HYDN

Smart Contract Security Assessment

April 03, 2023

Project Name:

Sablier

Prepared by:

HYDN

Contents

Introduction

About HYDN

HYDN is an industry leader in blockchain security and smart contract audits. Founded by Warren Mercer, a world renowned cybersecurity and blockchain expert, who has previously held senior roles at NYSE, Cisco, and Alert Logic. Having been involved in cryptocurrency for over 10 years, Warren is dedicated to making the blockchain ecosystem as secure as it can be for everyone. Warren serves as the CEO for HYDN and heads up the delivery team to ensure that work is carried out to the highest standard.

The HYDN delivery team has over 10 years combined experience in blockchain, having performed smart contract audits and built security systems for a large range of protocols and companies. HYDN have performed smart contract security services for the likes of SpookySwap, Swapsicle, Nau Finance, CrossWallet, Dancing Seahorse, Octane, Position Exchange, and more.

HYDN worked closely with Sablier to consider their unique business needs, objectives, and concerns. Our mission is to ensure the blockchain supports secure business operations for all and he has built the team at HYDN from the ground up to meet this very personal objective.

To keep up to date with our latest news and announcements, check out our website https://hydnsec.com/ or follow @hydnsecurity on Twitter.

About Sablier

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.

Sablier is compatible with crypto assets like DAI, USDC and most other ERC20 tokens. Their app offers a portal into the world of real-time finance, powered by permissionless smart contracts

Methodology

When tasked with conducting a security assessment, HYDN works through multiple phases of security auditing to ensure smart contracts are audited thoroughly. To begin the process automated tests are carried out, before HYDN then moves onto carrying out a detailed manual review of the smart contracts.

HYDN uses a variety of open-source tools and analyzers as and when they are required and alongside this, HYDN primarily focuses on the following classes of security and reliability issues:

Basic Coding Mistakes

One of the most common causes of critical vulnerabilities is basic coding mistakes. Countless projects have experienced hacks and exploits due to simple, surface level mistakes that could have been flagged and addressed by a simple code review. The HYDN automated audit process which includes model checkers, fuzzers, and theorem provers analyses the smart contract for many of these basic coding mistakes. Once the automated audit has taken place, HYDN then performs a manual review of the code to gain familiarity with the contracts.

Business Logic Errors

HYDN reviews the platform or projects design documents before analysing the code to ensure that the team has a deep understanding of the business logic and goals. Following this, HYDN reviews the smart contracts to ensure that the contract logic is in line with the expected functionality.

HYDN also analyses the code for inconsistencies, flaws, or vulnerabilities which could impact business logic such as Tokenomics errors, arbitrage opportunities, and share pricing.

Complex Integration Risks

Several high-profile exploits have been the result of not any bug within the contract itself, but rather an unintended consequence of its interaction with the broader DeFi ecosystem.
We perform a meticulous review of all of the contract's possible external interactions, and summarise the associated risks; for example: flash loan attacks, oracle price manipulation, MEV/sandwich attacks, etc.

Code Maturity

HYDN reviews the smart contracts for potential improvements in the codebase. These improvements ensure that the code follows industry best practices and guidelines, or code quality standards. Alongside this, HYDN makes suggestions for code optimization items such as gas optimization, upgradeability weaknesses, centralization risks, and more.

Impact Rankings

Once HYDN has completed the security assessment, each issue is assigned an impact rating. This impact rating is based upon both the severity and likelihood of the issue occurring. Each security assessment is different and the business needs and goals, such as project timelines, and Sablier threat modelling. are taken into account when the impact rankings are created.

HYDN assigns the following impact rankings: Critical, High, Medium, Low (listed by severity).

Disclaimer

This HYDN security assessment does not provide any warranties on finding all possible issues within the scope and the evaluation results do not guarantee the absence of any issues. **HYDN cannot make any guarantees on any further code which is added or altered after the security assessment has taken place**. As a single security assessment can never be considered fully comprehensive, HYDN always recommends multiple independent assessments paired with a bug bounty program. This security assessment report should not be considered as financial or investment advice.

Summary of Findings

Executive Summary

As part of this audit, HYDN was tasked with performing a Smart Contract Audit on the Sablier v-2 core contracts. In general, HYDN found the quality and security of the contracts to be extremely high with only Low severity issues found.

Project Summary

Platform	Ethereum
Language	Solidity
Scope	Sablier v2-core
Repository	https://github.com/sablierhq/v2-core
Initial Commits	8bd57ebb31fddf6ef262477e5a378027db8b85d8
Latest Commits	b343c60823e4f7a3e7b7be6af686fabb876f704f

Audit Summary

Delivery Date	April 03 2023
Audit Method	Static Automated Analysis + Manual Review

Breakdown of Findings

ID	Title	Severity	Status
1	**Weak Hook Validation**	Low	Resolved
2	**Flashloan Design Could Be Safer**	Low	Acknowledged
3	**Code Optimization**	Low	Resolved
4	**Code Mutualization**	Low	Resolved

Detailed Findings

1 Weak Hook Validation [Low]

1.1 Description

Hooks are called during modifications to the stream, affecting either the sender or the recipient of the stream. An attacker could exploit this feature by creating a stream intended to trigger a specific function on a target contract.

