RFC 7469 : Public Key Pinning Extension for HTTP

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 18 avril 2015

Date de publication du RFC : Avril 2015

http://www.bortzmeyer.org/7469.html

Depuis que les piratages de Comodo et DigiNotar ont fait prendre conscience du peu de sécurité qu'apportent les certificats X.509, plusieurs solutions ont été proposées pour combler les failles des autorités de certification. La proposition de ce RFC, HPKP ("*HTTP Public Key Pinning*"), consiste à permettre à un client d'**épingler** ("*to pin*") les clés cryptographiques d'un serveur HTTP utilisant TLS, c'est-à-dire à s'en souvenir pendant une durée spécifiée par le serveur. Ainsi, même si un faux certificat est émis par une AC, il ne sera pas accepté.

Un exemple plus récent d'un « vrai / faux certificat » (certificat vrai car émis par une AC reconnue, mais faux car émis sans l'autorisation du titulaire du nom de domaine, et dans l'intention de tromper les utilisateurs) est celui du ministère des finances français <http://www.01net.com/editorial/610140/comment-le-ministere-des-finances-espionne-le-trafic-web-de-ses-collaborateurs/>. Comme souvent avec les certificats mensongers, celui-ci visait Google (je soupçonne fort que le but était de lire le courrier Gmail des employés, mais, à Bercy, on dément, et on affirme que les buts étaient de pouvoir effectuer un filtrage du contenu sur le Web externe, et de passer les flux à l'antivirus) et c'est ce qui explique que cette compagnie soit très présente dans les forums comme l'IETF où on cherche des solutions aux sérieuses faiblesses de X.509. Parmi les solutions élaborées, il y a DANE, qui s'appuie sur le DNS (RFC 6698 [1]), mais Google préfère les certificats à la lumière du jour du RFC 6962 et l'épinglage ("*pinning*") des clés, dans ce tout nouveau RFC 7469.

À noter qu'il existe déjà des en-têtes HTTP permettant au serveur de spécifier au client son comportement, comme le HSTS du RFC 6797. Mais l'épinglage de clés est indépendant de HSTS.

L'épinglage fait partie des techniques TOFU ("*Trust On First Use*") : la première fois qu'un client se connecte au serveur, il ne peut pas savoir si la clé est bonne (il doit se fier à la validation X.509, avec ses

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc6698.txt

limites). À la première connexion, le client HTTP (par exemple un navigateur Web) doit espérer qu'il n'y a pas d'homme du milieu ayant réussi à obtenir un faux certificat (pour cette raison, une attaque persistente, comme celle du ministère des finances, citée plus haut, ne serait pas empêchée par l'épinglage). Aux connexions suivantes, le client HTTP pourra par contre vérifier l'identité du serveur, grâce à la clé épinglée, c'est-à-dire gardée en mémoire. Ce n'est pas idéal (imaginez un homme du milieu ayant un faux certificat pour la première connexion et épinglant ses propres clés, empêchant ainsi toute correction ultérieure !) mais c'est un progrès sur la situation actuelle (valider les clés est un problème difficile en cryptographie). À part des clés partagées à l'avance directement entre les correspondants (ce qui ne passerait évidemment pas à l'échelle pour le Web entier), il n'y a pas de solution parfaitement sûre pour la validation des clés.

La section 2 décrit les détails pratiques. Elle définit un nouvel en-tête HTTP, `Public-Key-Pins:`. Présent dans une réponse HTTPS, cet en-tête indique au client HTTP que ce serait souhaitable qu'il épingle, qu'il mémorise, la clé publique. Le contenu de l'en-tête `Public-Key-Pins:` est le condensat de la clé publique utilisée pour TLS, condensat encodé en Base64 (RFC 4648). L'algorithme de condensation est indiqué dans la valeur de l'en-tête. Aujourd'hui, c'est forcément SHA-256 (RFC 6234). La clé est l'encodage DER du champ `subjectPublicKeyInfo` du certificat X.509 (à noter que les cas des clés brutes du RFC 7250 ou des clés PGP du RFC 6091 ne semblent pas traités, car elles n'ont pas les problèmes des certificats X.509). Une directive `max-age` est obligatoire dans l'en-tête HTTP, pour indiquer le nombre de secondes que doit durer l'épinglage (après quoi le client HTTP « oublie » la clé). Voici un exemple :

```
Public-Key-Pins: max-age=604800;
       pin-sha256="d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM="
```

