第四章

網路

曾有一段時間，網管人員的主要職責，是想辦法將各種機器連接在一起，使它們能夠透過網路互通。然而，時過境遷，目前大多數網管人員最頭痛的問題，反而是如何適當限制網路存取 — 既要保障合法使用者的權益，又可摒除非必要的干擾。

幸好，Linux 核心內建的 Netfilter 防火牆【譯註】，提供了一個非常有彈性的操作介面 — `iptables` — 讓我們得以影響核心對網路封包的處理方式。藉由 `iptables` 來設置防火牆規則，我們可建構出相當細膩而且靈活的存取政策。Netfilter 不僅能依據通訊埠、介面、MAC 位址來過濾封包，就連封包所含的內容、甚至是收到封包的速率，全都可以成為過濾條件。這表示我們可用 Netfilter 來掃除各種攻擊企圖，包括 port floods（從遠端填滿你所有可用的通訊埠，使你無法發出連線）、病毒 ... 等等手法。

將機器連接在一起，會讓你有成就感；但是將「人」（可能是合法的使用者，也可能是罪大惡極的黑客）鎖在門外，可就沒那麼好玩了，畢竟電腦網路的原本初衷不是為了增加隔閡，而是為了讓人們能方便溝通。我們將看到一些控制網路交通的**不尋常**方法，包括「遠埠轉接」（remote port forwarding），以及各種形式的「IP 通道」（IP tunnelling）。在我們探討了 IP **封裝**（IP encapsulation）

譯註　Netfilter 是內建於 Linux 2.4 版核心的封包過濾機制，它不是獨立的套件，而是整合在核心內部的模組。`iptables` 是用來設置 Netfilter 作業參數的工具，真正執行過濾功能的是 Netfilter；不過，作者並不刻意強調這兩者的差異，請讀者注意。

與 vtun 之類的使用者空間通道（user space tunnels）【譯註 1】之後，將會示範如何把我們的網路搭在 Internet 之上，作出一些意想不到的驚奇效果。

HACK #45 簡易防火牆

使用 iptables 來享受專用防火牆的好處

內建於 Linux 2.4 版核心中的 Netfilter，是非常有彈性的防火牆機制。我們可在命令列以 iptables 直接調整、設置 Netfilter 的各種過濾條件。iptables 的語法有點囉唆，初次接觸的人不太容易上手，但也正因為如此，使得我們能用 iptables 建構出相當複雜（但願有用）的防火牆規則【譯註 2】。

即使你的機器不是「真正」的防火牆（也就是說，它只有一個網路介面，而且不是用來保護其它機器），**封包過濾**（packet filtering）的功能也非常有用。假設你打算讓同事能用 telnet（或 ssh）來存取你的機器，但不打算讓整個 Internet 上的人都能這樣做，那麼，你可以使用 tcpwrapper 來包裝你的 telnet daemon（在 /etc/hosts.allow 與 /etc/hosts.deny 設定適當的存取權限與服務對象）。或者，使用 iptables 也可以達成類似的效果：

```
# iptables -A INPUT -t filter -s ! 208.201.29.36 -p tcp --dport \
>   23 -j DROP
```

一般而言，最多人採用的防護策略，是只開放必要的公共服務，而只有少數幾台可信任的主機能享用全部的服務，對於曾經有不良記錄的主機，則完全隔絕。這裡有一個方法，讓你能同時使用白名單、黑名單、受限通訊埠：

```
#!/bin/sh
#
# 簡易的防火牆設置
#
WHITELIST=/usr/local/etc/whitelist.txt
BLACKLIST=/usr/local/etc/blacklist.txt
ALLOWED="22 25 80 443"

#
```

譯註 1　Linux 將整個記憶空間分成 kernel space（核心空間）與 user space（使用者空間），前者專供核心本身使用，後者用於一般應用程式。一般而言，涉及 IP 封裝的動作，都是在核心進行的，但是 vtun 卻能在 user space 來封裝 IP 封包，所以稱之為「使用者空間通道」（好詭異的名詞!!!）。

譯註 2　身為網路管理者的你，應該已經熟知 Netfilter 的架構與 iptables 的語法，因此作者在這方面並未著墨。不過，我相信並非每位讀者都熟悉 Netfilter，因此我在《附錄 A》整理了一些入門資料。如果你發現看不懂作者在 iptables 命令列玩什麼把戲，那表示你應該先翻閱《附錄 A》。

```
# 清掉所有現存的過濾規則
#
iptables -F

#
# 首先，批准 $WHITELIST 所列的主機與網路
#
for x in `grep -v ^# $WHITELIST | awk '{print $1}'`; do
   echo "Permitting $x..."
   iptables -A INPUT -t filter -s $x -j ACCEPT
done

#
# 阻隔 $BLACKLIST 所列的主機與網路
#
for x in `grep -v ^# $BLACKLIST | awk '{print $1}'`; do
   echo "Blocking $x..."
   iptables -A INPUT -t filter -s $x -j DROP
done

#
# 可有限度開放的通訊埠。
# 只要沒列在黑名單裡，就可使用這些通訊埠
#
for port in $ALLOWED; do
   echo "Accepting port $port..."
   iptables -A INPUT -t filter -p tcp --dport $port -j ACCEPT
done

#
# 任何不符合上述條件的存取企圖，必定是外來的連線嘗試【譯註】，
# 所以一律丟棄。
#
iptables -A INPUT -t filter -p tcp --syn -j DROP
```

