The sys admin's daily grind: Httptunnel

PIERCED WALLS

Just a couple of hours after completing this article, Charly headed off on vacation. Before he left, he indulged in a spot of piercing to help him work around the paranoid firewalls waiting for him in the Internet cafes at his holiday location.

BY CHARLY KÜHNAST

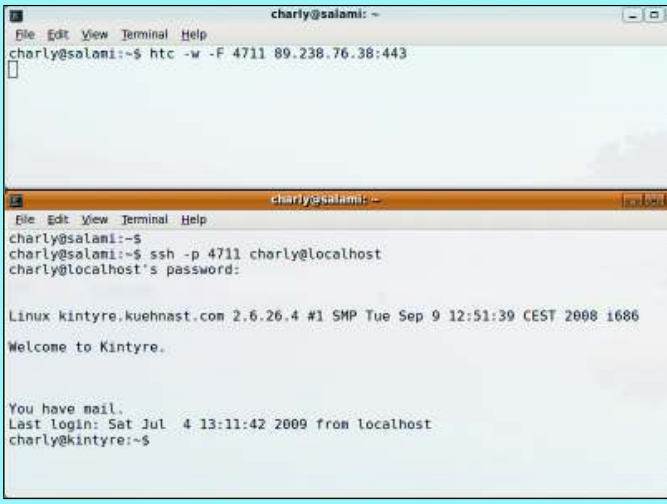

Figure 1: Charly tunneling through firewalls on vacation in France, just to report on the jellyfish-throwing contest on IRC

As a country boy, the first time I saw body piercing was in the nose of my grandfather's prize bull, long before people started to disfigure their faces and secondary sexual organs with bits of metal. Firewall piercing – setup tricks that route arbitrary TCP traffic through an existing hole such as HTTP(S) – started to become popular in the epoch between rings in bulls' noses and perforated humans, or about the time SUSE 5.3 was released.

Httptunnel [1], which I will be using on vacation, dates back to the same period. Although today, admins could replace the tool with just a couple of iptables lines, it has always been more user friendly, and it is available out of the box with most distributions.

A journey of approximately 12 hours will take me to Occitania [2], an area full of friendly people, beautiful landscapes, and Internet cafes, in which network access means strictly HTTP. Unfortunately, I was planning to publish the events of the international jellyfish-throwing contest, which is held in Occitania on IRC; in other words, I need SSH.

Httptunnel comprises the client and its target. The client, HTC, listens on any port; it will have to be a port above 1024 if you only have normal user privileges. The client encapsulates incoming connections in HTTP and sends them to its target, a port supported by the firewall. If you want to try this with port 443, the command line is:

```
htc -w -F 4711 ➥
kintyre.kuehnast.com:443
```

When you try this out, it makes sense to set the *-w* parameter, which keeps the process from running in the background as a daemon. As soon as I get everything working, I'll leave out this parameter. The *-F* stands for "Forward," and it is followed by the source and target for forwarding. Because the source is a port on my localhost, there is no need to specify a hostname, just a port. The target can be an IP address or a hostname, as shown in the example.

Back to Front

The whole thing needs to be back to front on the server. The HTS server component accepts incoming connections on port 443, strips off the HTTP layer, and passes them on to the configured target port. In the example here, this is port 22; after all, I need an SSH login:

```
hts -w -F localhost:22 443
```

Although this might not seem logical, I need to configure the target port 22 first, followed by the port that HTS uses to listen for incoming connections. To try it out, I'll open an SSH connection for localhost, port 4711:

```
ssh -p 4711 charly@localhost
```

Now I can log in to the server (Figure 1). Before you start to protest, the jellyfish-throwing contest was just a joke, but now that the vintners in Languedoc have stopped producing industrial alcohol and starting making really good wines, I'm sure you will all be really jealous. ■

INFO

[1] Httptunnel: *http://www.nocrew.org/software/httptunnel.html*

[2] Occitania: *http://en.wikipedia.org/wiki/Occitania*

THE AUTHOR

Charly Kühnast is a Unix operating system administrator at the Data Center in Moers, Germany. His tasks include firewall and DMZ security and availability. He divides his leisure time into hot, wet, and eastern sectors, where he enjoys cooking, fresh water aquariums, and learning Japanese, respectively.