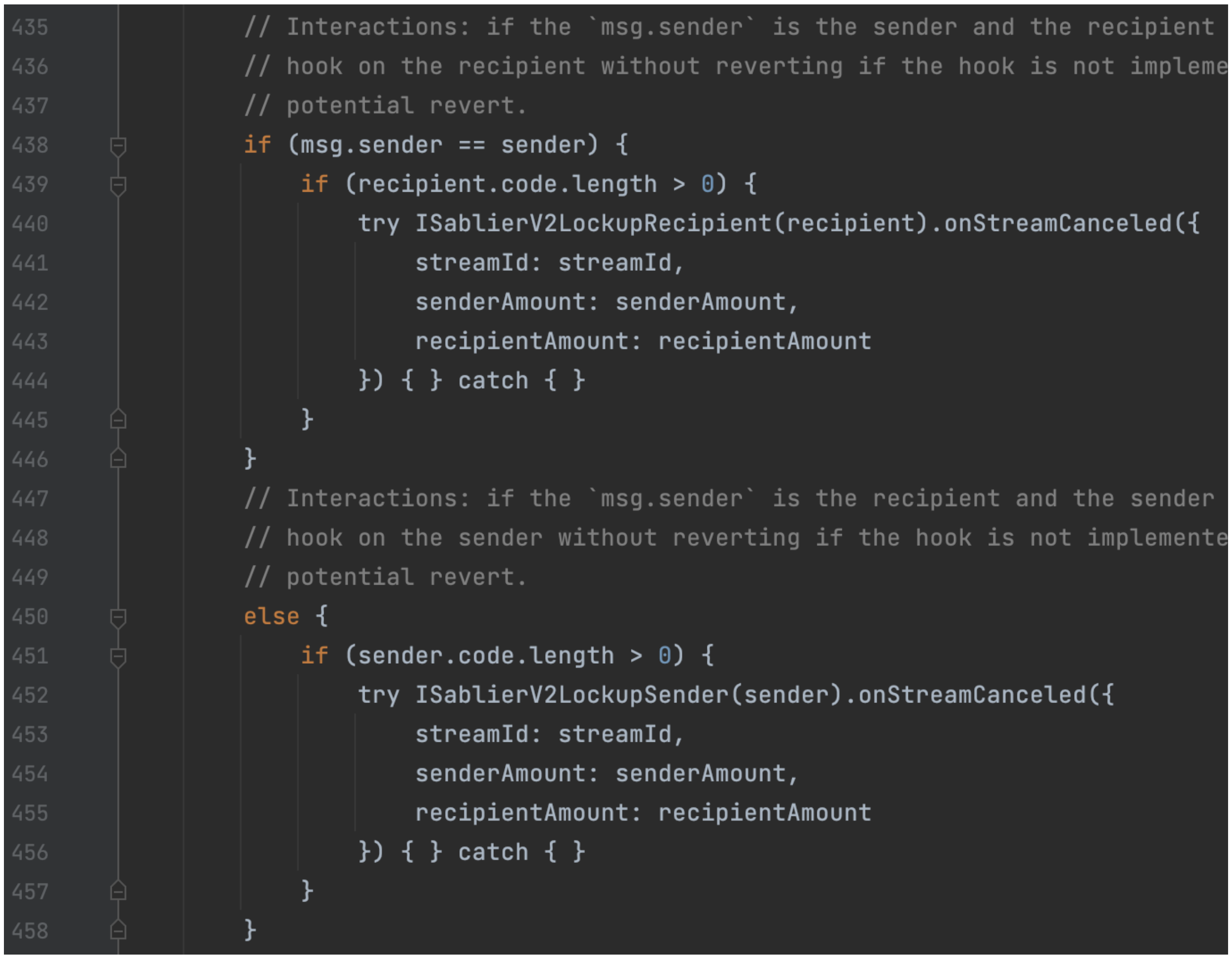
Although the SablierV2Lockup contract itself is not vulnerable, it places any sender/recipient implementing hooks in a risky context, where they need to implement additional validation.

