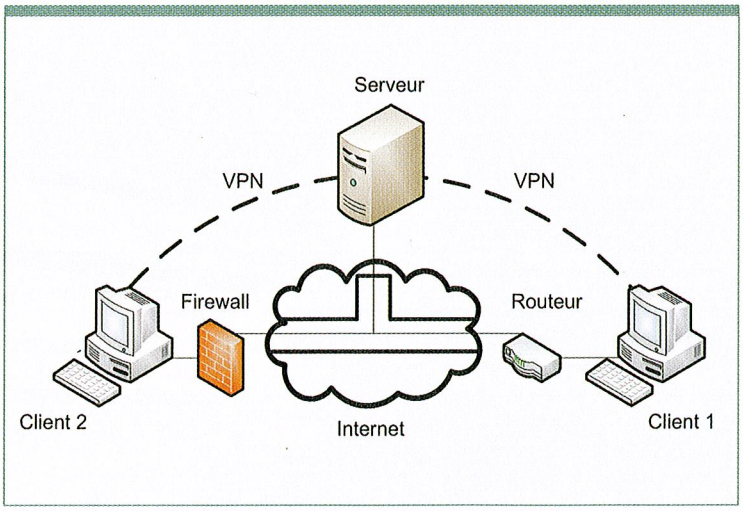
Denis Bodor

Authentification matérielle pour votre VPN

Dans le numéro 95 de juin dernier, nous avons vu comment mettre en œuvre rapidement un VPN basé sur OpenVPN et utilisant une authentification SSL/TLS. Voyons aujourd'hui comment prendre en charge un périphérique permettant de stocker des certificats SSL/TLS, ainsi que des clefs privées.

Fidèle lecteur, vous n'avez, sans doute, pas manqué le numéro 88 de novembre dernier. Celui-ci incluait deux articles traitant du support des *smartcards* sous GNU/Linux et des applications en mesure de se servir du *middleware* mis en œuvre. Étaient traités dans ces articles, des outils comme Apache, Firefox, OpenSSH, etc. Plus récemment (numéro 95), je traitais de la mise en œuvre d'OpenVPN selon la configuration illustrée ci-dessous :



L'idée est ici de montrer une configuration possible pour un VPN avec authentification SSL reposant sur un eToken Pro 32k d'Aladdin. Pour rappel, ce type de composant n'est pas une simple clef USB. Il s'agit d'un processeur cryptographique couplé à une petite mémoire Flash. Le contenu du *token*, comme celui d'une smartcard, n'est accessible que via un protocole strict et sécurisé. À l'instar des CB (qui sont également des smartcards), des violations répétées de ce protocole peuvent totalement verrouiller les données du token. Le plus bel exemple est l'utilisation récurrente d'un mauvais code PIN.

NOTE

Seules les versions récentes d'OpenVPN supportent l'utilisation d'une smartcard ou d'un token. L'actuelle version stable, 2.0.9, ne fonctionnera pas avec les explications qui suivent. Les essais ont été réalisés avec une version 2.1rc2 telle qu'empaquetée pour Debian Unstable. La dernière version, à la date où sont écrites ces lignes, est la 2.1rc4.

Initialisation et mise en œuvre de l'eToken

Il conviendra, avant toutes choses, d'installer et configurer le middleware OpenCT ainsi que les outils OpenSC. Après installation des paquets pour votre distribution, assurez-vous simplement que le périphérique est reconnu avec :

```
% openct-tool list
  0 Aladdin eToken PRO
```

On trouve ici le token Aladdin qui est à la fois le lecteur et la smartcard. On obtiendra plus d'information de la carte avec :

```
% opensc-tool --atr
3b:f2:98:00:ff:c1:10:31:fe:55:c8:03:15
```

L'ATR est *Answer To Reset*, la chaîne d'identification automatiquement émise suite à une réinitialisation de la carte (reset). Ceci devrait être suffisant pour vérifier le fonctionnement de l'installation.

