Liberté • Égalité • Fraternité
RÉPUBLIQUE FRANÇAISE
PREMIER MINISTRE

S . G . D . S . N
Agence nationale de la sécurité des systèmes d'information
CERTA

Paris, le 21 mars 2011
N° CERTA-2001-INF-003-002

Affaire suivie par :
CERTA

NOTE D'INFORMATION DU CERTA

Objet : Tunnels et pare-feux : une cohabitation difficile.

Gestion du document

Référence	CERTA-2001-INF-003-002
Titre	Tunnels et pare-feux : une cohabitation difficile.
Date de la première version	29 août 2001
Date de la dernière version	21 mars 2011
Pièce(s) jointe(s)	Capture de trames

TAB. 1 – *gestion du document*

Une gestion de version détaillée se trouve à la fin de ce document.

1 Résumé

Le désir d'utiliser de nouveaux protocoles (messagerie instantanée, partage de fichiers en point-à-point, etc.) amène certains utilisateurs à mettre en place des outils permettant de contourner les règles de filtrage du pare-feu des organisations jugées trop restrictives. Parmi toutes les techniques possibles, les tunnels HTTP rencontrent de plus en plus de succès. On peut citer à titre d'exemple, le cas de la voix sur IP ou de SKYPE.

Les tunnels applicatifs sont difficilement détectables. Il est donc important pour le responsable SSI de sensibiliser ses utilisateurs sur les dangers liés à l'utilisation de tunnels. En complément, cette note d'information précise un certain nombre de recommandations afin de se protéger des risques liés à ces tunnels.

L'étude du logiciel HTTHost, présenté en dernière partie de ce document, permet d'illustrer ces propos, en détaillant d'un point de vue technique le mécanisme de tunnel dans un protocole TCP/IP.

2 Introduction : les tunnels

Dans la littérature, l'expression « pile de protocoles » est souvent employée et rappelle que l'architecture de TCP/IP peut être vue comme un ensemble de protocoles empilés les uns sur les autres. Dans ce modèle en couches, les données issues d'un protocole de niveau haut sont encapsulées dans le protocole de niveau inférieur : ainsi une requête HTTP est transportée dans le champ « données » d'un segment TCP qui sera lui-même transporté à l'intérieur d'un datagramme IP, etc.

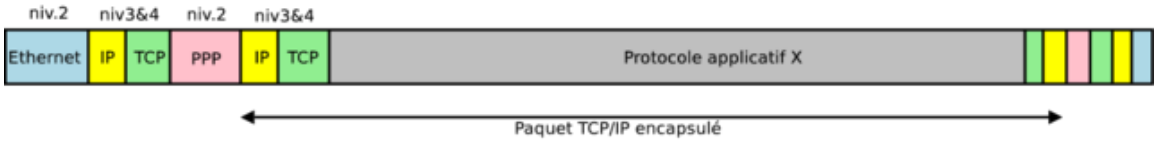

FIG. 1 – *Encapsulation PPP over IP.*

Dans certains cas, l'utilisation de cette propriété d'encapsulation peut paraitre déroutanteé: il arrive qu'un protocole de niveau bas soit encapsulé dans un protocole de niveau haut (ou de même niveau). On parle alors de tunnel : une « connexion » est créée entre deux noeuds du réseau pour transporter le protocole encapsulé.

On pourra citer en exemple l'encapsulation de PPP (*Point-to-Point Protocol*) dans un paquet IP, désignée par PPP *over* IP. Cette encapsulation permet, entre autre, de faire du contrôle d'accès sur un réseau TCP/IP classique par le biais de la couche PPP.

Les tunnels sont ainsi utilisés dans des applications « légitimes » ou non au regard de la politique de sécurité adoptée, telles que :

- les réseaux privés virtuels. Le tunnel permet l'usage d'une infrastructure partagée pour fournir une connectivité dédiée, souvent chiffrée ;
- le transport de protocoles non routables (l'encapsulation de datagrammes IP dans des datagrammes IP est ainsi préconisée pour les noeuds IP nomades) ;
- l'application SKYPE. Selon les informations aujourd'hui disponibles, elle encapsule des entrées/sorties chiffrées dans SSL/TLS. Comme les politiques de sécurité autorisent dans leur grande majorité, les protocoles HTTP et HTTPS, SKYPE traverse les pare-feux sans créer d'évènement particulier [10].

Cependant, l'utilisations de tunnels peut être moins classiques : l'utilisateur d'un réseau protégé peut utiliser un tunnel pour traverser le pare-feu et prendre en défaut la politique de sécurité.

3 Le cas des services web

3.1 Généralités

Il est intéressant d'observer le développement des services web sous l'angle de la problématique des tunnels. Comme le détaille [9], « les Web Services permettent de faire communiquer deux sous-systèmes d'architectures semblables ou différentes de manière standard dans le but d'échanger des services ou traitements applicatifs. Les Web Services sont un moyen de distribuer un service de manière standard grâce à XML tout en respectant un modèle de développement ayant déjà fait ses preuves ». En d'autres termes un service web permet de relier des composants logiciels différents sur le web.

L'échange de messages entre le service Web prestataire et le client est le plus souvent formalisé dans un document WSDL (*Web Services Description Language*) : l'interface, et utilise des protocoles comme SOAP, HTTP GET/POST et MIME. Par ailleurs de nombreux protocoles comme par exemple XKMS, XMLSign, XACML, SAML, etc. permettent de sécuriser les services web.

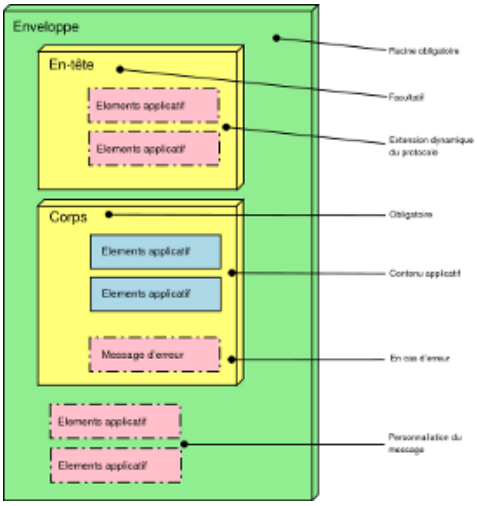

FIG. 2 – *Structure d'un message SOAP.*

3.2 Simple Object Access Protocol

Le protocole SOAP (*Simple Object Access Protocol*) est un protocole de transmissions de messages permettant d'invoquer des applications sur des réseaux hétérogènes et décentralisés. Pour réaliser le dialogue entre applications distribuées, le corps d'un message SOAP comprend une logique applicative qu'il transporte sur les réseaux. Pour réaliser ce transport d'éléments applicatifs, SOAP s'appuie sur un protocole de transport applicatif, souvent HTTP ou HTTPS, mais également SMTP ou POP. Comme tout protocole utilisé dans les services web, SOAP est bâti autour

d'XML. Un message SOAP encapsulé dans un flux HTTP et embarquant des éléments applicatifs est typiquement le cas d'un tunnel qui peut être construit pour contourner la politique de sécurité.

