# Prototype de canal caché dans le DNS

**Lucas Nussbaum et Olivier Richard**

*Laboratoire d'Informatique de Grenoble - projet MESCAL*
*Antenne de Montbonnot*
*ZIRST 51, avenue Jean Kuntzmann*
*38330 Montbonnot Saint Martin*
*{lucas.nussbaum,olivier.richard}@imag.fr*

RÉSUMÉ. *Avec la multiplication des réseaux ne permettant qu'un accès restreint à Internet, de nombreuses techniques ont vu le jour pour s'évader de ces restrictions. Les canaux cachés dans le DNS sont l'une d'elles, consistant à utiliser des requêtes et des réponses DNS pour transmettre et recevoir des informations via un serveur DNS complice. Après une présentation des différentes solutions existantes, nous présentons notre prototype, TUNS, et nous l'évaluons en le comparant aux autres implantations.*

ABSTRACT. *With more and more networks providing a restricted access to the Internet, numerous solutions have been developed to evade such networks. Covert channels inside DNS, which leverage DNS requests and replies to transmit and receive data using a rogue DNS server, are one of them. After describing the varous existing solutions, we introduce our own prototype, TUNS, and evaluate it by comparing it to other implementations.*

MOTS-CLÉS : *canal caché, DNS, mesure de performances*

KEYWORDS: *covert channel, DNS, performance evaluation*

## 1. Introduction

Avec la multiplication des réseaux ne permettant qu'un accès partiel à Internet (réseaux d'entreprises, d'hôtels, de restaurants), de nombreuses personnes ont essayé d'utiliser les quelques protocoles disponibles pour obtenir un accès complet à Internet. Cela se fait en général en transmettant les données en utilisant un protocole autorisé, vers un service complice situé ailleurs sur Internet, sur un réseau non limité.

Ainsi, il existe des implantations de tunnels utilisant les protocoles HTTP, HTTPS, ou ICMP. Dans ce poster, nous nous intéressons à l'utilisation du protocole DNS dans ce cadre.

Le protocole DNS est disponible sur de nombreux réseaux restreints, car il est nécessaire pour permettre d'utiliser la plupart des autres protocoles. Seuls les réseaux ne permettant qu'un accès au protocole HTTP peuvent se permettre de ne pas offrir un accès au protocole DNS : dans ce cas, la résolution DNS se fait sur le serveur proxy.

Toutefois, l'encapsulation de données dans des requêtes et des réponses DNS est difficile. Dans les requêtes, la seule possibilité est d'encoder les données dans le nom à résoudre, en codant les données sous une forme textuelle (Base32, ou Base64, en prenant quelques libertés avec la RFC 1035). Dans les réponses, les enregistrements TXT et NULL permettent d'envoyer de grosses quantités d'informations, mais ils sont souvent filtrés par les serveurs DNS. Une extension du protocole DNS définie dans la RFC 2671 permet aussi d'augmenter la taille des réponses.

Dans la suite de cet article, nous présenterons quelques implantations existantes de canaux cachés utilisant le DNS. Puis nous présenterons notre prototype, TUNS, ainsi que les résultats d'évaluations comparant TUNS aux autres implantations.

## 2. Implantations existantes

On peut distinguer deux catégories d'inplantations :

– Les solutions qui fournissent une connectivité au niveau IP (tunnels IP over DNS) ;

– Les solutions qui fournissent un seul canal de communication, permettant par exemple d'établir une connexion SSH.

Les tunnels fournissant une connectivité au niveau IP fournissent un périphérique `tun` ou `tap`, permettant à l'utilisateur de router les paquets souhaités à travers le tunnel. Leur utilisation est donc totalement transparente pour les applications.

NSTX [nst] est historiquement le plus ancien. Pour encoder les données dans les requêtes, il utilise un encodage Base64 non-conforme à la RFC 1035 (il utilise des caractères "_" en plus des 63 caractères autorisés par la RFC. Les réponses sont contenues dans des paquets TXT.

Iodine [iod] est un projet plus récent. Il utilise (au choix) un encodage Base32 ou Base64 pour encoder les données dans les requêtes, et des paquets de type NULL pour les réponses. Il utilise également EDNS0 pour augmenter la taille des réponses.

À la fois NSTX et Iodine découpent les paquets IP à envoyer en plusieurs paquets DNS, les envoyent séparément, et recomposent les paquets IP côté serveur.

La deuxième catégorie de tunnels ne fournit qu'une connexion TCP. L'utilisateur établit en général une connexion SSH, puis utilise les fonctionnalités de *port forwarding* et de proxy SOCKS de SSH.

La difficulté de ces solutions est qu'elles doivent fournir un canal fiable au dessus d'un protocole non fiable, et donc gérer les pertes, réordonnancements et duplications de paquets DNS.

OzymanDNS [ozy] est la solution la plus utilisée. Il utilise des enregistrements TXT, et l'extension EDNS0. Nous avons constaté des plantages très fréquents lors de nos tests.

dns2tcp [dns] est une autre solution. Il utilise des enregistrements TXT et un encodage Base64 non-conforme (utilisation de '/').

## 3. TUNS

Après une étude des solutions existantes, nous avons proposé notre propre solution, avec pour objectif de se restreindre à un fonctionnement le plus standard possible (et donc le plus difficile à empêcher). TUNS est un tunnel IP over DNS, écrit en Ruby. Contrairement aux autres solutions, qui utilisent des enregistrements TXT ou NULL, rarement utilisés de manière légitime, TUNS n'utilise que des enregistrements CNAME. Pour encoder les données émises côté client et serveur, un encodage Base32 est utilisé. Contrairement à NSTX et Iodine, TUNS ne découpe pas les paquets IP entre plusieurs paquets DNS plus petits : à la place, le MTU de l'interface du tunnel est réduit, et c'est donc le système d'exploitation qui se charge du découpage en utilisant la fragmentation IP.