請確定所有你打算開放的通訊埠都已納入 `$ALLOWED` 變數，如果你忘記 22，就無法使用 `ssh` 連線到此機器！

/usr/local/etc/blacklist.txt 檔案記載了 IP 位址、主機名稱、網路編號，像這樣：

```
1.2.3.4                    # Portscanned on 8/15/02
7.8.9.0/24                 # Who knows what evil lurks therein
r00tb0y.script-kiddie.coop # $0 s0rR33 u 31337 h4x0r!
```

譯註　要搭建一條 TCP 連線，必須經過三段步驟，而送出 SYN 封包是第一步。詳情請參考《TCP/IP 網路管理》第一章。

同樣地，/usr/local/etc/whitelist.txt 記載了可以不受其它規則限制的「好人」：

```
11.22.33.44         # My workstation
208.201.239.0/26    # the local network
```

由於我們只取用開頭不是 # 符號的列，所以你可以註解一整列，當你再次執行此指令稿時，所有註解都會被忽略，而且先前設定的過濾條件也會被清除（因為指令稿一開始就執行 `iptables -F`），所以，每當你修改了 blacklist.txt 或 whitelist.txt 檔案，或是在 `$ALLOWED` 變數指定其它通訊埠，只要重新執行一次本指令稿即可。

請注意到這個指令稿只處理 TCP 連線，如果你想要支援 UDP、ICMP 或其它傳輸層協定，請參考處理 `$ALLOWED` 的那個迴圈（記得將 `iptables` 的 `-p` 參數改成 `udp` 或 `icmp`）。

請務必小心使用白名單，任何出現在此名單的 IP 或網路，都可以任意存取你的機器。在某些狀況下，狡滑的壞蛋可能會送出偽裝封包，假冒自己來自這些 IP；如果被這些傢伙事先知道（或邏輯推論）你的白名單內容，就很難防範這方面的攻擊。當然，在實務上，要行使這樣的攻擊相當困難，但是理論上並非完全不可能。如果你感到疑慮，白名單裡就不應該出現 Internet routable 位址，只出現內部網路使用的 non-routable 位址【譯註】。

在設立新的防火牆規則時，最好能夠接觸到操控台（實際連接到主機的鍵盤與螢幕），以免遇到自己將自己阻擋在外的情況。當你在操作 `iptables` 時，如果忽略了自己所在的位置，你可以用 `iptables -L` 列出所有規則，或是用 `iptables -F` 重頭開始。如果 `iptables -L` 似乎當掉，建議試著使用 `iptables -L -n` 來顯示「數字形式」的規則（這可以避免使用到 DNS server）。

簡單的封包過濾功能，就已經妙用無窮。但是 `iptables` 的能耐還不只如此，好戲還在後頭。

相關參考：

- Netfilter HOWTO（位於 www.Netfilter.org）
- 《#47 iptables 的訣竅與技巧》

譯註　RFC 1918 所列的位址群（10.0.0.0/8、172.16.0.0/12、192.168.0.0/16）稱為 non-routable IP，又俗稱為「私有位址」（privte address）或「虛擬位址」（virtual address）。如果封包的 srcIP/destIP 在這些範圍內，Internet 閘道器將不予以轉送。

HACK #46 簡單的 IP 轉址

在十秒內設定好 NAT 閘道

如果你有一個私有網路，而且此網路上的所有主機必須共用同一條 Internet 連線（一個公開的 IP 位址），則必須在閘道器機器上設置 IP 轉址（IP Masquerading），而這只需要兩道命令就可以搞定了：

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
# iptables -t nat -A POSTROUTING -o $EXT_IFACE -j MASQUERADE
```

其中 `$EXT_IFACE` 是閘道器的連外界面。現在，任何位居於私有網路上的主機，都可以將它們的**預設路徑**（default route）設定成閘道器的連內界面（這就是閘道器必須啟動 IP Forwarding 的原因），當閘道器從內界網路收到要送往 Internet 的封包，就會自動將封包的來源位址改成 `$EXT_IFACE` 的位址；對 Internet 而言，經過轉換的封包，就像是閘道器本身發出的封包一樣，現在你知道為何這種轉址技術稱為 IP "轉址"。

使用 NAT 並非完全沒有顧慮，想像一下，如果外界網路對你的閘道器送出「來源位址為內部網路」的封包，則核心也會轉換這些封包的位址，"漂白"（有點類似「洗錢」的味道）成你的閘道器所發出的合法封包。換言之，你的 NAT 閘道器將成為不法之徒的跳板，而他們的罪行將嫁禍到無辜的你。

不過，這樣的問題已經過去了，因為新版核心免費提供你一組專門處理此問題的防火牆規則，稱為 `rp_filter`。這組規則促使核心檢查網路介面所收到的封包，看看其來源位址是否相符對應的路由表項目（比方說，從 eth0 收到應該只有 eth1 才可能收到的封包），如果不符，則予以丟棄。如此便能有效解決 IP 冒充的問題，避免我們的轉址功能被誤用。

如果 `rp_filter` 會對你造成問題（儘管不太可能），你可以輕易地取消它：