In a straightforward scenario, where a third party only manages a particular streamId, there are no issues. However, in other cases, it will be necessary to perform an on-hook call to the lookup contract, retrieve the complete stream details, and verify the sender and recipient values to ensure security.

onStreamCanceled:

An attacker could force the stream to trigger calls on a targeted contract:

- As a recipient on the stream
- As a sender of the stream

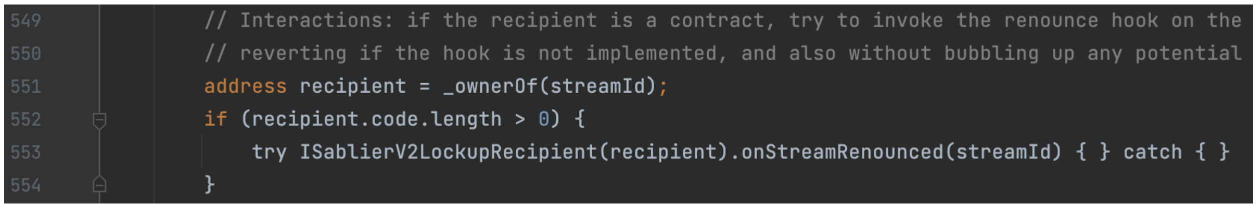


```
        // Interactions: if the `msg.sender` is the sender and the recipient
        // hook on the recipient without reverting if the hook is not impleme
        // potential revert.
        if (msg.sender == sender) {
            if (recipient.code.length > 0) {
                try ISablierV2LockupRecipient(recipient).onStreamCanceled({
                    streamId: streamId,
                    senderAmount: senderAmount,
                    recipientAmount: recipientAmount
                }) { } catch { }
            }
        }
        // Interactions: if the `msg.sender` is the recipient and the sender
        // hook on the sender without reverting if the hook is not implemente
        // potential revert.
        else {
            if (sender.code.length > 0) {
                try ISablierV2LockupSender(sender).onStreamCanceled({
                    streamId: streamId,
                    senderAmount: senderAmount,
                    recipientAmount: recipientAmount
                }) { } catch { }
            }
        }
```

Extract of SablierV2LockupDynamic.sol (logic is also in SablierV2LockupLinear.sol)

onStreamRenounced:

An attacker could force the stream to trigger calls on a targeted contract as a recipient of the stream.



```
        // Interactions: if the recipient is a contract, try to invoke the renounce hook on the
        // reverting if the hook is not implemented, and also without bubbling up any potential
        address recipient = _ownerOf(streamId);
        if (recipient.code.length > 0) {
            try ISablierV2LockupRecipient(recipient).onStreamRenounced(streamId) { } catch { }
        }
```

Extract of SablierV2LockupDynamic.sol (logic is also in SablierV2LockupLinear.sol)

1.2 Impact

An attacker could take advantage of the hook target contract logic in a case where the input call validation is not properly asserted by the recipient's third party contract. **It must be noted that this issue does not impact Sablier's contract, but could place the hook recipient third party in a risky position.**

1.3 Recommendation

The expected scenario is the caller of cancel() will be the sender or the recipient and for renounce it would only be the sender.

It would make life easier and safer by adding initiator parameters into the call hook, then the hook recipient would be able to check who is the original caller of the current transaction.

1.4 Remediation Completed

Following further discussions with the Sablier team, HYDN is happy with the team's decision regarding this issue. Sablier noted that attackers can create fake streams that target particular recipient contracts, but that it is impossible for Sablier to prevent this. The onus is on the recipient contract's developer to store streamIds, and validate that the provided streamId references a known, genuine stream. Attackers cannot impersonate other users' streams.

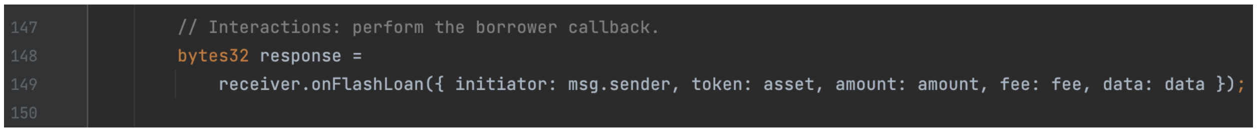
Sablier followed the recommendation to include the original "msg.sender" in the hook, but for a different reason. This reason is developer experience as having access to the original caller may prove useful for third-party integrations.

2 Flashloan Design Could Be Safer [Low]

2.1 Description

With the current design, anyone can request a flash loan to another recipient contract. This feature offers maximum flexibility, but also comes with some risks.

An improper implementation puts any implementer of the flash loan at risk.



```
        // Interactions: perform the borrower callback.
        bytes32 response =
            receiver.onFlashLoan({ initiator: msg.sender, token: asset, amount: amount, fee: fee, data: data });

```

Extract of SablierV2FlashLoan.sol

2.2 Impact

In situations where the initiator call parameter is not adequately validated, an attacker could exploit the flash loan recipient contract logic.

