Sablier V2 Audit Report

Version 1.0

palmcivet

May 31, 2024

Sablier V2 Audit Report

palmcivet

May 31st, 2024

Prepared by: palmcivet

Table of Contents

Protocol Summary

Sablier is a permissionless token distribution protocol for ERC-20 assets. It can be used for vesting, payroll, airdrops, and more.

The sender of a payment stream first deposits a specific amount of ERC-20 tokens in a contract. Then, the contract progressively allocates the funds to the stream recipient, also known as the Sablier NFT owner, who can access them as they become available over time. The payment rate is influenced by various factors such as the start time, the end time, the total amount of tokens deposited and the type of stream.

Disclaimer

The `palmcivet` 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

We use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

- Commit Hash: 43d7e752a68bba2a1d73d3d6466c3059079ed0c6

Scope

v2-core

```
src
#-- abstracts
|   #-- Adminable.sol - A minimalist implementation to handle admin access
|   #-- NoDelegateCall.sol - A minimalist implementation to prevent delegate calls
|   #-- SablierV2Lockup.sol - Handles common logic between all Sablier V2 Lockup contracts
#-- libraries
|   #-- Errors.sol - Library containing all custom errors used in the Sablier protocol
|   #-- Helpers.sol - Helpers to calculate and validate input data required to create streams
|   #-- NFTSVG.sol - Library to generate NFT SVG
|   #-- SVGElements.sol - Library to generate specific components of NFT SVG
#-- types
|   #-- DataTypes.sol - Implementation for a set of custom data types used in V2 core
#-- SablierV2LockupDynamic.sol - Creates and manages Lockup streams with a dynamic distribution function
#-- SablierV2LockupLinear.sol - Creates and manages Lockup streams with a linear distribution function
#-- SablierV2LockupTranched.sol - Creates and manages Lockup streams with a tranched distribution function
#-- SablierV2NFTDescriptor.sol - Generates the URI describing the Sablier V2 stream NFTs
```

v2-periphery

```
src
#-- abstracts
|   #-- SablierV2MerkleLockup.sol - Handles common logic between all Airstream campaigns
#-- libraries
|   #-- Errors.sol - Library containing all custom errors used in the Sablier protocol
#-- types
|   #-- DataTypes.sol - Implementation for a set of custom data types used in V2 periphery
#-- SablierV2BatchLockup.sol - Helpers to batch create Sablier V2 Lockup streams
#-- SablierV2MerkleLL.sol - Allows users to claim Airdrops using Merkle proofs. These airdrops are powered by Lockup Linear streams
#-- SablierV2MerkleLT.sol - Allows users to claim Airdrops using Merkle
     proofs. These airdrops are powered by Lockup Tranched streams
#-- SablierV2MerkleLockupFactory.sol - Factory for deploying Airdrop
    campaigns using CREATE
```

Roles

There are three roles assumed by actors in the Sablier protocol:

Recipient

Users who are the recipients of the streams. These users own the Sablier NFT which grants them the right to withdraw assets from the stream.

Sender

Users who create streams and are responsible for funding them. Senders are also authorized to cancel and renounce streams. These users can also trigger withdrawals on behalf of the recipients but only to the recipient's address.

Unknown caller

These are callers who are neither Sender nor Recipient but are allowed to trigger withdrawals on behalf of the recipients. This is because the withdraw function is publicly callable. Note that an unknown caller can withdraw assets only to the recipient's address.

Executive Summary

This was my first competitive security review on Codehawks. Sablier was a really well written codebase with excellent natspec and invariant tests. I enjoyed reading through it and struggled to find many issues. This report contains the submissions I made.

Issues found

Severity	Number of issues found
High	0
Medium	1
Low	1
Informational	0
Total	2

Findings

Medium

[M-1] ERC20s with pausable, upgradable or blocklist features may disrupt functionality

Description The Sablier README acknowledges the issues with rebased tokens and fee-on-transfer tokens. There are however, popular tokens, such as USDC or USDT, that have functionality which could severely disrupt the functionality of Sablier. Functionality such as pausable, blocklists and upgradability stem from the token issuer's centralised control.

Impact Senders may be unable to create streams and recipients may be unable to withdraw tokens they are owed.

Proof of Concepts

Code

Place the following code in a Foundry Forge test file.

```
pragma solidity >=0.8.22;

import { Test } from "../../lib/forge-std/src/Test.sol";
import { PausableMockERC20, IERC20 } from "./PausableMockERC20.sol";
import { SablierV2LockupLinear } from "../../src/SablierV2LockupLinear.
    sol";
import { SablierV2NFTDescriptor } from "../../src/
    SablierV2NFTDescriptor.sol";
import { ISablierV2NFTDescriptor } from "../../src/interfaces/
    ISablierV2NFTDescriptor.sol";
import { Lockup, LockupLinear, Broker } from "../../src/types/DataTypes
    .sol";
import { wrap } from "@prb/math/src/ud60x18/Casting.sol";