3.3 Services web et politique de sécurité

L'enjeu des services web est d'assurer l'interopérabilité nécessaire pour réaliser des applications distribuées mais, pour réaliser ce défi, les services web s'appuient sur HTTP ou HTTPS et peuvent devenir des outils de contournement d'une politique de sécurité. Il est essentiel de bien comprendre que les très nombreux projets ou déploiements d'applications fondées sur les services web (messagerie, agenda, etc.), en contournant l'IP risquent, de fait, de devenir des moyens d'éviter l'application de la politique de sécurité.

Le services web appartiennent à la famille des « appels de procédures distantes (RPC) » [9], famille qui possède une longue hitoire de failles (MSRPC) et de divulgations incontrôlées d'informations. Alors que, classiquement, le contrôle des « appels de procédures distantes » peut être grossièrement réalisé avec du filtrage de ports, l'utilisation d'un service légitime comme HTTP pour le transport, implique un filtrage de contenu complexe au niveau applicatif. Ce type de filtrage a de lourdes implications au niveau d'un pare-feu en terme de taille de code et donc de risque et de ressources nécessaires.

4 Contournement des pare-feux

La multiplication des applications type messagerie instantanée (ICQ, MSN, Instant Messenger...) ou le partage de fichiers en point-à-point (avec napster, gnutella,...) peuvent susciter chez l'utilisateur le besoin de contourner un pare-feu jugé trop restrictif.

La création d'un tunnel permet d'utiliser des protocoles non autorisés à travers le pare-feu et permet à des machines qui n'en ont pas le droit de communiquer avec l'extérieur, contournant ainsi la politique de sécurité du site.

Suivant le type de pare-feu, plusieurs techniques peuvent être employées :

4.1 Routeur filtrant

Si le pare-feu est un simple routeur filtrant, la création d'un tunnel pour utiliser un protocole non autorisé ne s'impose même pas : avec ce type de filtrage, le contrôle porte uniquement sur l'adressage (adresse IP, port source et port destination des paquets) et non sur le contenu de la communication. On peut donc utiliser un protocole X sur un port standardisé pour le protocole Y, par exemple, utiliser ICQ sur le port 80 normalement réservé à HTTP. On peut aussi émettre des paquets vers un port TCP non privilégié (1024 et supérieurs) en forçant le port d'émission TCP à 20 (FTP-DATA) : généralement, sur un routeur filtrant, les connexions TCP sur les ports non privilégiés depuis le port source 20 sont autorisées pour l'usage de FTP en mode actif.

Dans le cas d'un routeur filtrant, l'emploi d'un tunnel peut quand même se justifier : volonté de chiffrer des données (encapsulation d'un protocole dans SSL ou SSH par exemple) ou de permettre à une machine du réseau de communiquer vers l'extérieur ; le tunnel sera alors réalisé par une machine non filtrée du réseau interne qui pourra par exemple réaliser une encapsulation de PPP dans une simple session TCP.

4.2 Filtres applicatifs

Si le pare-feu intègre des filtres applicatifs (*proxies*), le tunnel sera réalisé en transportant le protocole interdit dans un protocole dont le passage est autorisé : DNS, HTTP, SMTP... L'encapsulation et la désencapsulation étant réalisées aux deux extrémités du tunnel, le protocole transporté dans un tunnel applicatif n'est pas visible au niveau des noeuds intermédiaires (notamment le pare-feu) sans une analyse de contenu des paquets. Si le contrôle de la sémantique du protocole réalisé par le filtre applicatif permet de s'assurer que c'est bien le protocole X sur le port Y, il ne permet pas de s'assurer qu'un protocole Z n'est pas transporté dans les données : par définition, le champ « données » est aléatoire !

Créer un tunnel nécessite seulement la coopération entre deux entités, les extrémités du tunnel. Sur l'Internet, des sociétés mettent à disposition des serveurs réalisant une extrémité de tunnel ainsi que des graticiels permettant de réaliser l'autre extrémité du tunnel sur la machine de l'utilisateur. Réaliser un tunnel pour contourner un pare-feu est à la portée du premier venu.

Le tunnel HTTP est de plus en plus souvent adopté, rares étant les sites interdisant l'utilisation de ce protocole.

5 Les risques du tunnel

La connexion virtuelle créée au moyen d'un tunnel revient à ouvrir sur l'extérieur une porte qui n'est absolument pas gardée : il n'y a plus de contrôle d'accès sur les paquets entrants qu'ils soient ou non chiffrés. Ainsi, pour peu que sa machine hôte soit configurée comme un routeur, un tunnel réalisé au moyen de l'encapsulation de PPP dans une connexion TCP permettrait à un utilisateur de rendre accessibles depuis l'Internet les machines d'un réseau interne, **quel que soit le filtrage mis en oeuvre au niveau du pare-feu**.

De plus, la multiplication des protocoles utilisés (et donc des applications) multiplie les vulnérabilités (débordement mémoire par exemple).

Enfin, il faut souligner que bien souvent l'utilisateur va employer un produit de provenance inconnue pouvant dissimuler une porte dérobée ou un cheval de Troie comprenant un tunnel.

6 Quelques solutions

6.1 Le pare-feu

L'emploi de filtres applicatifs permet de limiter les possibilités de création de tunnel, à condition toutefois de soigner l'écriture des règles de filtrages en limitant les ouvertures au strict minimum.

De nombreux articles [1] [2] traitent du filtrage de paquets ICMP. Limiter le type de paquets ICMP permet de se prémunir des attaques, par déni de service notamment. Mais permettre le passage des paquets ICMP_ECHO, ICMP_REPLY, ICMP_UNREACHABLE, ICMP_TIMEEXCEED laisse la porte ouverte à certains tunnels. Le projet LOKI [3] décrit l'utilisation de paquets ICMP_ECHO et ICMP_REPLY pour créer un tunnel à travers un pare-feu. Cependant, l'ICMP est nécessaire au bon fonctionnement d'IP. Il convient donc d'être très prudent sur son filtrage.

