RFC 7033 : WebFinger

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 28 septembre 2013. Dernière mise à jour le 2 mai 2017

Date de publication du RFC : Septembre 2013

https://www.bortzmeyer.org/7033.html

Ce RFC décrit (même si elle n'est pas présentée ainsi) la deuxième version du protocole WebFinger. La première était informelle et permettait de récupérer de l'information sur une personne ou une organisation à partir de son adresse de courrier. La deuxième version, la première officiellement normalisée, généralise WebFinger : la clé d'entrée dans l'information est un URI, qui peut être une adresse de courrier (`mailto:`, RFC 6068 [1]) mais pas forcément.

WebFinger doit son nom à l'antique protocole Finger, normalisé dans le RFC 1288. Finger permettait de récupérer de l'information sur une personne, identifiée par `login@nom-de-machine` en se connectant sur le port 79 de la dite machine. On obtenait une réponse non structurée (du texte libre). Au contraire, WebFinger tourne sur le port 80, celui de HTTP, utilise REST, et envoie une réponse structurée, en JSON, qui est donc analysable par un programme. On nomme cette réponse le JRD, "*JSON Resource Descriptor*".

Que contient ce JRD ? Ce qu'on veut. Pas grand'chose si on est soucieux de vie privée et, sinon, une adresse, un numéro de téléphone, une photo, etc. WebFinger peut être utilisé pour des entités non humaines (une imprimante, une machine à café, etc), pour obtenir de l'information sur leurs capacités. Par contre, WebFinger est prévu pour de l'information relativement statique, et pas pour se renseigner sur l'état actuel. Pour une imprimante, il peut servir à apprendre qu'elle sait faire de la couleur, ou seulement du noir & blanc, mais pas à connaître la quantité de feuilles de papier restant dans le bac.

(Pour des raisons historiques, certaines mises en œuvre de WebFinger distribuent un XRD, la même chose mais en XML. D'une manière générale, attention à ce que vous trouvez avec un moteur de recherche, lorsque vous demandez « webfinger » : ce seront souvent des informations complètement dépassées.)

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc6068.txt`

Et quel genre d'URI peut-on utiliser en argument ? Ce qu'on veut mais, en pratique, l'usage le plus courant aujourd'hui est avec les URI `acct:` (normalisés dans le RFC 7565).

WebFinger reste délibérement ouvert : le RFC spécifie un protocole et un format mais pas une sémantique. Ainsi, le JRD peut avoir un contenu très varié selon les applications. L'idée est que chaque application de WebFinger spécialisera ce protocole et ce format, en indiquant plus précisément le type d'URI attendu et les informations contenues dans le JRD de réponse. À noter que ce point avait fait l'objet de vives controverses à l'IETF, à la fin du processus de normalisation. Tel que spécifié, WebFinger semblait à certains terriblement vague, un "*framework*" plutôt qu'un protocole. Selon les mots de Pete Resnick lors de la discussion entre l'IESG et les auteurs, « "*I suspect that the semantics are so underspecified that there could not possibly be interoperable implementations without lots of out-of-band information*" ». C'est pour cela que le RFC intègre aujourd'hui ces précisions : la sémantique sera dans d'autres spécifications (la section 8 du RFC détaille ce choix et ses conséquences).

La section 3 donne deux exemples d'utilisation, le premier dans le cas d'OpenID Connect `<http://openid.net/specs/openid-connect-messages-1_0.html>` et le second pour récupérer des métadonnées sur une page Web. Dans le premier cas, Carol veut s'authentifier auprès d'un site Web et donne son identificateur OpenID Connect, `carol@example.com`. Le site qui authentifie va utiliser WebFinger (avec l'URI `acct:carol@example.com`, avec le plan `acct:`) pour trouver le fournisseur OpenID de Carol. Dans le second cas, le client WebFinger va interroger le site qui héberge la page Web et demander en utilisant comme URI celui de la page Web.

À noter que les versions préliminaires de ce RFC avaient également plein d'exemples très hypothétiques, jamais utilisés en vrai, et qui ont été ensuite supprimés. On y trouvait par exemple un mécanisme possible pour l'autoconfiguration du client de courrier électronique, qui n'apportait rien par rapport au RFC 6186.

