Détection de tunnels aux limites du périmètre

Guillaume Lehembre[1] and Alain Thivillon[1]

HSC, 4 Bis Rue de la Gare
92300 Levallois-Perret
http://www.hsc.fr/
{Alain.Thivillon, Guillaume.Lehembre}@hsc.fr

1 Introduction

Au fil des années, les entreprises ont cherché à cloisonner et filtrer leur système d'information afin d'en maîtriser au maximum les flux y circulant. Ces restrictions ont poussé un certain nombre de logiciels et d'individus à utiliser des méthodes de contournement pour s'affranchir des limites de la sécurité leur étant imposées. L'utilisation détournée de protocoles communément autorisés – en direct ou via des mécanismes de relayage – permet alors de véhiculer des flux non autorisés par le biais de mécanismes appelés tunnels. Ces tunnels, volontaires ou non, peuvent présenter de graves risques de fuite d'informations, de congestion réseau, etc. L'utilisation de tunnels au sein des entreprises est dorénavant devenu monnaie courante ce qui a eu pour effet de rendre les défenses périmétriques de plus en plus poreuses.

Cet article ne s'attachera pas à trouver des canaux cachés bas niveau mais s'intéressera à l'usage pratique des techniques non intrusives qu'un usager peut utiliser pour contourner la politique de sécurité de l'entreprise. Ces techniques s'appuient sur des protocoles standards tels HTTP, HTTPS, DNS, ICMP, etc. dans un réseau qu'on suppose filtré et bénéficiant d'une sécurité raisonnable (mise en place de relais applicatifs, etc.). L'article présentera par la suite comment détecter ces outils et plus généralement ces techniques pour finir sur la présentation d'un ensemble d'outils de détection de tunnels.

2 Types de tunnels et outils

2.1 Tunnels HTTP

Les tunnels HTTP représentent très certainement la forme de tunnels la plus simple et la plus rencontrée car ce protocole est fréquemment autorisé dans les entreprises par le biais de relais applicatifs.

Un utilisateur malveillant est en mesure de contourner les protections mises en place en utilisant le relais HTTP avec :

- la méthode CONNECT (nécessaire au fonctionnement des navigateurs en HTTPS au travers d'un relais applicatif),
- des requêtes HTTP classiques de type GET et/ou POST sur des scripts CGI ou en passant directement des informations dans les URL.

Relayage arbitraire TCP dans CONNECT Le cas le plus simple de tunnels utilisant la méthode CONNECT est l'usage d'un serveur SSH écoutant sur un port non standard (443 par exemple) autorisé à être accédé par le biais d'un relais applicatif. Un utilisateur malicieux pourra combiner l'utilisation de SSH avec un programme tel *Socat* [1] pour relayer sa connexion SSH de manière transparente au niveau du relais TCP :

```
$ socat tcp4-listen:2222 proxy:carbone.hsc.fr:unserveurssh.com:22
proxyport=8080

$ telnet carbone.hsc.fr 8080

Trying 192.70.106.49...
Connected to carbone.hsc.fr.
Escape character is '^]'.

CONNECT unserveurssh.com:443 HTTP/1.0
HTTP/1.0 200 Connection established

SSH-2.0-OpenSSH_4.2

$ ssh -p 2222 user@localhost

debug1: Connecting to localhost [127.0.0.1] port 2222.
debug1: Connection established.

[...]

debug1: Remote protocol version 2.0, remote software version
OpenSSH_4.2

[...]
```

Cette méthode est aussi utilisée par de nombreux logiciels tels Skype pour établir un canal de données avec plusieurs super-nodes (si des connexions directes n'aboutissent pas). Certaines connexions initiées à destination du port 443 ne sont pas du SSL/TLS standard (il n'y a pas d'échanges *Client Hello*, *Server Hello*, etc.) :

```
CONNECT 195.215.8.142:443 HTTP/1.0
HTTP/1.0 200 Connection established

..............A..../...A..-.......romiglups......A....(...A....
```

Utilisation de SSL Comme ci-dessus, les tunnels SSL utilisent pour passer les relais HTTP la méthode CONNECT, mais vont créer une vraie session SSL

(comme HTTPS) pour transporter des données arbitraires et chiffrées. Il devient alors extrêmement difficile de les détecter par analyse de protocole, puisque leur empreinte est similaire à celle laissée par un navigateur utilisant un site sécurisé.

On peut citer parmi ces tunnels SSL tous les VPN SSL commerciaux (dont la plupart utilisent une encapsulation de Socks dans SSL afin d'offrir un multiplexage) (Aventail, F5, Cisco, ...) et des solutions OpenSource telle que *SSL-Tunnel* [2] ou *Stunnel* [3].

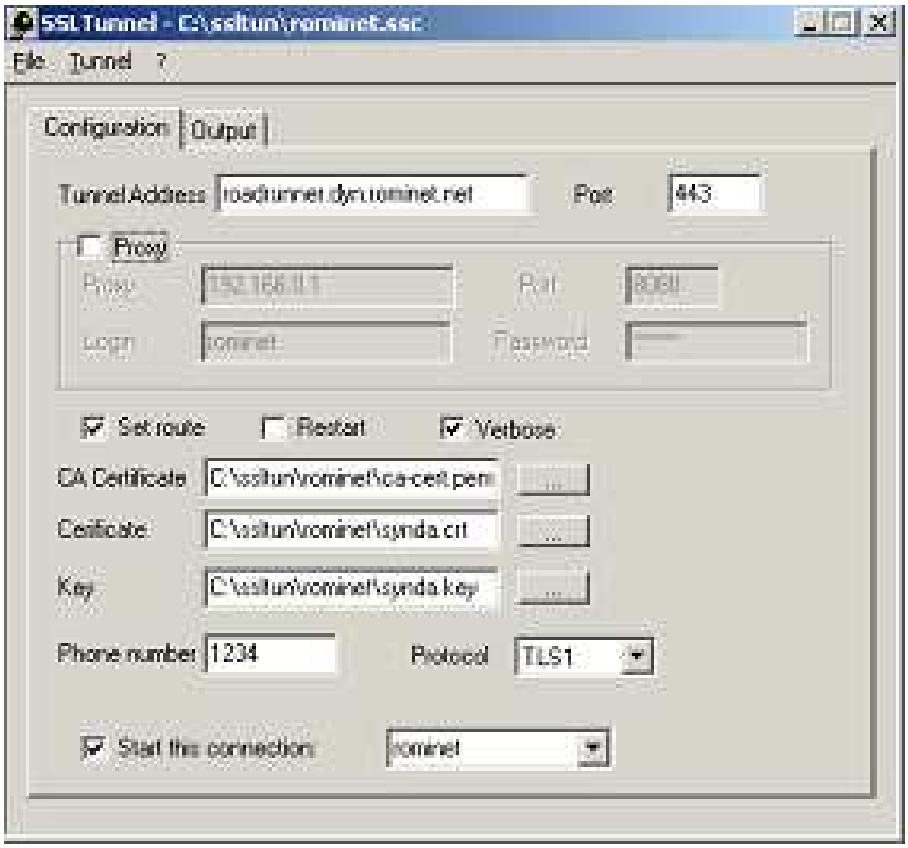


Fig. 1. Interface de configuration de SSLTunnel

Relayage en utilisant GET et POST A la différence des méthodes précédentes, dans lesquelles le relayage TCP était effectué de bout en bout, d'autres types de tunnels vont encapsuler leurs données dans des échanges HTTP valides, en utilisant les méthodes classiques POST et GET.