On passe à la suite en initialisant la carte, à commencer par les données de l'utilisateur :

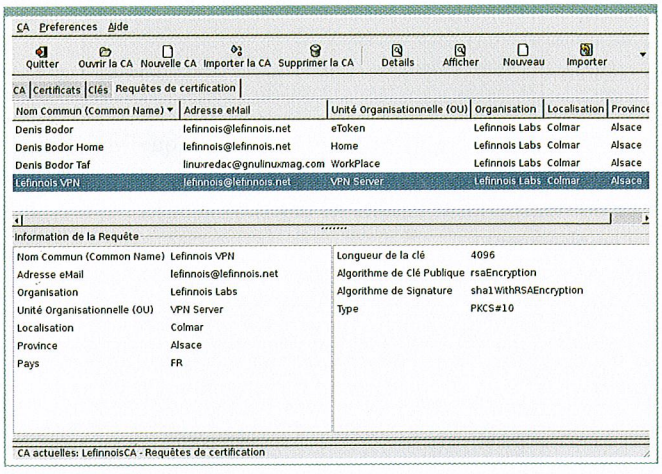
```
pkcs15-init -ECT
New Security Officer PIN (Optional - press return for no PIN).
Please enter Security Officer PIN:

% pkcs15-init -PT -a 1 -l denis -v
Connecting to card in reader Aladdin eToken PRO 64k...
Using card driver Siemens CardOS.
Found OpenSC Card
About to store PIN.
New User PIN.
Please enter User PIN: ****
Please type again to verify: ****
Unblock Code for New User PIN (Optional - press return for no PIN).
Please enter User unblocking PIN (PUK): ****
Please type again to verify: ****
```

Notre token est prêt à être utilisé.

Certificats, la voie de la facilité

Il est possible de configurer avec soin son installation d'OpenSSL, afin de mettre en œuvre une véritable autorité de certification. L'utilisateur avisé remarquera que, même si la commande `openssl` est relativement intuitive, il devient rapidement lassant de l'utiliser. En particulier, lorsqu'il s'agit de gérer plusieurs clefs, certificats, requêtes, listes de révocation, etc.

Le plus simple lorsque vous avez besoin de devenir une autorité de certification et de gérer un grand nombre de certificats est d'utiliser une application dédiée ou plus exactement une infrastructure dédiée : une PKI (*Public Key Infrastructure*) ou IGC (Infrastructure de Gestion de Clefs) en bon français. Il en existe un grand nombre, en logiciel libre ou non, adaptées ou non à vos besoins. Ici, nous voulons simplement remplacer la commande `openssl` par quelque chose de plus souple. Mon choix s'est porté sur TinyCA. Il ne s'agit pas vraiment d'un élément de PKI, mais plutôt d'une interface de gestion pour autorité de certification.



Dans l'interface, nous pouvons créer dans l'ordre :

- l'autorité de certification (AC) avec son certificat, sa clef privée et la liste de révocation ;
- les demandes de certificat pour chaque acteur du VPN ;
- les certificats découlant de la signature des demandes par l'AC ;

Quelques points importants sont à prendre en compte :

- le type et la taille de la clef sont importants en ce qui concerne le certificat qui sera stocké dans le token. Ici, notre eToken Pro 32k ne supporte que les clefs RSA/1024.
- le choix du champ CN (*Common Name*) et des autres renseignements de ce type (champs ST, L, O, OU... bref le `Subject` dans son ensemble) est très important. C'est via ces éléments que vous pourrez identifier le serveur, les clients et choisir le certificat dans le token.

Distribution des certificats

À présent que l'AC a fait son travail et dispose de certificats à distribuer aux acteurs du VPN, il ne reste plus qu'à déployer tout cela.

Sur le serveur OpenVPN, nous plaçons :

- Le fichier de paramètres Diffie Hellman `dh1024.pem` généré par `openssl dhparam -out dh1024.pem 1024`.
- Le certificat de l'AC exporté de tinyCA au format PEM : `LefinnoisCA-cacert.pem`.
- Le certificat du serveur : `LefinnoisCA-crl.pem`.
- La clef privée accompagnant le certificat : `serveur.pem`. Comme il s'agit d'un serveur, il est peut-être souhaité (mais non souhaitable) de voir le service OpenVPN démarrer automatiquement. Si tel est le cas, la demande de mot de passe protégeant la clef privée peut devenir un problème. Il faut donc retirer le chiffrement sur ce fichier avec `openssl rsa -in serveur.pem -out serveurnopass.pem`. Dès lors, la clef privée devient lisible par n'importe qui, et est donc potentiellement en danger.
- La liste de révocation `LefinnoisCA-crl.pem` afin que le serveur puisse refuser les connexions provenant du client dont le certificat n'est plus valide, car révoqué.

