Pratique

Problèmes d'authentification HTTP

Emilio Casbas

Degré de difficulté

Le protocole HTTP nous offre le mécanisme d'autorisation de type requête-réponse qui peut être exploité par un serveur en réseau ou un proxy pour accepter ou refuser l'accès aux ressources du réseau.

Actuellement dans le réseau, des millions de transactions ont lieu et les données privées et confidentielles sont rendues publiques. Dans le réseau, tout cela est possible, mais pour des raisons de sécurité, il faut savoir qui a accès aux données vulnérables et qui peut exécuter des opérations privilégiées. Nous devons être sûrs que les utilisateurs non autorisés ne peuvent pas consulter les documents auxquels ils n'ont pas d'accès. Les serveurs essaient de savoir qui est l'utilisateur et à partir de ces informations décider quel type d'action ceux-ci peuvent exécuter. L'authentification est une technique d'identification qui consiste à vérifier l'identité d'une personne à partir de certaines preuves telles que mot de passe ou PIN. HTTP fournit une fonctionnalité naturelle permettant une authentification.

En fait, HTTP définit deux protocoles officiels d'authentification : une *authentification de base* et une *authentification digest*. Dans cet article, je me concentrerai particulièrement sur la méthode d'*authentification de base* qui est largement utilisée par les clients et les serveurs réseau et est moins sûre.

Les domaines d'application de cette méthode d'authentification :

- Serveurs réseau sur Internet – c'est une situation la plus populaire. À partir de la maison ou d'un cyber-café, l'utilisateur se connecte sur un serveur sur lequel l'authentification HTTP est configurée pour accéder à certaines ressources. Si vous consultez les pages Web des entreprises, vous pouvez observer que beaucoup de pages utilisent ce type d'authentification pour donner accès aux parties protégées du site.
- Serveurs réseau dans un Intranet- dans ce cas, le domaine d'application est plus restreint

Cet article explique...

- Différents niveaux d'autorisation HTTP.
- Différences dans l'autorisation HTTP sur les niveaux différents.
- Exemples pratiques de communication HTTP.
- Faiblesses de l'autorisation.
- Solutions et alternatives.

Ce qu'il faut savoir...

- Modèle OSI.
- Connaissance du protocole HTTP.

Brève introduction à l'authentification digest

Le but de l'authentification digest n'est jamais d'envoyer *en clair* le mot de passe à travers le réseau, mais de l'envoyer en tant que *condensé* crypté de manière irréversible.

L'authentification digest a été développé de façon compatible comme une alternative plus sûre par rapport à l'authentification de base, mais ce n'est pas un unique protocole appelé sûr comparable à ceux utilisant les mécanismes de clé publique (SSL) ou les mécanismes d'échange de tickets (kerberos). L'authentification digest n'est pas une authentification forte et n'offre pas la confidentialité des données outre la protection du mot de passe (l'autre partie de la requête et de la réponses passent en texte clair).

Authentification de base :

- L'authentification de base est l'un des protocoles d'authentification HTTP les plus utilisés. Elle est implémentée dans la plupart des clients et serveurs réseaux. Pour mieux comprendre ce qu'est c'est, regardez sa description graphique.
- Au-dessous, nous présenterons les étapes précédents l'authentification HTTP.
- L'utilisateur veut accéder à une ressource réseau (par exemple une image)
- Le serveur vérifie que cette ressource est protégée et envoie au client une demande de mot de passe avec l'en-tête HTTP *Authorization Required* et le code 401.
- Le navigateur de l'utilisateur obtient le code et l'en-tête d'authentification et affiche la boîte de dialogue utilisateur/mot de passe. Quand l'utilisateur saisit les données, le navigateur effectue le chiffrement en base64 des données saisies et les envoie au serveur dans l'en-tête utilisateur *Authorization*.
- Le serveur déchiffre le nom de l'utilisateur et le mot de passe et vérifie s'il a accès à la ressource protégée.

Comme on voit, l'authentification de base envoie une paire *utilisateur:mot de passe* sous une forme non cryptée à partir du navigateur vers le serveur et en tant que telle ne devrait pas être employée pour les connexions sensibles, à moins qu'on ne travaille dans un environnement chiffré tel que SSL.

car il est limité à l'Intranet d'une entreprise, mais les problèmes liés à ce type d'authentification sont identiques à ceux étudiés dans le cas précédent, et les ressources disponibles à l'intérieur du réseau sont tout aussi vulnérables.

