributing grants, insurance premiums, loans interest, token ESOPs etc.

Disclaimer

The CC team makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit by the team is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity.

Audit Details

Scope

```
./src/
>FlowNFTDescriptor.sol
>SablierFlow.sol
=>abstracts/
   > Adminable.sol
   > Batch.sol
   > NoDelegateCall.sol
   > SablierFlowBase.sol
=>libraries/
   > Errors.sol
   > Helpers.sol
=>types/
    > DataTypes.sol
```

Roles

1. Protocol Admin: Protocol admin has the ability to set protocol fee, collect protocol revenue, recover surplus tokens and set NFT descriptor. Admin should not be able to change a stream's parameters.
2. Sender: The creator of the stream. Sender has the ability to change rps, pause, restart, void and refund from the stream. Sender can also withdraw from the stream as long as the `to` address is set to the recipient.
3. Recipient: The receiver of the stream. Recipient can withdraw streamed tokens to an external address. Recipient also has the ability to void the stream.
4. Unknown user: Anyone who is not the protocol admin, a sender or a recipient is considered as unknown. Unknown user has the ability to deposit into any stream. They can also withdraw from any stream as long as the `to` address is set to the recipient.

Executive Summary

Flow protocol by Sablier demonstrates State of The Art smart contract production, best practices as well as an extremely thorough and exhaustive test suite. Overall, it was extremely difficult to find any bugs that break core functionality given the assumptions and known issues highlighted beforehand. As such, the only vulnerability found was regarding the NFT functionality and being able to transfer streams that are in a non-flowing state. The recommendations have been provided and are not particularly difficult to implement.

Issues found

Severity	Number of Issues Found
High	0
Medium	1
Low	0
Informational	0
Gas	0
Total	1

Findings

Medium

[M-1] Missing critical stream information in NFT metadata could lead to misleading sales, particularly voided streams.

Description: The Sablier Flow protocol mints NFTs representing streams which can be made transferable. Currently, although the NFT has not been implemented yet, `IFlowNFTDescriptor.sol::tokenURI()` suggests only the `streamId` and the `sablierFlow` contract address will be used in the metadata; critical underlying information such as stream status, rate or balance will not be updated. This could lead to users purchasing NFTs representing depleted, void or paused streams on NFT marketplaces without understanding the true state of the stream.

Impact: - NFT marketplaces typically display only NFT metadata, making due diligence harder: - NFT buyers may purchase stream NFTs without understanding the actual value/status of the underlying stream. - Financial loss possible if NFTs are sold at prices not reflecting actual stream status - Trust issues for the protocol if stream NFTs are misused: - It would become social norm to avoid interacting with Flow NFTs, thus making the whole NFT functionality useless.

Proof of Concept: This PoC will be using a voided stream example as this scenario is the most impactful one and can be easily recreated.

1. Bad actor creates a new stream through a stream providing service.
2. Bad actor receives NFT for stream.
3. Bad actor has voided the stream intentionally immediately.
4. Bad actor sells NFT to buyer who was unaware of void function for streams.

As the access controls for voiding a stream can be done by the Sender/Recipient/Administrator, multiple different combinations of manipulation attacks could occur e.g the bad actor can create a stream themselves(sender) to themselves(recipient), void and list NFTs repeatedly.

Proof of Code Create the following test contract:

```solidity
// SPDX-License-Identifier: UNLICENSED

pragma solidity >=0.8.22;

import { Integration_Test } from "../Integration.t.sol";
import { UD21x18, ud21x18 } from "@prb/math/src/UD21x18.sol";
import { Flow } from "src/types/DataTypes.sol";
import { IERC721 } from "@openzeppelin/contracts/interfaces/
    IERC721.sol";

contract NFTTest is Integration_Test {
address streamProviderService = makeAddr("streamProviderService");
address badActor = makeAddr("badActor");
address userOne = makeAddr("userOne");

function setUp() public override {
// Call parent setUp which sets up admin, users, tokens etc
super.setUp();
}

function testMisleadingNFTSale() public {
UD21x18 initialRate = ud21x18(uint128(RATE_PER_SECOND.unwrap()));
uint256 block_i = block.number;

// streamProviderService creates transferable stream to badActor
    in block_i
vm.warp(block_i);
vm.startPrank(streamProviderService);
uint256 streamId = flow.create({
sender: streamProviderService,
recipient: badActor,
ratePerSecond: initialRate,
token: dai,
transferable: TRANSFERABLE
});
vm.stopPrank();

// badActor voids the stream in the next block
vm.warp(block_i + 1);
vm.startPrank(badActor);
flow.void(streamId);
vm.stopPrank();

// Assert stream is voided
// NOTE:: tokenId is 2 as this test inherits Integration_Test
    which creates previous streams.
// NOTE :: 4 == flow.status.VOIDED
assertEq(uint256(flow.statusOf(2)), 4);

// Assert NFT still exists and it owned by badActor
assertEq(flow.ownerOf(streamId), badActor);

// Simulate someone buying voided NFT :: transfer NFT to userOne
// After many days (10 days = 71680 blocks)
vm.warp(block_i + 71680);
vm.startPrank(badActor);
IERC721(flow).transferFrom(badActor, userOne, 2);
```

```
55  vm.stopPrank();
56
57  // Assert userOne owns voided stream NFT
58  assertEq(flow.ownerOf(streamId), userOne);
59  }
60  }
```

Recommended Mitigation: 1. Include critical stream information in dynamic NFT metadata format. 2. Consider making NFTs non-transferable when a stream is paused / depleted / void.