Avant de se plonger dans le protocole lui-même, la section 2 rappelle quelques points de vocabulaire. WebFinger étant, comme son nom l'indique, fondé sur le Web, il fait un grand usage des **liens**, tels que décrits dans le RFC 8288, pour indiquer une **relation**. La relation a un type et une information liée. En HTTP, le RFC 8288 utilise ainsi les attributs `rel` (type de la relation) et `href` (information liée) dans un en-tête `Link:`. WebFinger représente le même concept en JSON avec des objets JSON `links` (un tableau JSON), chacun comportant un membre `rel` et un membre `href`. Un exemple, pour une entité qui est un article publié sur le Web (le deuxième cas d'exemple cité plus haut, la recherche de métadonnées), les liens seraient :

```
"links" : [
    {
         "rel" : "copyright",
         "href" : "http://www.example.com/copyright"
    },
    {
         "rel" : "author",
         "href" : "http://blog.example.com/author/steve",
    }
]
```

Cet exemple se lit : l'entité interrogée via WebFinger a un "*copyright*" (qu'on peut trouver en `http://www.exampl` et un auteur, décrit en `http://blog.example.com/author/steve`.

La section 4 décrit le protocole complet. C'est du REST, donc au-dessus de HTTP. Le client WebFinger doit spécifier l'URI de l'entité sur laquelle il veut des informations et il peut aussi spécifier un ou plusieurs types de relations qui l'intéressent (par défaut, il va tout recevoir). À noter que WebFinger

impose l'usage de HTTPS, ce protocole étant souvent utilisé pour transporter des données sensibles (section 9.1, et c'était un des points de discussion les plus chauds à l'IETF). La requête WebFinger va utiliser un chemin qui utilise le préfixe `.well-known` du RFC 8615 et le suffixe `webfinger` (désormais enregistré dans les noms bien connus <https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml>). Si l'URI de l'entité qui nous intéresse contient un nom de machine, c'est cette machine que contacte le client WebFinger (sinon, il doit se débrouiller, d'une manière non précisée). La méthode HTTP utilisée est toujours `GET` (section 9.3 du RFC 2616). Prenons un exemple, celui de l'article `http://blog.example.com/article/id/314` sur lequel on voudrait plus d'informations. Le client WebFinger va se connecter à `blog.example.com` en HTTPS et envoyer la requête HTTP :

```
GET /.well-known/webfinger?resource=http%3A%2F%2Fblog.example.com%2Farticle%2Fid%2F314 HTTP/1.1
Host: blog.example.com
```

Le composant de requête (section 3.4 du RFC 3986) `resource` est l'URI (ici pour-cent encodé) de l'entité qui nous intéresse.

Si le serveur WebFinger connait l'entité en question, et accepte de répondre, il répond par le JRD (les données en JSON, étiquetées `application/jrd+json`, et décrites plus loin, en section 4.4 du RFC). Dans tous les autres cas, il répond par les codes HTTP traditionnels (400 « tu as oublié un truc, peut-être la `resource` », 403 « pas question que je te réponde à toi », 404 « je ne connais pas cette entité », 500 « j'ai un problème », 429 « trop de travail, je craque », etc).

Et si le client a inclus un ou plusieurs `rel` dans sa requête, indiquant qu'il n'est pas intéressé par tous les types de données mais seulement par certains ? Cela n'influe que sur le membre `links` du JRD, qui n'incluera alors que ce qui est demandé. Reprenons l'exemple de notre page Web et ne cherchons que l'auteur :

```
GET /.well-known/webfinger?resource=http%3A%2F%2Fblog.example.com%2Farticle%2Fid%2F314&rel=author HTTP/1.1
Host: blog.example.com
...
"links" : [
    {
         "rel" : "author",
         "href" : "http://blog.example.com/author/steve",
    }
]
```

Quel est l'intérêt (après tout, le client pourrait ainsi bien filtrer les types de liens après les avoir tous récupérés) ? Économiser des ressources sur le serveur (certaines informations peuvent nécessiter des requêtes compliquées dans une base de données) et diminuer le débit réseau. Notez toutefois, si vous écrivez un client, que tous les serveurs ne gèrent pas ce paramètre `rel` dans la requête et que le client risque donc toujours de tout récupérer, et de devoir trier ensuite.