- Serveurs proxy sur Internet – il se peut aussi que pour naviguer dans certaines ressources d'un réseau donné ou pour contrôler l'accès, on puisse le faire uniquement à travers le serveur proxy de cette institution. Pour cela, l'authentification HTTP peut être aussi implémentée dans le proxy pour que toutes les données passent via Internet.
- Serveurs proxy dans un Intranet – il arrive aussi souvent qu'au sein d'un réseau d'entreprise, l'unique possibilité d'accéder à Internet est réalisée via un serveur proxy, ce qui a pour but de contrôler pleinement l'utilisation d'Internet. C'est pourquoi la configuration du serveur proxy pour qu'il contrôle les utilisateurs est tout à fait normal dans ce cas. Normalement, ce type de validation sera intégrée à d'autres mécanismes existant dans le réseau intranet, ce qui aggrave le problème de Single Sign On dont nous nous occuperons plus tard.

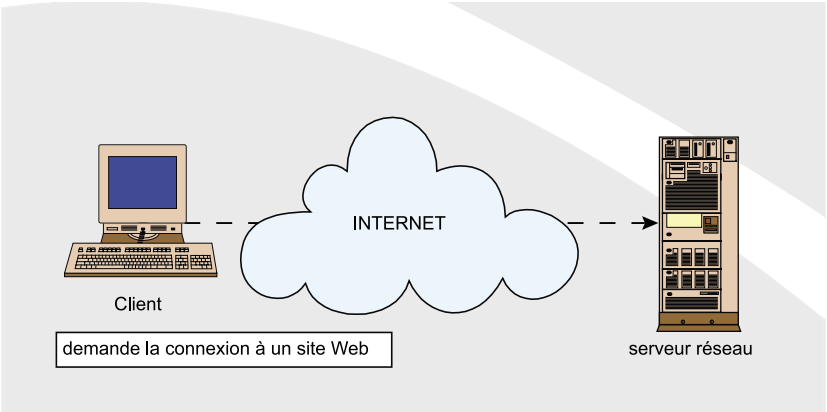


Figure 1. *Les serveurs réseau sur Internet*

Exemple pratique

Dès que nous savons déjà ce que c'est l'authentification HTTP et quelles sont les étapes de son fonctionnement, analysons ces étapes d'une façon plus détaillée, à l'aide de notre navigateur Web mozilla. Pour cela, nous aurons besoin d'une extension pour capturer les en-têtes HTTP que nous réalisons.

L'extension dont nous avons besoin s'appelle *livehttpheaders* et nous pouvons la télécharger à partir du site *http://livehttpheaders.mozdev.org/.* Nous l'installons suivant les pas décrits sur le site, et ensuite, dans mozilla, l'option Live HTTP Headers devient disponible (présentée sur la Figure 6).

Nous cliquons sur cette option et il s'affiche une fenêtre présentée sur la Figure 2.

Dès ce moment, nous obtiendrons toutes les informations sur les en-têtes HTTP de tous les sites que nous visitons en interceptant les en-têtes expliqués ci-dessous.

En-têtes et explications

Le client envoie une requête HTTP standard demandant une ressource donnée.

```
GET /doc/ecasbas/ HTTP/1.1\rn
Host: www.prueba.es\rn
User-Agent: Mozilla/5.0
 (Windows; U;
 Windows NT 5.1; en-US; rv:1.7.12)
Accept: text/xml,text/html;q=0.9,
 text/plain;q=0.8,
 image/png,*/*;q=0.1\rn
Accept-Language: en-us,en;
```