Côté configuration d'OpenVPN, nous avons :

```
local 9.851.6.814
port 1818
proto udp
dev tap
mode server
ifconfig 192.168.25.1 255.255.255.0
keepalive 10 60
client-config-dir /etc/openvpn/ccd
ccd-exclusive
client-to-client
push "route-gateway 192.168.25.1"
tls-server
dh /etc/openvpn/dh1024.pem
ca /etc/openvpn/LefinnoisCA-cacert.pem
cert /etc/openvpn/serveur.crt
key /etc/openvpn/serveur.pem
crl-verify /etc/openvpn/LefinnoisCA-crl.pem
reneg-sec 21600
comp-lzo
```

Rien de spécial ici, si ce n'est l'utilisation de `client-config-dir` spécifiant un répertoire dans lequel nous trouvons :

```
Denis_Bodor
Denis_Bodor_Home
Denis_Bodor_Taf
```

Ces fichiers sont nommées d'après le champ CN des certificats qui vont être présentés pour l'authentification (les espaces sont remplacées par des _). Chaque fichier contient une simple ligne permettant d'attribuer une adresse au client :

```
ifconfig-push 192.168.25.12 255.255.255.0
```

Rapidement, pour un client « normal » du VPN, nous avons les fichiers home.crt (certificat), home.pem (clef privée) et LefinnoisCA-cacert.pem (certificat de l'AC). Pour la configuration, cela se résume en :

```
client
remote 9.851.6.814 1818
dev tap
tls-client
ca /etc/openvpn/LefinnoisCA-cacert.pem
cert /etc/openvpn/home.crt
key /etc/openvpn/home.pem
reneg-sec 21600
comp-lzo
```

Et mon token ?

Nous arrivons à la partie la plus intéressante, celle du client nomade. La première chose à faire, comme pour les autres configurations est d'intégrer la clef privée et le certificat signé par l'AC dans le système. Cette fois, il ne s'agit pas d'une simple copie de fichiers :

```
% pkcs15-init -S token.pem -a 1 -u sign,decrypt --split-key
Please enter passphrase to unlock secret key:
User PIN required.
Please enter User PIN:

% pkcs15-init -X token.crt -v -a 1
Connecting to card in reader Aladdin eToken PRO...
Using card driver Siemens CardOS.
Found OpenSC Card
About to store certificate.
User PIN required.
Please enter User PIN:
```

L'option --split-key n'est nécessaire que pour certains tokens et smartcards qui ne peuvent utiliser une même clef publique pour la signature et le chiffrement. Cette option nous permet de cloner la clef en mémoire et donc de créer deux objets identiques. Notre token est prêt.

OpenVPN n'intègre pas directement les éléments nécessaires au dialogue avec un token ou une smartcard. À l'instar de la commande openssl, il peut charger un *provider* qui fournit une interface PKCS#11. PKCS#11 est le standard définissant l'API permettant l'accès aux périphériques cryptographiques. L'API elle-même s'appelle « cryptoki ».

Le provider PKCS#11 est installé en même temps qu'OpenSC. C'est /usr/lib/opensc-pkcs11.so. Ainsi, la première chose à faire est de s'assurer que tout cela fonctionne bien :

```
% openvpn --show-pkcs11-slots /usr/lib/opensc-pkcs11.so
Provider Information:
cryptokiVersion:        2.11
manufacturerID:         OpenSC Project
flags:                  0

The following slots are available for use with this provider.
Slots: (id - name)
        0 - Aladdin eToken PRO
        1 - Aladdin eToken PRO
        2 - Aladdin eToken PRO
        3 - Aladdin eToken PRO
[...]
```