La création de tunnel par encapsulation dans des requêtes DNS est évoqué dans un article posté sur la liste de diffusion Bugtraq [4]. Utiliser des relais applicatifs permet de limiter les risques : dans ce cas, seuls ces relais sont autorisés à accéder aux DNS externes.

Les pare-feux sont nécessaires. **Cependant, ils ne protègent pas contre toutes les menaces**. Ainsi, dès qu'une connexion est établie entre une machine du réseau interne et une machine sur l'Internet, un tunnel peut être créé. Et même si une configuration soignée permet d'empêcher l'usage de certains types de tunnels, la possibilité de créer des tunnels applicatifs (dans HTTP notamment) existe toujours.

6.2 La détection des tunnels

La détection des tunnels non autorisés est difficile : l'encapsulation peut revêtir différentes formes, depuis l'encapsulation dans un tunnel chiffré (SSH ou SSL par exemple) jusqu'à l'utilisation de protocole applicatif (HTTP, DNS, ...).

Plusieurs techniques peuvent toutefois donner des résultats :

- filtrer les connexions à des extrémités de tunnel qui sont connues (passerelles sur l'Internet en libre accès) ;
- identifier les « profils » des connexions atypiques et suspectes : connexions interactives longues, volume de données émis sur le port 80/TCP important (HTTP est un protocole de type requête/réponse. En utilisation normale, le volume de données en réception est bien supérieur au volume de données en émission) ;
- détecter l'occurrence de chaînes de caractères à l'intérieur des données. Il s'agit par exemple de repérer dans le début d'une session une chaîne de caractères identifiant la négociation associée à un protocole particulier comme « SSH-1. » ou « SSH-2. » ...

Filtrer des connexions en fonction de leur destination est très simple à mettre en oeuvre techniquement mais ressemble étrangement au jeu du chat et de la souris. L'administrateur du pare-feu est obligé de remettre très souvent à jour les règles de filtrage. Pour s'en convaincre il suffira de lire l'article [5] pour voir comment des étudiants essayent de bricoler pour obtenir l'utilisation sans restriction de napster, face à l'administration dont le souci est l'engorgement du réseau. Ceci est aussi directement transposable au cas de SKYPE.

Le filtrage de l'adresse destination est pourtant incontournable dans certains cas. En effet, des protocoles comme HTTPS (port 443/TCP) réalisent une encapsulation dans un tunnel chiffré, déjouant toute tentative de recherche d'occurrence de chaînes de caractères à l'intérieur des données (sauf à utiliser une technique de type attaque par le milieu). Pour ces protocoles, le relais applicatif du pare-feu est un simple redirecteur de session TCP sans aucun filtrage relatif à la sémantique du protocole applicatif.

Les deux dernières techniques évoquées (statistiques, signatures) sont similaires à celles employées par les sytèmes de détection des intrusions. Il peut donc être envisagé de réutiliser ce type d'outils avec toutefois les limitations associées (existence de faux positifs et de faux négatifs).

Dans [8], les auteurs ont mis en oeuvre ces techniques afin de détecter des portes dérobées (*backdoor*). Deux heuristiques ont été développées afin de mettre en évidence des sessions interactives :

– détection de sessions utilisant de nombreux paquets de taille réduite ;

– détection de sessions dans lesquelles le temps inter-paquets est très courts.

Ces travaux se basent sur des mesures de trafic réel dans lequelles il apparaît que les protocoles interactifs comme TELNET ou RLOGIN utilisent à 99,7% des paquets de moins de 20 octets et que le temps inter-paquets n'est pas constant mais suit une loi de distribution de Pareto.

6.3 La politique de sécurité

Empêcher l'utilisation de tunnels non autorisés est avant tout un problème de politique de sécurité. Dans certains articles, le tunnel est présenté comme un outil permettant de « contourner les limitations du pare-feu ». Cet argument peut être jugé recevable par une personne n'ayant pas reçu d'information sur la politique de sécurité en vigueur sur le site et les dangers liés à l'utilisation des tunnels.

L'efficacité du pare-feu est donc dépendante du niveau de maturité entretenu sur le site interne. Communiquer sur la politique de sécurité et informer les utilisateurs sur les aspects techniques touchant à la création du tunnel est indispensable.

Au-delà des solutions techniques, le RSSI doit aussi s'attacher déterminer clairement le périmètre du système d'information à protéger. Cela signifie en particulier de définir les droits de chacun afin de maîtriser les applicatifs installés sur les machines hôtes du réseau. En effet, si les utilisateurs ont la possibilité d'installer des outils, par ailleurs disponibles sur Internet, permettant de contourner les pare-feux (sans oublier les ActivX, AppletJava et, d'une façon générale, les codes mobiles) les mesures techniques recommandées dans ce document ne seront d'aucune utilité. De plus, il est essentiel pour le RSSI de maîtriser les hôtes connectés (problème de la mobilité des hôtes et donc des connexions intermittentes) au système à protéger.

Des choix sont proposés pour un cas particulier dans la circulaire du 09 août 2005 [11].

7 Etude de cas : HTTPort et HTTHost

Cen cas concret permet de toucher du doigt ce qu'est réellement un tunnel et les techniques permettant de le détecter.

Dans cette étude de cas, HTTPort 3.SN et HTTHost 1.5.1 ont été installés pour les tests. La version de HTTHost avec chiffrement n'a pas été testée.

7.1 Le besoin

Un utilisateur du réseau interne souhaite se connecter à un serveur baptisé « dicton-du-jour ».

Le protocole « dicton-du-jour » mis en oeuvre entre le client et le serveur est très simple :

– le client se connecte sur le port 1234 du serveur. Le client envoie le prénom (chaîne de caractères terminée par le signe égal, « = ») ;

– le serveur répond par le dicton du jour correspondant au prénom.

Exemple :

– le client envoie « Rose= » ;

– le serveur répond « Á la sainte Rose, pas de pause. ».

Malheureusement, notre utilisateur ne peut se connecter au serveur dicton-du-jour externe à son réseau puisque le pare-feu (utilisant un relais applicatif) ne permet que le passage du protocole HTTP.

Notre utilisateur va donc installer HTTPort (disponible gratuitement sur l'Internet) afin de créer un tunnel HTTP permettant de contourner le filtrage réalisé par le pare-feu.