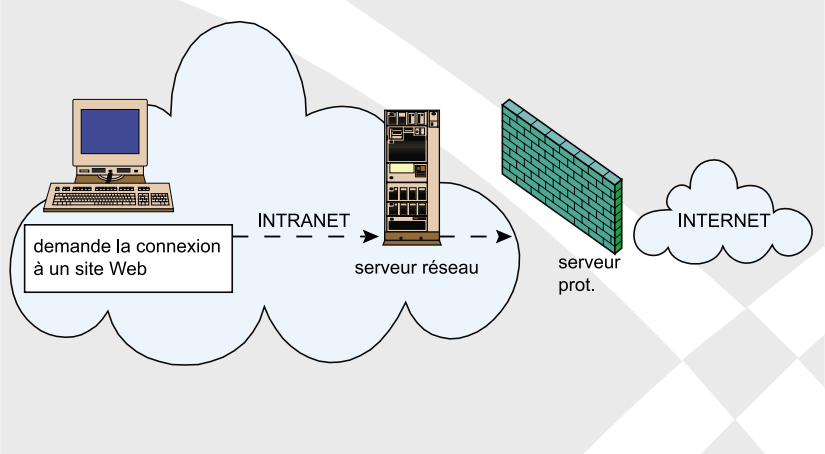


Figure 2. *Les serveurs réseaux dans un Intranet*

```
 q=0.7,es;q=0.3\rn
Accept-Encoding: gzip,deflate\rn
Accept-Charset: ISO-8859-1,
 utf-8;q=0.7,*;q=0.7\rn
Keep-Alive: 300\rn
Connection: keep-alive\rn
```

Le serveur lit son archive de configuration et détermine si la ressource sollicitée est protégée par un mot de passe. Le serveur peut donner accès à celle-ci seulement aux utilisateurs connus. Alors le serveur répond au client par une demande d'autorisation en l'indiquant par le code HTTP 401.

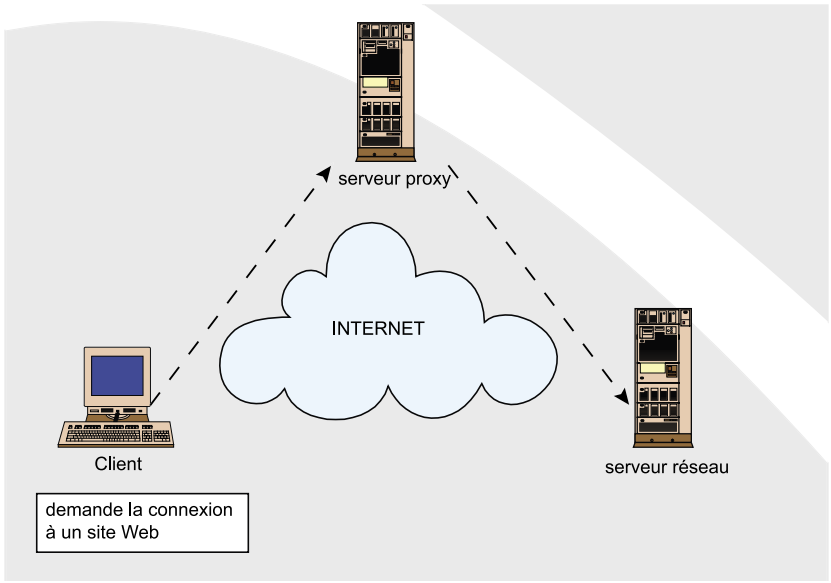
```
HTTP/1.0 401 Unauthorized\rn
Date:
 Mon, 16 Jan 2006 11:17:51 GMT\rn
Server: Apache/2.0.55 (Unix)
 mod_ssl/2.0.55 OpenSSL/0.9.7g
 PHP/5.1.1\rn
WWW-Authenticate:
 Basic realm="ByPassword"\rn
Accept-Ranges: bytes\rn
Content-Length: 3174\rn
Content-Type: text/html\rn
X-Cache:
 MISS from www.prueba.es\rn
Connection: keep-alive\rn
```

Le navigateur interprète ce code HTTP 401 comme requête à authentifier et affiche alors l'invite utilisateur: mot de passe en montrant le nom de l'hôte et realm, ce qui est illustré sur la Figure 3.

Le client renvoie la requête avec les données sur l'utilisateur/le mot de passe saisies :

```
GET /doc/ecasbas/ HTTP/1.1\rn
Host: www.unav.es\rn
User-Agent: Mozilla/5.0
 (Windows; U; Windows NT 5.1;
 en-US; rv:1.7.12)
Accept: text/tml+xml,text/html,
 image/jpeg,image/gif;
 q=0.2,*/*;q=0.1\rn
Accept-Language:
 en-us,en;q=0.7,es;q=0.3\rn
Accept-Encoding: gzip,deflate\rn
Accept-Charset:
 ISO-8859-1,utf-8;q=0.7,*;q=0.7\rn
Keep-Alive: 300\rn
Connection: keep-alive\rn
Authorization:
 Basic ZWNhc2JhczpwcnVlYmE=\rn
```

Ensuite, le serveur compare l'information provenant du client avec sa liste des utilisateurs/mots de passe.