Il existe aussi deux directives facultatives. `includeSubDomains` indique que l'épinglage s'applique aussi aux serveurs dont le nom est un sous-domaine de celui-ci. Si le client se connecte en HTTPS à `bar.example.com` et reçoit un `Public-Key-Pins:` avec `includeSubDomains`, et qu'il se connecte plus tard à `foo.bar.example.com`, il exigera de trouver la clé épinglée (voir la section 4.2 pour quelques pièges de cette directive). Autre directive facultative, `report-uri` indique à quel URI le client doit signaler les erreurs résultant de l'épinglage. En effet, on peut prévoir que, comme avec toute technique de sécurité, il y aura des problèmes. Par exemple, des webmestres changeront de clé et oublieront de modifier le `Public-Key-Pins:`. Il est donc important que ces problèmes soient détectés rapidement. Ce signalement se fera selon la technique décrite plus loin en section 3. Attention aux conséquences pour la vie privée ! La très détaillée section 5 couvre ce risque d'intrusivité. Voici un exemple avec ces deux directives, et l'utilisation de plusieurs clés (par exemple parce qu'un remplacement est en cours) :

```
Public-Key-Pins: max-age=2592000;
       pin-sha256="E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=";
       pin-sha256="LPJNul+wow4m6DsqxbninhsWHlwfp0JecwQzYpOLmCQ=";
       includeSubDomains;
       report-uri="https://other.example.net/pkp-report"
```

À noter qu'un deuxième en-tête est défini, `Public-Key-Pins-Report-Only:`, qui a exactement la même syntaxe, mais une sémantique différente : le client HTTP teste la clé épinglée et, si elle est fausse, le signale à l'URI indiqué par `report-uri` (qui devient donc obligatoire), mais n'empêche pas l'accès au serveur. C'est utile pour tester avant le vrai déploiement. Les deux en-têtes sont désormais dans le registre des en-têtes <`https://www.iana.org/assignments/message-headers/message-headers.xml`>.

Comme indiqué plus haut, la section 3 décrit la façon dont un client HTTP peut signaler une erreur de validation. Pour cela, le client HTTP doit fabriquer un message en JSON (RFC 8259) et l'envoyer avec la méthode `POST` de HTTP à l'URI indiqué dans la directive `report-uri`. Voici un exemple d'un tel message (la date est au format du RFC 3339) :

```
{
   "date-time": "2014-04-06T13:00:50Z",
   "hostname": "www.example.com",
   "port": 443,
   "effective-expiration-date": "2014-05-01T12:40:50Z"
   "served-certificate-chain": [
     "-----BEGIN CERTIFICATE-----\n
     MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTAlVT\n
     ...
   ],
   "validated-certificate-chain": [
     "-----BEGIN CERTIFICATE-----\n
     MIIEBDCCAuygAwIBAgIDAjppMA0GCSqGSIb3DQEBBQUAMEIxCzAJBgNVBAYTAlVT\n
     ...
   ],
   "known-pins": [
     'pin-sha256="d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM="',
     "pin-sha256=\"E9CZ9INDbd+2eRQozYqqbQ2yXLVKB9+xcprMF+44U1g=\""
   ]
 }
```

Comme souvent en sécurité, le diable est dans les détails, et c'est ce qui explique que la section 4, l'obligatoire « "*Security Considerations*" » soit particulièrement longue. Par exemple, l'épinglage peut avoir des conséquences néfastes si on s'est fait voler sa clé privée : normalement, on génère une nouvelle clé, un nouveau certificat et on demande à l'AC de re-signer. Mais on ne pourra pas l'utiliser car la clé publique correspondant à la clé volée est toujours épinglée dans des tas de navigateurs. On peut réduire ce problème en diminuant la durée d'épinglage. Mais, alors, on réduit aussi la durée de la protection dans le cas où une AC génère un certificat mensonger. Un compromis est donc nécessaire. Une approche possible est d'épingler, non pas la clé du serveur mais une clé intermédiaire dans la chaîne des certificats. Supposons que la société Example, qui gère `example.com` soit cliente de l'AC SmallAC qui a elle-même son certificat signé par une AC très largement reconnue, BigAC. En épinglant la clé de SmallAC, on se protège contre un vol de la clé privée d'`example.com`, **et** contre un faux certificat, même émis par BigAC. La société Example reste vulnérable à un faux certificat produit par SmallAC mais cela diminue sérieusement le nombre d'acteurs qui peuvent attaquer Example. (Notez que DANE a les mêmes possibilités : on peut publier dans le DNS sa propre clé, ou bien celle d'une AC.)

On a vu plus haut qu'on pouvait avoir plusieurs clés dans l'en-tête `Public-Key-Pins:`. Une des utilisations de cette possibilité est le cas où on a une clé de réserve ("*backup pin*"), non utilisée mais gardée en un endroit différent de la clé du certificat actuel. Ainsi, si, pour une raison ou pour une autre, la clé est inutilisable, la clé de réserve peut prendre le relais immédiatement puisqu'elle est déjà épinglée.

Autre cas où l'épinglage peut avoir un effet néfaste, celui de l'épinglage hostile. Imaginons un méchant qui obtienne un faux certificat. Il détourne le trafic (par exemple par un empoisonnement DNS) et envoie les gens vers son serveur HTTPS, qui semblera légitime, en raison du faux certificat. Avec l'épinglage, il peut même prolonger son attaque en épinglant **sa** clé. Si le site légitime n'avait pas été visité avant, ou s'il ne faisait pas d'épinglage, l'attaque peut réussir, bloquant le client HTTPS sur la clé de l'agresseur pour la durée `max-age` spécifiée par ledit agresseur. Il n'y a pas de solution miracle pour ce problème. Une possibilité est de précharger les clés de sites connus dans le navigateur. Une autre est d'utiliser les certificats au grand jour du RFC 6962. (Et, bien sûr, il y a DANE, RFC 6698, que Google prend toujours soin de ne pas mentionner.)