```
# echo "0" > /proc/sys/net/ipv4/conf/all/rp_filter
```

相關參考：

- 《#47 iptables 的訣竅與技巧》

HACK #47 iptables 的訣竅與技巧

你的防火牆不是只能過濾封包而已

`iptables` 是 Netfilter 的第二代工具，它不僅延續了前一代（`ipchains`）的所有功能性，還額外支援了**狀態隔絕**（stateful firewalling），甚至提供了一套架構（framework，模組的 API），讓管理者可外掛模組來擴增新功能。這裡有一些可用`iptables` 基本功能辦到的事，以及一些好用的 `iptables` 外掛模組。

為了方便舉例說明，我們假設系統已經設置了下列環境變數：

$EXT_IFACE

防火牆的對外（公開）介面。

$INT_IFACE

防火牆的對內（私有）介面。

$DEST_IP

封包的最終目的地。

你可以使用 iptables 限制外來的 TCP 封包之流入速率，避免遭受**癱瘓服務**（Denial of Service，DoS）攻擊：

```
# 製作一個 syn-flood chain，用於偵測 DoS 攻擊
iptables -t nat -N syn-flood

# 限制每秒 12 次連線（在瞬間到達每秒 24 次之後）
iptables -t nat -A syn-flood -m limit --limit 12/s --limit-burst \
   24 -j RETURN
iptables -t nat -A syn-flood -j DROP

# 檢查是否受到 DoS 攻擊
iptables -t nat -A PREROUTING -i $EXT_IFACE -d $DEST_IP -p tcp \
    --syn -j syn-flood
```

這些規則檢查外來 TCP 連線的增長速率，如果在一秒之內暴增了超過 24 條新連線（在 1 秒之內連續收到 24 個 SYN 封包【譯註】），則將新連線的增長速率限制在每秒 12 次以下。

藉由 iptables 之助，我們還可以設置一個透明的 Squid proxy，幫我們自動快取、記錄所有送到 Internet 的 HTTP requests。這讓我們無須改變使用者瀏覽器的設定值，也可以阻擋不想要的內容。以下是設置代理伺服器的規則，這些規則必須放在 PREROUTING chain 的頂端才有效：

```
# 為內部網路設置一個透明的 Squid proxy
#
# 關於 Squid 的設置細節，請參考：
# http://www.linuxdoc.org/HOWTO/mini/TransparentProxy.html
#
```

譯註　TCP 協定採用「三段式交握」（three-way handshake）程序來搭建連線，交握程序的第一個步驟，就是送出設立了 SYN（同步）位元的 TCP 封包（簡稱「SYN 封包」）。所以，計算「SYN 封包」的個數，就可算出 TCP 連線的搭建速率。

```
iptables -t nat -A PREROUTING -i $INT_IFACE -p tcp --dport 80 \
         -j REDIRECT --to-port 3128
```

此規則將原本送到外界 TCP port 80 的封包，轉向到防火牆本身的 Squid proxy（位於 TCP port 3128）。

`iptables` 容許你比對任何 TCP 旗標。這表示你可用下列規則阻擋聖誕樹封包（所有旗標皆被設立的 TCP 封包），或是空封包（所有旗標皆被歸零）：

```
# 丟掉 XMAS & NULL 兩種 TCP 封包
iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL ALL -j DROP
iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
```

iptables 的進階功能

`iptables` 還提供了好些先進的防火牆功能，但是必須修補（patch）核心才能得到這些功能。你可從 http://www.Netfilter.org/ 取得與你的核心相符的 patch-o-matic 版本【譯註】。所謂的 Patch-o-matic 版本，是指那些尚未被納入主流 Linux 核心的 `iptables` 功能，其中某些功能尚在實驗階段，所以必須小心使用。

藉由 `psd` 功能（尚在實驗階段），`iptables` 可以偵測並阻擋外來的通訊埠掃瞄（inbound port scan）：

```
# DROP 外來的 port scans
iptables -t nat -A PREROUTING -i $EXT_IFACE -d $DEST_IP -m psd -j DROP
```

藉由 `iplimit` 功能（也還在實驗階段），`iptables` 可以限制來自特定 IP 位址的 TCP 連線數量：

```
# 當某特定主機同時有 16 條連線之後，就 DROP 後續到來的連線要求
iptables -t nat -A PREROUTING -i $EXT_IFACE -p tcp --syn -d $DEST_IP \
         -m iplimit --iplimit-above 16 -j DROP
```

最有用的 Netfilter 補強功能之一，是讓你能依據封包內容來進行過濾。實驗性的字串比對功能（string-matching patch）讓你能篩檢出含有特定字串的封包。這功能對於 web server 特別有用，因為你可以用它來過濾掉 CodeRed 或 Nimda 之類的病毒，讓我們的 web server 不受到這些病毒的騷擾：