Si l'autorisation ne réussit pas, le serveur renvoie l'en-tête de l'autorisation souhaitée HTTP 401. Si les données saisies sont correctes, le serveur affichera la ressource voulue. Ensuite, le serveur passe à la ressource sollicitée :

```
HTTP/1.0 200 OK\rn
Date: Mon, 16 Jan 2006 11:17:58 GMT\rn
Server: Apache/2.0.55 (Unix)
 mod_ssl/2.0.55
 OpenSSL/0.9.7g PHP/5.1.1\rn
Last-Modified:
 Fri, 13 Jan 2006 10:31:02 GMT\rn
ETag: "125b019-5-f636a580"\rn
Accept-Ranges: bytes\rn
Content-Length: 5\rn
Content-Type: text/html\rn
X-Cache:
 MISS from www.prueba.es\rn
Connection: keep-alive\rn
```

Dans les champs précédents, nous avons vu les champs spéciaux qui ont été ajoutés aux différentes entêtes HTTP. Dans l'étape 3, quand le serveur envoie la réponse avec l'entête 401, il ajoute un champ spécial.

```
WWW-Authenticate:
 Basic realm="ByPassword"\rn
```

La valeur *Basic* montre que nous demandons au navigateur d'utiliser l'authentification de base. La chaîne de caractères *realm* est une chaîne

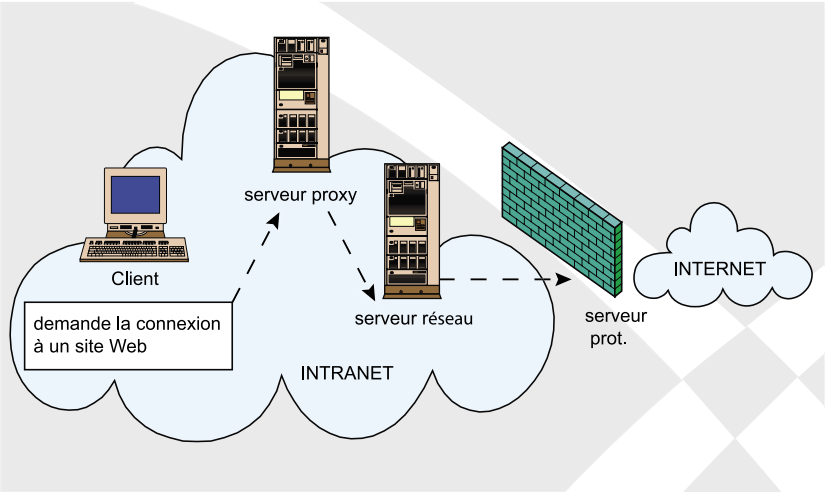


Figure 4. *Les serveurs proxy dans un Intranet*

arbitraire envoyée pour afficher à l'utilisateur l'information sur le type d'authentification. La Figure 3 montre comment la fenêtre de mozilla demande l'authentification en affichant le realm et l'hôte.

L'utilisateur remplit le formulaire et l'envoie. Le navigateur renvoie automatiquement sa demande. Comme on a pu voir auparavant, certains champs ont été ajoutés à une requête HTTP standard :

```
Authorization:
 Basic ZWNhc2JhczpwcnVlYmE=\rn
```

À ce moment, le navigateur Web envoie une information sur l'autorisation actuelle au serveur. On voit que le champ *Authorization* se compose de deux valeurs. Le mot *Basic* montre que le nom de l'utilisateur a été envoyé conformément à la méthode d'authentification de base. Le bloc de données qui vient après celui-ci est le nom de l'utilisateur actuel envoyé par le navigateur. Nos données concernant le nom de l'utilisateur n'apparaissent pas directement, mais ce n'est pas le chiffrement, mais seulement le codage en 64. En résumé, le codage en base64 présente les séquences d'octets arbitraires de façon illisible pour un être humain. Les algorithmes de codage et de décodage sont simples, mais les données codées sont d'habitude de 33% que les données non codés. Pour en savoir plus, lisez l'encadré *Sur le réseau*.