Lorsque des données à transmettre sont reçues par TUNS côté client, elles sont immédiatement transmises dans une requête DNS.

Pour recevoir des données du serveur, le client sonde régulièrement le serveur avec des requêtes quasi-vides. Le serveur répond immédiatement à ces requêtes si des données à transmettre au client sont disponibles. Si ce n'est pas le cas, il attend quelques dixièmes de seconde, au cas où des données à transmettre arriveraient pendant cette attente.

## 4. Évaluation

Nous avons évalué Iodine, NSTX et TUNS.

Tout d'abord, nous avons vérifié le bon fonctionnement des trois solutions sur différents réseaux. Il est apparu que Iodine et NSTX n'étaient pas utilisable sur de nombreux réseaux, probablement à cause des libertés prises par rapport à la RFC 1035. Par contre, nous n'avons pas trouvé de réseau sur lequel TUNS ne fonctionnait pas.

Dans un deuxième temps, nous nous sommes intéressés aux performances de ces 3 outils. Nous avons réalisé des expériences à l'aide d'un émulateur réseau, permettant de modifier la latence entre le client et le serveur. Trois machines de Grid'5000 ont été utilisées avec l'émulateur TC+Netem inclu dans Linux.

Nous avons d'abord mesuré le *Round Trip Time* à l'aide de *pings* envoyés à travers le tunnel, initiés par le serveur. Si NSTX et TUNS ne modifient pas significativement la latence (la latence mesurée à travers le tunnel est proche de la latence physique), on constate (figure 1) que Iodine provoque une latence bien plus importante. Cela est probablement dû à l'algorithme de sondage utilisé par le client.

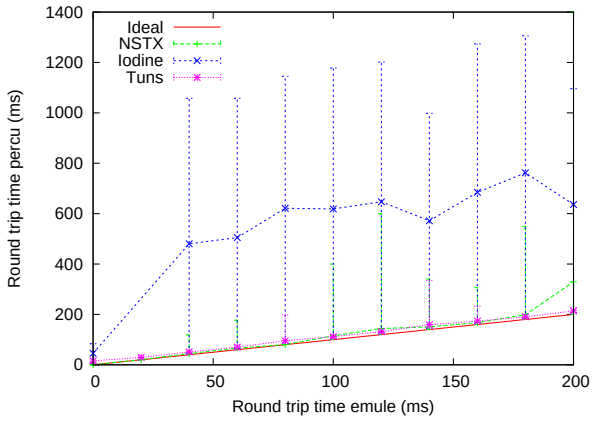


**Figure 1.** *Round Trip Time mesuré à travers tunnel, comparé au RTT réel entre le client et le serveur*

Puis nous nous sommes intéressés au débit maximal disponible à travers le tunnel. Encore une fois, nous l'avons mesuré en émulant une latence plus importante entre le client et le serveur. À l'aide de l'émulateur, nous avons également émulé 2% de pertes de paquets. Le comportement des tunnels face à des pertes de paquets est important, puisqu'ils sont souvent utilisés sur des réseaux WiFi de mauvaise qualité. Nous constatons (figure 2) que les performances de TUNS sont inférieures à celles de Iodine et NSTX. Mais face à des pertes de paquets, TUNS est moins affecté que les deux autres solutions, probablement car il ne découpe pas les paquets IP en plusieurs paquets DNS (la perte d'un paquet DNS ayant alors des conséquences plus importantes).

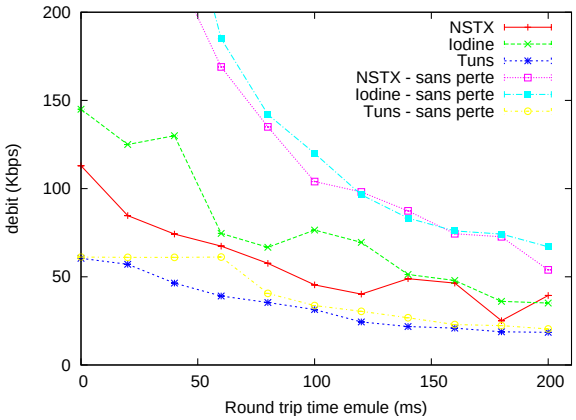


**Figure 2.** *Débit montant (client vers serveur) disponible à travers le tunnel, en fonction du RTT réel entre le client et serveur*

## 5. Conclusion

Après une présentation des différentes solutions existantes permettant de réaliser un canal d'informations caché dans des requêtes DNS, nous avons proposé notre propre prototype, TUNS, conçu dans l'objectif d'être le plus conforme possible au protocole DNS, et nous l'avons évalué en le comparant aux autres implantations existantes.

En terme de latence, nous avons constaté que TUNS proposait des performances équivalentes à celles des autres solutions. Toutefois, en terme de débit, ses performances sont moins bonnes. Cela s'explique, d'une part, par l'absence de découpage des paquets IP : l'utilisation de la fragmentation diminue la part utile de données dans les paquets DNS. D'autre part, nous avons constaté une importante utilisation du processeur dans nos expériences : il semblerait que l'implantation en utilisant un langage de script, en particulier de la bibliothèque gérant la construction et le décodage des paquets DNS, est assez peu performante.

## 6. Bibliographie

[dns] « Dns2tcp », `http://hsc.fr/ressources/outils/dns2tcp/`.

[iod] « Iodine », `http://code.kryo.se/iodine/`.

[nst] « NSTX », `http://thomer.com/howtos/nstx.html`.

[ozy] « OzymanDNS », `http://www.doxpara.com/`.