```
# 默默丟掉 CodeRed 與 Nimda 病毒發出的感染封包
iptables -t filter -A INPUT -i $EXT_IFACE -p tcp -d $DEST_IP --dport http \
         -m string --string "/default.ida?" -j DROP
```

譯註　你必須用 CVS 取得這些 patch，在「修補」了核心原始碼之後，還必須然後重新編譯核心，最後才能使用這裡提到的功能。關於詳細操作程序，請參考 Netfilter-extensions-HOWTO。

```
iptables -t filter -A INPUT -i $EXT_IFACE -p tcp -d $DEST_IP --dport http \
              -m string --string ".exe?/c+dir" -j DROP
iptables -t filter -A INPUT -i $EXT_IFACE -p tcp -d $DEST_IP --dport http \
         -m string --string ".exe?/c+tftp" -j DROP
```

通訊埠轉接（Port forwarding）目前已經是 `iptables` 的標準功能了，稱為 DNAT（Destination NAT）。比方說，只要在 nat 表的 PREROUTING chain 設置下列規則，就可以將本機收到的 HTTP requests 轉遞到內部網路上某系統（例如 10.0.0.3）：

```
# 使用 DNAT 轉接 HTTP port
iptables -t nat -A PREROUTING ! -i $INT_IFACE -p tcp \
        --destination-port 80 -j DNAT --to 10.0.0.3:80
```

UDP 封包也可以被轉遞。你無須在 INPUT chain 設置一條接受該 UDP port 的規則，就可以直接將特定 UDP port 收到的封包轉遞到他處，但是這個「他處」必須在與本機介面相接的網路上（換言之，不能轉遞到外界網路）；如果你真的需要轉遞到遠端網路，請參考《#48 將 TCP Ports 轉接到任意機器》的作法，使用 `rinetd` 或 `nportredird` 之類的工具來處理。

如果使用「通訊埠轉遞」將 HTTP requests 轉遞到內部主機，你還可以在 FORWARD chain 設置下列規則來過濾掉 CodeRed 病毒：

```
iptables -t filter -A FORWARD -p tcp --dport http \
         -m string --string "/default.ida?" -j DROP
```

剛開始接觸 `iptables` 可能會覺得頗為棘手，因為它的設計實在太靈活、太有彈性。但也正因為如此，使得 `iptables` 成為相當有用的利器。如果你在構思自己的過濾條件時遇到困難（保證你會！），記得 `iptables -L -n` 與 `tcpdump`（或 `ethereal`）是你最好的兩位朋友。

相關參考

- 《#48 將 TCP Ports 轉接到任意機器》
- 《The Linux Firewall HOWTO》

HACK #48

將 TCP Ports 轉接到任意機器

移花接木 — 使遠端的服務出現在本端通訊埠上

正如《#47 iptables 的訣竅與技巧》的展示，`iptables` 能將防火牆的 TCP/UDP port 轉接到內部主機。但是，如果要將來自任意位址的封包轉接到遠端（非當地網路）的主機上，那又該怎麼辦？很顯然地，我們無法在傳輸層做這件事，所以必須借助應用層的轉遞工具，例如 `rinetd`。

`rinetd` 是一個小程式，而且也已經有好幾年歷史了，但是它相當精簡、有效率，剛好可完美解決我們的問題。從 http://www.boutell.com/rinetd/ 取得原始碼壓縮檔並解開之後，依序使用 `make` 與 `make install` 完成編譯與安裝【譯註 1】，就會得到一個簡潔的 `rinetd` 程式。你可用它來施行移花接木之計 — 將本機特定 TCP port 收到的封包轉換到其它機器上。`rinetd` 唯一的美中不足之處，在於它不支援 UDP port。

`rinetd` 的組態檔的語法簡單到不行：

```
[SrcAddress] [SrcPort] [DestAddress] [DestPort]
```

每一列各代表一個要被轉接的通訊埠。*SrcAddress* 與 *DestAddrss* 可以是數字形式的 IP 位址，也可以是文字形式的主機名稱。若將 *SrcAddress* 設定成 0.0.0.0，表示 `rinetd` 將繫結（bind）【譯註 2】到每一個屬於本地主機的 IP 位址。