Si l'on possède le code précédent, le nom de l'utilisateur en texte clair peut être facilement déchiffré au format utilisateur:mot de passe.

```
ZWNhc2JhczpwcnVlYmE=
 --> base64Decode() --> "ecasbas:essai"
```

L'implémentation de l'authentification *digest* est exactement le même processus que l'authentification de base. L'unique différence est dans le nombre d'arguments soumis par le navigateur et dans le format du nom de l'utilisateur retourné.

Les deux types d'authentification, *digest* et *basic* sont utilisés par les clients et les serveurs Web, mais il ne faut pas les utiliser pour la protection des informations sensibles ou l'accès sécurisé. L'utilisation du même nom de l'utilisateur et mot de passe pour différents services est très fréquente,

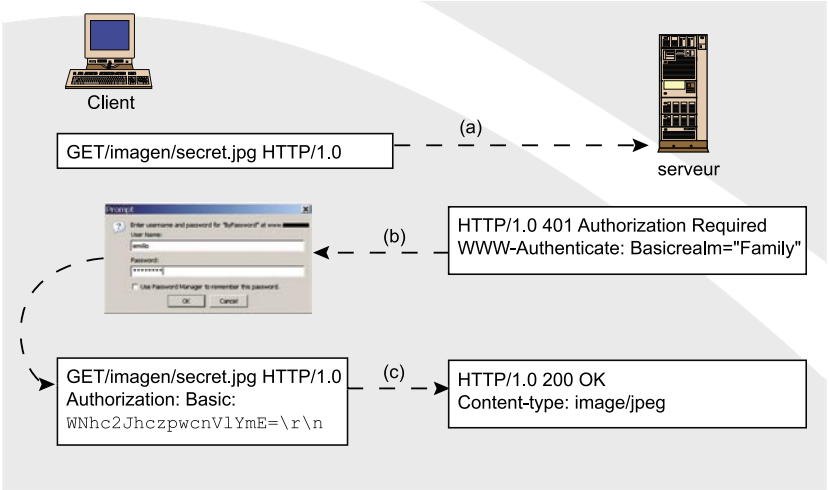


Figure 5. *L'authentification de base*

Listing 1. *Le code perl pour décoder la chaîne en base64*

```
#!/usr/bin/perl
use MIME::Base64;
while (<>) {
  print MIME::Base64::decode_
                        base64($_);
}
```

mais il faut faire attention à ce que les ressources que l'on veut ainsi protéger ne soient pas trop vulnérables et que ces authentifications ne fonctionnent pas dans d'autres services tels que le courrier électronique et l'accès aux informations privées.

Authentification proxy

Les séquences précédentes concernent le client demandant au serveur Web un accès à une ressource protégée. Mais cette procédure peut être aussi employée si le serveur proxy nécessite une authentification pour obtenir un accès à une ressource. Nous analyserons aussi cette situation et verrons quels codes sont affichés dans le cas d'un proxy.

Avec un serveur proxy configuré dans le navigateur, nous exécutons une requête pour pouvoir naviguer.

```
GET
 http://www.google.com/ HTTP/1.1\rn
Host: www.google.com\rn
User-Agent: Mozilla/5.0 (Windows; U;
 Windows NT 5.1; en-US; rv:1.7.12)
Accept: application/x-shockwave-flash,
text/xml,application/xml,*/*;q=0.1\rn
```

