Incident documentation/20191231-search-api-traffic-block

< Incident documentation

document status: draft

Contents [hide]

Summary

In an attempt to deploy a change to block excessive scraping of the search API from a particular User-Agent running on AWS hosts, instead, by accident, all enwiki search API traffic was blocked.

Impact

In an interval of just under five hours, we wrongly returned HTTP 403 for 59.1k HTTP queries. This is all global traffic against `Host: en.wikipedia.org` with a URI path of `/w/api.php` and a URI query that included the string `srsearch=`. AIUI this affected both bots and also mobile app users, but not website users.

The particular AWS bot we were attempting to block was not active for these five hours.

Detection

Detection was a human report in #wikimedia-operations. No automated detection.

(There is a larger architectural issue here about the 'proper' interface between Traffic and other services; see discussion below in actionables.)

Timeline

All times in UTC.

- 2019-12-31 16:15 dcausse posts a diagnosis of elasticsearch elevated latency on T241421, and also mentions the excessive scraping in #mediawiki_security.
- 18:56: cdanis, after looking at query logs for a while, uploads the first patchset of I7970d3c9, then iterates several times on the criteria of the block, and its location within our VCL configs
- 19:42: cdanis self-+2s, merges, and deploys I7970d3c9, which in the process of writing it, was badly refactored to be far too broad **OUTAGE BEGINS**
- 20:12: After half an hour, Puppet should have run on all cp servers; outage at 100%.
- 2020-01-01 00:43: in #wikimedia-operations, dbrant reports Search API 403 errors. Reedy and paladox notice and begin digging.
- 00:46: Reedy pings cdanis, who begins prepping a revert.
- 00:55: cdanis self-+2s and merges I50a2cd79 to roll back the erroneous block, and begins a cumin run to invoke puppet on all text CPs

- 01:05: fix deployed to 100% of text CPs **OUTAGE ENDS**

Conclusions

What went well?

- Once recognized, outage was root-caused quickly.

What went poorly?

- It was a holiday, and no one else looked to be around, so the erroneous change was self-+2'd. A quick look by a second person would have easily caught the mistakes.
- While automated tests for our Varnish configuration exist, and they were ran, they (intentionally) don't cover all kinds of traffic.

Where did we get lucky?

- Reedy was paying attention, had seen the bad change go by, and pinged cdanis.
- cdanis was able to respond quickly.

How many people were involved in the remediation?

- 2 engineers for under half an hour.

Links to relevant documentation

- Varnish#Blacklist_an_IP

Actionables

There's two sections here: one for easy and obvious things, and another for larger ideas that require more discussion.

- Update docs to mention the recently-added `public_cloud_nets` IP list recently made available. ✓ **Done**
- Add some more documentation on how to write and run VTC tests. Consider adding a test suite of common requests expected to return 200.

Grander schemes

- A 'differ' tool that would re-play the last ~1h of HTTP requests (or a set of canned queries) into an 'old' and 'new' Varnish, and show differences between the routing/rejection decisions that each made, would have caught this mistake before it was deployed, and also probably would be generically useful.
- An external monitoring system, if explicitly configured to probe the `srsearch` API, would have caught this mistake quickly after it was deployed.
- It is likely desirable to add more structure to how traffic blocks are configured. Currently the options are "add one line in the private Puppet repo to block all traffic from a given IP range", or "write arbitrary VCL to do something else". We could make more options available by filling out a few lines of a data structure instead of writing code (and knowing where to place said code).
  - As we scale the organization/our technical infrastructure/the number of services we run, this will probably prove necessary, along with other measures to make it more self-service: service owners shouldn't have to escalate to Traffic/SRE to implement blocks, nor should each of them have to implement their own blocking logic in every application.
  - If we made the notion of a 'traffic blocking rule' into a first-class entity, we could also add instrumentation around them -- and know how many rps were being blocked by which blocking rule, etc.

Categories: Incident documentation | Incident documentation drafts