Seul le premier *slot* nous intéresse. Nous avons besoin des informations sur le token, tel que vu par openssl et le provider PKCS#11. Voyons les objets disponibles dans ce premier slot :

```
% openvpn --show-pkcs11-objects /usr/lib/opensc-pkcs11.so 0
PIN:
Token Information:
          label:                OpenSC Card (denis)
          manufacturerID:       OpenSC Project
          model:                PKCS #15 SCard
          serialNumber:         2221AA071518
          flags:                0000040c

You can access this token using
--pkcs11-slot-type "label" \
--pkcs11-slot "OpenSC Card (denis)" options.
```

openvpn est fort sympathique : il nous explique comment, en ligne de commande, accéder directement à ce slot/token. Il liste ensuite les objets contenus à cet emplacement :

```
The following objects are available for use with this token.
Each object shown below may be used as a parameter to
--pkcs11-id-type and --pkcs11-id options.

Object
          Type:                         Private Key
          CKA_ID:               45
          CKA_LABEL:            Private Key
          CKA_SIGN:             TRUE
          CKA_SIGN_RECOVER:     TRUE
Object
          Type:                         Certificate
          CKA_ID:               45
          CKA_LABEL:            Certificate
          subject:              /C=FR/ST=Alsace/
                                L=Colmar/O=Lefinnois Labs/
                                OU=eToken/CN=Denis Bodor/
                                emailAddress=lefinnois@lefi
nnois.net
          serialNumber:                 01
          notBefore:            070611131442Z
Object
          Type:                         Public Key
          CKA_ID:               45
          CKA_LABEL:            Certificate
Object
          Type:                         Public Key
          CKA_ID:               45
          CKA_LABEL:            Public Key
```

Nous obtenons toutes les informations utiles pour construire notre fichier de configuration :

```
client
remote 9.851.6.814 1818
dev tap
tls-client
tls-remote Lefinnois_VPN
ca /etc/openvpn/LefinnoisCA-cacert.pem
reneg-sec 21600
comp-lzo
pkcs11-providers /usr/lib/opensc-pkcs11.so
pkcs11-slot-type label
pkcs11-slot "OpenSC Card (denis)"
pkcs11-id-type subject
pkcs11-id "/C=FR/ST=Alsace/L=Colmar/
O=Lefinnois Labs/OU=eToken/CN=Denis Bodor/
emailAddress=lefinnois@lefinnois.net"
```

On retrouve les éléments classiques, ainsi que les directives spécifiques au support PKCS#11 (notez que les trois dernières lignes n'en forment qu'une dans le fichier). OpenVPN a besoin de connaître l'emplacement du token. Nous ne devons pas utiliser une valeur numérique et décidons donc de nous baser sur le label intégrant le nom d'utilisateur. Idem pour l'objet à utiliser. L'ID 45 n'est pas une bonne solution. Mieux vaut utiliser les champs du certificat à notre disposition. Ceci nous permet également de placer plusieurs certificats sur le token et, pourquoi pas, de l'utiliser pour se connecter à plusieurs VPN.

Utilisation

La mise en route est un jeu d'enfant. Nous nous baserons sur les scripts d'init pour les postes sédentaires, ainsi que sur le serveur et utiliserons directement le binaire `openvpn` pour le compte nomade. Ainsi, ceci fera simplement l'affaire :

```
% sudo openvpn --config token.conf
Enter OpenSC Card (denis) token Password:
TUN/TAP device tap0 opened
Initialization Sequence Completed
```

Notez l'utilisation de `sudo` en raison de la mise en œuvre de l'interface TUN/TAP. La fin de session dans le VPN est décidée par l'arrêt d'`openvpn`. Le retrait du token de son emplacement USB peut également provoquer la coupure de connexion tout comme la non-réponse du serveur.

Les options passées par la directive `keepalive` de la configuration du serveur, ici `10 60`, permettent de « *pinguer* » après 10 secondes de non-réception de paquets. En cas de non-réponse, 60 secondes plus tard, le serveur redémarrera (voir SIGUSR1 dans la *manpage*). Mais, l'option la plus intéressante dans ce type de situation est `reneg-sec`, via laquelle on précise une durée en secondes (3600 par défaut). Cette directive permet de spécifier le délai entre deux renégociations de la clef de session. Avec le support SSL/TLS, cette négociation utilise, bien entendu, l'authentification et donc le token. Notez que, dans la plupart des cas, le code PIN est stocké en cache et ne sera pas demandé à nouveau. Mais le plus important : un retrait précipité du token et la connexion est coupée. Celle-ci peut être rattrapée en réintroduisant le token et le code PIN, dans le délai spécifié via `keepalive`. Attention tout de même, la négociation de clef de session utilise du CPU et de la bande passante. On évitera donc ce genre de choses dans des conditions critiques (systèmes embarqués).

Denis Bodor,

db@ed-diamond.com
lefinnois@lefinnois.net