```
0.0.0.0  80  some.othersite.gov  80
216.218.203.211  25 123.45.67.89  25
0.0.0.0  5353  my.shellserver.us  22
```

將組態內容存入 /etc/rinetd.conf 檔案，並將 `rinetd` 複製到你認為順手的地方（例如 /usr/local/sbin/），然後直接執行 `rinetd` 就可以啟動它。

在我們的例子中，第一組規則將送到本機任何 IP 位址的 port 80 的封包，一律轉接到 some.othersite.gov 的 port 80。當然，必須在本機沒有任何其它行程（例如 Apache）佔用 port 80 的情況下，`rinetd` 才能發揮作用。

第二組規則外界送到本機 216.218.203.211 位址的 SMTP 封包，轉送到位於 123.45.67.89 的郵件伺服器（不影響繫結在本機其它 IP 的 SMTP agent）。最後一個例子，將外界送到本機 port 5353（不管任何 IP 位址）的封包，轉送到位於 my.shellserver.us 的 SSH server。

這些動作都不必用到 NAT 或系統核心的任何特殊功能，因為 `rinetd` 是在應用層（通常在 user space）處理來往細節。只要組態檔正確設定無誤，在執行了 `rinetd` 之後，它會自動留駐在記憶體，並繫結到你所指定的任何本機通訊埠。

譯註 1　在 RedHat 系統上安裝時，你可能要修改一下它的 Makefile，修正 man page 的安裝目錄（將 /usr/man/man8 改成 /usr/share/man/man8）。

譯註 2　在網路程式設計的術語中，「繫結」（bind）是伺服軟體佔用本機某通訊埠，以靜待外來連線。相對地，「連接」（connect）是指本機（通常是用戶端軟體）對遠端通訊埠（通常是伺服端）發出連線要求的動作（送出 SYN 封包）。

當你打算改變伺服器的 IP 位址時（搬家、換 ISP、換機器、重整位址配置...），rinetd 可以幫你順利處理過渡期的交通，讓客戶可以透過原本的 IP 位址使用新位址提供的服務。

如果你要轉接的 TCP port 全部都位於 1024 以上，rinetd 甚至可以不需要 root 權限。此外，rinetd 也提供了一些擴充選項，讓你能控制存取對象（就像 inetd 與 xinetd 那樣），並保留日誌記錄。總之，當你需要轉遞 TCP port 時，rineted 這個小工具實在是太好用了！

相關參考：

- http://www.boutell.com/rinetd/
- 《#47 iptables 的訣竅與技巧》

HACK #49 要在 iptables 使用自訂的規則集

使用自訂的規則集來控制你的防火牆

Netfilter 的 filter 表，內建了三組串鏈（chains），分別是 INPUT、FORWARD 與 OUTPUT。當你要設置許多繁複的規則時，甚至可另外自己定義額外的串鏈。Netfilter 處理自訂串鏈的程序，與一般內建串鏈沒兩樣，唯一的要求，是自訂串鏈必須決定封包的最終命運。

要建構一個新串鏈，請使用 -N 選項：

```
root@mouse:~# iptables -N funfilter
```

不管在任何時刻，都可用標準的 -L 選項來檢視 filter 表裡有哪些串鏈：

```
root@mouse:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain funfilter (0 references)
target prot opt source destination
```

要使新建的自訂串鏈能發揮作用，必須在預設串鏈的某處設置一個進入點。比方說，在 INPUT 串鏈設置一條會跳到（-j）funfilter 的規則：

```
root@mouse:~# iptables -t filter -A INPUT -j funfilter
```

現在，你可以任意規則加入自訂串鏈。例如，依據來源主機的 MAC 位址來進行過濾：

```
root@mouse:~# iptables -A funfilter -m mac \
> --mac-source 11:22:33:aa:bb:cc   -j ACCEPT
root@mouse:~# iptables -A funfilter -m mac \
> --mac-source de:ad:be:ef:00:42 \  -j ACCEPT
root@mouse:~# iptables -A funfilter -m mac \
> --mac-source 00:22:44:fa:ca:de \
> -j REJECT --reject-with icmp-host-unreachable
root@mouse:~# iptables -A funfilter -j RETURN
```

在串鏈或表的最末端，往往會設置一條 RETURN jump 規則，讓處理程序回到當初的進入點之後。在此例中，這會使得所有不符合 funfilter 規則的封包回到 INPUT 串鏈。設定好所有規則之後，也是同樣使用 -L 選項來觀察設定結果：

```
root@mouse:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
funfilter all -- anywhere anywhere

Chain FORWARD (policy ACCEPT)
target prot opt source destination

Chain OUTPUT (policy ACCEPT)
target prot opt source destination

Chain funfilter (0 references)
target prot opt source destination
ACCEPT all -- anywhere anywhere MAC 11:22:33:AA:BB:CC
ACCEPT all -- anywhere anywhere MAC DE:AD:BE:EF:00:42
REJECT all -- anywhere anywhere MAC 00:22:44:FA:CA:DE reject-with
icmp-host-unreachable
RETURN all -- anywhere anywhere
```

你可以設置任意多個自訂串鏈，甚至在這些串鏈之間跳來跳去。當你在規劃複雜的防火牆規則時，不妨將新規則定義在自訂串鏈裡，將它們與標準系統規則隔開，如此一來，我們就可輕易地取消或啟用新規則；例如，如果臨時想取消新規則，只要將 INPUT 串鏈裡的進入點刪掉即可，而不必刪除所有新規則，像這樣：

```
root@mouse:~# iptables -t filter -D INPUT -j funfilter
```

當然，如果你決定要刪除整個自訂串鏈，請用 -X 選項：