Plusieurs outils peuvent être cités, utilisant des techniques parfois différentes :

– *Firepass* [4] est un outil de tunnel HTTP utilisant des requêtes POST sur un script CGI d'un serveur distant. Le script CGI appelé est systématiquement sous la forme /cgi-bin/fpserver.cgi et il est accédé avec des en-têtes de requête HTTP correctement formés (champs Host, Content Type, User-Agent, etc.). Les connexions effectuées ne sont pas permanentes, un mécanisme de polling régulier est mis en place pour permettre au serveur d'émettre ces données en réponse à des requêtes POST lorsqu'une session active est établie :

```
Hypertext Transfer Protocol
POST http://firepass.hsc.fr:80/cgi-bin/fpserver.cgi HTTP/1.1\r\n

Content-Type: application/octet-stream\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)\r\n
Host: firepass.hsc.fr\r\n

Content-Length: 0\r\n
X-Session: 1\r\n
X-Counter: 198\r\n
X-Connection: alive\r\n

Proxy-Connection: Keep-alive\r\n
Pragma: no-cache\r\n
```

– *GNU HTTPTunnel* [5] utilise directement les URL pour transmettre les données à l'aide de requêtes POST et GET. Un canal de communication est ouvert dans chaque sens : POST pour envoyer et GET pour recevoir. Ces canaux sont réinitialisés au bout d'une certaine quantité de données échangées ($\sim$ 65ko) ce qui permet de ne pas avoir à utiliser de mécanisme de polling. L'URL accédée se présente toujours sous la même forme /index.html ?crap=XYZ et aucun chiffrement natif n'est mis en place ce qui permet de voir transiter en clair certaines bannières caractéristiques :

```
Client -> Proxy
Hypertext Transfer Protocol

POST http://tun.hsc.fr:80/index.html?crap=1143719371
  HTTP/1.1\r\n

[...]

Client -> Proxy
Hypertext Transfer Protocol

GET http://tun.hsc.fr:80/index.html?crap=1143719371
  HTTP/1.1\r\n
```

```
[...]

Client -> Proxy (banniere SSH du client)
Hypertext Transfer Protocol
Data (519 bytes)

0000 53 53 48 2d 32 2e 30 2d 4f 70 65 6e 53 53 48 5f
SSH-2.0-OpenSSH_

0010 34 2e 32 70 31 20 44 65 62 69 61 6e 2d 35 0a 00
4.2p1 Debian-5..

[...]

Proxy -> Client (banniere SSH du serveur distant)
Hypertext Transfer Protocol

Data (22 bytes)

0000 00 14 53 53 48 2d 32 2e 30 2d 4f 70 65 6e 53 53
..SSH-2.0-OpenSS

0010 48 5f 34 2e 32 0a H_4.2.

[...]
```

Des sociétés à vocation commerciale ont aussi investi le créneau des tunnels HTTP pour faire profiter à leurs clients, moyennant finance, des divers logiciels de messagerie instantanée et autre Peer to Peer en toute impunité. Ces logiciels proposent dans la plupart des cas des clients graphiques généralement sous Windows.

- *Loophole* [6] est l'exemple même d'un tunnel HTTP avancé implémentant du chiffrement Blowfish nativement et utilisant des POST sur des URL partiellement aléatoires afin de ne pas éveiller de soupçons – de prime abord – sur la nature des flux échangés. De la même façon que pour *Firepass*, un mécanisme de polling est nécessaire à son bon fonctionnement. C'est un logiciel commercial utilisant une interface graphique en Java et pouvant fonctionner en relais SOCKS pour l'encapsulation de flux autres que TCP.

```
Hypertext Transfer Protocol

POST http://tun.hsc.fr:80/display/mode/forum/minimize.asp
HTTP/1.0\r\n

Request Method: POST
```

```
Request URI: http://tun.hsc.fr:80/display/mode/forum/minimize.asp
Request Version: HTTP/1.0

Content-Type: application/octet-stream\r\n
Content-Length: 480\r\n

Hypertext Transfer Protocol
POST http://tun.hsc.fr:80/pack/imply.asp
 HTTP/1.0\r\n

Request Method: POST
Request URI: http://tun.hsc.fr:80/pack/imply.asp
Request Version: HTTP/1.0
Content-Type: application/octet-stream\r\n
Content-Length: 312\r\n
```

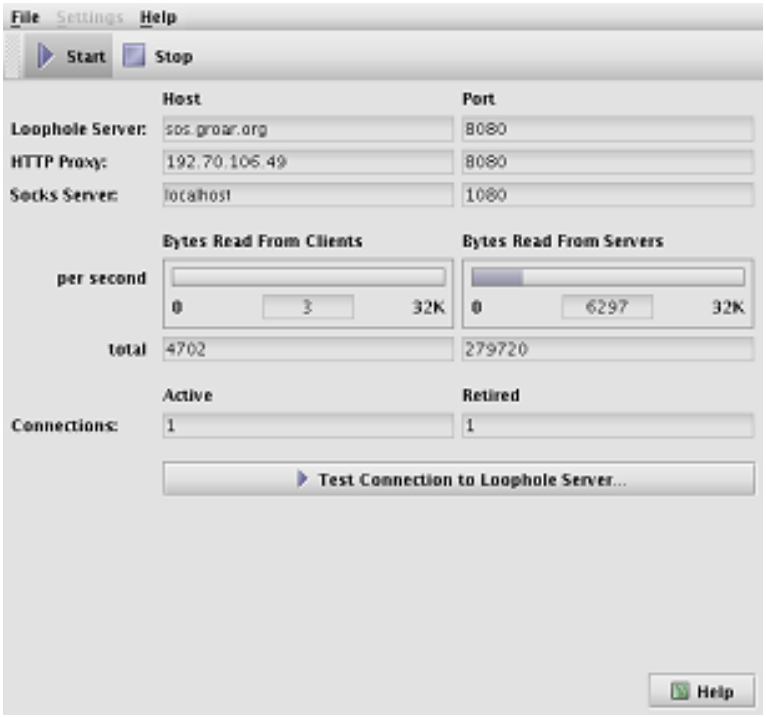


Fig. 2. Interface Java de LoopHole

– D'autres sociétés proposent les mêmes types de services commerciaux : *Socks2HTTP* [7], *Hopster* [8], *HTTP Tunnel* [9], etc avec des techniques de dissimulation plus ou moins évoluées.

2.2 Tunnels ICMP

Le protocole ICMP peut lui aussi servir à monter des tunnels par le biais de requêtes Echo Request (type 8) et Echo Reply (Type 0). *PingTunnel* [10] est un exemple d'outil utilisant ces requêtes contenant des messages d'états, des numéros de séquence à usage interne et un champ spécial permettant de différencier les requêtes ICMP du tunnel de celles utilisées par ping. A ces caractéristiques de paquets est associé un protocole de communication et de retransmission en cas de perte. Il est important de noter que la tête de tunnel ICMP doit être configurée pour empêcher sa propre pile IP de répondre aux messages ICMP. En effet, un firewall protégeant un client n'acceptera qu'un seul paquet Echo Reply par Echo Request émis (avec un même numéro de séquence), bloquant ainsi l'établissement du tunnel.

```
Du client vers la tête de tunnel ICMP :
 Internet Control Message Protocol

Type: 8 (Echo (ping) request)
Code: 0

Checksum: 0x68b5 [correct]
Identifier: 0xac12
Sequence number: 0x0000
Data (28 bytes)

0000 d5 20 08 80 52 e8 c6 85 00 00 00 16 40 00 00 00
    . ..R.......@...
0010 00 00 ff ff 00 00 00 00 00 00
    ac 12 ..........

La réponse de la tête de tunnel ICMP :
 Internet Control Message Protocol

Type: 0 (Echo (ping) reply)
Code: 0
Checksum: 0x957c [correct]

Identifier: 0xac12
Sequence number: 0x0000
Data (48 bytes)

0000 d5 20 08 80 00 00 00 00 00 00 00 00 80 00 00 02
```

```
    . ..............
0010 00 00 00 01 00 00 00 14 00 00 ac 12 53 53 48 2d
    ............SSH-

0020 32 2e 30 2d 4f 70 65 6e 53 53 48 5f 34 2e 32 0a
2.0-OpenSSH_4.2.
```

