RFC 6960 : X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 7 juin 2013

Date de publication du RFC : Juin 2013

http://www.bortzmeyer.org/6960.html

Le protocole OCSP permet à un client X.509 (par exemple un navigateur Web engagé dans une connexion HTTPS) de s'informer en temps réel sur l'état d'un certificat, notamment afin de savoir s'il est révoqué ou pas. Ce nouveau RFC remplace (avec de légers changements) l'ancienne norme OCSP qui était dans le RFC 2560 [1].

Normalement, le destinataire d'un certificat X.509 (par exemple le navigateur Web cité plus haut), authentifie ce dernier en utilisant les clés publiques des AC qu'il a dans son magasin, et en vérifiant dans une liste de CRL, mise à jour périodiquement, que le certificat est toujours d'actualité, qu'il n'a pas été révoqué par l'AC émettrice (section 3.3 du RFC 5280). En complément, ou à la place de ces CRL, le destinataire peut utiliser le protocole OCSP normalisé dans ce RFC. OCSP est un protocole synchrone : le destinataire du certificat émet une requête à un serveur OCSP et attend sa réponse avant de valider le certificat. Il peut ainsi obtenir des informations plus à jour qu'avec des CRL.

La requête OCSP contient (un survol du protocole figure en section 2 de ce RFC) le numéro de série du certificat qu'on veut authentifier. La réponse dépend de si le répondeur (le serveur OCSP) a l'information sur ce certificat. Si oui, il répond positivement (`good`, le certificat est valable) ou négativement (`revoked`, le certificat ne doit pas être accepté), en indiquant l'heure (un certificat valable à un moment peut être révoqué par la suite) et sa réponse est signée (typiquement avec le certificat de l'AC). Si non, si le répondeur n'a pas l'information, il répond `unknown`.

Les réponses possibles sont donc `good`, `revoked` et `unknown`. Notez que, de manière surprenante, `good` ne dit pas que le certificat existe, simplement qu'il n'a pas été révoqué. (Cette question a fait l'objet

1. Pour voir le RFC de numéro NNN, `https://www.ietf.org/rfc/rfcNNN.txt`, par exemple `https://www.ietf.org/rfc/rfc2560.txt`

de chauds débats dans le groupe de travail : que doit faire un répondeur OCSP lorsqu'il reçoit une requête pour un certificat qu'il est censé connaître mais qu'il n'a pas émis ?)

Le répondeur peut aussi être dans l'incapacité de fournir une réponse et, dans ce cas, le message d'erreur résultant n'est pas signé. Parmi les causes possibles d'erreur, une requête incorrecte, une erreur dans le répondeur (équivalent du `500 Internal server error` de HTTP), un problème temporaire qui nécessite que le client réessaie plus tard, un client non autorisé, etc.

La réponse inclut souvent des temps : dernier moment où l'information retournée était correcte, moment de la prochaine mise à jour de l'information, moment de la signature de l'information (les réponses peuvent être pré-produites) ou moment où la révocation a eu lieu.

Mais comment un destinataire de certificat sait-il où trouver le serveur OCSP ? Il est typiquement indiqué dans le certificat qu'on teste, dans l'extension "*Authority Information Access*" (section 4.2.2.1 du RFC 5280). Si la méthode d'accès est `id-ad-ocsp`, le contenu de cette extension est un URL pointant vers le serveur OCSP. Voici un exemple dans un certificat récupéré sur l'Internet :

```
% openssl x509 -text -in /tmp/site.pem
...
            Authority Information Access:
                OCSP - URI:http://rapidssl-ocsp.geotrust.com
                CA Issuers - URI:http://rapidssl-aia.geotrust.com/rapidssl.crt
```

Mais cette information peut aussi être codée en dur dans le client.

La plupart du temps, l'URL en question sera un URL HTTP, et OCSP tournera donc au dessus de ce protocole (parfois en HTTPS). L'annexe A décrit en détail ce mécanisme. OCSP peut utiliser `GET` ou `POST`. Dans ce dernier cas, la requête HTTP aura le type `application/ocsp-request` et la requête OCSP forme le corps de la requête `POST`, après son encodage en DER. Même chose pour la réponse, type `application/ocsp-response`, et du DER dans le corps HTTP.