```
root@mouse:~# iptables -X funfilter
```

如果你先刪除自訂串鏈，而忘了刪除該串鏈在預設串鏈裡的進入點，將很難保證不發生（或發生）什麼奇怪的事。如果搞亂了所有串鏈，建議使用 `-F` 刷掉所有串鏈裡的規則，然後重新開始。

只要處理得當，即使是複雜的規則集也可以很容易看懂，訣竅是為你的自訂串鏈取一個名符其實的名稱。

相關參考：

- 《#47 iptables 的訣竅與技巧》

HACK #50

通道技術：IPIP 封裝

使用 Linux IPIP 模組來搭建 IP 通道。

如果你從沒用過 IP 通道，在你繼續看下去之前，建議先去研讀《Advanced Router HOWTO》。基本上，IP 通道的作用很類似於 VPN，只不過並非每一種 IP 通道都涉及資料加密。在一台具有通道的機器上，若該通道連接到另一個網路，則該機器上會有一個虛擬介面，但是此介面的 IP 位址不屬於本地端網路，而屬於相連的遠端網路。通常，所有（或大部份）封包都會被導入通道，所以，遠端看起來就像是本地端的網路服務。或者，更通俗的說法，就是通道將兩個私有網路連接在一起，而 Internet 成了兩端網路之間的產業道路。

最簡單的通道協定是 IPIP，用於在兩台機器之間搭建簡單的 IP-within-IP 通道。由於 IPIP 協定的實作方式相當簡單，所以各種作業系統（*BSD、Solaris、甚至 Windows）幾乎都支援。提醒你，IPIP 只是一種通道協定而已，它並不涉及任何加密，而且 IPIP 也只能傳導直播封包（unicast packets），如果你需要通導群播封包（multicast packets），請參考《#51 通道技術：GRE 封裝》所介紹的 GRE 通道。

在開始動手打穿通道之前，你必須先取得一份 Advanced Routing Tools（ART，通常稱為 iproute2 套件），特別是其中的 `ip` 工具程式。最新、最權威的版本，可從 ftp://ftp.inr.ac.ru/ip-routing/ 取得。話說在前頭，ART 的操作介面並不友善，但是其功能確實非常強大，它能做到的效果，大概僅略次於直接動手調整 Linux 核心的網路引擎。

假設你有兩個私有網路（10.42.1.0/24 與 10.42.2.0/24），而且分別以各自的 Linux NAT 閘道器連接到 Internet。第一個 Linux NAT 閘道器的 IP 位址分別是 10.42.1.1（連接 10.42.1.0/24 內部網路）與 240.101.83.2（連接 Internet）。而第二個閘道器的的 IP 位址分別是 10.42.2.1（連接 10.42.2.0/24 內部網路）與 251.4.92.217（連接 Internet）。整個網路佈局相當簡單，所以讓我們直接開始動手。

首先，兩端的 Linux NAT 閘道器都要載入 ipip 核心模組：

```
# modprobe ipip
```

接著，在第一個閘道器（位於 10.42.1.0/24 網路），做下列動作：

```
# ip tunnel add mytun mode ipip remote 251.4.92.217 \
>   local 240.101.83.2 ttl 255
# ifconfig mytun 10.42.1.2
# route add -net 10.42.2.0/24 dev mytun
```

第一步是搭建一條通過 Internet 到遠端的 IPIP 通道，然後指定一個內部網路上的可用 IP 位址（10.42.1.2）給新的 mytun 介面，最後，將遠端內部網路（10.42.2.0/24）的路徑加入路由表。

在第二個閘道器上（位於 10.42.2.0/24 網路），也是依樣畫葫蘆：

```
# ip tunnel add mytun mode ipip remote 240.101.83.2 \
>   local 251.4.92.217 ttl 255
# ifconfig mytun 10.42.2.2
# route add -net 10.42.1.0/24 dev mytun
```

「mytun」這個名稱是我隨意取的，如果你喜歡，你可以另外取一個更有意義的名稱。一切搞定之後，應該可以在第一個閘道器上 ping 到 10.42.2.2，而在第二閘道器上 ping 到 10.42.1.2。事實上，在第一網路（10.42.1.0/24）【譯註】上的任何主機，都可以直接 ping 第二網路上的主機，另一方面，在第二網路（10.42.2.0/24）上的任何主機，也都可以直通 10.42.1.0/24 網路上的每一台主機，感覺就像兩網路之間沒有 Interent 隔閡一樣。

如果你還在跑 Linux 2.2x 核心，那你走運了。這裡有一個讓你不用 ART也能達到同樣效果的密技。在第一台閘道器：

```
# ifconfig tun10 10.42.1.2 pointopoint 251.4.92.217
# route add -net 10.42.2.0/24 dev tun10
```

在第二台閘道器（位於10.42.2.0/24）：

```
# ifconfig tun10 10.42.2.2 pointopoint 240.101.83.2
# route add -net 10.42.1.0/24 dev tun10
```

搞定了，就這樣。

譯註　10.42.1.0/24 網路上的主機，其預設閘道器必須設定為 10.42.1.2（NAT 的 mytun 介面）或 10.42.1.1（NAT 接在內部網路上的介面）。而 10.42.2.0/24 網路上的主機，其預設閘道器必須設定為 10.42.2.1 或 10.4.2.2。

如果你能 ping 到對方的閘道器，但是對方網路上的機器所發出的封包，似乎都通不過該閘道器，那或許是該閘道器沒啟動 IP forward 功能，以致於兩網路介面上的封包不能互相穿梭到對方。

