



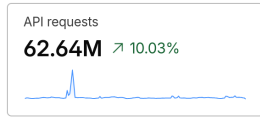
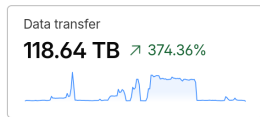
Understanding the Recent DDoS Attack Against Read the Docs

Sep 8, 2026 David Fischer

Analytics

HTTP Traffic

Review incoming [HTTP traffic](#) for all domains connected to Cloudflare under Read the Docs Production. Only includes traffic for [proxied](#) hostnames.



HTTP Traffic

[Print report](#) [Download data](#)

[Add filter](#)

Last 7 days (PDT)

Requests over time

Review spikes in [HTTP requests](#) and the top request attributes.

All [Referer host](#) [Host](#) [Country](#) [Path](#) [Edge status code](#) ...

● Total requests

19.75B



Analytics from the June 2026 attack

In mid-to-late June 2026, Read the Docs experienced the largest and most sophisticated distributed denial-of-service (DDoS) attack in our history. At its peak, our infrastructure was hit with over 5.5 million requests per minute, about 100 times our normal baseline traffic.

The incident lasted for nearly ten days, testing our infrastructure, our edge defenses, and our incident response processes. Unlike simpler traffic floods we've seen in the past, this attack was more distributed, it adapted to our defenses rapidly, and it purposefully attacked areas that bypassed caching.

Now that our small ops team is back to sleeping at normal hours, we wanted to walk through the anatomy of this kind of attack, why our existing rate limiting only partially mitigated it, and what strategies actually helped us (mostly) maintain availability throughout the attack.

Evolution of DDoS attacks

Read the Docs has historically been very tolerant toward spiders and bots scraping documentation we host, and IP-based rate limiting solved most abuse problems. Starting about two years ago, we began seeing a significant uptick as [AI crawlers](#) became more prevalent and it seems [other members of the dev infrastructure community](#) are seeing similar issues. It became straightforward to plug an AI-generated scraper into a proxy network. Our defenses adapted to that fairly easily, but the June attack was over 10x larger than anything we had faced.

Key characteristics of this attack included:

- Massive volume: At peak, we received 5.5 million requests per minute, compared to our normal daily peak of under 100k requests per minute.
- Global distribution: We saw malicious requests originating from millions of unique IP addresses across hundreds of networks (ASNs) globally. This included residential IP blocks as well as major and minor hosting providers.
- Header & TLS randomization: The attackers systematically randomized HTTP request headers and TLS connection parameters to evade signature-based filters (JA3/JA4).
- Limitations of automated CDN defenses: Read the Docs uses Cloudflare and while Cloudflare's automated DDoS protection mitigated some traffic originating from what they called "known botnets", a big part of the attack passed that first check and got through to our rate limiting and WAF rules.
- Cache evasion: Attackers found and deliberately targeted URLs that resulted in cache misses, such as non-existent pages with unique paths (404s) as well as temporary redirects (302s).
- Adaptive behavior: When we implemented blocks or rate limits, the botnet adjusted its request rates, rotating through different target paths and spreading traffic across broader IP pools to probe our defense boundaries.

Scale and breadth

Previous minor DDoS attacks or large distributed scrapers we'd seen were typically concentrated in some way. The requests either originated from a small set of countries, or a small set of IP blocks, or they had a small set of browser signatures. This attack was truly global. It came from every country all at once, which is a nightmare when rate limiting rules

are applied per Cloudflare colo. It's hard to craft rules that can limit a distributed attack while not hitting legitimate bots scraping at a reasonable rate from a single IP or subnet.

At one point when the attackers focused on redirects, they were overwhelming a hardcoded Nginx redirect (a simple `rewrite` regex directive) with enough traffic to cause dropped requests even on horizontally scaling infrastructure. An Nginx redirect like that can easily handle thousands of requests per second.

In addition to spreading across the globe, it also attacked multiple Read the Docs properties. Public community documentation was attacked as well as our commercially hosted docs. We also saw attackers try to take down our author-facing dashboards that require logins.

While it was an option (Cloudflare's "Under Attack Mode") to simply give every site visitor, legitimate or otherwise, a JavaScript challenge, we didn't want to do that. This would break every API integration and cause lots of friction for the hundreds of thousands of real docs readers. Instead, we relied on rate limiting and targeted challenges combined with more caching and pushing more features out to the edge.

“There's no way we could have handled this attack without Cloudflare.”

Adapting to our defenses

Read the Docs uses Cloudflare heavily for caching and rate limiting, and there's no way we could have handled this attack without Cloudflare. We have dozens of rate limiting rules (managed through [Terraform](#)) to protect our infrastructure based on IPs, on the thousands of hostnames and hundreds of thousands of subdomains Read the Docs hosts, on ASNs, browser fingerprints, and on combinations of all of these.