Le format complet du JRD ("*JSON Resource Descriptor*", annexe A du RFC 6415 et dérivé du XRD <http://docs.oasis-open.org/xri/xrd/v1.0/xrd-1.0.html>) figure en section 4.4. C'est un objet JSON (RFC 8259) comprenant les membres `subject`, `aliases`, `properties` et `links` que nous avons déjà vu. `subject`, le seul obligatoire, est un identificateur de l'entité sur laquelle on se renseigne (en général le même que le paramètre `resource`), `properties` sont les informations sur l'entité (un registre IANA <https://www.iana.org/assignments/webfinger/webfinger.xhtml#properties> les stocke, en échange d'une spécification écrite, cf. section 10.4.2) et `links` les liens. `links` est le plus complexe. Chaque lien est un objet JSON comportant plusieurs membres. `rel` est le seul obligatoire et sa valeur est, soit un type enregistré à l'IANA <https://www.iana.org/assignments/link-relations/link-relations.xhtml> selon le RFC 8288, soit un URI (on peut ainsi « créer » ses propres types). Les autres membres possibles sont `type` (un type MIME), `href` (pointe vers la valeur du lien), `titles` (un texte humainement lisible, par exemple pour le présenter à l'utilisateur, marqué par une étiquette de langue) et `properties` (informations diverses). Voici un exemple complet, tiré du RFC, toujours au sujet de notre page Web intéressante :

```
{
       "subject" : "http://blog.example.com/article/id/314",
       "aliases" :
       [
         "http://blog.example.com/cool_new_thing",
         "http://blog.example.com/steve/article/7"
       ],
       "properties" :
       {
         "http://blgx.example.net/ns/version" : "1.3",
         "http://blgx.example.net/ns/ext" : null
       },
       "links" :
       [
         {
           "rel" : "copyright",
           "href" : "http://www.example.com/copyright"
         },
         {
           "rel" : "author",
           "href" : "http://blog.example.com/author/steve",
           "titles" :
           {
             "en" : "The Magical World of Steve",
             "fr" : "Le Monde Magique de Steve"
           },
           "properties" :
           {
             "http://example.com/role" : "editor"
           }
         }
      ]
}
```

La section 7 du RFC couvre un cas délicat, celui de services WebFinger hébergés. Si on souhaite sous-traiter WebFinger à un tiers, comment l'indique-t-on ? La seule solution est de faire une redirection HTTP depuis son site. Par exemple, avec Apache, on mettra dans la configuration :

```
Redirect /.well-known/webfinger http://wf.example.net/.well-known/webfinger
```

Et les requêtes WebFinger qu'on recevra seront gérées par le prestataire `wf.example.net` par le biais d'une redirection HTTP.

La section 8 décrit ce que veut dire « spécifier l'usage de WebFinger pour une application ». On a vu que WebFinger fournissait un protocole et un format très général. Chaque application qui compte se servir de WebFinger doit préciser un certain nombre de choses, notamment le contenu du JRD attendu. Si vous voulez vous servir de WebFinger pour un nouveau service très cool, vous allez devoir lire cette section et rédiger les détails. Première chose, le type d'URI attendu (`acct:` ? un autre ?) Deuxième chose, comment trouver le serveur à interroger. Si l'URI utilise le plan `http:`, c'est trivial. Mais pour les `acct:` ou les `mailto:` ? L'application doit donc préciser comment on trouve le serveur WebFinger (cela peut être aussi simple que d'avoir un serveur WebFinger central, codé en dur dans les programmes, pour tous les URI de l'application...)

Enfin, l'application doit spécifier le contenu attendu : quelles `properties` sont obligatoires dans la réponse, par exemple ? Même chose pour les liens : quels types `rel` peuvent/doivent être utilisés dans les liens ?

