Port Knocking

Master 2 I2L

Année 2009/2010

(unverified ☹)

D'après la version de Jean-Christophe Soulié (2007-2008)

D. Duvivier
LIL – Université du Littoral Côte d'Opale
duvivier@lil.univ-littoral.fr

1 Installation

Sous Debian, il existe un package qui implémente le protocole du Port Knocking. Ce package s'appelle « knockd ». Il suffit de l'installer :

```
# aptitude install knockd
```

Le package « knockd », fournit le serveur : « knockd », et un client : « knock »

2 Présentation

« knockd » est un serveur « port-knock ». Il écoute tout le trafic sur une interface ethernet (ou PPP) et il attend une séquence spéciale de knock sur les ports de la machine.

Le client envoie des séquences de knock en envoyant des paquets TCP (ou UDP) sur les ports du serveur. Quand le serveur détecte une séquence de knock sur ses ports, il exécute une commande définie dans le fichier de configuration. Cela peut être utilisé, par exemple, pour ouvrir momentanément un firewall pour un accès.

3 Options de knockd

- -i, --interface <int> : Spécifie l'interface où les knock vont être écoutés (eth0 par défaut)
- -d, --daemon : Permet à knockd de devenir un démon, pour un fonctionnement « server-like »
- -c, --config <file> : Spécifie un fichier de configuration alternatif. Par défaut, le fichier se trouve dans /etc/knockd.conf
- -D, --debug : Affiche les messages de debug
- -l, --lookup : Utilise la résolution de nom DNS pour les logs
- -v, --verbose : Affiche le mode verbeux de knockd
- -V, --version : Affiche la version
- -h, --help : Aide sur la syntaxe

4 Configuration

knockd lit toute la configuration définie dans un fichier de configuration. Chaque événement knock commence avec un mot clé sous la forme [nom] où nom est le nom qui apparaîtra dans le fichier log. Un mot clé spécial : **[options]** est utilisé pour définir les options globales à la configuration.

5 Configuration : options globales

- **UseSyslog** : logue les actions via syslog(). Celà va insérer une entrée dans le fichier log /var/log/messages
- **LogFile = /path/to/file** : logue les actions directement dans un fichier. La plupart du temps dans : /var/log/knockd.log
- **PidFile = /path/to/file** : PidFile à utiliser lorsque knockd est lancé en mode démon. Par défaut : /var/run/knock.pid
- **Interface = <interface_name>** : l'interface réseau sur laquelle on va écouter les événements knock

6 Configuration : directives événement knock

- `Sequence = <port1>[:<tcp|udp>][,<port2>[:<tcp|udp>] ...]` : spécifie la séquence de `knock` à laquelle le serveur va réagir. Le protocole utilisé peut être `TCP` ou `UDP` (`TCP` par défaut).
- `One_Time_Sequences = /path/to/one_time_sequences_file` : fichier contenant une séquence unique à utiliser. Plutôt que d'utiliser une séquence de `knock` fixée, `knockd` va lire les séquences utilisées dans un fichier spécifié par l'option. A chaque fois qu'une séquence est utilisée, elle est désactivée en rajoutant un `#` en début de ligne contenant la séquence. Chaque ligne contient une séquence exactement comme définit ci-dessus. **Cette option est très pratique si on veut éviter qu'une personne ne découvre une séquence fixée en sniffant les paquets.**
- `Seq_Timeout = <timeout>` : le temps (en seconde) que l'on a pour compléter une séquence. Si le temps est écoulé avant que la séquence soit complète les entrées déjà validées sont annulées.
- `TCPFlags = fin|syn|rst|psh|ack|urg` : les flags `TCP` classiques. Utiliser « `TCPFlags = syn` » est le plus courant. Séparer les flags multiples avec de « , » (par exemple : « `TCPFlags = syn,ack,urg` »). Des flags peuvent être explicitement exclus avec un « ! » (par exemple : « `TCPFlags = syn,!ack` »). Les `TCP Flags` représentent des informations supplémentaires :
  - `URG` : si ce drapeau est à 1 le paquet doit être traité de façon urgente
  - `ACK` : si ce drapeau est à 1 le paquet est un accusé de réception.
  - `PSH (PUSH)` : si ce drapeau est à 1, le paquet fonctionne suivant la méthode `PUSH`
  - `RST` : si ce drapeau est à 1, la connexion est réinitialisée
  - `SYN` : Le `Flag TCP SYN` indique une demande d'établissement de connexion
  - `FIN` : si ce drapeau est à 1 la connexion s'interrompt
- `Start_Command = <command>` : spécifie la commande à exécuter lorsque le client fait une séquence de `knock` correcte. Toutes les instances de `%IP%` sont remplacées par l'adresse `IP` du `knocker`. La directive `Command` est un alias de `Start_Command`.
- `Cmd_Timeout = <timeout>` : temps à attendre entre un `Start_Command` et un `Stop_Command`. Cette directive, optionnelle, est nécessaire seulement si la directive `Stop_Command` est utilisée.
- `Stop_Command = <command>` : Spécifie la commande à exécuter quand `Cmd_Timeout` secondes sont passées depuis que `Start_Command` a été exécutée. Toutes les instances de `%IP%` sont remplacées par l'adresse `IP` du `knocker`. Cette directive est optionnelle.