```
# echo "1" > /proc/sys/net/ipv4/ip_forward
```

如果你需要通達 10.42.1.0 與 10.42.2.0 之外的網路，只需在路由表裡添加額外的路徑（用 route add -net … ）即可。每個網路上的個別主機都不必調整組態，只要主機的預設路徑（default route）指向該網路的閘道器即可（原本就應該這樣，畢竟那是它們的「閘道器」）。

要撤除通道，兩端閘道器都必須關掉通道介面；必要時，你甚至可以刪除通道介面：

```
# ifconfig mytun down
# ip tunnel del mytun
```

或者，在 Linux 2.2：

```
# ifconfig tun10 down
```

在介面被關閉之後，核心會很聰明地清掉路由表裡的相關路徑。

相關參考：

- 《Advanced Routing HOWTO》（http://www.tldp.org/HOWTO/Adv-Routing-HOWTO/）
- Advanced Routing Tools（位於 ftp://ftp.inr.ac.ru/ip-routing/）

HACK #51 通道技術：GRE 封裝

支援群播的 IP 通道

GRE 是 Generic Routing Encapsulation 的縮寫，它是類似於 IPIP（見 《#50 通道技術：IPIP 封裝》的一種非加密式的封裝協定。使用 GRE 取代 IPIP 的理由之一，是 GRE 支援**群播封包**（multicast packets），但是另一個更具說服力的理由，是 GRE 能相容於 Cisco 閘道器。

且讓我們沿用 《#50 通道技術：IPIP 封裝》的例子，來示範如何設定 GRE 通道。假設你有兩個私有網路（10.42.1.0/24 與 10.42.2.0/24），而且都各自都透過自己的 Linux NAT 閘道器連接到 Internet，其中第一個閘道器的 "真實" IP 位址是 240.101.83.2，而第二個閘道器的 “真實” IP 位址是 251.4.92.217。

如同 IPIP 通道，你需要先取得 iproute2 套件（Linux 2.2.x 也需要，因為我不知道如何用 2.2x 版核心來模擬 GRE 通道）。在安裝好 iproute2 套件之後，我們必須在兩端的 Linux 閘道器都載入 GRE 核心模組：

```
# modprobe ip_gre
```

在第一個網路（10.42.1.0/24）的閘道器上，設置一個新的通道裝置：

```
# ip tunnel add gre0 mode gre remote 251.4.92.217 local \
>   240.101.83.2 ttl 255
# ip addr add 10.42.1.254 dev gre0
# ip link set gre0 up
```

你可以為通道裝置取任何名稱，`gre0` 只是一個例子而已，不是硬性規定。此外，通道裝置的 IP 位址，可以是內部網路上的任何可用位址，但不可以是 10.42.1.1（因此此位址已經分配給閘道器的內側網路介面）。現在，可在路由表裡加入一筆通過 `gre0` 介面的網路路徑（network route）：

```
# ip route add 10.42.2.0/24 dev gre0
```

第一個網路就這樣搞定了，第二個網路也是依樣畫葫蘆：

```
# ip tunnel add gre0 mode gre remote 240.101.83.2 local \
>   251.4.92.217 ttl 255
# ip addr add 10.42.2.254 dev gre0
# ip link set gre0 up
# ip route add 10.42.1.0/24 dev gre0
```

同樣地，你可用第二網路上任何尚未被佔用的 IP 位址來取代 10.42.2.254。必要時，使用多次 **`ip route add .. dev gre0`** 命令將該介面通達的所有網路之路徑都放進路由表。

就這樣！現在兩個網路之間應該可以互傳封包了，就好像 Internet 不存在一樣！如果用 `tracroute` 追查路徑，你會發現兩閘道器竟然是相鄰的，只不過這兩站之間的傳輸延遲可能會有點長（除非兩站之間的 Internet 真的是高速線路）。如果你遇到麻煩，回頭看看 #50 的例子，不要灰心。當你在解決新網路環境的問題時，封包取樣工具（packet sniffer）是你最好的朋友，所以，如果真的想不透，那就祭出你的 `tcpdump` 或 `ethereal` 看看吧！比方說，你可以一邊 `ping`，一邊使用 **`tcpdump 'proto \icmp'`** 抓封包，然後你將會清楚問題發生在哪裡。

分別在兩邊閘道器都執行下列命令，可撤銷通道：

```
# ip link set gre0 down
# ip tunnel del gre0
```

相關參考

- Advanced Routing HOWTO（http://www.tlpd.org/HOWTO/Adv-Routing-HOWTO/）
- Advanced Routing Tools（iproute2，ftp://ftp.inr.ac.ru/ip-routing/）

HACK #52 使用 VTun over SSH 來避開 NAT

用 VTun 與一條 ssh 連線來連接兩個私有網路

VTun 是一套可在「使用者空間」（相對於「核心空間」）運作的通道伺服軟體，它利用 Linux 的 Universal TUN/TAP 模組來接通兩個以 IP、PPP 或 SLIP 連線的網路，這表示我們也可以只用一個 TCP port（也就是一條 ssh 連線）來加密、傳輸所有 IP 封包。如果你的網路伺服器被防火牆擋得死死的，這技術就派上用場了！因為它可以封裝任何網路服務的 IP 封包（所以才會稱為「Universal Tunnel」）。

且讓我們以實例來說明如何使用 VTun。假設我們有一台只有 non-routable IP 位址的主機，其 IP 位址是 192.168.0.114（後文簡稱此主機為「client」），此主機透過一個 NAT 閘道器（其內側位址是 192.168.0.7）連接到 Internet：