Note : Outre la création de tunnel HTTP, HTTPort propose aussi la création de tunnels au moyen de la méthode « CONNECT »". L'usage de cette méthode est très peu documenté. Elle n'est que brièvement évoquée dans le RFC2616 (HyperText Transfert Protocol version 1.1). Toutefois, il apparaît que certains relais applicatifs permettent l'utilisation de ce type de requêtes pour créer des tunnels (voir [6]). Cette fonctionnalité est dangereuse car cela revient quasiment à mettre un serveur socks (RFC 1928) en libre accès sur le pare-feu.

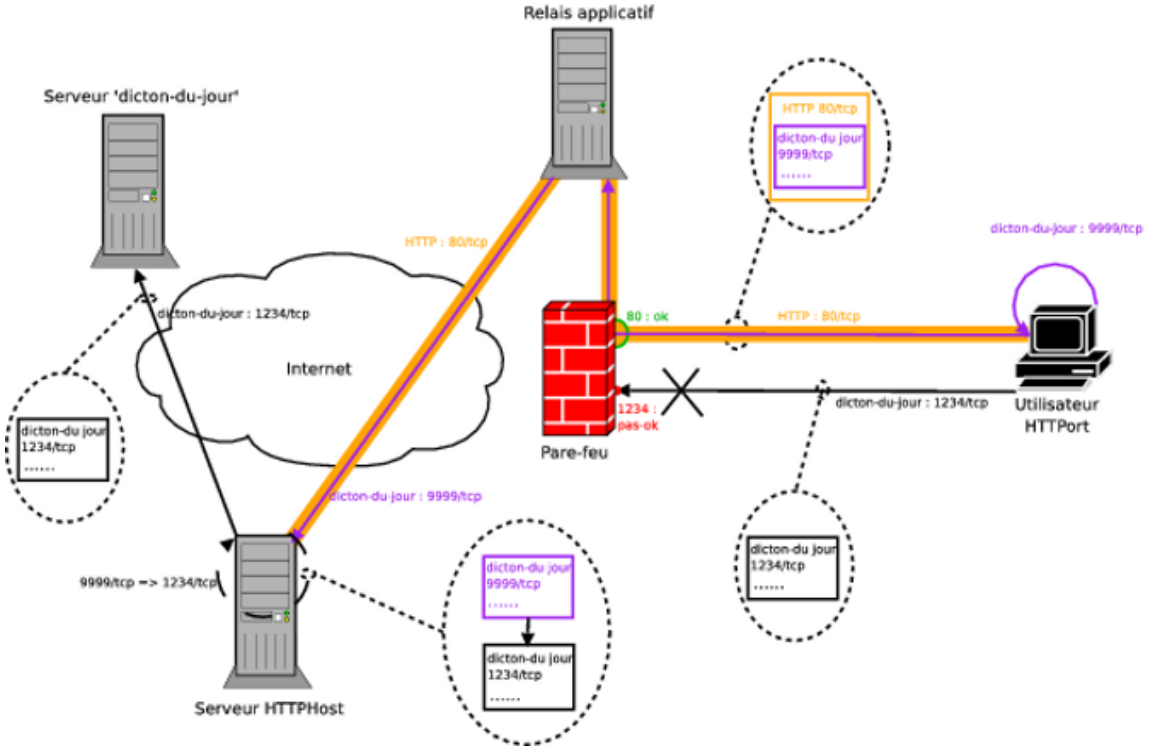

FIG. 3 – *Schéma récapitulatif du mode de fonctionnement de HTTHost et HTTPort.*

7.2 Le paramétrage de HTTPort sur la machine cliente

L'utilisateur indique tout d'abord à HTTPort l'adresse d'un serveur HTTHost qui jouera le rôle de l'extrémité du tunnel sur l'Internet. Ce serveur peut être une machine publique ou une machine privée exécutant une version de HTTHost Personnal Edition.

Pour chaque application utilisant le tunnel, deux étapes sont nécessaires. Ainsi, pour l'application « dicton-du-jour » :

- l'utilisateur paramètre le client « dicton-du-jour » pour envoyer les requêtes, non plus directement vers le port TCP 12324 du serveur « dicton-du-jour », mais vers le port TCP 9999 de son poste, localhost. Les requêtes seront ainsi interceptées par HTTPort.
- l'utilisateur indique ensuite à HTTPort que les connexions qui sont réalisées sur le port TCP 9999 de localhost devront être envoyées vers le port TCP 1234 du serveur « dicton-du-jour » à travers le relais HTTP hébergé sur PROXY, port 8080/TCP.

7.3 Comment cela fonctionne-t-il ?

Á l'arrivée d'un paquet sur le port 9999, HTTPort crée un tunnel HTTP vers HTTHost. La requête du client « dicton-du-jour » est envoyée (encodée) dans une en-tête HTTP à travers le proxy applicatif vers HTTHost. Á l'autre extrémité du tunnel, HTTHost décode l'en-tête HTTP et envoie la requête ainsi obtenue sur le port TCP 1234 du serveur « dicton-du-jour ». Pour le serveur « dicton-du-jour », c'est la machine hébergeant HTTPHost qui joue le rôle de client.

La réponse reçue du serveur « dicton-du-jour » par le serveur HTTHost est ensuite retournée dans une page HTML à HTTPort. Ces données sont extraites par HTTPort et réacheminées vers le client « dicton-du-jour » qui tourne sur la même machine.

7.4 Analyse du trafic

Note : en annexe figurent les trames échangées entre HTTPort et HTTHost.

Les tests ont été réalisés sur un réseau qui n'est pas connecté à l'Internet. Les adresses IP sont des adresses non officielles.

Le trafic (vu du relais applicatif du pare-feu) ressemble à un trafic HTTP normal :

- le format des requêtes en provenance de la station de l'utilisateur et à destination de HTTHost est conforme au format des requêtes HTTP tel que défini dans le RFC 2616 ;

– le format des réponses en provenance du serveur HTTHost et à destination de la station de l'utilisateur est conforme au format des réponses HTTP tel que défini dans le RFC 2616.

