RFC 7235 : Hypertext Transfer Protocol (HTTP/1.1): Authentication

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 juin 2014

Date de publication du RFC : Juin 2014

https://www.bortzmeyer.org/7235.html

Dans la série de RFC sur le protocole HTTP 1.1 <https://www.bortzmeyer.org/http-11-reecrit.html>, ce nouveau document normalisait l'authentification faite en HTTP. Le RFC 9110 [1] l'a remplacé depuis.

Le HTTP originel avait été conçu uniquement pour de l'accès à des documents publics. Toutefois, l'authentification a été assez vite ajoutée, afin de permettre de mettre en ligne des documents dont l'accès est restreint à une certaine catégorie d'utilisateurs. HTTP dispose aujourd'hui de plusieurs mécanismes d'authentification, décrits dans ce RFC. Il faut quand même noter que tous ont des limites et, qu'en pratique, l'authentification des utilisateurs humains se fait presque toujours par des mécanismes non-HTTP, et pas par ceux de ce RFC.

Le principe de base de l'authentification HTTP est le défi/réponse. Un client cherche à accéder à une ressource non-publique : le serveur lui envoie à la place un défi (code de réponse 401 et en-tête `WWW-Authenticate:`), le client fait une nouvelle demande, avec la réponse au défi (en-tête `Authorization:`). À noter qu'il n'y a pas que les serveurs d'origine qui peuvent exiger une authentification, les relais peuvent le faire également (la réponse de défi est alors un 407 et la réponse du client est dans un en-tête `Proxy-Authorization:`). Les codes de retour liés à l'authentification (301 et 307) sont décrits en détail dans la section 3 et stockés dans le registre IANA des codes de retour <https://www.iana.org/assignments/http-status-codes>. Voici un exemple, vu avec curl, d'un site faisant une authentification par mot de passe :

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc9110.txt`

```
% curl -v https://icinga.example.org/icinga/
...
> GET /icinga/ HTTP/1.1
> Host: icinga.example.org
>
< HTTP/1.1 401 Unauthorized
< WWW-Authenticate: Basic realm="Icinga Access"
...
```

Et, si on indique via curl le mot de passe (si on s'en sert souvent, on peut aussi mettre le mot de passe dans `/.netrc`) :

```
% curl -u icinga:vachementsecret -v https://icinga.example.org/icinga/
...
> GET /icinga/ HTTP/1.1
> Authorization: Basic aWNpbmdhOnZhY2hlbWVudHNlY3JldA
> Host: icinga.example.org
...
< HTTP/1.1 200 OK
```

Notez que l'authentification se fait sur le canal HTTP (le mot de passe n'est pas chiffré, juste un peu brouillé) et que celui-ci a donc intérêt à être protégé (par exemple par TLS).

Un navigateur Web, recevant le défi (401), va automatiquement demander à l'utilisateur le mot de passe et le renverra, dans une seconde requête (et, en général, il mémorisera ce mot de passe, pour éviter de le demander à chaque requête HTTP ; HTTP est sans état, c'est le client qui mémorise et donne l'illusion d'une « session »).

Une fois qu'on est authentifié, on n'est pas forcément autorisé <`https://www.bortzmeyer.org/authentifier-et-autoriser.html`> : si un client reçoit un 403, c'est qu'il n'est pas autorisé, même si son authenticité ne fait pas de doute.

Un concept important de l'authentification HTTP est celui de « royaume » ("*realm*", section 2.2). Un royaume est un ensemble de ressources sur le serveur qui ont la même base d'utilisateurs. Un serveur peut partitionner ses ressources en plusieurs royaumes, par exemple pour utiliser plusieurs sources d'authentification différentes. Le royaume utilisé est transmis par le serveur dans le défi (en-tête `WWW-Authenticate:`).

La section 4 décrit en détail les en-têtes liés à l'authentification. Le `WWW-Authenticate:` contient le défi. Il est composé d'une indication du type d'authentification, avec le royaume utilisé, et d'éventuels paramètres. Par exemple, pour le type le plus simple, "*Basic*", où le défi est vide, et où le client doit simplement envoyer identificateur et mot de passe, l'en-tête ressemblera à (pour le royaume "*Staff*") :

```
WWW-Authenticate: Basic realm="Staff"
```

Quant au champ `Authorization:`, il est envoyé par le client en réponse à un défi (ou préventivement si le client sait que l'accès à la ressource est authentifié). Les informations qu'il contient dépendent du type d'authentification. Avec "*Basic*", ce sont un identificateur et un mot de passe, séparés par un deux-points et encodés en Base64 (c'est normalisé dans le RFC 7617, section 2).

Au fait, la configuration du serveur Apache qui a servi pour les exemples plus haut :

```
<Directory>
  AuthName "Icinga Access"
  AuthType Basic
  AuthUserFile /etc/icinga/htpasswd.users
  Require valid-user
  ...
```

Et le fichier `/etc/icinga/htpasswd.users` est géré avec la commande `htpasswd`.

L'authentification auprès d'un relais, est identique, avec `Proxy-Authenticate:` et `Proxy-Authorization:` au lieu de `WWW-Authenticate:` et `Authorization:`.

Les différents mécanismes d'authentification (comme "*Basic*") sont notés dans un registre IANA <https://www.iana.org/assignments/http-authschemes/>. Le registre inclut un lien vers la spécification de chaque mécanisme. Ainsi, le traditionnel "*Basic*" est dans le RFC 7617 et le récent "*Bearer*" dans le RFC 6750. Si vous voulez enregistrer un nouveau mécanisme, lisez aussi le RFC 7236 pour voir des exemples.

Tout nouveau mécanisme doit tenir compte des caractéristiques de HTTP. Par exemple, comme HTTP est sans état, le serveur ne doit pas avoir à se souvenir des requêtes précédentes : toutes les informations pour authentifier un client doivent être dans la requête.

La section 6 résume les problèmes de sécurité liés à l'authentification. Par exemple, le cadre général d'authentification de HTTP, spécifié dans ce RFC 7235, ne précise pas de mécanisme assurant la confidentialité des lettres de créance présentées. Ainsi, dans le cas du mécanisme "*Basic*", le mot de passe est transmis en clair (on ne peut pas considérer Base64 comme un chiffrement...) Il faut donc prendre des précautions supplémentaires (TLS...) Notez qu'il n'y a pas que le mot de passe qui soit une donnée sensible : dans le cas de "*Basic*", l'identificateur de l'utilisateur est également confidentiel.

Le mécanisme HTTP d'authentification, reposant sur un protocole sans état, ne fournit pas de durée de vie de l'authentification. Un client peut garder le mot de passe éternellement alors que le serveur voudrait, par exemple, obliger les clients qui avaient été absents longtemps à se ré-authentifier. Et si c'est le client qui veut annuler l'effet de son authentification, le mécanisme HTTP ne permet pas de fonction de « déconnexion ». C'est une des principales raisons pour lesquelles les applications qui authentifient les utilisateurs ne se servent pas souvent de l'authentification HTTP. Une autre raison est l'impossibilité d'adapter le dialogue de "*login*", par exemple en mettant un lien vers une page de documentation. En pratique, la plupart des applications se servent donc d'un formulaire Web pour demander le mot de passe, puis d'un "*cookie*" (RFC 6265) pour chaque requête.

Les changements depuis les RFC 2616 et RFC 2617? Ils sont cités dans l'annexe A. Les deux principaux sont la révision de la grammaire des valeurs des en-têtes `Authorization:` et l'introduction d'une procédure d'enregistrement des nouveaux mécanismes d'authentification (un premier exemple est celui des mécanismes existants, dans le RFC 7236).