```

```
contract BlacklistTest is Test, Events {
    PausableMockERC20 token;
    SablierV2LockupLinear sablier;
    SablierV2NFTDescriptor nftDescriptor;

    address sender = makeAddr("sender");
    address recipient = makeAddr("recipient");
    address tokenController = makeAddr("tokenController");
    address sablierAdmin = makeAddr("sablierAdmin");

    uint256 ONE_THOUSAND_TOKENS = 1000 * 1e18;

    LockupLinear.CreateWithTimestamps params;

    function setUp() public {
        nftDescriptor = new SablierV2NFTDescriptor();
        sablier = new SablierV2LockupLinear(sablierAdmin,
            ISablierV2NFTDescriptor(nftDescriptor));

        vm.startPrank(tokenController);
        token = new PausableMockERC20();
        token.mint(ONE_THOUSAND_TOKENS);
        token.transfer(sender, ONE_THOUSAND_TOKENS);
        vm.stopPrank();

        // Define the parameters for createWithTimestamps
        params = LockupLinear.CreateWithTimestamps({
            sender: sender,
            recipient: recipient,
            totalAmount: uint128(ONE_THOUSAND_TOKENS),
            asset: IERC20(token),
            cancelable: true,
            transferable: false,
            timestamps: LockupLinear.Timestamps({
                start: uint40(block.timestamp), // Stream starts now
                cliff: uint40(block.timestamp + 60), // Cliff period
                    ends in 60 seconds
                end: uint40(block.timestamp + 3600) // Stream ends in 1
                     hour
             }),
            broker: Broker({ account: address(0), fee: wrap(0) })
        });
    }

    function test_blacklist() public {
        // stream recipient is blacklisted
        vm.prank(tokenController);
        token.addToBlacklist(recipient);

        // create stream
        vm.startPrank(sender);
        token.approve(address(sablier), ONE_THOUSAND_TOKENS);
        sablier.createWithTimestamps(params);
        vm.stopPrank();

        // Set block timestamp to the end time of the stream
        vm.warp(params.timestamps.end);

        // expect withdraw to revert because recipient is blacklisted
        vm.expectRevert();
        sablier.withdraw(1, recipient, uint128(ONE_THOUSAND_TOKENS));
    }

    function test_pausableToken() public {
        // create stream
        vm.startPrank(sender);
        token.approve(address(sablier), ONE_THOUSAND_TOKENS);
        sablier.createWithTimestamps(params);
        vm.stopPrank();

        // token is paused
        vm.prank(tokenController);
        token.pause();

        // Set block timestamp to the end time of the stream
        vm.warp(params.timestamps.end);

        // expect withdraw to revert because token is paused
        vm.expectRevert();
        sablier.withdraw(1, recipient, uint128(ONE_THOUSAND_TOKENS));
    }
```

Recommended Mitigation Consider adding a disclaimer for users to be aware of the functionality of tokens they are using for payment streams.

Low

[L-1] Streams can be created with ERC721s as the underlying payment asset

Description When a sender creates a payment stream, they are intended to specify the address of an ERC20 token, which the streamed payment will be in. The README states that the Sablier Protocol is `not compatible with any token standard other than ERC20`. However it is possible for a sender to create a payment stream by specifying an ERC721 as the underlying payment asset, if they also pass the ERC721's tokenID as the `params.totalAmount`.

Impact Senders who create streams with ERC721s as the underlying payment asset will lose access to their ERC721 token and cause confusion to recipients who observed events such as `CreateLockupLinearStream` being emitted.

Proof of Concept

Code

Place the following code in a Foundry Forge test file.

```solidity
import { Events } from "../utils/Events.sol";
.
.
.
function test_stream_nft() public {
    vm.startPrank(sender);
    TestNft nft = new TestNft();
    address nftAddress = address(nft);

    // Define the parameters for createWithTimestamps
    params = LockupLinear.CreateWithTimestamps({
        sender: sender,
        recipient: recipient,
        totalAmount: uint128(1),
        asset: IERC20(nftAddress),
        cancelable: true,
        transferable: true,
        timestamps: LockupLinear.Timestamps({
            start: uint40(block.timestamp), // Stream starts now
            cliff: uint40(block.timestamp + 60), // Cliff period ends in 60 seconds
            end: uint40(block.timestamp + 3600) // Stream ends in 1 hour
         }),
        broker: Broker({ account: address(0), fee: wrap(0) })
    });

    // create stream and expect emit
    nft.approve(address(sablier), 1);
    vm.expectEmit({ emitter: address(sablier) });
    emit CreateLockupLinearStream(
        1,
        sender,
        sender,
        recipient,
        Lockup.CreateAmounts({ deposit: 1, brokerFee: 0 }),
        IERC20(nftAddress),
        true,
        true,
        LockupLinear.Timestamps({
            start: uint40(block.timestamp),
            cliff: uint40(block.timestamp + 60),
            end: uint40(block.timestamp + 3600)
        }),
        address(0)
    );
```

```
        sablier.createWithTimestamps(params);
        vm.stopPrank();

        // assert sablier is now the owner of the nft
        assertEq(nft.ownerOf(1), address(sablier));

        // Set block timestamp to the end time of the stream
        vm.warp(params.timestamps.end);

        // recipient cannot withdraw the "payment" nft locked in sablier
        vm.prank(recipient);
        vm.expectRevert();
        sablier.withdraw(1, recipient, uint128(1));
    }
.
.
.
import { ERC721 } from "@openzeppelin/contracts/token/ERC721/ERC721.sol";

contract TestNft is ERC721 {
    address private immutable owner;

    constructor() ERC721("Test NFT", "TNFT") {
        owner = msg.sender;
        _mint(msg.sender, 1);
    }
}
```

Recommended Mitigation Consider checking the `params.asset` has decimals() (or something else unique to ERC20s that ERC721s do not have).

```
+   function isERC20(address token) internal view returns (bool) {
+       (bool success, bytes memory data) = token.staticcall(
+           abi.encodeWithSignature("decimals()")
+       );
+       return success && data.length > 0;
+   }

+   modifier onlyERC20(address token) {
+       require(isERC20(token), "Token is not an ERC20 token");
+       _;
+   }

    function createWithTimestamps(LockupDynamic.CreateWithTimestamps calldata params)
        external
        override
        noDelegateCall
+       onlyERC20(address(params.asset))
```

```
        returns (uint256 streamId)
    {
        // existing code
    }
```