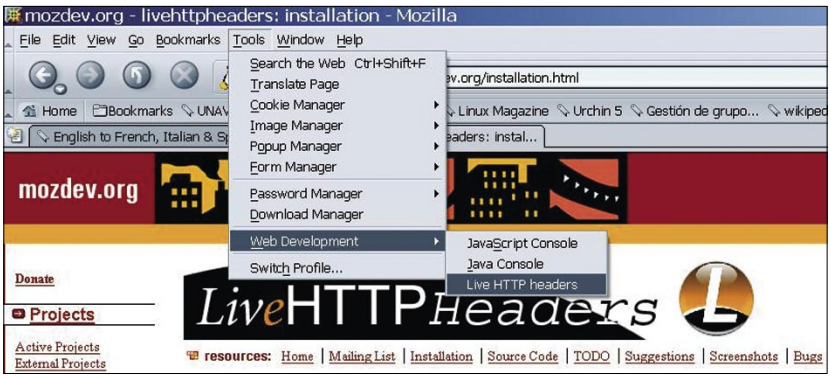


Figure 6. *L'option Live HTTP Headers*

```
Accept-Language:
 en-us,en;q=0.7,es;q=0.3\rn
Accept-Encoding: gzip,deflate\rn
Accept-Charset:
 ISO-8859-1,utf-8;q=0.7,*;q=0.7\rn
Keep-Alive: 300\rn
Proxy-Connection: keep-alive\rn
```

Le proxy nous répond en nous indiquant que pour pouvoir naviguer, une autorisation est nécessaire.

```
HTTP/1.0 407
 Proxy Authentication Required\rn
Server: squid/2.5.STABLE12\rn
Mime-Version: 1.0\rn
Date: Mon, 16 Jan 2006 13:01:19 GMT\rn
Content-Type: text/html\rn
Content-Length: 3283\rn
Expires:
 Mon, 16 Jan 2006 13:01:19 GMT\rn
X-Squid-Error:
 ERR_CACHE_ACCESS_DENIED 0\rn
Proxy-Authenticate: Basic realm=
""Proxy Authentication
  (user/passwd)""\rn
X-Cache: MISS from proxy.es\rn
Proxy-Connection: keep-alive\rn
```

Alors notre navigateur interprète cela comme requête/réponse pour l'authentification de base et affiche l'invite pour y saisir le données exigées, ce qui est illustré sur la Figure 9.

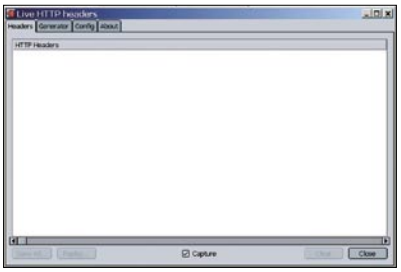


Figure 7. *La fenêtre Live HTTP Headers*

Certains navigateurs n'interprètent pas correctement *realm*, c'est pourquoi ils affiche dans la fenêtre présentée ci-dessous le message *Proxy Authentication (user/passwd)*. Ce cas est un peu différent, mais il nous servira d'exemple.

Nous saisissons le nom de l'utilisateur et le mot de passe, et notre client envoie les données de retour suivantes au proxy.

```
GET
http://www.google.com/ HTTP/1.1\rn
Host: www.google.com\rn
User-Agent: Mozilla/5.0
 (Windows;
 U; Windows NT 5.1;
 en-US; rv:1.7.12)
Accept:
 application/x-shockwave-flash,
 text/xml,
 image/gif;q=0.2,*/*;q=0.1\rn
Accept-Language:
 en-us,en;q=0.7,es;q=0.3\rn
Accept-Encoding:
 gzip,deflate\rn
Accept-Charset:
 ISO-8859-1,utf-8;q=0.7,*;q=0.7\rn
Keep-Alive: 300\rn
Proxy-Connection:
 keep-alive\rn
Proxy-Authorization:
 Basic
 ZWNhc2Jhc0B1bmF2Lm
 VzOnBydWViYTAx\rn
```

Le serveur proxy vérifie intérieurement si effectivement le nom de l'utilisateur et le mot de passe sont valides et nous donne accès à la ressource.

```
HTTP/1.0 200 OK\rn
Cache-Control: private\rn
Content-Type: text/html\rn
Content-Encoding: gzip\rn
Server: GWS/2.1\rn
Content-Length: 1408\rn
Date: Mon, 16 Jan 2006 13:05:40 GMT\rn
X-Cache: MISS from filter\rn
Proxy-Connection: keep-alive\rn
```

Au lieu du code 401 HTTP, le serveur proxy affiche le code 407 (authentification par proxy nécessaire), et l'en-tête ajouté dans le cas du serveur réseau WWW-authenticate, pour le serveur proxy est Proxy-Authenticate. Tout le processus est identique à celui de l'accès restreint à une ressource réseau, excepté ces petites différences.

