RFC 9309 : Robots Exclusion Protocol

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 14 septembre 2022

Date de publication du RFC : Septembre 2022

https://www.bortzmeyer.org/9309.html

Quand vous gérez un serveur Internet (pas forcément un serveur Web) prévu pour des humains, une bonne partie du trafic, voire la majorité, vient de robots. Ils ne sont pas forcément malvenus mais ils peuvent être agaçants, par exemple s'ils épuisent le serveur à tout ramasser. C'est pour cela qu'il existe depuis longtemps une convention, le fichier `robots.txt`, pour dire aux robots gentils ce qu'ils peuvent faire et ne pas faire. La spécification originale était très limitée et, en pratique, la plupart des robots comprenait un langage plus riche que celui du début. Ce nouveau RFC documente plus ou moins le format actuel.

L'ancienne spécification, qui date de 1996, est toujours en ligne <http://www.robotstxt.org/orig.html> sur le site de référence <http://www.robotstxt.org/> (qui ne semble plus maintenu, certaines informations ont des années de retard). La réalité des fichiers `robots.txt` d'aujourd'hui est différente, et plus riche. Mais, comme souvent sur l'Internet, c'est assez le désordre, avec différents robots qui ne comprennent pas la totalité du langage d'écriture des `robots.txt`. Peut-être que la publication de ce RFC aidera à uniformiser les choses.

Donc, l'idée de base est la suivante : le robot qui veut ramasser des ressources sur un serveur va d'abord télécharger un fichier situé sur le chemin `/robots.txt`. (Au passage, si cette norme était faite aujourd'hui, on utiliserait le `/.well-known` du RFC 8615 [1].) Ce fichier va contenir des instructions pour le robot, lui disant ce qu'il peut récupérer ou pas. Ces instructions concernent le chemin dans l'URI (et `robots.txt` n'a donc de sens que pour les serveurs utilisant ces URI du RFC 3986, par exemple les serveurs Web). Un exemple très simple serait ce `robots.txt` :

```
User-Agent: *
Disallow: /private
```

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc8615.txt`

Il dit que, pour tous les robots, tout est autorisé sauf les chemins d'URI commençant par `/private`. Un robot qui suit des liens (RFC 8288) doit donc ignorer ceux qui mènent sous `/private`.

Voyons maintenant les détails pratiques (section 2 du RFC). Un `robots.txt` est composé de **groupes**, chacun s'appliquant à un robot ou un groupe de robots particulier. Les groupes sont composés de **règles**, chaque règle disant que telle partie de l'URI est interdite ou autorisée. Par défaut, tout est autorisé (même chose s'il n'y a pas de `robots.txt` du tout, ce qui est le cas de ce blog). Un groupe commence par une ligne `User-Agent` qui va identifier le robot (ou un astérisque pour désigner tous les robots) :

```
User-Agent: GoogleBot
```

Le robot va donc chercher le ou les groupes qui le concernent. En HTTP, normalement, l'identificateur que le robot cherche dans le `robots.txt` est une sous-chaine de l'identificateur envoyé dans le champ de l'en-tête HTTP `User-Agent` (RFC 9110, section 10.1.5).

Le robot doit combiner tous les groupes qui s'appliquent à lui, donc on voit parfois plusieurs groupes avec le même `User-Agent`.

Le groupe est composé de plusieurs règles, chacune commençant par `Allow` ou `Disallow` (notez que la version originale de la spécification n'avait pas `Allow`). Si plusieurs règles correspondent, le robot doit utiliser la plus spécifique, c'est-à-dire celle dont le plus d'octets correspond. Par exemple, avec :

```
Disallow: /private
Allow: /private/notcompletely
```

Une requête pour `/private/notcompletely/foobar` sera autorisée. L'ordre des règles ne compte pas (mais certains robots sont bogués sur ce point). C'est du fait de cette notion de correspondance la plus spécifique qu'on ne peut pas compiler un `robots.txt` en une simple expression rationnelle, ce qui était possible avec la spécification originelle. Si une règle `Allow` et une `Disallow` correspondent, avec le même nombre d'octets, l'accès est autorisé.

Le robot a le droit d'utiliser des instructions supplémentaires, par exemple pour les "*sitemaps*" `<https://www.sitemaps.org/>`.

En HTTP, le robot peut rencontrer une redirection lorsqu'il essaie de récupérer le `robots.txt` (code HTTP 301, 302, 307 ou 308). Il devrait dans ce cas la suivre (mais pas infinement : le RFC recommande cinq essais au maximum). S'il n'y a pas de `robots.txt` (en HTTP, code de retour 404), tout est autorisé (c'est le cas de mon blog). Si par contre il y a une erreur (par exemple 500 en HTTP), le robot doit attendre et ne pas se croire tout permis. Autre point HTTP : le robot peut suivre les instructions de mémorisation du `robots.txt` données, par exemple, dans le champ `Cache-control` de l'en-tête.