Depending on the receiver logic implementation, the attacker could potentially deplete their balance in the worst-case scenario. Another outcome occurs when there is a pending or infinite approval to the Sablier lockup, resulting in a balance drain that benefits Sablier in the form of fees. In this case, an attacker would request a flash loan with the target as the receiver, and then the Lockup would pull back the flash loan amount along with the associated fees.

2.3 Recommendations

EIP-3156 Flash Loans are widely adopted, similar to Aave-style flash loans, and are considered safer than Balancer-style loans, which lack the initiator address parameter. Uniswap v3 Flash Swap offers an even safer pattern, where the receiver must be the initiator of the flash loan. While this reduces modularity, it significantly mitigates risks for the requester.

Although EIP-3156 is a solid implementation, it exposes a security vulnerability on the requester side. Inadequate implementation or weak validation may result in heightened risks for the implementer.

In practical applications, it is highly likely that the caller will also be the receiver.

2.4 Remediation Completed

Sablier has acknowledged that in most cases, the caller and the receiver of a flash loan are the same, and therefore, technically, they could make this a hard requirement.

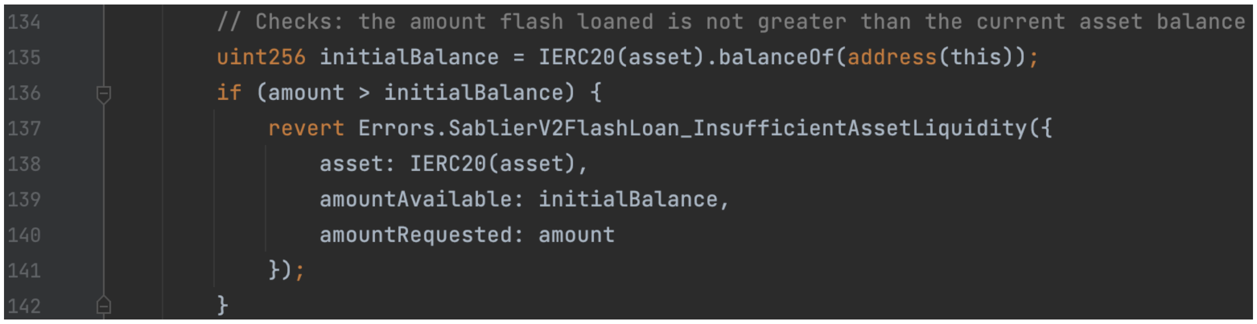
However, Sablier wants to stick with ERC-3156, so will accept this as a risk of the flash loan feature, which they will mitigate by raising awareness through their documentation site.

HYDN has accepted this decision by Sablier.

3 Code Optimization [Low]

3.1 Description

The Flash Loan logic includes a balance check to ensure that the lockup balance is sufficient to cover the requested flash loan amount. This validation appears to be excessive, as in cases where the balance is insufficient, the transfer of the asset will revert the flash loan call request.



```
        // Checks: the amount flash loaned is not greater than the current asset balance
        uint256 initialBalance = IERC20(asset).balanceOf(address(this));
        if (amount > initialBalance) {
            revert Errors.SablierV2FlashLoan_InsufficientAssetLiquidity({
                asset: IERC20(asset),
                amountAvailable: initialBalance,
                amountRequested: amount
            });
        }
```

Extract of SablierV2FlashLoan.sol

3.2 Impact

Unnecessary checks can lead to wasted gas consumption.

3.3 Recommendations

It seems to be safe to rely on the token's internal balance accounting, and just fully removing this check will reduce the gas cost of the flash loan.

3.4 Remediation Completed

Sablier accepted this recommendation and has implemented the change.

4 Code Mutualization [Low]

4.1 Description

Lockup Dynamic and Lockup Linear inherit from the same abstract contract, SablierV2Lockup. Multiple parts are duplicated in both inherited contracts and can be moved directly into SablierV2Lockup.

4.2 Impact

A variety of impacts depending upon the recommendation (all of Low severity).

4.3 Recommendations

HYDN recommends the following changes:

4.3.1 - _nextStreamId slot variable can be moved

4.3.2 - ERC721 heritage can be moved up (pass through the constructor parameters)

4.3.3 - After ERC721 moves, getRecipient(), tokenURI(), _isApprovedOrOwner(), can be moved. _burn() will not be necessary

4.3.4 - withdrawableAmountOf() can be moved

4.3.5 - renounce() and _cancel() share a large part of the codebase, asset transfer + hook call. A specific contract can modify state, grab all necessary data from the stream storage and pass it to the abstract contract

4.4 Remediation Completed

Following discussions with Sablier, recommendations **4.3.2 and 4.3.2** have been implemented whilst the remaining three have been acknowledged, but do not require a change currently.

HYDN has accepted Sablier's response to these three recommendations.