Vision générale de l'authentification HTTP

Le schéma d'authentification de base n'est pas une méthode sûre

Figure 8. *La fenêtre Live HTTP Headers*

Tableau 1. *L'authentification de serveur réseau et l'authentification proxy*

Serveur réseau	Serveur Proxy.
Code d'état sans autorisation : 401	Code d'état sans autorisation : 407
WWW-authenticate	Proxy-Authenticate
Authorization	Proxy-authorization
Authentication-Info	Proxy-Authentication-Info.

Figure 9. *La fenêtre Live HTTP Headers*

d'authentification des utilisateurs. Elle ne protège pas l'identité de l'utilisateur et celle-ci est transmise à travers le réseau en texte clair. D'autre part, HTTP ne refuse pas l'utilisation des schémas d'authentification additionnels et des mécanismes de cryptage qui augmentent et améliorent la sécurité d'authentification de base.

Malgré les faiblesses de ce type d'authentification, comme on a pu voir ci-dessus, cette méthode est utilisée dans différents environnements où le plus grand danger est lié à l'existence de Single Sign On, quand un seul mot de passe vous sert à vous identifier auprès de toutes les ressources. Ainsi, il n'est pas important si vous utilisez les mécanismes d'accès sécurisé par le biais de SSL, les connexions chiffrées au réseau (IPSEC), les programmes avec une connexion sécurisée, etc. Si l'une des ressources emploie ce type d'authentification, vous avez un accès immédiat à tous les services disponibles.

Un champ idéal pour implémenter ce type d'authentification serait, par exemple, l'accès aux statistiques d'utilisation d'un serveur donné ou l'accès à une autre ressource, si vous trouvez qu'un accès illégal à celle-ci n'est pas dangereux. De cela, les authentifications d'accès doivent être fournies par l'administrateur du site ou par un programme générateur. Elles ne doivent jamais être choisies par l'utilisateur car ainsi on se retrouvera avec le même problème qu'auparavant. Les gens n'ont pas l'habitude d'utiliser différentes authentifications pour différents services, mais les réutilisent.

Conclusion

L'authentification HTTP de base est simple et commode, mais ce n'est pas une méthode sûre. Il faut l'employer dans les cas où l'accès aux informations doit avoir le caractère privé et son utilisation ne peut pas compromettre la sécurité des autres systèmes.

Les gens utilisent souvent le même nom d'utilisateur et le même mot de passe à des fins différentes. De cela, même si ceux-ci sont utilisés dans des environnements de confiance et dans les cas d'un accès aux informations non sensibles, il existe toujours un risque que les mêmes authentifications pourraient nous donner accès aux services plus importants tels que courrier électronique, documents privés ou bases de données.

Au moyen d'un sniffeur réseau et de quelques scripts permettant d'interpréter le trafic intercepté, en quelques minutes on peut obtenir des centaines de paires *utilisateur/mot de passe* par la méthode décrite ci-dessus. Dans l'authentification HTTP, les mots de passe traversent le réseau en texte clair, et lors de la connexion, les en-têtes avec les mots de passe le traversent plus d'une fois. Pendant la connexion, ils sont renvoyés pour chaque transaction réalisée. Cela est dû à la propriété du protocole HTTP qui ne conserve pas l'état et il est nécessaire de rappeler les données qui sont fournies pendant chaque connexion au serveur ou au proxy. Pour améliorer cette manière d'authentification ou la remplacer par d'autres méthodes plus sûres, il faudrait :

- le combiner avec SSL pour renforcer la sécurité en chiffrant toutes les données de la transmission
- le remplacer par l'authentification digest
- utiliser Kerberos
- le supprimer de tous les endroits où il n'est pas nécessaire. ●

Sur Internet

- *http://livehttpheaders.mozdev.org/* – le plug-in pour mozilla,
- *ftp://ftp.isi.edu/in-notes/rfc2617.txt* – l'authentification HTTP : l'authentification Basic et Digest,
- *http://www.faqs.org/rfcs/rfc2045.html* – le chiffrage du transfert en Base64,
- *http://httpd.apache.org/docs/1.3/howto/auth.html* – la configuration d'apache pour qu'il demande l'authentification,
- *http://rfc.sunsite.dk/rfc/rfc2617.html* – RFC 2617.