Ce n'est pas un peu indiscret, toutes ces informations distribuées à tout vent ? Si, et ce point a fait l'objet de vives discussions à l'IETF, ce qui a fini par donner naissance aux sections 6 et 9.2 de ce RFC. Le principal avantage de WebFinger (un seul endroit où aller pour récupérer toutes les informations sur une entité, et sous une forme structurée, ce qui est très pratique pour les programmes qui vont l'analyser) est aussi son principal risque (comme dit le RFC « " *The easy access to user information via WebFinger was a design goal of the protocol, not a limitation*" »). Le RFC cite l'exemple de données qui permettraient le harcèlement d'une personne. L'article « "*Abusing social networks for automated user profiling*" `<https://www.eurecom.fr/en/publication/3042/download/rs-publi-3042_1.pdf>` » illustrait bien comment le recoupement d'informations provenant de différents réseaux sociaux permettait de découvrir plein de choses sur les utilisateurs.

Ces sections « vie privée » du RFC rappellent qu'un serveur WebFinger ne distribue que ce qu'il veut. En cas de demande d'information sur une personne, par exemple, la norme technique qu'est ce RFC ne spécifie pas qu'on doive distribuer l'adresse et le numéro de téléphone. C'est un choix des administrateurs du serveur. (Au passage, c'est exactement la même chose pour le protocole whois, RFC 3912, un protocole dont les usages sont proches de ceux de WebFinger. Le RFC spécifie un protocole, pas une politique de distribution des données.)

Ensuite, le serveur n'est pas obligé d'être ouvert à tout le monde. Il peut parfaitement utiliser l'authentification HTTP (ou d'autres mécanismes de contrôle d'accès comme l'adresse IP du client) pour restreindre la distribution d'informations à certains. Un serveur WebFinger est également, cela va de soi, autorisé à fournir des réponses différentes selon le client. Par exemple, on peut imaginer une réponse minimale pour les clients inconnus, et davantage de détails pour ceux qui s'authentifient. Le RFC ne cite pas les questions légales (hors sujet pour une norme technique) mais, par exemple, un serveur WebFinger d'une entreprise qui distribuerait des détails personnels sur ses employés, comme des photos, sans leur autorisation, serait certainement en violation de la directive européenne sur la protection des données personnelles.

La section 9.2 demande donc que, pour tout service WebFinger, il existe une interface permettant aux utilisateurs d'indiquer de manière simple s'ils veulent que des informations à leur sujet soient publiées ou pas, et lesquelles. Par exemple, pour un réseau social typique, on peut imaginer que les utilisateurs choisissent quels éléments d'information sur eux soient publics et, dans ce cas, que seuls les éléments ainsi marqués soient distribués par WebFinger. Le RFC demande aussi que, par défaut, rien ne soit publié (ce qui n'est certainement pas la pratique des gros silos de données comme Facebook).

Les liens fournis en réponse à une requête WebFinger peuvent d'ailleurs eux aussi pointer vers des ressources dont l'accès est contrôlé ou limité. Bref, ce n'est pas de la faute de WebFinger si des informations sensibles circulent, il n'est qu'un outil, à utiliser intelligemment.

Autre problème de sécurité avec WebFinger, le fait que la réponse est différente selon que la ressource existe ou pas (code HTTP 200 dans le premier cas et 404 dans le second). Ainsi, même si la réponse est vide, un client WebFinger peut, par essais répétés, se constituer une liste des ressources existantes. Cela peut permettre d'énumérer les utilisateurs d'un réseau social, ou bien les adresses de courrier valides (information certainement utile pour un spammeur). Le RFC recommande donc que des mesures techniques, comme une limitation du trafic par adresse IP du client, soient déployées.

Autre cas où l'utilisation maladroite de WebFinger peut avoir des conséquences néfastes, les requêtes automatiques. Supposons un MUA qui ferait automatiquement une requête WebFinger sur le champ `From:` du message lorsque celui-ci est lu. Un spammeur pourrait générer un champ `From:` différent par destinataire et les requêtes WebFinger entrantes lui diraient quels destinataires ont lu le message... Le RFC recommande donc de ne pas effectuer de requêtes WebFinger automatiquement.