2.3 Tunnels DNS

Le protocole DNS est – à tort – considéré comme un protocole « inoffensif ». Il occupe une place très importante dans le bon fonctionnement des systèmes d'informations et autorise dans la plupart des cas l'interrogation de serveurs de noms externes. De ce fait, il peut représenter une possibilité de fuite d'informations vers un serveur contrôlé par une personne malveillante. *Nstx* [11] et *Dns2tcp* [12] sont deux logiciels exploitant ces propriétés pour encapsuler des données dans des requêtes DNS (encapsulation d'IP dans DNS pour *Nstx* et TCP sur DNS pour *Dns2tcp*). Certains types de requêtes et de réponses du protocole DNS sont particulièrement adaptés au transport de données arbitraires : c'est le cas des enregistrements TXT et KEY par exemple qui sont transmis de bout en bout dans la chaîne de serveurs DNS. Des données encodées en base64 peuvent alors y être insérées :

```
Domain Name System (query)
Transaction ID: 0xe5c4

Flags: 0x0100 (Standard query)
0... .... .... .... = Response: Message is a query
.000 0... .... .... = Opcode: Standard query (0)
.... ..0. .... .... = Truncated: Message is not truncated
.... ...1 .... .... = Recursion desired: Do query recursively
.... .... .0.. .... = Z: reserved (0)
.... .... ...0 .... = Non-authenticated data OK: Non-authenticated
  data is unacceptable

Questions: 1
Answer RRs: 0
Authority RRs: 0
Additional RRs: 0

Queries
bTgc1fuuaacICreaaqazAU7bgAHXs5mAf34WafIANOx7vOCKXubadGn2xaabH.\
   nstxd.hsc.fr:
type TXT, class IN

Name:
bTgc1fuuaacICreaaqazAU7bgAHXs5mAf34WafIANOx7vOCKXubadGn2xaabH.\
```

```
   nstxd.hsc.fr

Type: TXT (Text strings)
Class: IN (0x0001)
```

Un mécanisme de polling est mis en place par le client pour envoyer à intervalles de temps régulier des interrogations DNS, ces requêtes étant ensuite bufferisées pendant un certain temps côté serveur pour un éventuel envoi de données « Serveur vers Client ».

Au sein d'une entreprise, seuls les relais applicatifs devraient pouvoir effectuer des requêtes DNS vers l'extérieur, les machines internes devant se limiter à l'interrogation d'un DNS privé.

Ce type de tunnels est plus préoccupant pour les fournisseurs de HotSpot WiFi car ils ne peuvent pas facilement empêcher les résolutions DNS avant l'authentification de l'utilisateur sur le portail captif Web. Un utilisateur astucieux est donc en mesure d'utiliser le service sans payer. Une solution simpliste serait d'utiliser un DNS temporaire (suite à une redirection au niveau des règles de filtrage) avant l'authentification de l'utilisateur renvoyant systématiquement une même adresse ... mais cela pose un problème au niveau du cache DNS de la machine cliente (sous Windows en natif avec un navigateur comme Internet Explorer par exemple) car celle-ci ne pourra pas résoudre correctement l'adresse qu'elle aura saisie avant d'être redirigée sur le portail captif. Cette solution étant inacceptable pour des utilisateurs légitimes du service, on retrouve systématiquement les résolutions DNS externes autorisées avant l'authentification et donc la possibilité de monter ce type de tunnels. A noter également que certains fournisseurs d'accès sans fil ne prennent même pas la peine de vérifier que le protocole DNS est utilisé sur le port 53/udp, laissant ainsi la possibilité de monter un tunnel PPP sur UDP par exemple.

2.4 Détection en temps réel

2.5 Architectures et Contraintes

La détection de tunnels en temps réel permet de réagir immédiatement, et d'observer « in situ » le trafic généré. Elle permet également de détecter plus de types de tunnels, et de constituer au fil de l'eau une liste de machines ou d'utilisateurs au comportement suspect dont il sera possible d'auditer le trafic par des moyens plus traditionnels.

Cette détection peut s'effectuer selon différents procédés :

- par écoute passive du trafic, en différents points du réseau :
  - firewalls,
  - routeurs,
  - serveurs applicatifs relayant le trafic des utilisateurs (relais HTTP, relais génériques, ...).

Cette écoute doit être suivie d'une analyse du trafic, et sa confrontation par rapport à une liste de critères, parfois dynamiques, à déterminer.

- par mise en place de contrôles dans les relais eux-mêmes : la détection peut alors s'effectuer à un niveau plus élevé, en analysant de bout en bout le trafic émis et en vérifiant sa cohérence avec le protocole demandé.
- par mise en place de moyens de détection dans chaque poste client, par exemple par interception des communications TCP/IP comme le font les anti-virus et les HIDS.

D'une manière pratique, seule la première solution (écoute passive) est suffisamment facile à déployer et générique pour être utilisable dans un grand réseau basé sur des relais HTTP fermés ou difficiles à étendre (sources indisponibles).

Malheureusement, cette solution a également des inconvénients difficiles à maîtriser et à borner :

- nécessité de pouvoir écouter tout le trafic échangé entre les postes clients et le monde extérieur : un réseau de grande taille peut disposer de plusieurs points d'accès à Internet, certains étant implantés dans des lieux différents (filiales, ...).
- nécessité de pouvoir écouter un volume de trafic IP parfois conséquent, ce qui suppose de disposer d'un moyen d'écoute et d'un système d'exploitation assez rapides pour ne pas perdre de paquets.

Il est en effet fondamental de ne manquer aucun paquet réseau, en particulier en ce qui concerne le trafic TCP qu'il va être nécessaire de ré-assembler, afin de pouvoir :

- Analyser le début des connexions, en particulier l'établissement de la session applicative entre le client sur le réseau local et le serveur situé sur le réseau externe.
- Être prévenu de la fin de la connexion TCP, afin de pouvoir journaliser ses caractéristiques pour les analyses statistiques.

Ce ré-assemblage doit évidemment tenir compte :

- des fragments IP
- des paquets TCP réémis, à l'identique ou par concaténation de paquets non acquittés,
- de la fenêtre TCP actuelle.

Il est bien sûr important que l'écoute puisse résister à des contournements triviaux, comme l'utilisation de logiciels comme *fragrouter* [13] effectuant une fragmentation au niveau IP, ou de relais TCP permettant de scinder les paquets TCP comme *socat*. En cela, les précautions à prendre et la qualité d'un détecteur sont très proches de celles requises pour un système de détection d'intrusion réseau, et les vulnérabilités et limites d'un tel système doivent être bien comprises.

2.6 Analyse du protocole

Cette analyse est la plus « simple » : elle vise à s'assurer que les ports assignés à un protocole par l'IANA sont utilisés par les protocoles prévus.

Cette analyse doit permettre de détecter par exemple les connexions SSH sur un port non standard, l'encapsulation sur UDP/53 (DNS) d'un protocole UDP autre.

