RFC 7616 : HTTP Digest Access Authentication

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 8 octobre 2015

Date de publication du RFC : Septembre 2015

https://www.bortzmeyer.org/7616.html

Dans les techniques d'authentification normalisées pour un client HTTP, la plus connue et la plus simple est le mécanisme "*Basic*" du RFC 7617[1]. Mais elle a plusieurs failles de sécurité, comme le transmission explicite du mot de passe (et qui se fait en clair si on n'utilise pas HTTPS). D'où cette technique alternative, que je crois peu répandue, car plus complexe (et pas forcément plus sûre) : le mécanisme "*Digest*". Dans ce cas, le serveur HTTP envoie un défi au client, défi auquel le client répondra sans divulguer au serveur aucun secret.

Avec les RFC 7615 et RFC 7617, ce nouveau RFC remplace le très ancien RFC 2617 (qui normalisait tous les mécanismes d'authentification de HTTP). Désormais, l'authentification HTTP comporte un cadre général (le RFC 7235) et un RFC spécifique par mécanisme d'authentification. Celui de notre nouveau RFC, le "*Digest*", a assez peu changé depuis le RFC 2617.

L'essentiel du RFC actuel est dans la section 3. Le client s'authentifie avec un nom et un mot de passe mais le mot de passe n'est pas envoyé tel quel au serveur (quant au nom, ça dépend). Comme son nom l'indique ("*digest*" = condensation), les données sont condensées avant d'être envoyées au serveur, empêchant le serveur (ou un indiscret qui écoute) d'accéder à la valeur initiale.

Lorsque le client HTTP tente d'accéder à une ressource protégée, le serveur lui répond avec un code HTTP 401 et un en-tête `WWW-Authenticate:`, indiquant le mécanisme "*Digest*". Cet en-tête contient le **royaume** ("*realm*", un identificateur caractérisant le groupe d'utilisateurs attendus, cf. RFC 7235, section 2.2), le **domaine** (une liste d'URI pour qui cette protection s'applique), un **numnique <https://www.bortzmeyer.org/nonce.html>**, une chaîne de caractères **opaque**, que le client renverra telle quelle, une indication de l'**algorithme** de condensation utilisé et quelques autres détails. Le numnique doit être différent à chaque réponse du serveur et de son mécanisme de génération dépend largement la

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc7617.txt`

sécurité de ce mécanisme d'authentification. Un attaquant ne doit pas pouvoir deviner le numnique qui sera utilisé. À part cela, le mécanisme de génération des numniques n'est pas spécifié, c'est une décision locale. Une méthode suggérée par le RFC est que le numnique soit la concaténation d'une estampille temporelle et de la condensation de la concaténation de l'estampille temporelle, de l'`ETag` de la ressource et d'un secret connu du seul serveur. L'estampille temporelle sert à éviter les attaques par rejeu. Elle est en clair au début du numnique, pour faciliter les comparaisons floues (estampille « pas trop ancienne »). Notez que le RFC ne conseille pas d'inclure l'adresse IP du client dans son calcul, cela défavoriserait trop les clients qui changent d'adresse IP, par exemple parce qu'ils passent par plusieurs relais de sortie possibles. (Alors que la section 5.5 du même RFC donne le conseil opposé...)

Le client HTTP doit répondre à ce défi en envoyant un en-tête `Authorization:` qui inclut la **réponse condensée**, le **nom** de l'utilisateur (qui peut être condensé ou pas, selon la valeur du paramètre `userhash`, cf. section 3.4.4 pour les détails), un **numnique du client**, et un **compteur** du nombre de tentatives d'authentification, qui permet de repérer les rejeux.

La réponse condensée est calculée (section 3.4.1) en condensant la concaténation de plusieurs valeurs, comprenant notamment une condensation de la concaténation du nom de l'utilisateur, de son mot de passe et du royaume, le numnique du serveur et celui du client. Un attaquant ne peut donc pas savoir à l'avance quelle sera la réponse, puisqu'il ne connait pas le numnique.

En recevant cette réponse à son défi, le serveur doit récupérer le mot de passe de l'utilisateur dans sa base de données, et refaire les mêmes opérations de concaténation et de condensation que le client, puis vérifier qu'il trouve le même résultat. (Notez qu'il n'y a pas besoin de stocker le mot de passe en clair, et que c'est même déconseillé, il suffit d'avoir accès à son condensat.) Par exemple, avec Apache, l'outil htdigest <`http://httpd.apache.org/docs/2.4/programs/htdigest.html`> gère tout cela.