The attack started on a small number of domains where attackers discovered temporary redirects (302) that were not cached at the edge and were served by our Python backend rather than something like Nginx. Within a few minutes, our operations team had been paged due to a short outage (users may not always notice our outages because cached documentation keeps serving), and within half an hour or so we had moved these redirects to be served at the edge by Cloudflare instead of our servers. We thought that might be the end of it, but instead the attackers tried different tactics across various hosts and services for another week and a half.

HTTP traffic for readthedocs.io

Print report Download data ►

+ Add filter

Live

Last 30 min... (PDT) 📅

Served by equals origin ✕

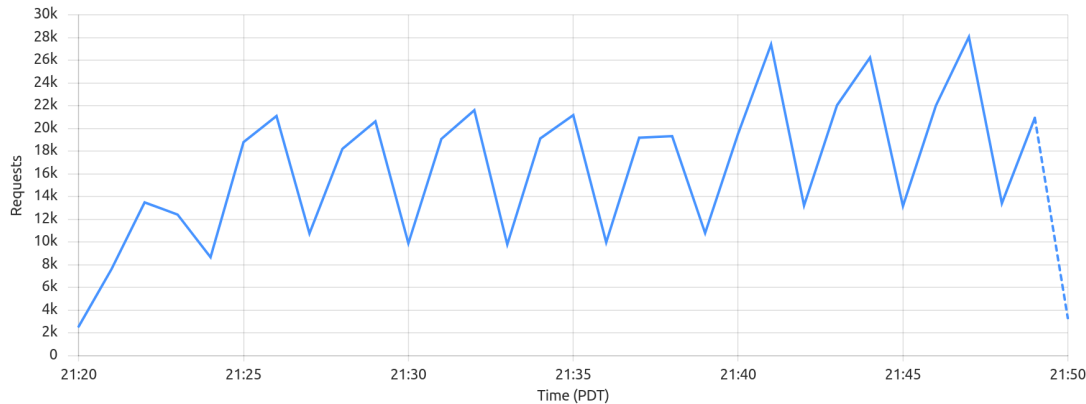
Requests over time 🕒

Review spikes in [HTTP requests](#) and the top request attributes.

All Referrer host Host Country Path Edge status code ...

● Total requests

503.47k



Analytics showing oscillating "Yo-Yo" traffic levels during the attack.

Attackers would ramp up to discover our rate limit thresholds and then back off to let the rate limit windows expire. This is called a yo-yo pattern, and it's designed to maximize the financial costs of auto-scaled infrastructure and cause intermittent service degradation. The attackers knew we were running a web app firewall (WAF) with rate limits and knew how to cause as much damage as possible in spite of that.

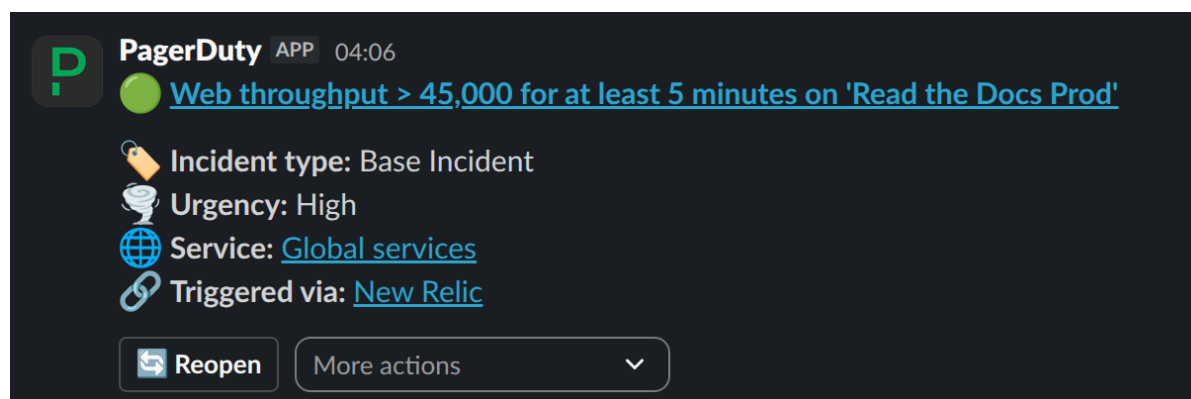
Defending against volumetric DDoS attacks

Defending against multi-million request-per-minute floods requires a defense-in-depth approach with edge caching, web app firewalls, rate limiting, local caches, and request fingerprinting. The fastest request is the one served by the CDN or the web app firewall.

Edge caching

The first line of defense, and one we were already using heavily, is a proper CDN and ensuring that as few requests as possible hit origin servers. Serving docs from a CDN has a lot of benefits. For Read the Docs, where docs sites change infrequently, we cache fairly aggressively but purge the cache for a particular docs site whenever new documentation is pushed to git and rebuilt. CDNs also make fetching documentation much faster for people geographically further away from our origin servers.