Le tunnel HTTP encapsule les données de deux manières différentes :

– l'envoi des données (depuis HTTPort vers HTTHost) est réalisé dans une pseudo en-tête HTTP. En regardant de plus près le paquet 1.5, on s'aperçoit que l'avant-dernière en-tête (nommée « Uxaxr ») contient les données émises par le client « dicton-du-jour » (soit la chaîne de caractères « Rose= ») codée en base64 (RFC 1521) ;
– le transport des données (depuis HTTHost vers HTTPort) est réalisé dans le corps d'un pseudo message HTML. Dans le paquet 1.8, la réponse du serveur (soit la chaîne de caractères « Á la sainte Rose, pas de pause. ») est encadrée par la balise `<fsswihxh>`. Le codage utilisé est également base64.

7.5 Comment détecter l'utilisation de HTTHost ?

Ici, l'extrémité du tunnel situé sur le réseau externe n'est pas une machine publique. On ne peut donc pas mettre en place un filtre sur le nom du serveur puisque ce nom n'est pas connu à priori.

La première pistee venant à l'esprit est la détection des balises « exotiques » employées dans les documents HTML renvoyés par HTTHost en réponse à une requête HTTP. Ainsi, l'apparition de la chaîne de caractères `<ghlbjug>` ou `<fsswihxh>` dans un paquet TCP réémis par le relais HTTP du pare-feu et à destination d'une station du réseau interne est probablement la signature d'un dialogue établi entre HTTPort et HTTHost.

Hélas, différents tests montrent que les balises changent à chaque redémarrage de HTTHost. La recherche de balises « exotiques » n'est donc pas la bonne méthode.

Si on regarde plus attentivement les requêtes HTTP créées par HTTPort, on s'apercoit que, dans la pseudo URL, une séquence de caractères revient à chaque fois :

```
Paquet 1.1 : /Oxgllxs?gUHHlIs6rIqDfEHNHCukfFtjxOSiuBg.....
Paquet 1.2 :  /Vmhmco?pUjHrIS6AIcDZENNqCWkZFujpOviHBe.....
                       U H I 6 I F E N C k F j O i B
```

L'intégralité de la séquence « U H I 6 D... » constitue un résultat de codage de type base64. Une fois décodée, cette chaîne devient :

```
Pr: 1
Ac: cn
Hs: 100.100.100.6
Pt: 1234
Sr: -1
Pp: 03001C4F8B0129DA
Ki:
```

On imagine facilement que si certaines parties de cette chaîne de caractères sont susceptibles de changer, comme `100.100.100.6` et `1234` qui indiquent à l'extrémité du tunnel l'adresse de réacheminement des données encapsulées, une partie de la chaîne de caractères est invariante. La détection de « U H I 6 D... », correspondant à `Pr:` en base64 (le codage en base64 utilise 4 octets pour représenter 3 octets), semble donc être une signature permettant de détecter l'utilisation de HTTHost. Afin d'éliminer d'éventuels faux positifs, on pourra rajouter le point d'interrogation indiquant le début de la partie paramètres dans l'URL.

Un outil de détection d'intrusions tel que SNORT [7] permet de programmer ses propres signatures. Sous SNORT, la signature HTTHost s'écrira :

```
alert tcp $HOME_NET any -> $PROXY 8080 (msg:"Possible tunnel HTTHost"; /
                    content:"\??U?H?I?6"; regex; offset: 18; depth:50;)
```

Les versions récentes de SNORT permettent d'utiliser des expressions régulières : « \? » correspond au caractère « ? » (« \ » indique une séquence d'échappement) car « ? » correspond à un caractère quelconque. Les mot-clefs `depth` et `offset` permettent d'optimiser la recherche. Dans l'exemple, celle-ci commence à partir du 18ème caractère et finit au 50ème.

8 Bibliographie

1. Building bastion routers using Cisco IOS par Brett - Phrack Magazine vol 9 issue 55
2. ICMP Packet Filtering par Rob Thomas
   http://www.enteract.com/ robt/Docs/Articles/icmp-messages.html

3. Project Loki par daemon9 - Phrack Magazine vol 7 issue 49
4. Message Bugtraq du 13 Avril 1998 "DNS Tunnel through bastion hosts" par Oskar Pearson
5. Bypassing the Western Residence Napster Prohibition :
   http://stuffclub.org/western.html
6. SSL tunneling and the proxy
   http://developper.netscape.com/docs/manuals/proxy/ProxyUnx/SSL-TUNL.HTM
7. SNORT :
   http://www.snort.org
8. Detecting Backdoors - Yin Zhang, Vern Paxson - Proceedings of the 9th USENIX Security Symposium, Denver, Colorado, Août 2000
9. Mémento sur la sécurité des services web - Bureau conseil de la DCSSI - Octobre 2004
10. Design of a Voice Over IP Systems that circumvent NAT Jem Berkes, Timothy Czyrnyj, Justin Olivier, Dominic Schau March 2004, Universitty of Manitoba, Canada -
    http://www.sysdesign.ca/archive/VoIP_System.pdf
11. Recommandations de sécurtié pour l'applications Skype - n2328/SGDN/DCSSI du 09 août 2005 :
    http://www.circulaires.gouv.fr/pdf/2009/04/cir_1289.pdf

Gestion détaillée du document

23 août 2001 version initiale.

07 octobre 2005 mise à jour concernant les tunnels HTTP, SOAP, ajout d'illustrations pour l'encapsulation, SOAP et HTTPort/Host.

21 mars 2011 Bibliographie, élimination de coquilles.

Annexe

Capture des trames échangées entre HTTPort et HTTHost.

Les acteurs impliqués dans la communication sont les suivants :

client-dicton-du-jour<—>HTTPort(port 9999)<——>proxy(8080)<—>personal HTTHost<—>serveur-client-du-jour(port 1234)