Le même mécanisme peut s'utiliser pour s'authentifier auprès d'un relais, mais dans ce cas les entêtes se nomment `Proxy-Authenticate:` et `Proxy-Authorization:`.

Bon, c'est bien compliqué tout cela, place aux exemples. Un client veut accéder à `http://www.example.org/di` (section 3.9.1 du RFC). Le royaume est `http-auth@example.org`, le nom d'utilisateur Mufasa et le mot de passe "*Circle of Life*" (n'oubliez pas les deux espaces entre les mots). Le serveur envoie le défi :

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
       realm="http-auth@example.org",
       qop="auth, auth-int",
       algorithm=SHA-256,
       nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
       opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

Le serveur demande qu'on condense avec SHA-256. Le client répond au défi :

```
Authorization: Digest username="Mufasa",
       realm="http-auth@example.org",
       uri="/dir/index.html",
       algorithm=SHA-256,
       nonce="7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v",
       nc=00000001,
       cnonce="f2/wE4q74E6zIJEtWaHKaf5wv/H5QzzpXusqGemxURZJ",
       qop=auth,
       response="753927fa0e85d155564e2e272a28d1802ca10daf4496794697cf8db5856cb6c1",
       opaque="FQhe/qaU925kfnzjCev0ciny7QMkPqMAFRtzCUYo5tdS"
```

Le nom de l'utilisateur était en clair (non condensé). Le numnique et la chaîne opaque sont renvoyés tels quels. La réponse a été calculée en suivant l'algorithme de la section 3.4.1. Si vous voulez toutes les étapes (condensat calculé avec la commande Unix `sha256sum`) :

— A1 = Mufasa :http-auth@example.org :Circle of Life
— H(A1) = 7987c64c30e25f1b74be53f966b49b90f2808aa92faf9a00262392d7b4794232 (avec le nom de l'utilisateur, qui sert de clé d'accès, c'est la seule information à garder sur le serveur, cela évite des mots de passe stockés en clair)
— A2 = GET :/dir/index.html (la réponse dépend de la partie locale de l'URI, ce qui limite l'ampleur d'un éventuelle attaque par rejeu)
— H(A2) = 9a3fdae9a622fe8de177c24fa9c070f2b181ec85e15dcbdc32e10c82ad450b04
— data = 7ypf/xlj9XXwfDPEoM4URrv/xwf94BcCAzFZH4GiTo0v :00000001 :f2/wE4q74E6zIJEtWaHKaf5wv/H5Qzzp
— secret :data = 7987c64c30e25f1b74be53f966b49b90f2808aa92faf9a00262392d7b4794232 :7ypf/xlj9XXwfDPEoM4URrv
— KD (réponse) = 753927fa0e85d155564e2e272a28d1802ca10daf4496794697cf8db5856cb6c1

Parmi les points qui peuvent compliquer ce mécanisme (qui est ennuyeux mais simple à mettre en œuvre), l'internationalisation (section 4). Si le nom ou le mot de passe ne se limitent pas à ASCII, le serveur a intérêt à utiliser le paramètre `charset` pour indiquer le jeu de caractères dont le client devra se servir. La seule valeur légale de ce caractère est UTF-8, et les chaînes de caractères doivent être normalisées en NFC (cf. RFC 5198, notamment la section 3). Un exemple se trouve dans la section 3.9.2.

Maintenant que ce mécanisme est décrit, analysons sa sécurité (section 5 du RFC). D'abord, les mots de passe utilisés peuvent être trop faibles. Ce mécanisme "*Digest*" permet les attaques par dictionnaire (on essaie tous les mots d'un dictionnaire comme mots de passe) et, si le mot de passe figure dans les dictionnaires habituels (c'est le cas de `azertyuipop` et de `123456789`), il finira par être trouvé. Cette attaque marche quelles que soient les qualités de l'algorithme de condensation employé. Il faut donc essayer de s'assurer qu'on n'utilise que des mots de passe forts. (Si la communication n'utilise pas HTTPS, et qu'un attaquant écoute passivement les défis et les réponses, il peut également les tester contre un dictionnaire, sans avoir besoin de faire des essais qui peuvent donner l'alarme.)

