Conserver sa clé avec Let's Encrypt, certbot et dehydrated

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 2 décembre 2017

https://www.bortzmeyer.org/letsencrypt-certbot-keep-key.html

Beaucoup de gens utilisent désormais l'AC Let's Encrypt. Ce n'est pas la première autorité de certification qui permet d'avoir un certificat en faisant tout en ligne, ni la première gratuite, mais elle est néanmoins très utilisée (au point de devenir un nouveau SPOF de l'Internet). Par défaut, les outils Let's Encrypt comme certbot <https://certbot.eff.org/> créent une nouvelle clé quand le certificat est renouvelé. Dans quels cas est-ce gênant et comment éviter cela ?

Un petit rappel sur les certificats : un certificat, c'est tout bêtement une clé publique, avec quelques métadonnées dont les plus importantes sont la signature de l'AC et la date d'expiration (qui, avec Let's Encrypt, est très rapprochée). Renouveler un certificat, c'est demander une nouvelle signature à l'AC. Si la clé n'est pas trop ancienne, ou n'a apparemment pas été compromise, il n'est pas nécessaire de la changer.

Mais les outils existants le font quand même systématiquement (c'est un choix des outils, comme certbot <https://certbot.eff.org/> ou dehydrated <https://dehydrated.de/>, ce n'est pas une obligation du protocole ACME, ni de l'AC Let's Encrypt). Cette décision a probablement été prise pour garantir que la clé soit renouvelée de temps en temps (après tout, il est raisonnable de supposer que, tôt ou tard, elle sera compromise, et ce ne sera pas forcément détecté par le propriétaire).

Et pourquoi est-ce gênant de changer de clé à chaque renouvellement (donc tous les trois mois avec Let's Encrypt) ? Cela ne pose pas de problème pour l'utilisation habituelle d'un serveur HTTPS. Mais c'est ennuyeux si on utilise des techniques de sécurité fondées sur un épinglage de la clé, c'est-à-dire une authentification de la clé publique utilisée. Ces techniques permettent de résoudre une grosse faille de X.509, le fait que n'importe quelle AC, même si vous n'en êtes pas client, puisse émettre un certificat pour n'importe quel domaine. Parmi ces techniques de sécurité :

— DANE, normalisé dans le RFC 6698 [1]. Si on met les utilisations 1 (« PKIX-EE ») ou 3 (« DANE-EE »), DANE déclare la clé, un changement de celle-ci invalide donc l'enregistrement DANE.

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

— HPKP (RFC 7649).
— Et les autres solutions d'épinglage comme celle spécifiée dans la section 4.2 du RFC 7858 pour les serveurs DNS-sur-TLS. La supervision des serveurs publics utilisant ce protocole <https://dnsprivacy.org/jenkins/job/dnsprivacy-monitoring/> les authentifie ainsi (« *"Strict mode - SPKI only"* »).

Si on utilise l'outil certbot <https://certbot.eff.org/>, qui est celui officiellement recommandé par Let's Encrypt, la méthode normale d'utilisation est, la première fois :

```
% sudo certbot certonly --webroot --webroot-path /var/lib/letsencrypt -d www.example.com
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Obtaining a new certificate
Performing the following challenges:
http-01 challenge for www.example.com
Using the webroot path /var/lib/letsencrypt for all unmatched domains.
Waiting for verification...
Cleaning up challenges
Generating key (2048 bits): /etc/letsencrypt/keys/0000_key-certbot.pem
Creating CSR: /etc/letsencrypt/csr/0000_csr-certbot.pem
...
```

Let's Encrypt a testé la présence du défi sur le serveur, on le voit dans le journal du serveur HTTP :

```
2600:3000:2710:200::1d - - [13/Sep/2017:16:08:46 +0000] "GET /.well-known/acme-challenge/2IlM1PbP9QZlAA22xvF
```

On va alors créer l'enregistrement TLSA (DANE) :

```
% tlsa --create --selector 1  www.example.com
Got a certificate with Subject: /CN=www.example.com
_443._tcp.www.example.com. IN TLSA 3 1 1 f582936844ec355cfdfe8d9d1a42e9565940602c71c7abd2c36c732daa64b9db
Got a certificate with Subject: /CN=www.example.com
_443._tcp.www.example.com. IN TLSA 3 1 1 f582936844ec355cfdfe8d9d1a42e9565940602c71c7abd2c36c732daa64b9db
```

