29 JULY 2019

The Livepeer slashing vulnerability

tl;dr

Improper input validation meant that the same claim could be submitted twice as proof of transcoder double claiming. This would have resulted in loss of funds as well as denial of service of the Livepeer network.

What is Livepeer?

Livepeer is a platform for streaming live video over the internet. One of the ways it facilitates this is by operating a marketplace where GPU time for transcoding videos can be bought and sold.

This marketplace roughly works as follows. Transcoders who want to sell their GPU time must stake Livepeer Tokens (LPT) and/or have others stake LPT tokens for them. If they're in the list of top 25 transcoders by tokens staked, then they're eligible to receive work through the marketplace.

When a broadcaster wants to buy GPU time to transcode a video, they submit a job through the smart contract system. An active transcoder is randomly matched with this job, and the broadcaster can coordinate with the selected transcoder off-chain.

The broadcaster breaks down the video they want transcoded into small chunks, called *segments*. As the transcoder receives segments, they must record on-chain that they've begun work on those segments by submitting a *claim*. Each claim can only contain sequential segments, and certain segments are randomly subjected to further verification by the Livepeer network.

In order to ensure that malicious transcoders are held accountable, any third party can *slash* transcoders if certain conditions are met. If a transcoder is slashed, a portion of their staked tokens will be burnt, and they'll be removed from the list of active transcoders.

Currently, transcoders may be slashed if they meet one of these two conditions

1. The transcoder does not submit the necessary verification for a segment
2. The transcoder submitted two claims for the same segment

The Vulnerability

Here's the code which determines whether a transcoder can be slashed for double claiming a segment.

```
function doubleClaimSegmentSlash(
 uint256 _jobId,
 uint256 _claimId1,
 uint256 _claimId2,
 uint256 _segmentNumber
) /* snip */ {
    Job storage job = jobs[_jobId];
    Claim storage claim1 = job.claims[_claimId1];
    Claim storage claim2 = job.claims[_claimId2];

    // Claim 1 must not be slashed
    require(claim1.status != ClaimStatus.Slashed);
```

```
        // Claim 2 must not be slashed
        require(claim2.status != ClaimStatus.Slashed);
        // Segment must be in claim 1 segment range
        require(_segmentNumber >= claim1.segmentRange[0] && _segme
        // Segment must be in claim 2 segment range
        require(_segmentNumber >= claim2.segmentRange[0] && _segme

        // Slash transcoder and provide finder params
        bondingManager().slashTranscoder(job.transcoderAddress, ms

        refundBroadcaster(_jobId);

        // Set claim 1 as slashed
        claim1.status = ClaimStatus.Slashed;
        // Set claim 2 as slashed
        claim2.status = ClaimStatus.Slashed;
    }
```

Source

At first glance, this code looks perfectly fine. Both claims must be valid, and the supposedly double-claimed segment must be included in both claims. While it's unfortunate that the Checks-Effects-Interaction pattern isn't used, there's no risk of re-entrancy by `bondingManager`.

However, the function does not check that the two specified claims are *distinct*. This means that any claim of any segment is eligible to be slashed just by calling `doubleClaimSegmentSlash` with the same claim ID.

The Solution

Here's the one (two if you count comments) line solution, presented in all its glory.

```
// The provided claims must be different (i.e different IDs)
```

```
    require(_claimId1 != _claimId2);
```

Source

The Impact

When a transcoder is slashed for double claiming, 3% of their staked LPT is deducted as penalty, with 5% of the penalty going to the finder and the remaining 95% burned. Furthermore, the transcoder is removed from the pool of active transcoders.

This means that for the top transcoder, who has staked ~1,100,000LPT, a mere 10 false slashes would bring their stake down to ~800,000LPT. Another 40 false slashes (for a total of 50) would result in a meager ~240,000LPT still staked.

Given that the attacker has the history of the entire world to play with, and can also manipulate transcoders into doing legitimate work by submitting jobs, this scenario is not entirely unrealistic. Furthermore, through the finder's fee the attacker would have netted 5% of the penalized LPT, resulting in a cool ~43,000LPT (~1500ETH) for their troubles.

Key Takeaways

Consider the assumptions made and ensure that they are true

When writing code, assumptions about what sorts of input will be received are made. As the developer works on the code and becomes more and more familiar with it, it's all too easy to fall into the trap of believing that these assumptions are etched into stone.

As such, when it comes time to write unit tests (or, even when just reviewing code), it's a good idea to list out the assumptions that were made and check whether they are really true, or just assumed to be true. In this case the assumption that both claim IDs would be distinct was made, but not enforced.

Have a response plan

Audited code does not equate to secure code, and as such response plans should always be developed in case a situation like this one arises. Due in part to having responded to security incidents before, the Livepeer team was able to quickly triage and resolve this problem. This was aided by their modular contract layout, which allowed them to pause and upgrade implementations as needed.

Further Reading

- The original Livepeer announcement
- Livepeer's post-mortem

samczsun
Read more posts by this author.

Read More

Taking undercollateralized loans for fun and for profit

Price manipulation, now with 100% more blockchain

The 0x vulnerability, explained

An in-depth look at how 0x's Exchange contract was vulnerable

17 MIN READ

5 MIN READ