Note : les paquets sont capturés sur le réseau interne, seul le trafic entre la station de l'utilisateur et le relais applicatif du pare-feu est visible. Pour des raisons de clarté, seul le protocole applicatif (HTTP) est représenté.

```
Paquet 1.1: HTTPort -> proxy:8080
47 45 54 20 68 74 74 70 3A 2F 2F 31 30 30 2E 31  GET http://100.1
30 30 2E 31 30 30 2E 35 2F 4F 78 67 6C 6C 78 73  00.100.5/Oxgllxs
3F 67 55 48 48 6C 49 73 36 72 49 71 44 66 45 48  ?gUHHlIs6rIqDfEH
4E 48 43 75 6B 66 46 74 6A 78 4F 53 69 75 42 67  NHCukfFtjxOSiuBg
6A 34 62 49 67 6E 30 34 4B 57 53 52 48 55 4D 2F  j4bIgn04KWSRHUM/
36 69 49 6B 44 61 45 58 77 61 4D 52 43 31 34 44  6iIkDaEXwaMRC14D
78 52 4D 52 44 6C 41 31 75 62 4D 45 54 56 41 45  xRMRDlA1ubMETVAE
77 65 4C 4B 6A 4E 59 4E 4E 4D 43 38 6C 61 42 63  weLKjNYNNMC8laBc
30 4C 4F 67 69 4C 41 6E 78 6A 4D 46 6A 6A 4D 55  0LOgiLAnxjMFjjMU
30 69 44 70 51 33 70 4F 54 32 63 50 6A 77 6F 75  0iDpQ3pOT2cPjwou
67 34 4C 34 54 53 45 64 4E 56 43 4E 6C 68 42 64  g4L4TSEdNVCNlhBd
77 78 4F 4E 69 5A 41 30 77 4D 4D 69 7A 46 41 56  wxONiZA0wMMizFAV
77 73 4D 61 55 79 4D 52 30 6B 52 4E 6A 42 68 51  wsMaUyMR0kRNjBhQ
43 78 4D 38 44 70 45 4E 79 53 4F 65 55 4A 52 69  CxM8DpENySOeUJRi
42 45 44 44 51 48 70 2F 4C 62 61 6F 54 75 6F 46  BEDDQHp/LbaoTuoF
67 37 44 34 51 33 6F 39 4E 72 43 39 67 26 3D 35  g7D4Q3o9NrC9g&=5
3D 20 48 54 54 50 2F 31 2E 31 0D 0A 43 6F 6E 6E  = HTTP/1.1..Conn
65 63 74 69 6F 6E 3A 20 6B 65 65 70 2D 61 6C 69  ection: keep-ali
76 65 0D 0A 43 61 63 68 65 2D 43 6F 6E 74 72 6F  ve..Cache-Contro
6C 3A 20 6E 6F 2D 63 61 63 68 65 0D 0A 50 72 61  l: no-cache..Pra
67 6D 61 3A 20 6E 6F 2D 63 61 63 68 65 0D 0A 55  gma: no-cache..U
73 65 72 2D 41 67 65 6E 74 3A 20 4D 6F 7A 69 6C  ser-Agent: Mozil
6C 61 2F 34 2E 30 20 28 63 6F 6D 70 61 74 69 62  la/4.0 (compatib
6C 65 3B 20 4D 53 49 45 20 35 2E 30 3B 20 57 69  le; MSIE 5.0; Wi
6E 64 6F 77 73 20 39 35 29 0D 0A 48 6F 73 74 3A  ndows 95)..Host:
20 31 30 30 2E 31 30 30 2E 31 30 30 2E 35 0D 0A   100.100.100.5..
0D 0A                                            ..

Paquet 1.2: Proxy:8080 -> HTTHost
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65  .Connection: kee
70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65 2D 43  p-alive..Cache-C
6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65  ontrol: no-cache
0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63 61 63  ..Pragma: no-cac
68 65 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65  he..Content-Type
3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43 6F 6E  : text/html..Con
74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 32 34 34  tent-Length: 244
0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C 68 65 61 64 3E  ....<html><head>
3C 67 68 6C 62 6A 75 67 3E 3C 71 6F 6F 70 6A 77  <ghlbjug><qoopjw
76 3E 3C 66 73 73 77 69 68 68 3E 3C 67 79 6F  v><fsswihxh><gyo
7A 73 65 3E 3C 74 6D 6B 72 79 77 75 3E 3C 6F 62  zse><tmkrywu><ob
68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70 3E 3C  hmpxnl><uybmip><
2F 68 65 61 64 3E 3C 62 6F 64 79 3E 3C 67 68 6C  /head><body><ghl
62 6A 75 67 3E 3C 2F 67 68 6C 62 6A 75 67 3E 3C  bjug></ghlbjug><
71 6F 6F 70 6A 77 76 3E 6F 6B 78 77 76 68 75 65  qoopjwv>okxwvhue
3C 2F 71 6F 6F 70 6A 77 76 3E 3C 66 73 73 77 69  </qoopjwv><fsswi
68 78 68 3E 3C 2F 66 73 73 77 69 68 78 68 3E 3C  hxh></fsswihxh><
67 79 6F 7A 73 65 3E 32 3C 2F 67 79 6F 7A 73 65  gyozse>2</gyozse
3E 3C 74 6D 6B 72 79 77 75 3E 3C 2F 74 6D 6B 72  ><tmkrywu></tmkr
79 77 75 3E 3C 6F 62 68 6D 70 78 6E 6C 3E 3C 2F  ywu><obhmpxnl></
6F 62 68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70  obhmpxnl><uybmip
3E 3C 2F 75 79 62 6D 69 70 3E 3C 2F 62 6F 64 79  ></uybmip></body
3E 3C 2F 68 74 6D 6C 3E                          ></html>

Paquet 1.3: HTTHost -> Proxy:8080
47 45 54 20 68 74 74 70 3A 2F 2F 31 30 30 2E 31  GET http://100.1
30 30 2E 31 30 30 2E 35 2F 56 6D 68 6D 63 6F 3F  00.100.5/Vmhmco?
70 55 6A 48 72 49 53 36 41 49 63 44 5A 45 4E 4E  pUjHrIS6AIcDZENN
71 43 57 6B 5A 46 75 6A 70 4F 76 69 48 42 65 79  qCWkZFujpOviHBey
4F 59 4E 77 6D 30 54 4B 66 53 5A 57 4D 51 48 36  OYNwm0TKfSZWMQH6
31 49 39 47 35 39 59 72 45 65 61 48 53 64 6F 32  1I9G59YrEeaHSdo2
6E 61 39 48 5A 56 34 6C 4D 44 32 51 49 70 31 54  na9HZV4lMD2QIp1T
78 63 31 54 53 6F 63 67 6D 4D 73 51 35 30 39 4B  xc1TSocgmMsQ509K
31 53 42 32 71 6B 32 36 2F 49 6B 41 75 30 38 4B  1SB2qk26/IkAu08K
4C 44 46 51 47 6F 78 3D 20 48 54 54 50 2F 31 2E  LDFQGox= HTTP/1.
31 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B  1..Connection: k
65 65 70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65  eep-alive..Cache
2D 43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63  -Control: no-cac
68 65 0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63  he..Pragma: no-c
61 63 68 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74  ache..User-Agent
3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20 28 63  : Mozilla/4.0 (c
6F 6D 70 61 74 69 62 6C 65 3B 20 4D 53 49 45 20  ompatible; MSIE
35 2E 30 3B 20 57 69 6E 64 6F 77 73 20 39 35 29  5.0; Windows 95)
0D 0A 48 6F 73 74 3A 20 31 30 30 2E 31 30 30 2E  ..Host: 100.100.
```

```
31 30 30 2E 35 0D 0A 0D 0A                        100.5....

Paquet 1.4: Proxy:8080 -> HTTHost
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65  .Connection: kee
70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65 2D 43  p-alive..Cache-C
6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65  ontrol: no-cache
0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63 61 63  ..Pragma: no-cac
68 65 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65  he..Content-Type
3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43 6F 6E  : text/html..Con
74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 32 34 34  tent-Length: 244