Dans le cadre de la détection de tunnels dans un réseau déjà filtré et n'utilisant que des relais applicatifs, l'analyse à ce niveau doit conduire par exemple à :

- vérifier que le trafic sur le port 80 utilise convenablement le protocole HTTP, que les requêtes et réponses sont convenablement formatées,
- vérifier que le protocole utilisé dans une requête de type CONNECT vers un relais HTTP est bien SSL.

Il est bien évident que le protocole ne peut pas être suivi de bout en bout sans sacrifier les performances, et qu'un outil réaliste ne s'appuiera que sur une détection du début de la connexion.

HTTP En HTTP, on s'assurera notamment :

- Que les requêtes adressées au relais HTTP (ou en direct) utilisent une méthode HTTP standard (GET, POST, HEAD) ou à la limite Webdav (bien que l'utilisation de ces méthodes soit déjà très peu courante sur Internet).
- Que les requêtes POST incluent un mode de transfert valide (*multipart-form/data* ou *application/x-www-form-urlencoded*) et un en-tête *Content-Length* conforme ; ou bien utilisent le mode HTTP/1.1 *Chunked*.
- Que les requêtes incluent un en-tête *User-Agent* et un en-tête *Host* (optionnel en HTTP/1.0 mais utilisé par tous les navigateurs depuis 1996).
- Que les réponses du serveur incluent un en-tête *Content-Type* ou *Transfer-Encoding* contenant la valeur *Chunked*.

Proxy HTTPS Dans le cas d'une utilisation de relais HTTPS (méthode CONNECT), qui est on l'a vu la méthode la plus utilisée afin de contourner la politique de sécurité, on devra vérifier :

- que l'adresse demandée est de type DNS FQDN (un certificat SSL public et signé par une autorité valide ne devant jamais contenir une adresse IP, il est exclu qu'un usage normal de SSL fasse appel à une adresse IP numérique). Si des exceptions sont nécessaires, elles doivent être gérées au coup par coup. Cette caractéristique permet de détecter de nombreux logiciels utilisant CONNECT pour établir des connexions arbitraires, notamment Skype, de nombreux logiciels de VPN SSL, ...
- que le port demandé au relais est bien 443, ou tout autre port explicitement prévu par l'administrateur.
- que les en-têtes *Host* et *User-Agent* sont présents et valides dans la requête CONNECT (cette disposition n'est pas une obligation du protocole, mais les navigateurs la suivent).

Une requête CONNECT bien formée par un navigateur moderne se présente ainsi sous la forme :

```
CONNECT www.verisign.com:443 HTTP/1.1

User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.6)
```

```
  Firefox/1.0.1
Proxy-Connection: keep-alive

Host: www.verisign.com
HTTP/1.0 200 Connection established
```

SSL En ce qui concerne le suivi de SSL (que ce soit en direct ou au travers de la méthode CONNECT), on pourra vérifier assez simplement :

- que le premier paquet de données est émis par le client (Client Hello) : cette caractéristique permet par exemple de détecter les connexions SSH émises sur le port 443, ou tout autre protocole dans lequel le serveur envoie sa bannière avant que le client n'émette (et c'est une majorité).
- que les données échangées dans les deux premiers paquets sont conformes à SSLv2 ou SSLv3, et en particulier qu'ils sont de type « Client Hello » ou « Client Handshake », puis « Server Hello » ou « Server Handshake ».

La mise en place du chiffrement immédiatement après cet échange (cas d'utilisation de Diffie Hellman, réutilisation d'une clé de session existante) ne permet pas d'aller plus loin, on peut toutefois vérifier que les paquets suivants ont bien un type SSL connu (Application Data, Cipher Spec, ...). Etendre les contrôles (vérification notamment des longueurs d'enregistrement SSL) est sans doute difficile, superflu et coûteux en CPU, en particulier à cause du réassemblage nécessaire des enregistrements SSL.

Une détection intelligente (mais très coûteuse et difficile à généraliser sur un grand réseau) est de calculer l'entropie du flux de données : un vrai flux chiffré générera une entropie maximale, une encapsulation de protocole non chiffré sera moins aléatoire. Cette idée à donné naissance au logiciel *Net-Entropy* [14] développé par Julien Olivain du laboratoire LSV de l'ENS Cachan. Cette méthode ne permet pas en revanche de détecter les encapsulations chiffrées (par exemple l'utilisation de clients VPN comme CheckPoint SecuRemote).

Analyse des en-têtes Le comportement des navigateurs utilisés dans le réseau de l'entreprise est en général relativement facile à déterminer :

- Utilisation de logiciels plus ou moins standardisés, installés dans un « master ».
- Volume important de requêtes et usage régulier des logiciels, ce qui rend possible une analyse de l'existant et une détection assez rapide des exceptions.

Il est donc possible d'examiner le trafic entre le réseau de l'entreprise et le proxy, afin de lister notamment :

- les en-têtes « User-Agent » générés par les logiciels agrées,
- les éventuels logiciels non standards utilisés et leur comportement.

Un logiciel de détection peut donc s'appuyer sur cette connaissance pour signaler tout autre logiciel non prévu, ce qui permettra ensuite à l'administrateur réseau d'enquêter. De même, l'absence d'en-tête User-Agent est fortement suspecte. Ces simples tests permettent de récupérer un nombre importants de logiciels non

voulus, que ce soit de simples spywares ou de tunnels plus évolués. Évidemment, la mise en conformité de ces logiciels est possible relativement simplement, mais les utilisateurs les moins avertis ne seront pas capables de le changer, et surtout la configuration par défaut, utilisée lors de leurs premiers tests, est donc repérable.

La liste de User-Agent ci-dessous a été découverte en quelques heures de trafic sur un réseau de quelques centaines de PC :

```
442 Acrobat Messages Updater
  7 Adobe Online Manager
  5 Avant Browser (http://www.avantbrowser.com)
 13 BW-C-2.0
  4 CryptRetrieveObjectByUrl::InetSchemeProvider
  9 Google Talk
  7 LAN-Console [workstation] 2004 Server
  6 McAfee AutoUpdate
  3 Microsoft BITS/6.2
  2 Microsoft Office Protocol Discovery
  2 Microsoft(r) Windows(tm) FTP Folder
  1 Microsoft-CryptoAPI/5.131.2600.2180
  6 Microsoft-CryptoAPI/5.131.3790.0
  3 Microsoft-WebDAV-MiniRedir/5.1.2600
 35 Mozilla/3.0 (compatible; Acrobat SOAP 1.0)
  3 Mozilla/4.0 (Compatible; MyWay)
  3 Mozilla/4.0 (Compatible; MyWaySearchAssistant)
  2 Mozilla/4.0 (compatible; GoogleToolbar 3.0.131.0-big;
     Windows 2000 5.0)
 44 Mozilla/4.0 (compatible; GoogleToolbar 3.0.131.0-big;
     Windows XP 5.1)
 79 Mozilla/4.0 (compatible; Lotus-Notes/6.0; Windows-NT)
111 Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1;
     .NET CLR
    1.1.4322; MSN Messenger 7.0.0777)}
  4 NSISDL/1.2 (Mozilla)
  2 NSPlayer/4.1.0.3928
  4 NetClient/5.09.6
  1 QuickTime (qtver=6.4;os=Windows NT 5.0Service Pack 3)
  6 RMA/1.0 (compatible; RealMedia)
  2 Toolbar
  1 Windows-Media-Player/9.00.00.3344
  5 pmx-ppm/4.7.1.128075 (S662C1589D; linux) libwww-perl/5.69
  1 session name
```

On note en particulier que MSN Messenger, quand il est encapsulé dans HTTP, utilise un *User-Agent* assez reconnaissable. De même Google Talk est facilement repérable dans ce trafic.