(L'option `--selector 1` est pour faire apparaitre dans l'enregistrement TLSA la clé publique seulement et non pas tout le certificat, ce que ferait le sélecteur par défaut, 0. C'est expliqué plus en détail plus loin.) À ce stade, on a un certificat Let's Encrypt, un enregistrement DANE qui correspond et tout le monde est heureux :

```
% tlsa --verify --resolvconf="" www.example.com
SUCCESS (Usage 3 [DANE-EE]): Certificate offered by the server matches the TLSA record (x.y.z.t)
```

Maintenant, si on renouvelle le certificat quelques mois plus tard :

```
% certbot --quiet  --webroot --webroot-path /usr/share/nginx/local-html renew
```

Cela change la clé. Regardez avec OpenSSL :

```
# Avant
 % openssl x509 -pubkey -in /etc/letsencrypt/live/www.example.com/fullchain.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsUd3mG5QK6EdTtYh0oLJ
nkIovYkWXV8QLQMvAthzURbyeIlQ8CXeTNCT6odh/VVyMn49IwkRJl6B7YNhYiRz
pbmIxxzceAhKNAg6TF/QylHa1HWvHPniZF02NJAXCxMO5Y8EZ7n0s0cGz4XD5PGA
XctV6ovA3fR8b2bk9t5N+UHklWvIOT7x0nVXWmWmrXzG0LX/P4+utZJjRR6Kf5/H
9GDXprklFCbdCTBkhyPBgdiJDnqzdb6hB1aBEsAMd/Cplj9+JKtu2/8Pq6MOtQeu
364N+RKcNt4seEr6uMOlRXzWAfOHI51XktJT64in1OHyoeRMV9dOWOLWIC2KAlI2
jwIDAQAB
-----END PUBLIC KEY-----

# Après
% openssl x509 -pubkey -in /etc/letsencrypt/live/www.example.com/fullchain.pem
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6MF8Dw3JQ58n8B/GvWYI
Vd+CG7PNFA+Ke7B+f9WkzEIUPzAHq7qk1v7dOitD3WsRKndJDPxZAq7JrgOiF/0y
4726HhYR1bXOTziAbk0HzR+HwECo1vz26fqPnNpZ3M46PQFQU9uq2pgHtBwCVMQ+
Hi1pYKnB2+ITl11DBLacSHP7WZZGHXbEqW5Cc710m6aTt18L+OgqxuQSgV+khh+W
qWqd2bLq32actLEVmfR4uWX7fh/g6I7/p3ohY7Ax4WC30RfWZk3vLyNc/8R0/67c
bVIYWmkDhgXy6UlrV2ZgIO2K8oKiJBMHjnaueHIfu1ktubqM1/u1yLKwXW16UAxm
5QIDAQAB
-----END PUBLIC KEY-----
```

À partir de là, l'enregistrement DANE ne correspond plus, la clé copiée à l'extérieur n'est plus la clé utilisée.

Avec certbot, la solution est de ne pas laisser le client ACME choisir le certificat, mais de fabriquer un CSR et de lui indiquer de l'utiliser systématiquement (cf. la documentation officielle `<https://certbot.eff.org/faq/#can-i-use-an-existing-private-key-or-certificate-signing-request-csr`

```
% certbot certonly --webroot -w /usr/share/nginx/local-html -d dns-resolver.yeti.eu.org --csr /etc/letsencrypt-l
No handlers could be found for logger "certbot.crypto_util"
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Performing the following challenges:
http-01 challenge for dns-resolver.yeti.eu.org
Using the webroot path /var/lib/letsencrypt for all unmatched domains.
Waiting for verification...
Cleaning up challenges
Server issued certificate; certificate written to /etc/letsencrypt-local/dns-resolver.yeti.eu.org.pem
```