0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C 68 65 61 64 3E  ....<html><head>
3C 67 68 6C 62 6A 75 67 3E 3C 71 6F 6F 70 6A 77  <ghlbjug><qoopjw
76 3E 3C 66 73 73 77 69 68 78 68 3E 3C 67 79 6F  v><fsswihxh><gyo
7A 73 65 3E 3C 74 6D 6B 72 79 77 75 3E 3C 6F 62  zse><tmkrywu><ob
68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70 3E 3C  hmpxnl><uybmip><
2F 68 65 61 64 3E 3C 62 6F 64 79 3E 3C 67 68 6C  /head><body><ghl
62 6A 75 67 3E 3C 2F 67 68 6C 62 6A 75 67 3E 3C  bjug></ghlbjug><
71 6F 6F 70 6A 77 76 3E 6F 6B 78 77 76 68 75 65  qoopjwv>okxwvhue
3C 2F 71 6F 6F 70 6A 77 76 3E 3C 66 73 73 77 69  </qoopjwv><fsswi
68 78 68 3E 3C 2F 66 73 73 77 69 68 78 68 3E 3C  hxh></fsswihxh><
67 79 6F 7A 73 65 3E 34 3C 2F 67 79 6F 7A 73 65  gyozse>4</gyozse
3E 3C 74 6D 6B 72 79 77 75 3E 3C 2F 74 6D 6B 72  ><tmkrywu></tmkr
79 77 75 3E 3C 6F 62 68 6D 70 78 6E 6C 3E 3C 2F  ywu><obhmpxnl></
6F 62 68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70  obhmpxnl><uybmip
3E 3C 2F 75 79 62 6D 69 70 3E 3C 2F 62 6F 64 79  ></uybmip></body
3E 3C 2F 68 74 6D 6C 3E                          ></html>

Paquet 1.5: HTTHost -> Proxy:8080
47 45 54 20 68 74 74 70 3A 2F 2F 31 30 30 2E 31  GET http://100.1
30 30 2E 31 30 30 2E 35 2F 56 64 62 67 7A 3F 6D  00.100.5/Vdbgz?m
55 71 48 7A 49 75 36 51 49 57 44 38 45 59 4E 65  UqHzIu6QIWD8EYNe
43 76 6B 6A 46 56 6A 5A 4F 4F 69 2F 42 74 7A 51  CvkjFVjZOOi/BtzQ
62 39 67 62 30 77 4B 75 53 45 57 59 51 76 36 62  b9gb0wKuSEWYQv6b
49 6F 47 6E 39 26 72 4E 65 69 48 46 64 46 32 30  IoGn9&rNeiHFdF20
61 5A 48 33 56 7A 6C 49 44 74 51 2F 70 38 46 7A  aZH3VzlIDtQ/p8Fz
62 4B 6A 75 6F 47 67 59 56 43 58 35 68 58 68 41  bKjuoGgYVCX5hXhA
65 6F 48 33 49 54 4E 4D 43 53 6C 72 4E 6D 78 6C  eoH3ITNMCSlrNmxl
4F 32 69 5A 41 57 79 62 44 44 51 59 70 38 4C 4B  O2iZAWybDDQYp8LK
61 67 54 5A 6F 30 67 69 44 52 51 66 6F 79 4E 49  agTZo0giDRQfoyNI
43 41 67 54 3D 62 3D 20 48 54 54 50 2F 31 2E 31  CAgT=b= HTTP/1.1
0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65  ..Connection: ke
65 70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65 2D  ep-alive..Cache-
43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68  Control: no-cach
65 0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63 61  e..Pragma: no-ca
63 68 65 0D 0A 55 73 65 72 2D 41 67 65 6E 74 3A  che..User-Agent:
20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20 28 63 6F   Mozilla/4.0 (co
6D 70 61 74 69 62 6C 65 3B 20 4D 53 49 45 20 35  mpatible; MSIE 5
2E 30 3B 20 57 69 6E 64 6F 77 73 20 39 35 29 0D  .0; Windows 95).
0A 55 78 61 78 72 3A 20 55 6D 39 7A 5A 54 30 3D  .Uxaxr: Um9zZT0=
0D 0A 48 6F 73 74 3A 20 31 30 30 2E 31 30 30 2E  ..Host: 100.100.
31 30 30 2E 35 0D 0A 0D 0A                       100.5....

Paquet 1.6: Proxy:8080 -> HTTHost
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65  .Connection: kee
70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65 2D 43  p-alive..Cache-C
6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65  ontrol: no-cache
0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63 61 63  ..Pragma: no-cac
68 65 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65  he..Content-Type
3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43 6F 6E  : text/html..Con
74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 32 34 34  tent-Length: 244
0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C 68 65 61 64 3E  ....<html><head>
3C 67 68 6C 62 6A 75 67 3E 3C 71 6F 6F 70 6A 77  <ghlbjug><qoopjw
76 3E 3C 66 73 73 77 69 68 78 68 3E 3C 67 79 6F  v><fsswihxh><gyo
7A 73 65 3E 3C 74 6D 6B 72 79 77 75 3E 3C 6F 62  zse><tmkrywu><ob
68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70 3E 3C  hmpxnl><uybmip><
2F 68 65 61 64 3E 3C 62 6F 64 79 3E 3C 67 68 6C  /head><body><ghl
62 6A 75 67 3E 3C 2F 67 68 6C 62 6A 75 67 3E 3C  bjug></ghlbjug><
71 6F 6F 70 6A 77 76 3E 6F 6B 78 77 76 68 75 65  qoopjwv>okxwvhue
3C 2F 71 6F 6F 70 6A 77 76 3E 3C 66 73 73 77 69  </qoopjwv><fsswi
68 78 68 3E 3C 2F 66 73 73 77 69 68 78 68 3E 3C  hxh></fsswihxh><
67 79 6F 7A 73 65 3E 32 3C 2F 67 79 6F 7A 73 65  gyozse>2</gyozse
3E 3C 74 6D 6B 72 79 77 75 3E 3C 2F 74 6D 6B 72  ><tmkrywu></tmkr
79 77 75 3E 3C 6F 62 68 6D 70 78 6E 6C 3E 3C 2F  ywu><obhmpxnl></
6F 62 68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70  obhmpxnl><uybmip
3E 3C 2F 75 79 62 6D 69 70 3E 3C 2F 62 6F 64 79  ></uybmip></body
3E 3C 2F 68 74 6D 6C 3E                          ></html>

Paquet 1.7: HTTHost -> Proxy:8080
47 45 54 20 68 74 74 70 3A 2F 2F 31 30 30 2E 31  GET http://100.1
30 30 2E 31 30 30 2E 35 2F 47 78 78 6B 64 64 66  00.100.5/Gxxkddf
3F 2F 55 32 48 52 49 39 36 32 49 65 44 38 45 70  ?/U2HRI962IeD8Ep
4E 69 43 4C 6B 44 46 6C 6A 43 4F 32 69 79 42 34  NiCLkDFljCO2iyB4
79 74 59 2F 77 63 30 65 4B 6A 53 74 57 6A 51 64  ytY/wc0eKjStWjQd
36 48 49 6C 47 71 39 5A 72 2F 65 33 48 41 64 4D  6HIlGq9Zr/e3HAdM
32 76 61 2F 48 31 56 30 6C 67 44 6D 51 38 70 43  2va/H1V0lgDmQ8pC
```

```
54 75 63 56 54 2F 6F 72 67 2F 4D 65 77 61 30 58  TucVT/org/Mewa0X
4B 65 53 75 32 44 6B 6A 36 31 49 41 41 42 30 65  KeSu2Dkj61IAAB0e
4B 63 44 46 51 26 6F 48 3D 20 48 54 54 50 2F 31  KcDFQ&oH= HTTP/1
2E 31 0D 0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20  .1..Connection:
6B 65 65 70 2D 61 6C 69 76 65 0D 0A 43 61 63 68  keep-alive..Cach
65 2D 43 6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61  e-Control: no-ca
63 68 65 0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D  che..Pragma: no-
63 61 63 68 65 0D 0A 55 73 65 72 2D 41 67 65 6E  cache..User-Agen
74 3A 20 4D 6F 7A 69 6C 6C 61 2F 34 2E 30 20 28  t: Mozilla/4.0 (
63 6F 6D 70 61 74 69 62 6C 65 3B 20 4D 53 49 45  compatible; MSIE
20 35 2E 30 3B 20 57 69 6E 64 6F 77 73 20 39 35   5.0; Windows 95
29 0D 0A 48 6F 73 74 3A 20 31 30 30 2E 31 30 30  )..Host: 100.100
2E 31 30 30 2E 35 0D 0A 0D 0A                    .100.5....
```