Il est à noter également que le téléchargement des CRLs par Windows utilise un *User-Agent* spécifique (*Microsoft-CryptoAPI*).

La mise en place d'une liste de clients connus permet donc de repérer rapidement tout nouveau logiciel.

3 Détections statistiques

3.1 Principes

La détection statistique va s'attacher à déterminer par une étude a-posteriori des connexions sortantes quelles sont celles qui sont susceptibles d'avoir été des tunnels.

Cette détection va s'appuyer :
- sur le relevé des caractéristiques effectué par écoute du trafic,
- sur les journaux des relais applicatifs,
- sur quelques propriétés intrinsèques de chaque protocole.

On s'intéressera principalement dans la suite à HTTP et HTTPS qui sont les protocoles généralement autorisés à sortir du réseau de l'entreprise, et qui sont les moyens les plus génériques et « tout-terrain » afin de passer des informations.

L'étude statistique doit s'intéresser :
- aux connexions une par une,
- aux requêtes HTTP dans leur ensemble d'un client vers un serveur.

Pour que les statistiques aient un sens, il est nécessaire qu'elles incluent l'adresse IP réelle du client, et que la capture du trafic (ou les journaux) soit effectuée entre le poste client et le relais applicatif. Si HTTP 1.1 est utilisé, la connexion TCP doit être scindée en autant de requêtes HTTP que nécessaire pour l'analyse des requêtes unitaires.

3.2 Métriques

Afin de pouvoir exploiter au mieux les informations, il est nécessaire de connaître pour chaque requête HTTP adressée au relais :
- Les adresses IP source, destination, les ports source et destination.
- L'URL accédée (ou le serveur et le port dans le cas de HTTPS).
- Le nombre de paquets de données échangés dans chaque sens (ce qui est différent évidemment du nombre de paquets TCP : on ne s'intéresse ici qu'aux paquets portant une charge, en ignorant les acquits vides).
- Le volume de données échangées dans chaque sens.
- La durée de la connexion depuis son établissement jusqu'à sa fin TCP.
- La durée de la connexion dans chaque sens (période utile pendant lesquelles des données ont transité).
- L'intervalle moyen entre deux paquets de données dans chaque sens.
- L'écart type de ces intervalles.

On constate rapidement que les journaux applicatifs ne suffisent pas : la notion du nombre paquets de données leur sera en général étrangère. De même, un firewall s'intéressera et fournira des informations sur le nombre de paquets TCP dans la connexion, mais ne distinguera pas les paquets utiles des simples ACK TCP sans charge.

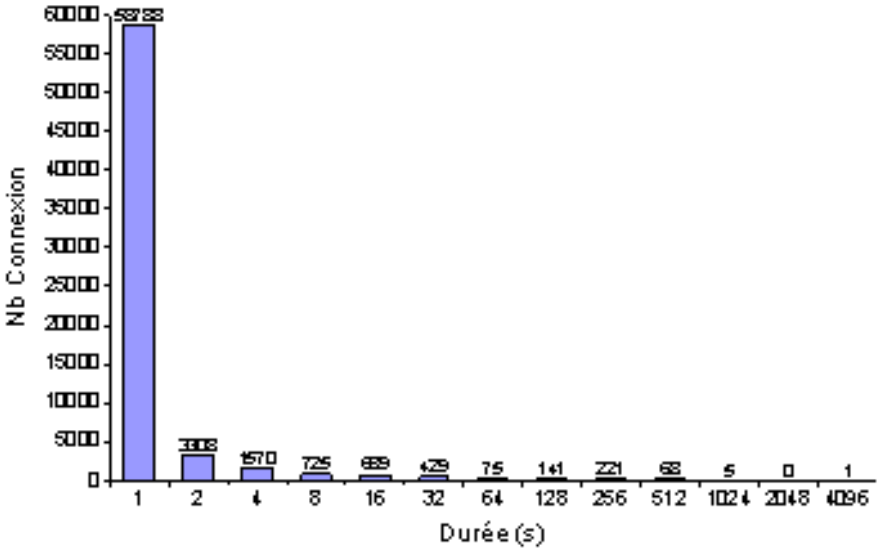
A partir de ces données, il est possible de déterminer :
- la taille moyenne des paquets échangés dans chaque sens,
- la bande passante utilisée,
- le ratio upload/download.

Si l'on s'intéresse maintenant à un dialogue sur le moyen terme entre un client un serveur, il est intéressant de connaître :
- le nombre total de requêtes adressées par un client à un serveur en particulier (figure 3.3),
- la somme des volumes échangés dans chaque sens avec ce serveur,
- l'intervalle moyen entre deux requêtes,
- l'écart-type de cet intervalle.

Le problème lors de ce calcul étant de déterminer quelles sont les requêtes appartenant à une même « série » afin de distinguer deux sessions de tunnels. D'une manière arbitraire, on peut estimer qu'un tunnel de données n'ayant rien échangé pendant plusieurs minutes est relancé (ce qui bien évidemment exclut de l'étude les tunnels « lents » qui pourraient permettre de faire sortir des données sur le long terme).

3.3 Durée de la connexion

Le graphe ci-dessous montre la distribution de la durée des connexions HTTP ou HTTPS a l'entrée d'un relais Squid. Il montre de manière claire que les connexions de plus de 1000 secondes sont très rares, même s'il faudrait corréler cette information avec le volume de données échangées (gros téléchargements notamment).

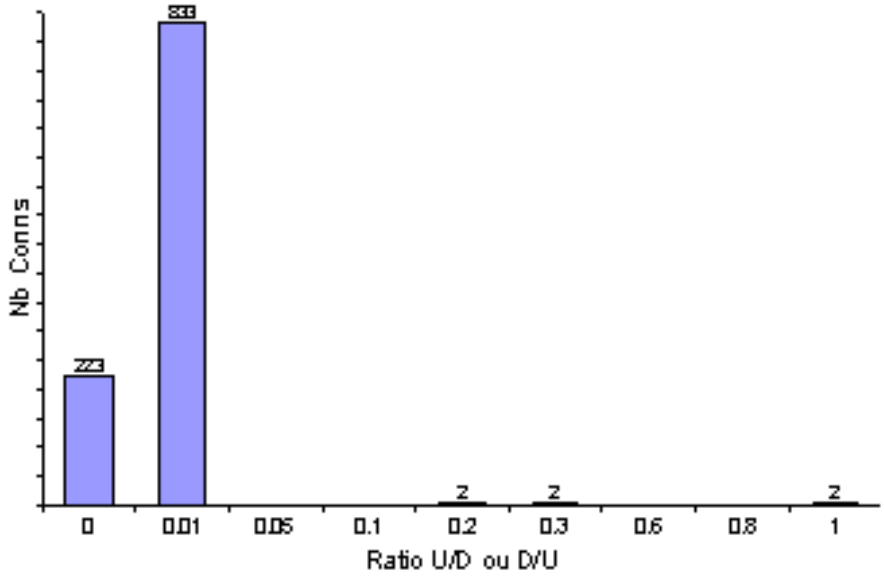


3.4 Ratio Upload/Download

Une caractéristique très intéressante de HTTP est qu'il est très rare que les échanges y soient symétriques : la plupart des échanges sont orientés vers le navigateur qui reçoit plus de données qu'il n'en émet. Il est possible également d'envoyer des données en grand nombre (upload de fichiers), mais la plupart du temps la réponse du serveur sera alors plus courte que la requête.