Comment on avait fabriqué un CSR ? OpenSSL le permet. Faisons-en un joli, utilisant la cryptographie à courbes elliptiques :

```
%  openssl ecparam -out yeti-resolver.pem -name prime256v1 -genkey

%  openssl req -new -key yeti-resolver.pem  -nodes -days 1000 -out yeti-resolver.csr
You are about to be asked to enter information that will be incorporated
into your certificate request.
...
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Dahu
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:dns-resolver.yeti.eu.org
Email Address []:yeti@eu.org
```

Avec ce CSR, et en appelant certbot depuis cron avec les options indiquées plus haut, indiquant le même CSR (`certbot certonly --webroot -w /usr/share/nginx/local-html -d dns-resolver.yeti.eu --csr /etc/letsencrypt-local/yeti-resolver.csr --cert-path /etc/letsencrypt-local/tmp.pem`),

---

https://www.bortzmeyer.org/letsencrypt-certbot-keep-key.html

la clé reste constante, et DANE et HPKP fonctionnent. Petit inconvénient : avec ces options, certbot renouvelle le certificat à chaque fois, même quand ça n'est pas nécessaire. (Depuis l'écriture initiale de cet article, certbot a ajouté l'option <https://github.com/certbot/certbot/pull/5901> `--reuse-key`, qui résout proprement le problème, et est donc une meilleure solution que d'utiliser son CSR.)

Et si on utilise dehydrated <https://dehydrated.de/> au lieu de certbot, comme client ACME ? Là, c'est plus simple, on met dans le fichier de configuration `/etc/dehydrated/config` l'option :

```
PRIVATE_KEY_RENEW="no"
```

Et cela suffit. Ceci dit, dehydrated a un gros inconvénient, il est bavard <https://github.com/lukas2511/dehydrated/issues/455>. Quand on le lance depuis cron, il affiche systématiquement plusieurs lignes, même s'il n'a rien à dire.

Pour revenir à la question du choix du sélecteur DANE (RFC 6698 et RFC 7218), il faut noter que tout renouvellement change le certificat (puisqu'il modifie au moins la date d'expiration). Il ne faut donc pas utiliser le sélecteur 0 « `Cert` » (qui publie le condensat du certificat entier dans l'enregistrement TLSA) mais le sélecteur 1 « `SPKI` » (qui ne met que le condensat de la clé). Le problème existe avec toutes les AC mais est plus aigu pour Let's Encrypt où on renouvelle souvent. L'annexe A.1.2 du RFC 6698 l'explique bien.

Enfin, un avertissement de sécurité : avec les méthodes indiquées ici, le client ACME ne change plus de clé du tout. C'est donc désormais à vous de penser à créer une nouvelle clé de temps en temps, pour suivre les progrès de la cryptanalyse.

Si vous voulez des exemples concrets, `dns-resolver.yeti.eu.org` (uniquement en IPv6) utilise certbot, alors que `dns.bortzmeyer.org` et `mercredifiction.bortzmeyer.org` utilisent dehydrated. Prenons `dns.bortzmeyer.org`. Son enregistrement TLSA :

```
% dig TLSA _443._tcp.dns.bortzmeyer.org
...
;; ANSWER SECTION:
_443._tcp.dns.bortzmeyer.org. 80968 IN TLSA 1 1 1 (
C05BF52EFAB00EF36AC6C8E1F96A25CC2A79CC714F77
055DC3E8755208AAD0E4 )
...
;; Query time: 0 msec
;; SERVER: 127.0.0.1#53(127.0.0.1)
;; WHEN: Sat Dec 02 17:03:18 CET 2017
;; MSG SIZE  rcvd: 2311
```

Il déclare une contrainte sur le certificat du serveur (champ "*Certificate usage*" à 1, PKIX-EE), ne testant que la clé (champ "*Selector*" à 1), l'enregistrement est un condensat SHA-256 (champ "*Matching type*" à 1). On peut vérifier que l'enregistrement DANE est correct avec hash-slinger <http://people.redhat.com/pwouters/hash-slinger/> :

```
% tlsa --verify --resolvconf="" dns.bortzmeyer.org
SUCCESS (Usage 1 [PKIX-EE]): Certificate offered by the server matches the one mentioned in the TLSA record
SUCCESS (Usage 1 [PKIX-EE]): Certificate offered by the server matches the one mentioned in the TLSA record
```

ou bien avec .

---

https://www.bortzmeyer.org/letsencrypt-certbot-keep-key.html