Paquet 1.8: Proxy:8080 -> HTTHost

```
48 54 54 50 2F 31 2E 31 20 32 30 30 20 4F 4B 0D  HTTP/1.1 200 OK.
0A 43 6F 6E 6E 65 63 74 69 6F 6E 3A 20 6B 65 65  .Connection: kee
70 2D 61 6C 69 76 65 0D 0A 43 61 63 68 65 2D 43  p-alive..Cache-C
6F 6E 74 72 6F 6C 3A 20 6E 6F 2D 63 61 63 68 65  ontrol: no-cache
0D 0A 50 72 61 67 6D 61 3A 20 6E 6F 2D 63 61 63  ..Pragma: no-cac
68 65 0D 0A 43 6F 6E 74 65 6E 74 2D 54 79 70 65  he..Content-Type
3A 20 74 65 78 74 2F 68 74 6D 6C 0D 0A 43 6F 6E  : text/html..Con
74 65 6E 74 2D 4C 65 6E 67 74 68 3A 20 32 38 38  tent-Length: 288
0D 0A 0D 0A 3C 68 74 6D 6C 3E 3C 68 65 61 64 3E  ....<html><head>
3C 67 68 6C 62 6A 75 67 3E 3C 71 6F 6F 70 6A 77  <ghlbjug><qoopjw
76 3E 3C 66 73 73 77 69 68 78 68 3E 3C 67 79 6F  v><fsswihxh><gyo
7A 73 65 3E 3C 74 6D 6B 72 79 77 75 3E 3C 6F 62  zse><tmkrywu><ob
68 6D 70 78 6E 6C 3E 3C 75 79 62 6D 69 70 3E 3C  hmpxnl><uybmip><
2F 68 65 61 64 3E 3C 62 6F 64 79 3E 3C 67 68 6C  /head><body><ghl
62 6A 75 67 3E 3C 2F 67 68 6C 62 6A 75 67 3E 3C  bjug></ghlbjug><
71 6F 6F 70 6A 77 76 3E 6F 6B 78 77 76 68 75 65  qoopjwv>okxwvhue
3C 2F 71 6F 6F 70 6A 77 76 3E 3C 66 73 73 77 69  </qoopjwv><fsswi
68 78 68 3E 51 53 42 73 59 53 42 7A 59 57 6C 75  hxh>QSBsYSBzYWlu
64 47 55 67 55 6D 39 7A 5A 53 77 67 63 47 46 7A  dGUgUm9zZSwgcGFz
49 47 52 6C 49 48 42 68 64 58 4E 6C 4C 67 6F 3D  IGRlIHBhdXNlLgo=
3C 2F 66 73 73 77 69 68 78 68 3E 3C 67 79 6F 7A  </fsswihxh><gyoz
73 65 3E 31 3C 2F 67 79 6F 7A 73 65 3E 3C 74 6D  se>1</gyozse><tm
6B 72 79 77 75 3E 3C 2F 74 6D 6B 72 79 77 75 3E  krywu></tmkrywu>
3C 6F 62 68 6D 70 78 6E 6C 3E 3C 2F 6F 62 68 6D  <obhmpxnl></obhm
70 78 6E 6C 3E 3C 75 79 62 6D 69 70 3E 3C 2F 75  pxnl><uybmip></u
79 62 6D 69 70 3E 3C 2F 62 6F 64 79 3E 3C 2F 68  ybmip></body></h
74 6D 6C 3E                                      tml>
```