Cette règle est évidemment à préciser : de nombreuses réponses à des requêtes HTTP ou HTTPS sont très courtes (petites images, pages d'erreur, Javascript, ...). Il est donc nécessaire de s'intéresser au nombre de paquets de données échangées (somme des deux sens) et d'appliquer alors une heuristique.

Le graphe ci-dessous montre le ratio upload/download ou download/upload (on choisit le plus faible) pour 1062 connexions HTTP ou HTTPS émises à travers un relais Squid. Ces connexions contenaient chacune plus de 150 paquets au total, afin que le ratio calculé soit significatif : On constate de manière très



claire que seules quelques connexions ont un ratio supérieur à 0.5, et que l'immense majorité a un ratio inférieur à 1 sur 100. Ce critère permet notamment de détecter :

- les VPN SSL comme SSLTunnel ou Aventail [15] utilisés pour accéder à des ressources interactives comme une émulation de terminal, Windows Terminal Services ou Citrix.
- les connexions aux services de chat en utilisant la méthode CONNECT (Google Talk notamment),

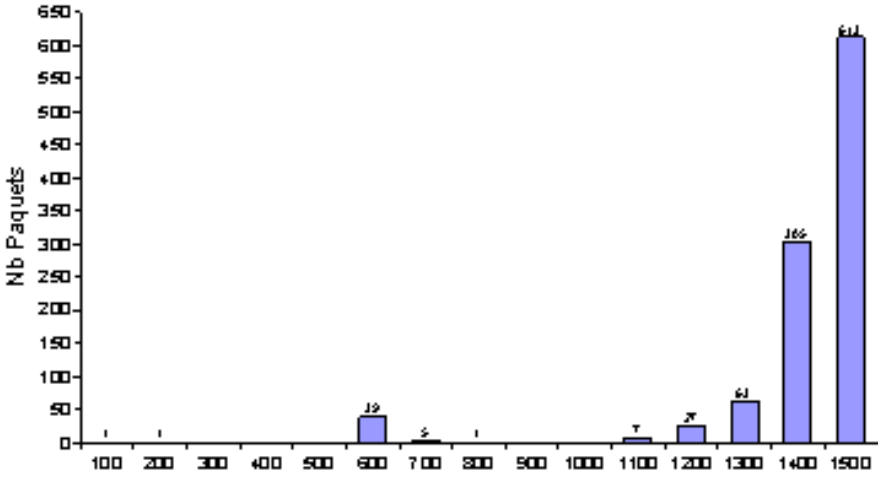
– les connexions à des services comme Webex [16] autorisant la mise en partage de données entre participants.

Il est bien évident que si le tunnel est utilisé principalement pour télécharger de gros fichiers, il est probable que ce critère ne soit plus déterminant, puisque le déséquilibre sera rétabli.

3.5 Taille des paquets

Une requête/réponse HTTP est la plupart du temps une transaction rapide, dans laquelle la connexion TCP va être utilisée au maximum afin que les informations arrivent vite au client ou au serveur : les paquets de données TCP seront en général remplis au maximum de la capacité du lien (MSS du serveur ou du client), et donc la plupart du temps entre 1400 et 1500 octets.

Le graphe ci-dessous est un histogramme de la taille moyenne des paquets reçus ou émis par un client HTTP placé derrière un relais Squid. La encore, les échanges inférieurs à 150 paquets ont été ignorés. On constate que la plupart des



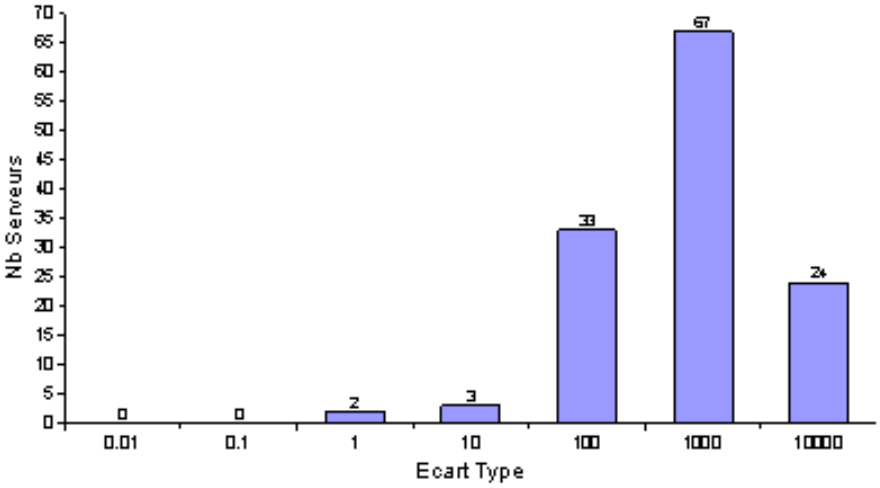
échanges ont une taille supérieure à 700 octets. Les deux connexions inférieures à 200 sont effectivement des tunnels (SSLTunnel et WebEx). Le cas des données entre 600 et 700 octets est plus difficile à trancher : il est à noter en particulier qu'en HTTPS, il est impossible de déterminer le contenu des échanges HTTP, et qu'en particulier l'utilisation du *KeepAlive* en HTTP 1.1 à l'intérieur de SSL peut amener, par exemple lors du chargement d'une page complexe dans la même connexion TCP, à produire de nombreux paquets de petite taille. Pour ces échanges, il est alors intéressant de vérifier si d'autres critères peuvent confirmer la suspicion (longueur, ratio, bande passante, URL évidemment, ...).

3.6 Intervalle entre requêtes

Une métrique intéressante à analyser est l'intervalle de temps écoulé entre chaque requête, pour les serveurs accédés de manière récurrente (par exemple plus de 200 requêtes/jour) par le même client.

Si le nombre de requêtes est élevé et que l'intervalle est réduit (moins d'une minute en moyenne), il est alors intéressant de calculer l'écart-type de cet intervalle : la distribution ci-dessous montre qu'un écart type inférieur à 10 est très rare, et dans les faits, le signal de requêtes automatiques (aspirateur comme *wget*) ou de tunnels dans HTTP nécessitant qu'un « poll » régulier du serveur soit effectué.

Ce critère permet aussi par exemple de détecter l'utilisation de passerelles Chat ↔ HTTP. HTTP comme celle utilisée par le client Microsoft MSN Mes-



senger quand il est situé derrière un relais HTTP.

4 Moltunnel

4.1 Principe et fonctionnement

Moltunnel est un logiciel développé par HSC permettant de mettre en pratique quelques unes des méthodes décrites ci-dessus.

Il est développé en C sous Unix (FreeBSD et Linux), en utilisant la bibliothèque LIBNIDS [17].

Cette bibliothèque, déjà utilisée dans de nombreux projets similaires ou connexes (*dsniff* [18], *tcpstatflow* [19], *netentropy*, *kill-p2p* [20]), permet notamment :

- D'écouter le trafic IP au travers de la bibliothèque *libpcap*.
- De réassembler de manière relativement fiable les flux TCP, et de présenter au programme les données telles qu'elles sont vues par les applications à chaque extrémité,
- De fermer brutalement une connexion écoutée en envoyant un paquet RST bien formé à chaque extrémité de la connexion.