7 Un premier exemple

Soit le fichier de configuration suivant :

```
[options]
     UseSyslog
[openSSH]
     sequence = 7000,8000,9000
     seq_timeout = 5
     command = /sbin/iptables -A INPUT -s %IP% -p tcp -dport 22 -j ACCEPT
     tcpflags = syn

[closeSSH]
     sequence = 9000,8000,7000
     seq_timeout = 5
     command = /sbin/iptables -D INPUT -s %IP% -p tcp -dport 22 -j ACCEPT
     tcpflags = syn
```

Remarque : « `UseSyslog` » peut être remplacé par « `logfile = /var/log/knockd.log` »

Et nous avons les `iptables` configurées de manière suivante (configuration de base) :

```
# iptables -L -n -v
Chain INPUT (policy ACCEPT 3219 packets, 1106K bytes)
 pkts bytes target prot opt in out source destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target prot opt in out source destination

Chain OUTPUT (policy ACCEPT 2255 packets, 664K bytes)
 pkts bytes target prot opt in out source destination
```

On lance le serveur de la manière suivante :

```
# knockd -i lo -v
listening on lo...
```

Provoquez les `knocks` sur les ports en utilisant dans un premier temps la commande `telnet` :

```
# telnet localhost 7000
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
# sleep 6
# telnet localhost 8000
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
# ... INUTILE DE CONTINUER
```

Attention, vous avez 5 secondes maximum pour réaliser cette séquence ! Comme nous avons temporisé plus de 5 secondes entre les deux premiers appels à telnet, la séquence de knock échoue.
Et au niveau du serveur, on obtient les retours suivants :

```
127.0.0.1: openSSH: Stage 1
127.0.0.1: openSSH: sequence timeout (stage 1)
```

Pour réussir, nous lançons trois commandes telnet en séquence pour y parvenir dans les temps :

```
# telnet localhost 7000 ; telnet localhost 8000 ; telnet localhost 9000
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

Et au niveau du serveur, on obtient les retours suivants :

```
127.0.0.1: openSSH: Stage 1
127.0.0.1: openSSH: Stage 2
127.0.0.1: openSSH: Stage 3
127.0.0.1: openSSH: OPEN SESAME
openSSH: running command: /sbin/iptables -A INPUT -s 127.0.0.1 -p tcp --dport 22 -j ACCEPT
```

Au niveau du `OPEN SESAME`, les `iptables` sont les suivantes :

```
# iptables -L -n -v
Chain INPUT (policy ACCEPT 3206 packets, 1105K bytes)
pkts bytes target     prot opt   in    out   source     destination
0    0     ACCEPT     tcp  --    *     *     127.0.0.1   0.0.0.0/0 tcp dpt:22

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt   in    out   source     destination

Chain OUTPUT (policy ACCEPT 2243 packets, 662K bytes)
pkts bytes target     prot opt   in    out   source     destination
```

La séquence est validée cette fois ! Essayons la seconde séquence :

```
# telnet localhost 9000 ; telnet localhost 8000 ; telnet localhost 7000
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
Trying 127.0.0.1...
telnet: Unable to connect to remote host: Connection refused
```

Et au niveau du serveur, on obtient les retours suivants :

```
127.0.0.1: closeSSH: Stage 1
127.0.0.1: closeSSH: Stage 2
127.0.0.1: closeSSH: Stage 3
127.0.0.1: closeSSH: OPEN SESAME
closeSSH: running command: /sbin/iptables -D INPUT -s 127.0.0.1 -p tcp --dport 22 -j ACCEPT
```

Au niveau du second `OPEN SESAME`, nous revenons à la situation initiale :

```
# iptables -L -n -v
Chain INPUT (policy ACCEPT 3219 packets, 1106K bytes)
pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 2255 packets, 664K bytes)
pkts bytes target     prot opt in     out     source               destination
```

Autre exemple utilisant le `knock` pour accéder au port 22 : Après avoir reçu la bonne séquence de `knocks`, `knockd` exécute `Start_Command` puis `Stop_Command` après avoir attendu `cmd_timeout` :