However, attackers probing our defenses quickly discovered which requests were cached and which ones weren't by how fast the CDN responded. This means that finding just a few requests that aren't cached gives attackers an angle of attack. We are still finding more paths and endpoints that aren't cached, but even very short-lived cached responses (using the [Cache-Control header](#)) for both redirects and normal 200 responses will help with this kind of attack.



A Slack notification when Read the Docs is getting 45k uncached reqs/min. Without caching and rate limiting, auto-scaling infrastructure will just scale out to handle the load at our expense.

Rate limiting and fingerprinting

Since we didn't want to give a JavaScript challenge to all users, we used targeted rate limiting rules combining [bot probability scores](#) with per-IP rate limits to challenge suspicious traffic while letting legitimate users and well-behaved bots browse uninterrupted. Everyone who has solved a JavaScript challenge knows they cause a lot of friction. We decided that it was better to let some malicious traffic through and err on the side of not challenging real users. With that said, we still needed rate limits to protect our infrastructure.

Instead of focusing on where the request came from (the IP, the country), defenses must focus on what the request looks like:

- Cipher suite and TLS anomalies: Automated scrapers and bot clients frequently present abnormal TLS connections different from browsers. Cloudflare's bot detection has [specific tools](#) to detect these.
- Too many bad requests: Legitimate users and good bots are almost always served successful (200) responses, not redirects or 404s. Since 200s are always cached, they almost never present a problem. When we see too many more expensive requests like redirects or 404s, we begin rate limiting the browser fingerprint, the ASN, or even possibly the specific domain as a whole. Adding these rules, which we call the "penalty box", probably made the biggest difference in automatically mitigating the attack as it changed over time.
- Protocol inconsistencies: Malicious tools often declare modern User-Agent strings while using older HTTP/1.1 connections. Unfortunately, this inconsistency wasn't very useful in this attack, which was entirely HTTP/2 and HTTP/3.

- Client fingerprinting: Everyone using the Golang HTTP client or the Python requests module with the same TLS cipher suites will have the same [JA4 fingerprint](#). This fingerprinting is specific to a browser or tool, not specific to a user. These fingerprints also weren't very helpful in this attack as the attackers were randomizing their TLS parameters.
- IP block classification: One area we are still working on is to classify more IP blocks into different categories with their own limits. For a service like Read the Docs which receives lots of automated traffic and wants to allow bots, we know we're going to get a lot of traffic from major cloud ASNs like Amazon, Google Cloud, and Azure. They should have higher limits than most residential or minor hosting providers.

Give users an escape hatch

One decision we made is to always give real users an escape hatch. Read the Docs very rarely issues outright blocks or bans to specific IPs or user agents. Instead, our "worst" is a JavaScript challenge, and if a user solves a challenge, they are very unlikely to get challenged again for the next day or so.

Lessons learned and key takeaways

The June 2026 DDoS attack reinforced several critical takeaways for running high-traffic infrastructure:

- IP blocking is obsolete for distributed attacks: Botnets or large scrapers use proxy services which make simple IP blocks useless. We already knew that, but this incident underscored it. Defenses need to have broader rate limits across more than just IPs (ASNs, hostnames, etc.).
- Aggressively cache: Cache everything, whether it's a simple static file, a 404, or a temporary redirect. Even setting a short cache window of a few minutes will ensure that these resources can't be used to attack our infrastructure. The default settings on the CDN and in most web frameworks are not what a service like Read the Docs wants.
- Protect cache-miss surfaces: Attackers actively search for non-cacheable paths (e.g. dynamic redirects, search endpoints, and 404s). Cache where possible, and if caching isn't feasible, try to handle as much on the edge as possible.
- Targeted challenges beat blunt instruments: Combining bot management heuristics with rate limits allowed us to mitigate the attack with minimal impact on legitimate users.
- Infrastructure as Code is essential: Managing edge and WAF rules via Terraform enabled us to review, test, version-control, and roll out complex filtering rules quickly and safely.

We're still seeing low levels of background traffic from the attack IP blocks, but our infrastructure is in a much stronger position today than it was before the attack. As AI tooling and proxy networks make attacks like this cheaper and more accessible, they are no longer reserved for big enterprise targets. They are becoming the baseline reality for any high-profile

public service. Our ops team is back to getting a full night's sleep, but we're viewing this as more like an extended reprieve rather than attacks being a thing of the past.

[See more posts from our blog](#)

STAY UPDATED

[Blog](#)

[Newsletter](#)

[Status](#)



LEARN MORE

[Documentation](#)

[Tutorial](#)

[Configure your project](#)

[Popular documentation tools](#)

[Compare to other platforms](#)

PRODUCT

[Features](#)

[Pricing](#)

[Privacy Policy](#)

[Terms of Service](#)

COMPANY

[About us](#)

[Support](#)

[Advertise with Us](#)

[Contribute](#)