Voilà, vous savez l'essentiel, la section 4 décrit tout le protocole, en ASN.1 (en s'appuyant sur les modules de l'annexe B). Par exemple, une requête est une `TBSRequest` qui contient plusieurs `Request`, chacune concernant un certificat :

```
OCSPRequest     ::=     SEQUENCE {
       tbsRequest                  TBSRequest,
       optionalSignature   [0]     EXPLICIT Signature OPTIONAL }

   TBSRequest      ::=     SEQUENCE {
       version             [0]     EXPLICIT Version DEFAULT v1,
       requestorName       [1]     EXPLICIT GeneralName OPTIONAL,
       requestList                 SEQUENCE OF Request,
       requestExtensions   [2]     EXPLICIT Extensions OPTIONAL }

 ...

  Request         ::=     SEQUENCE {
       reqCert                     CertID,
       singleRequestExtensions     [0] EXPLICIT Extensions OPTIONAL }

   CertID          ::=     SEQUENCE {
       hashAlgorithm       AlgorithmIdentifier,
       issuerNameHash      OCTET STRING, -- Hash of Issuer's DN
       issuerKeyHash       OCTET STRING, -- Hash of Issuers public key
       serialNumber        CertificateSerialNumber }
```

Et la réponse est un code de retour et la réponse elle-même :

```
OCSPResponse ::= SEQUENCE {
      responseStatus         OCSPResponseStatus,
      responseBytes          [0] EXPLICIT ResponseBytes OPTIONAL }

   OCSPResponseStatus ::= ENUMERATED {
       successful            (0),  --Response has valid confirmations
       malformedRequest      (1),  --Illegal confirmation request
       internalError         (2),  --Internal error in issuer
       tryLater              (3),  --Try again later
                                   --(4) is not used
       sigRequired           (5),  --Must sign the request
       unauthorized          (6)   --Request unauthorized
   }

   ResponseBytes ::=       SEQUENCE {
       responseType   OBJECT IDENTIFIER,
       response       OCTET STRING }
```

Avec `response` comportant à son tour des structures ASN.1 arrivant finalement à :

```
  SingleResponse ::= SEQUENCE {
      certID                       CertID,
      certStatus                   CertStatus,
      thisUpdate                   GeneralizedTime,
      nextUpdate         [0]       EXPLICIT GeneralizedTime OPTIONAL,
      singleExtensions   [1]       EXPLICIT Extensions OPTIONAL }

   CertStatus ::= CHOICE {
       good        [0]     IMPLICIT NULL,
       revoked     [1]     IMPLICIT RevokedInfo,
       unknown     [2]     IMPLICIT UnknownInfo }

   RevokedInfo ::= SEQUENCE {
       revocationTime              GeneralizedTime,
       revocationReason    [0]     EXPLICIT CRLReason OPTIONAL }
```

On note que la raison de la révocation est indiquée. Elle n'est pas toujours indiquée par le répondeur OCSP (cela peut être une information sensible). La présence de cette information dans les CRL a permis à l'EFF de réaliser une intéressante étude sur les raisons des révocations `<https://www.eff.org/deeplinks/2011/10/how-secure-https-today>` (la section « *"How often are these attacks occurring?"* »).

La section 5 revient sur l'analyse de sécurité d'OCSP. On notera que les problèmes de vie privée n'y sont pas mentionnés. Pourtant, la requête OCSP indique au répondeur (en général l'AC) quels sites Web ont été visités. C'est d'ailleurs ainsi qu'il a été possible de prouver l'utilisation en Iran des vrais/faux certificats de DigiNotar, en regardant les requêtes OCSP de leurs navigateurs vers l'AC.

Un autre problème de sécurité n'est pas mentionné : que doit faire le navigateur Web qui tente d'authentifier un certificat lorsque la requête OCSP n'aboutit pas (peut-être parce qu'un Homme du Milieu bloque l'accès au serveur OCSP) ? S'il refuse d'authentifier le certificat, la connexion échoue. Mais s'il accepte, OCSP ne protège plus tellement.

Les changements depuis le RFC 2560 sont mineurs, et sont résumés dans la section 1.

Merci à Erwann Abalea pour sa relecture précise.