Limiter le nombre de requêtes sur des scripts WSGI

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 3 décembre 2012

https://www.bortzmeyer.org/rate-limiting-wsgi.html

Le standard WSGI <https://www.bortzmeyer.org/wsgi.html> permet de développer facilement des scripts en Python pour réaliser un service accessible via le Web. Ces services nécessitent souvent pas mal de traitement côté serveur et risquent donc de faire souffrir celui-là si un client maladroit appelle le service en boucle (ou, pire, si un client méchant essaye délibérement de planter le serveur en lançant plein de requêtes). Il est donc souhaitable de limiter le nombre de requêtes.

Voici la technique que j'utilise (il y en a plein d'autres, avec Netfilter <https://www.bortzmeyer.org/rate-limiting-dos.html>, ou avec Apache <https://www.bortzmeyer.org/limit-apache.html>). On crée un seau qui fuit (dans le fichier (en ligne sur https://www.bortzmeyer.org/files/LeakyBucket.py)). Chaque requête en provenance d'une adresse IP donnée ajoute une unité dans le seau. Le temps qui passe vide le seau. Lorsqu'une requête arrive, on vérifie si le seau est plein et on refuse la requête autrement. Cela nécessite une entrée par client dans la table des seaux (chaque client a un seau différent). Avec le (en ligne sur https://www.bortzmeyer.org/files/LeakyBucket.py), cela donne :

```
if buckets[ip_client].full():
    # Refus
else:
    buckets[ip_client].add(1)
```

En pratique, on va regrouper les adresses IP des clients en préfixes IP, de manière à limiter le nombre de seaux et à éviter qu'un attaquant ne disposant d'un préfixe contenant beaucoup de machines ne puisse éviter la limitation en lançant simplement beaucoup d'adresses IP différentes à l'attaque. De manière assez arbitraire, on a mis 28 bits pour IPv4 et 64 pour IPv6. Pour faire les calculs sur les adresses IP, on se sert de l'excellent module Python netaddr <http://netaddr.googlecode.com/> :

```
ip_client = netaddr.IPAddress(environ['REMOTE_ADDR'])
if ip_client.version == 4:
    ip_prefix = netaddr.IPNetwork(environ['REMOTE_ADDR'] + "/28")
elif ip_client.version == 6:
    ip_prefix = netaddr.IPNetwork(environ['REMOTE_ADDR'] + "/64")
# Et on continue comme avant :
if buckets[ip_prefix.cidr].full():
...
```

Et comment se manifeste le refus ? On renvoie le code HTTP 429, "*Too Many Requests*", normalisé par le RFC 6585 [1]. En WSGI :

```
status = '429 Too many requests'
output = "%s sent too many requests" % environ['REMOTE_ADDR']
response_headers = [('Content-type', 'text/plain'),
                    ('Content-Length', str(len(output)))]
start_response(status, response_headers)
return [output]
```

Cela donne quoi en pratique ? Testons avec curl :

```
% for i in $(seq 1 50); do
    curl --silent --output /dev/null \
         --write-out "$i HTTP status: %{http_code} ; %{size_download} bytes downloaded\n" \
      https://www.bortzmeyer.org/apps/counter
done
1 HTTP status: 200 ; 278 bytes downloaded
2 HTTP status: 200 ; 278 bytes downloaded
3 HTTP status: 200 ; 278 bytes downloaded
4 HTTP status: 200 ; 278 bytes downloaded
5 HTTP status: 200 ; 278 bytes downloaded
6 HTTP status: 200 ; 278 bytes downloaded
7 HTTP status: 200 ; 278 bytes downloaded
8 HTTP status: 200 ; 278 bytes downloaded
9 HTTP status: 200 ; 278 bytes downloaded
10 HTTP status: 200 ; 278 bytes downloaded
11 HTTP status: 429 ; 36 bytes downloaded
12 HTTP status: 429 ; 36 bytes downloaded
13 HTTP status: 200 ; 278 bytes downloaded
14 HTTP status: 200 ; 278 bytes downloaded
15 HTTP status: 429 ; 36 bytes downloaded
16 HTTP status: 429 ; 36 bytes downloaded
...
```

On voit bien l'acceptation initiale, puis le rejet une fois le nombre maximal de requêtes fait, puis à nouveau l'acceptation lorsque le temps a passé. Attention en configurant la taille du seau (`default_bucket_size` dans (en ligne sur `https://www.bortzmeyer.org/files/LeakyBucket.py`)) : il y a plusieurs démons WSGI qui tournent et le nombre de requêtes acceptées est donc la taille du seau multipliée par le nombre de démons.

Comme souvent avec les mesures de sécurité, elles peuvent avoir des effets secondaires, parfois graves. Ici, le serveur alloue une table indexée par le nombre de préfixes IP. Si un attaquant méchant peut envoyer des paquets depuis d'innombrables adresses IP usurpées (c'est non trivial en TCP), il peut vous faire allouer de la mémoire en quantité impressionnante.

Si vous voulez les fichiers authentiques et complets, voir les fichiers de ce blog <`http://www.bortzmeyer.org/files/blog-support-files.tar.gz`> (fichiers `wsgis/LeakyBucket.py` et `wsgis/dispatcher.py`).

Merci à David Larlet pour l'amélioration du code.

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6585.txt`