```
[options]
     UseSyslog

 [openccloseSSH]
     sequence = 2222:udp,3333:tcp,4444:udp
     seq_timeout = 15
     tcpflags = syn,ack
     start_command = /usr/sbin/iptables -A INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
     cmd_timeout = 10
     stop_command  = /usr/sbin/iptables -D INPUT -s %IP% -p tcp --syn --dport 22 -j ACCEPT
```

Le port `ssh` sera fermé automatiquement. Comment faire pour le rendre persistant une fois la connexion établie ?

Un autre exemple avec l'utilisation d'un fichier de séquences :

```
[openHello]
     one_time_sequences = /root/sequences.seq
     seq_timeout = 5
     command = echo "Je suis ouvert"
     tcpflags = syn

[closeHello]
     sequence = 9100,9500,9900
     seq_timeout = 5
     command = echo "Je suis ferme"
     tcpflags = syn
```

Le fichier /root/sequences.seq :

```
 6110,6120,6130
 6140,6150,6160
```

Il vaut mieux laisser un espace en début de ligne, il sera remplacé par # pour inhiber la séquence :

```
# knock localhost 6110 6120 6130
# cat /root/sequences.seq
#6110,6120,6130
 6140,6150,6160
```

La première séquence est maintenant inutilisable car précédée d'un « # ».

A tester, que se passe t'il quand il n'y a plus de séquence ?

Encore un exemple :

```
[openFTP]
     sequence = 23077:tcp,12044:udp,13001:tcp,42967:udp
     seq_timeout = 5
     command = /sbin/iptables -I INPUT -d xxx.xxx.xxx.xxx -s %IP% -p tcp
--dport 21 -j ACCEPT
     tcpflags = syn
[closeFTP]
     sequence = 23078:tcp,12045:udp,13002:tcp,42968:udp
     seq_timeout = 5
     command = /sbin/iptables -D INPUT -d xxx.xxx.xxx.xxx -s %IP% -p tcp
--dport 21 -j ACCEPT
     tcpflags = syn
[openWEBMIN]
     sequence = 44295:tcp,50155:udp,11932:tcp,24085:udp
     seq_timeout = 5
     command = /sbin/iptables -I INPUT -d xxx.xxx.xxx.xxx -s %IP% -p tcp
--dport 10000 -j ACCEPT
     tcpflags = syn
[closeWEBMIN]
     sequence = 44296:tcp,50156:udp,11933:tcp,24086:udp
     seq_timeout = 5
     command = /sbin/iptables -D INPUT -d xxx.xxx.xxx.xxx -s %IP% -p tcp
--dport 10000 -j ACCEPT
     tcpflags = syn
[REBOOT]
     sequence = 23654:tcp,50443:udp,20978:tcp,31855:udp
     seq_timeout = 5
     command = /sbin/shutdown -r +1
     tcpflags = syn
```

Dans cet exemple, nous allons à chaque fois frapper à 4 ports.

Pour ouvrir le port 21 nous frapperons, dans l'ordre, aux ports 23077, 12044, 13001 et 42967. Pour ouvrir le port 10000, nous frapperons aux ports 44295, 50155, 11932 et 24085 et pour rebooter nous utiliserons les ports 23654, 50443, 20978 et 31855. Pour compliquer les choses, nous utiliserons des séquences `TCP` (valeur par défaut) pour chaque 1er et 3ème ports et `UDP` pour les autres.

Pour fermer les deux ports, nous incrémenterons d'une unité ces valeurs (23077 => 23078, 12044 => 12045 …) et la totalité de la commande devra être exécutée en moins de 5 secondes (`seq_timeout`) ce qui est bien plus qu'il n'en faut en réalité.

On doit, bien entendu, remplacer dans l'exemple ci-dessus les `xxx.xxx.xxx` par l'`IP` de votre serveur. On peut cependant supprimer l'option de destination « `-d xxx.xxx.xxx.xxx` » mais aussi utiliser le paramètre `-A` (ajouter) au lieu de `-I` (insérer) suivant la configuration du serveur et du firewall. Quant à la variable `%IP%`, `knockd` la remplacera lui-même par l'adresse `IP` du client de connexion ce qui permettra de n'ouvrir le port que pour celui-ci.

Ensuite, on bloque complètement l'accès aux ports 21 et 10000 avec `iptables`, puis on lance knockd (en `lo`, c'est un peu simpliste, vous pouvez essayer sur `eth0` avec la machine du voisin)

```
# knockd -i lo -v
```

Ensuite, il n'y a plus qu'à lancer `knock` et vérifier à chaque fois ce que ça donne au niveau d'`iptables`