Autres risques, ceux pour la vie privée (section 5). Un serveur peu sympathique peut utiliser l'épinglage comme une sorte de "*cookie*" discret. Par exemple, il peut mettre une petite image dans un sous-domaine, épingler des fausses clés pour ce sous-domaine (une différente par visiteur) et attendre les signalements à `report-uri` pour identifier un client HTTPS unique. Des sites différents peuvent même coopérer pour avoir le même `report-uri`, pour suivre un client à travers plusieurs sites.

À noter aussi que les signalements d'un problème de validation contiennent la chaîne de certificats utilisée. Cela peut permettre d'identifier l'utilisation de faux certificats par telle ou telle organisation. C'est plutôt une bonne chose pour détecter des attaques comme celle effectuée au ministère des finances <http://www.zdnet.fr/actualites/des-faux-certificats-de-l-anssi-detectes-et-bloques-p
htm> mais cela ne sera évidemment pas apprécié de ceux qui veulent faire du détournement de trafic.

Comme toujours en sécurité, l'**utilisabilité** est cruciale (section 7). Lorsqu'un épinglage a eu lieu, et que le trafic est ensuite détourné vers un faux serveur, le client HTTPS (par exemple le navigateur Web) va refuser l'accès à ce faux serveur ce qui, du point de vue de M. Toutlemonde devant son navigateur, est un déni de service. Il va donc falloir bien expliquer à M. Toutlemonde ce qui se passe, pour éviter qu'il n'accuse le site légitime.

Le RFC recommande aussi que l'utilisateur puisse accéder facilement à la liste des clés épinglées et aussi, ce qui me semble plus contestable, qu'il puisse la modifier, par exemple en désépinglant les clés. Cela me semble dangereux car cela peut ouvrir une voie à l'ingénierie sociale (« il y a un petit problème technique, si l'accès vous est refusé, cliquez sur "*Clear all pinned keys*" »).

L'annexe A contient un exemple, utilisant OpenSSL en ligne de commande, pour générer le `Public-Key-Pins:`. J'en ai fait un petit script (en ligne sur `http://www.bortzmeyer.org/files/make-pin.sh`) qui s'utilise ainsi :

```
% make-pin.sh www.ietf.org
...
DHKscLdFrBmZiBGxMdZiyaxp29vnyyPltRS1ZmTF09Y=
```

Et hop, on n'a plus qu'à mettre `Public-Key-Pins: max-age=604800; pin-sha256="DHKscLdFrBmZiBGx` dans la réponse du serveur HTTP. (Tom me fait remarquer à juste titre qu'une clé de secours est obligatoire - section 4.3 - et que donc il faut toujours au moins deux `pin-sha256`. J'ai simplifié.) Les premières versions de l'"*Internet-Draft*" qui a mené à ce RFC utilisaient un programme en Go, (en ligne sur `http://www.bortzmeyer.org/files/make-pin.go`) :

```
% openssl s_client -servername www.ietf.org -connect www.ietf.org:443 > ietf.pem
...
% ./make-pin ietf.pem
Hex: 0c72ac70b745ac19998811b131d662c9ac69dbdbe7cb23e5b514b56664c5d3d6
Base64: DHKscLdFrBmZiBGxMdZiyaxp29vnyyPltRS1ZmTF09Y=
```

À noter que l'annexe B contient quelques conseils pratiques à l'usage des gérants de serveurs HTTPS, sur le bon déploiement de l'épinglage. Par exemple, il est recommandé de commencer doucement, en mode "*report-only*" avant de se jeter dans le grand bain et de risquer de bloquer ses clients. Comme avec n'importe quelle technique de sécurité, on risque de se planter soi-même si on ne fait pas attention : n'épinglez pas bêtement !

Question mises en œuvre, Chrome et Firefox avaient déjà de l'épinglage, avec une série de clés mise en dur dans le logiciel (voir la description de Mozilla <https://wiki.mozilla.org/SecurityEngineering/Public_Key_Pinning> et leur annonce <https://blog.mozilla.org/security/2014/09/02/public-key-pinning/>). Vous pouvez tester ici <https://projects.dm.id.lv/Public-Key-Pins_test> si votre navigateur gère l'épinglage. Une description de l'épinglage des clés dans Chrome avait été faite par Adam Langley <https://www.imperialviolet.org/2011/05/04/pinning.html> et sa lecture est recommandée (par exemple pour comprendre pourquoi on épingle juste des clés et pas des certificats entiers). Pour le monde Microsoft, voir leur article <http://blogs.technet.com/b/srd/archive/2013/05/08/emet-4-0-s-certificate-trust-feature.aspx>. Il y aussi un article détaillé sur le concept, chez OWASP <https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning>, et l'article de Robert Love <http://blog.rlove.org/2015/01/public-key-pinning-hpkp.html> avec des détails pratiques,

Merci à Florian Maury et Kim Minh Kaplan pour leur relecture.