Cette bibliothèque est donc utilisée dans *Moltunnel* pour intercepter le trafic (filtré ou non par une expression *pcap*). La liste des connexions en cours de traitement est visualisable par un socket TCP de commandes :

```
% telnet 127.0.0.1 19266
Trying 127.0.0.1...
Connected to localhost.hsc.fr.
Escape character is '^]'.

dumplist
192.70.106.70:47015->192.70.106.49:8080 : <- 18650638/21739 ->
1780562/15960 8836s)

192.70.106.49:4016->82.67.212.152:443 : <- 18650598/21928 ->
1780488/15143 8836s)

192.70.106.134:32855->192.70.106.49:8080 :
    <- 9698/13 -> 1872/6 25s)

192.70.106.134:43052->192.70.106.49:8080 :
    <- 1460/1 -> 428/1 75s)

192.70.106.68:39609->192.70.106.49:8080 :
    <- 2468/12 -> 1885/4 181s)

192.70.106.49:2595->72.14.205.19:443 :
    <- 2429/11 -> 1679/3 181s)

192.70.106.131:52116->$192.70.106.49:8080 :
    <- 39317037/27172 -> 165/1
54s)
```

Les traitements suivants sont ensuite effectués :

- Ignorer le trafic suivant les critères : adresse IP source, adresse IP destination, port destination, URL dans le cas de HTTP (filtrage par expression rationnelle),
- Dans le cas de trafic HTTP (ports à examiner paramétrables), vérification que la requête HTTP est bien formée (méthodes, syntaxe), qu'elle contient un en-tête *User-Agent*, un en-tête *Host*. Une liste noire de serveurs est paramétrable (expression rationnelle).

- Vérification pour le trafic HTTP que l'en-tête User-Agent est dans une liste autorisée (white list) ou dans une liste noire (black list) (expressions rationnelles).
- Vérification que la méthode CONNECT est utilisée sur des ports autorisés, et que le nom du serveur demandé n'est pas entièrement numérique.
- Dans le cas de trafic SSL (direct sur des ports paramétrables ou après ouverture du tunnel TCP via la méthode CONNECT), vérification que le premier paquet est bien issu du client, et que les deux premiers paquets sont bien *Client Hello* et *Server Hello* (ou *Client Handshake*, *Server Handshake* avec un sous type *Hello*).

Chacun de ces tests est désactivable en fonction de l'adresse IP source, destination ou par couple (source, destination). Dans le cas de HTTP, l'adresse destination est également cherchée dans le nom du serveur placé dans l'URI envoyée au proxy.

Chacun des tests positifs produit un avertissement dans l'erreur standard.

A la fin de chaque connexion, *Moltunnel* écrit dans un fichier plat au format CSV une ligne contenant les données suivantes :

- date de début en ms de la connexion (format Unix)
- adresse IP source, port source, adresse IP destination, port destination
- nombre de paquets reçus par le client, volume en octets reçus par le client, taille moyenne de ces paquets, intervalle moyen de ces paquets,
- mêmes informations pour les paquets envoyés au serveur,
- URI demandée dans le cas de HTTP, User-Agent

Ce fichier CSV peut ensuite être traité par des outils d'analyse basique de statistiques, écrits en Perl, qui pourront signaler :

- les connexions plus longues qu'un temps paramétrable par port destination (par exemple 1000s),
- les connexions dont le ratio upload/download est supérieur à un seuil paramétrable par port,
- les connexions dont la taille moyenne des paquets est inférieure à un seuil paramétrable par port,
- les couples (clients/serveur final) détectés plus de 200 fois, avec un intervalle moyen entre requêtes inférieur à 60 secondes et un écart type inférieur à 40 (seuils paramétrables).

Enfin les scripts effectuent un tri et une édition :

- n clients les plus consommateurs en nombre de requêtes et volumes échangés,
- n serveurs les plus accédés (nombre de requêtes, volumes échangés).

4.2 Tunnels détectés

Même avec des métriques et des recherches si simples, il est relativement aisé de détecter l'utilisation de nombreux outils cités précédemment. Les exemples ci-dessous ne sont évidemment pas exhaustifs, le principe étant évidemment d'offrir un outil générique.

4.3 Détection temps réel

– Détection de Skype utilisé à travers une connexion SSL directe :
```
09:52:39 192.70.106.104:58741->212.72.49.141:443 :
   Invalid SSL client Hello
09:52:40 192.70.106.104:58741->212.72.49.141:443 :
   Invalid SSL server Hello
```

– Détection de Skype dans un proxy HTTPS utilisant la méthode CONNECT, par au moins trois méthodes :
```
16:45:24 192.70.106.104:56478->192.70.106.49:8080 :
    CONNECT all numeric : 212.72.49.141:443
16:45:24 192.70.106.104:56478->192.70.106.49:8080 :
    No User-Agent for : CONNECT 212.72.49.141:443

 16:45:24 192.70.106.104:56478->192.70.106.49:8080 :
    No Host for : CONNECT 212.72.49.141:443
```

– Détection de *GNU-HTTPTUNNE*L par détection de l'absence d'en-tête *User-Agent* :
```
11:20:26 192.70.106.104:35205->192.70.106.49:8080 :
    No User-Agent for : POST http://roadrunner.dyn.\
    rominet.net:8080/index.html?crap=1142504424
```

– Détection de Google Talk par interception de l'en-tête *User-Agent* :
```
08:50:47 192.168.128.108:1227->192.70.106.49 :8080 :
    Bad User-Agent (Google Talk) for :
    CONNECT www.google.com:443
```

– Détection de logiciels mal configurés émettant des requêtes vers l'extérieur pour des serveurs internes (ici le cas typique d'un client Windows essayant de monter une ressource en utilisant le redirecteur WebDav) :
```
09:33:37 192.70.106.80:1027->192.70.106.49:8080 :
  Bad User-Agent (Microsoft-WebDAV-MiniRedir/5.1.2600)
  for : OPTIONS http://hsc/
```

– Utilisation d'un client MSN (ici Gaim) utilisant CONNECT :
```
09:47:28 192.70.106.103:46240->192.70.106.49:8080 :
    CONNECT all numeric : 65.54.228.31:1863
09:47:28 192.70.106.103:46240->192.70.106.49:8080 :
    No User-Agent for : CONNECT 65.54.228.31:1863
```

– Utilisation d'un protocole non autorisé sur le port 443 :
```
10:50:39 192.70.106.49:3494->81.62.237.118:443 :
    Invalid SSL trafic (Server Banner})
```

Détection statistique

- Détection de l'utilisation de WebEx pour une prise de contrôle à distance par alerte sur le ratio U/D ou D/U et la taille des paquets :

```
Wrong Ratio 0.334;1144151045.457;192.70.106.104;32840;
  192.70.106.49;
  8080;45670;80;11417;142.7;503;239;64529;270.0;191;
  "CONNECT elcb14.webex.com:443";

Small Packets 142.7 / 270.0:
1144151045.457;192.70.106.104;32840;192.70.106.49;8080;
   45670;80;11417;142.7;503;239;64529;270.0;191;
   "CONNECT elcb14.webex.com:443";
```

- Détection de l'utilisation de *SSLTunnel* par les mêmes métriques :

```
Wrong Ratio 0.992;1144163372.934;192.70.106.103;40149;
   192.70.106.49;8080;876118;501;111465;222.5;1748;505;
   86801;171.9;1734;"CONNECT smash.dyndns.org:443";

Small Packets 222.5 / 171.9:
1144163372.934;192.70.106.103;40149;192.70.106.49;8080;
   876118;501;111465;222.5;1748;505;86801;171.9;1734;
   "CONNECT smash.dyndns.org:443";
```

- Détection de l'utilisation de Google Talk à travers un relais HTTPS :

```
Small Packets 166.5 / 154.3:
1144136838.355;192.70.106.103;46236;192.70.106.49;
8080;5602559;45;7494;166.5;122660;109;16824;154.3;51399;
  "CONNECT talk.google.com:5222";
```

- Détection d'une connexion longue (3400 secondes) vers le serveur de tunnel commercial *HTTP-TUNNEL* :

```
Long connection: 1144144696.472;192.70.106.103;47894;
   192.70.106.49;8080;3422533;2901;2842855;980.0;1176;
   1;182;182.0;0;"POST http://209.8.233.161/data.htm";
```

- Connexion longue (12529 secondes) vers une passerelle HTTP ↔ IRC :

```
Long connection:
1144148340.272;192.70.106.104;58656;192.70.106.49;8080;
12529072;1895;389335;205.5;6611;1;828;828.0;0;"GET
 http://webchat.xs4all.nl/cgi-bin/ircnet/nph-irc.cgi";
```

Les analyses des requêtes multiples sont plus difficiles à analyser, on peut lister ci-dessous le tableau émis par *Moltunnel* pour les ensembles de connexion de plus de trois minutes, avec un nombre de requêtes supérieur à 10, et un écart type inférieur à 60.