Avant de passer à la pratique, un peu de sécurité. D'abord, il faut bien se rappeler que le respect du `robots.txt` dépend de la bonne volonté du robot. Un robot malveillant, par exemple, ne tiendra certainement pas compte du `robots.txt`. Mais il peut y avoir d'autres raisons pour ignorer ces règles. Par exemple, l'obligation du dépôt légal fait que la BNF annonce explicitement qu'elle ignore ce fichier `<https://www.bnf.fr/fr/capture-de-votre-site-web-par-le-robot-de-la-bnf>`. (Et un programme comme wget a une option, `-e robots=off`, pour débrayer la vérification du `robots.txt`.)

Bref, un `robots.txt` ne remplace pas les mesures de sécurité, par exemple des règles d'accès aux chemins définies dans la configuration de votre serveur HTTP. Le `robots.txt` peut même diminuer la sécurité, en indiquant les fichiers « intéressants » à un éventuel attaquant.

Passons maintenant à la pratique. On trouve plein de mises en œuvre de `robots.txt` un peu partout mais en trouver une parfaitement conforme au RFC est bien plus dur. Disons-le clairement, c'est le bordel, et l'auteur d'un `robots.txt` ne peut jamais savoir comment il va être interprété. Il ne faut donc pas être trop subtil dans son `robots.txt` et en rester à des règles simples. Du côté des robots, on a un problème analogue : bien des `robots.txt` qu'on rencontre sont bogués. La longue période sans spécification officielle est largement responsable de cette situation. Et tout n'est pas clarifié par le RFC. Par exemple, la grammaire en section 2.2 autorise un `Disallow` ou un `Allow` à être vide, mais sans préciser clairement la sémantique associée.

Pour Python, il y a un module standard <`https://docs.python.org/3/library/urllib.robotparser.html`>, mais qui est loin de suivre le RFC. Voici un exemple d'utilisation :

```
import urllib.robotparser
import sys

if len(sys.argv) != 4:
    raise Exception("Usage: %s robotstxt-file user-agent path" % sys.argv[0])
input = sys.argv[1]
ua = sys.argv[2]
path = sys.argv[3]
parser = urllib.robotparser.RobotFileParser()
parser.parse(open(input).readlines())
if parser.can_fetch(ua, path):
    print("%s can be fetched" % path)
else:
    print("%s is denied" % path)
```

Et, ici, on se sert de ce code :

```
% cat ultra-simple.txt
User-Agent: *
Disallow: /private

% python test-robot.py ultra-simple.txt foobot /public/test.html
/public/test.html can be fetched

% python test-robot.py ultra-simple.txt foobot /private/test.html
/private/test.html is denied
```

Mais ce module ne respecte pas la précédence des règles :

```
% cat precedence.txt
User-Agent: *
Disallow: /
Allow: /bar.html

% python3 test-robot.py precedence.txt foobot /bar.html
/bar.html is denied
```

En application de la règle la plus spécifique, `/bar.html` aurait dû être autorisé. Si on inverse `Allow` et `Disallow`, on obtient le résultat attendu. Mais ce n'est pas normal, l'ordre des règles n'étant normalement pas significatif. Autre problème du module Python, il ne semble pas gérer les jokers, comme l'exemple `*.gif$` du RFC. Il existe des modules Python alternatifs pour traiter les `robots.txt` mais aucun ne semble vraiment mettre en œuvre le RFC.

La situation n'est pas forcément meilleure dans les autres langages de programmation. Essayons avec Elixir. Il existe un module pour cela <https://hex.pm/packages/robots>. Écrivons à peu près le même programme qu'en Python :

```
usage = "Usage: test robotstxtname useragent path"
filename =
  case Enum.at(System.argv(), 0) do
    nil -> raise RuntimeError, usage
    other -> other
  end
content =
  case File.read(filename) do
    {:ok, data} -> data
    {:error, reason} -> raise RuntimeError, "#{filename}: #{reason}"
  end
ua =
  case Enum.at(System.argv(), 1) do
    nil -> raise RuntimeError, usage
    other -> other
  end
statuscode = 200
{:ok, rules} = :robots.parse(content, statuscode)
path =
  case Enum.at(System.argv(), 2) do
    nil -> raise RuntimeError, usage
    other -> other
  end
IO.puts(
  case :robots.is_allowed(ua, path, rules) do
    true -> "#{path} can be fetched"
    false -> "#{path} can NOT be fetched"
  end)
```

Et testons-le :

```
% mix run test-robot.exs ultra-simple.txt foobot /public/test.html
/public/test.html can be fetched

% mix run test-robot.exs ultra-simple.txt foobot /private/test.html
/private/test.html can NOT be fetched
```

Il semble avoir moins de bogues que le module Python mais n'est quand même pas parfait.

Comme dit plus haut, le `robots.txt` n'est pas réservé au Web. Il est utilisé par exemple pour Gemini. Ainsi, le robot de collecte Lupa <gemini://gemini.bortzmeyer.org/software/lupa/> lit les `robots.txt` et ne va pas ensuite récupérer les URI interdits. En septembre 2022, 11 % des capsules Gemini <gemini://gemini.bortzmeyer.org/software/lupa/stats.gmi> avaient un `robots.txt`.