Ces mots de passe doivent être stockés sur le serveur. Bien sûr, on peut stocker uniquement leur forme condensée mais, comme c'est elle qui sert d'entrée à l'algorithme de calcul de la réponse, un attaquant qui met la main sur cette base a un accès complet aux ressources du serveur, sans étape de décryptage (comme cela serait le cas avec un attaquant mettant la main sur un `/etc/passwd` ou `/etc/shadow` Unix). Notez que le mécanisme "*Digest*" ne permet pas de saler les mots de passe. (À titre personnel, c'est pour cela que je ne trouve pas forcément ce mécanisme forcément plus sûr que "*Basic*", contrairement à ce que dit le RFC en 5.13. Aujourd'hui, les piratages de bases des serveurs sont fréquents. La documentation d'Apache est sceptique aussi, disant « "*this [Digest authentication] does not lead to a significant security advantage over basic authentication*" » et « "*the password storage on the server is much less secure with digest authentication than with basic authentication*" »). C'est en raison de cette faiblesse que le royaume est inclus dans la condensation qu'on stocke dans la base : au moins l'attaquant ne pourra pas attaquer les autres sites, même si on a le même nom d'utilisateur et mot de passe.

Notez au passage que, comme le mécanisme "*Basic*" du RFC 7617, ce "*Digest*" permet au serveur d'authentifier le client, mais pas le contraire. Si on veut plus fort, il va falloir utiliser l'authentification permise par TLS (avec des certificats).

Et les attaques par rejeu ? La principale protection est fournie par le numnique. S'il est stupidement généré (par exemple, un simple compteur incrémental, trivial à deviner), les attaques par rejeu deviennent possibles. Avec l'algorithme conseillé, les attaques par rejeu sont plus difficiles mais pas impossibles : comme on peut réutiliser le numnique pendant un certain temps, un attaquant rapide peut faire plusieurs essais. Si c'est intolérable, la seule solution est d'avoir des numniques qui ne sont absolument pas réutilisables (par exemple, des numniques aléatoires, en prenant les précautions du RFC 4086). L'inconvénient est qu'il faut que le serveur les mémorise, au lieu de simplement les recalculer.

Il y a aussi les attaques classiques de l'Homme du Milieu (par exemple dans un relais Web). Un tel attaquant peut, par exemple, remplacer l'algorithme dans le défi par un algorithme plus faible, voir remplacer le défi de "*Digest*" par un appel à utiliser un mécanisme plus faible, tel que "*Basic*". Le client HTTP peut prendre quelques mesures (se souvenir de l'authentification utilisée et avertir l'utilisateur si un site qui utilisait "*Digest*" passe à "*Basic*") mais aucune n'est parfaite.

Le RFC recommande aussi de n'utiliser l'authentification qu'au-dessus de TLS. Même si le mot de passe n'est pas transmis, l'observation de l'authentification peut donner des informations (et donner accès au contenu des pages Web alors que, si on authentifie, cela peut être parce qu'il est confidentiel). Et TLS avec authentification du serveur protège contre les attaques de l'Homme du Milieu.

Le mécanisme "*Digest*" est dans le registre IANA des mécanismes d'authentification `<https://www.iana.org/assignments/http-authschemes/http-authschemes.xml>`. Un nouveau registre est créé pour stocker les algorithmes de condensation `<https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xml#hash-alg>`.

Les changements depuis le RFC 2617 sont décrits dans l'annexe A. Les principaux ? SHA-256 et SHA-512 sont ajoutés aux algorithmes de condensation. MD5 est conservé, pour des raisons de compatibilité mais très déconseillé (RFC 6151). La possibilité de condenser le nom de l'utilisateur est ajoutée. L'internationalisation est ajoutée.

Si vous utilisez curl pour vous connecter au serveur HTTP, l'option `--digest` vous permettra d'utiliser le mécanisme de ce RFC (ou bien vous mettez l'option `--anyauth` et vous laissez curl se débrouiller pour trouver). Côté serveur, pour Apache, c'est bien documenté `<http://httpd.apache.org/docs/2.4/mod/mod_auth_digest.html>`. Notez la directive Apache `AuthDigestDomain` pour spécifier le domaine, c'est-à-dire la liste des URI protégés. Pour Nginx, il n'y a apparement pas de solution standard, il faut utiliser un module supplémentaire (ou utiliser le mécanisme "*Basic*" + TLS, ce qui est probablement meilleur).

Deux bons textes à lire si vous voulez creuser la question, avec des exemples, un chez Microsoft `<https://technet.microsoft.com/en-us/library/cc780170(v=ws.10).aspx>`, et sur le blog perso de Chua Hock-Chuan `<https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Authentication.html>`.