Client-Serveur	Tps écoulé	Requ	Moyenne	EC
192.70.106.101-207.46.1.4	19407	978	19.84	2.87
3-192.70.106.132-www.lemondeinformatique.fr	303	356	0.85	39.07
192.70.106.132-www.lemondeinformatique.fr	303	356	0.85	39.07
192.70.106.102-www.immostreet.fr	309	403	0.77	20.29
4-192.70.106.132-www.lemondeinformatique.fr	303	356	0.85	39.07
192.70.106.102-www.3suisses.fr	238	366	0.65	26.60
1-192.70.106.132-www.lemondeinformatique.fr	303	356	0.85	39.07
192.70.106.101-207.46.1.8	1376	85	16.19	58.16
2-192.70.106.132-www.lemondeinformatique.fr	303	356	0.85	39.07
192.70.106.131-ftp.de.debian.org	599	285	2.10	37.94
192.70.106.132-www.eecis.udel.edu	185	195	0.95	15.22
192.70.106.132-ng-prod1.cisco.com	782	192	4.07	56.07
192.70.106.102-www.super-secretaire.com	656	1405	0.47	18.67
1-192.70.106.131-ftp.de.debian.org	599	285	2.10	37.94
192.70.106.132-www.networkcomputing.com	334	135	2.48	17.40
1-192.70.106.132-www.eecis.udel.edu	185	195	1.18	25.15
192.70.106.134-www.ter-sncf.com	189	276	0.69	24.17
\textbf{192.70.106.104-192.70.106.166:8080	675	854	0.79	0.32
1-192.70.106.132-www.networkcomputing.com	334	135	2.48	17.40
192.70.106.78-altern.org	217	55	3.95	53.67
192.70.106.102-sec.3suisses.fr	185	326	0.57	14.97
192.70.106.80-www.conrad.fr	241	204	1.18	25.15
192.70.106.103-smash.dyndns.org:80	1260	1119	1.13	0.85
2-192.70.106.131-ftp.de.debian.org	599	285	2.10	37.94
2-192.70.106.132-www.networkcomputing.com	334	135	2.48	17.40
192.70.106.78-www.channel4.com	182	80	2.28	43.93

Chercher les écart-types inférieurs à 5 permet de repérer trois tunnels ou assimilés effectifs :

- une utilisation très longue de MSN Messenger via la gateway HTTP 207.-46.1.4,
- une utilisation de *Loophole*, reconnaissable à un écart-type très faible,
- une utilisation de *FirePass*, par le même moyen.

On voit aussi que certains sites sont à la limite du faux positif (www.super-secretaire.com) si le critère « écart-type » est trop monté, et qu'il serait intéressant de corréler ces résultats avec d'autres métriques à déterminer.

5 Limitations et bugs, évolutions

A l'heure actuelle, Moltunnel est trop dépendant des problèmes éventuels de *libnids* : certaines connexions vers le proxy (en particulier depuis Internet Explorer) n'étaient jamais détectées, ce qui pose évidemment de gros problèmes :

- pas d'analyse statistique possible,
- encombrement de la mémoire.

Un correctif permettant de pallier ce problème (envoi simultané des segments FIN par les deux parties) a été développé. Un autre problème, en cours de

résolution, est l'incapacité de *libnids* à gérer des tailles de fenêtres supérieures à 64Ko : hors certaines piles TCP, dont Windows, utilisent la technique du *TCP Window Scaling* lors de gros téléchargements, ce qui aboutit à des buffers trop larges non acquittés :

```
Apr 4 15:50:29 fw libnids: Too much data in TCP receive queue,from
192.70.106.49:8080 to 192.70.106.132:35380
```

Cela pose un problème assez sérieux quand de nombreux téléchargements importants sont effectués, puisque *libnids* stoppe l'observation de la connexion.

Il est évident que les contournements possibles des détections effectuées sont nombreux :

- Correction et mise en place des en-têtes adéquats dans les logiciels de tunnels,
- Utilisation dans les logiciels d'une connexion montante et d'une descendante afin que les ratios échangés restent corrects (ce qui est déjà le cas pour *Gnu-HTTPTUNNEL*),
- Rupture de la connexion à intervalles réguliers afin de conserver des requêtes de durée normale,
- Mise en place de fausses requêtes mélangées avec celles transportant des données afin de perturber les détections liées à l'écart-type,
- Utilisation de paquets d'échange ressemblant à du SSL (cas déjà semble t-il de Skype pour certaines de ses connexions),

Un des axes d'évolution possibles, qui permettrait d'être plus générique et plus résistant, serait de faire passer les profils de connexion dans des réseaux de neurones, afin d'identifier des patterns :

- à la fois dans la connexion TCP elle-même,
- dans les connexions effectuées vers un serveur en particulier.

D'autre part, d'un point de vue plus pratique, il serait nécessaire :

- de prévoir une infrastructure agent/collecteur, dans lequel des agents d'écoute suivent les connexions TCP en de multiples points du réseau et envoient leur données à un serveur central qui effectue l'analyse statistique, avec une configuration centralisée.
- d'évaluer la possibilité d'utiliser d'autres bibliothèques que NIDS, et en particulier de valider s'il est possible d'utiliser des moteurs plus évolués tels que Snort [21] pour ce travail.

Le code de *Moltunnel* devrait être disponible sous licence BSD au second semestre 2006.

Références

1. SOCAT : `http://www.dest-unreach.org/socat/`
2. SSLTunnel : `http://www.hsc.fr/ressources/outils/ssltunnel/`
3. Stunnel : `http://www.stunnel.org`
4. FirePass : `http://www.gray-world.net/pr_firepass.shtml`

5. Gnu-HTTPTunnel : `http://www.nocrew.org/software/httptunnel.html`
6. LoopHole : `http://www.loopholesoftware.com/`
7. TotalRC : `http://www.totalrc.net/`
8. Hopster : `http://www.hopster.com/`
9. HTTP-TUNNEL : `http://www.http-tunnel.com/`
10. PingTunnel :`http://www.cs.uit.no/~daniels/PingTunnel/`
11. NSTX : `http://freshmeat.net/projects/nstx/`
12. Outil interne HSC
13. Fragrouter : `http://www.anzen.com/research/nidsbench/fragrouter.html`
14. NetEntropy : `http://www.lsv.ens-cachan.fr/~olivain/net-entropy/`
15. Aventail : `http://www.aventail.com/`
16. Webex : `http://www.webex.com/`
17. LibNids : `http://libnids.sourceforge.net/`
18. DSNIFF : `http://monkey.org/~dugsong/dsniff/`
19. Tcpstatflow : `http://www.geocities.com/fryxar/`
20. Kill-p2p : `https://cvs.orion.education.fr/savane/projects/kill-p2p/`
21. Snort : `http://www.snort.org/`