```
[root@client ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:D0:59:0D:13:FD
          inet addr:192.168.0.114  Bcast:192.168.0.255
          Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:91714 errors:46 dropped:0 overruns:0 frame:46
          TX packets:60932 errors:0 dropped:0 overruns:0 carrier:0
          collisions:4002 txqueuelen:100
          RX bytes:24433872 (23.3 Mb)  TX bytes:36467254 (34.7 Mb)
          Interrupt:9 Base address:0x90c0 Memory:81400000-81400038

[root@client ~]# route -n
Kernel IP routing table
Destination  Gateway      Genmask        Flags Metric Ref   Use Iface
192.168.0.0  0.0.0.0      255.255.255.0  U     0      0       0 eth0
127.0.0.0    0.0.0.0      255.0.0.0      U     0      0       0 lo
0.0.0.0      192.168.0.7  0.0.0.0        UG    0      0       0 eth0
```

從 client 到 tw.yahoo.com 的路線如下：

```
[root@client ~]# traceroute -n tw.yahoo.com
traceroute to tw.yahoo.com (202.1.237.21), 30 hops max, 38 byte packets
1  192.168.0.7  0.970 ms  0.268 ms  0.200 ms
2  211.23.29.121  1.124 ms  1.084 ms  1.065 ms
3  10.23.29.254  46.288 ms  45.958 ms  45.320 ms
4  168.95.82.194  43.882 ms  44.433 ms  43.587 ms
5  211.22.34.6  44.593 ms  44.469 ms  43.775 ms
6  211.22.35.169  56.854 ms  43.707 ms  43.640 ms
7  211.22.41.89  43.821 ms  43.783 ms  44.337 ms
8  202.1.237.21  43.820 ms  43.775 ms  44.331 ms
```

在 Internet 的另一端有一台 VTun 伺服器，其 IP 位址為 211.23.161.10（後文簡稱此伺服器為「server」），該伺服器所在的區域網路上還有兩個備用的 routable

IP 位址，可用來當成通道兩端的 IP 位址，分別是 211.23.161.13 與 211.23.161.14。而我們的任務，是將這兩個分居於 Internet 兩端的網路連接在一起，讓 Internet 上的其它主機可透過 211.23.161.14 這個 routable IP 位址來存取 192.168.0.114 這台藏在 NAT 後面的網路伺服器。

現在，讓我們開始著手架設通道。第一步是在兩端機器都載入 tun 模組（除非核心本身已經內建 TUN/TAP 模組）【譯註】：

```
# modprobe tun
```

值得注意的是，如果 server 與 client 兩端的核心版本不相同，tun 模組有時候可能會掛掉。為了保險起見，建議兩端機器都使用最新（而且相同版本，例如 2.4.21）的核心。

在 server 機器上，將下列內容寫到 /usr/local/etc/vtund.conf 組態檔：

```
options {
  port 5000;
  ifconfig /sbin/ifconfig;
  route /sbin/route;
  firewall /sbin/iptables;
  syslog auth;
}

default {
  compress no;
  speed 0;
}

# home 是通道名稱
home {
  type tun;
  proto tcp;
```

譯註　tun 是一種字元式裝置（char device），其裝置代碼為 10,200，使用者空間的應用程式（例如 vtund）必須透過裝置節點（通常是在 /dev/tun 或 /dev/net/tun）才能使用 tun 的功能，然而，有些 Linux distribution 並未幫你製作 tun 裝置節點，因此，在繼續設定 vtund 之前，請先檢查自己的系統是否有 tun 裝置節點：

```
[root@client ~]# ls -l /dev/net
總計 0
crw-------    1 root     root      10, 200  1月 30  2003 tun
```

如果沒有，那麼，你可用下列程序自己製作 tun 裝置節點：

```
# mkdir /dev/net   （如果 /dev/net/ 目錄不存在的話）
# mknod /dev/net/tun c 10 200
# chmod 600 /dev/net/tun
```

```
  stat yes;
  keepalive yes;
  pass sHHH;   # client 與 server 兩端的密碼必須相符！

# 譯註：
# up{} 是連線啟動時會被自動執行的程序
# 211.23.161.13 是 PPP 連線在本地端（server）的 IP 位址
# 211.23.161.14 是 PPP 連線在遠端（client）的 IP 位址
# 192.168.0.0/24 是 client 所在的區域網路
# 「%%」是通道裝置的名稱（tun0）。
  up {
    # 啟動 tun0 介面
    ifconfig "%% 211.23.161.13 pointopoint 211.23.161.14";
    # 讓 tun0 與 eth0 能收到 destIP=211.23.161.14 的封包
    program /sbin/arp "-Ds 211.23.161.14 %% pub";
    program /sbin/arp "-Ds 211.23.161.14 eth0 pub";
    route "add -net 192.168.0.0/24 gw 211.23.161.14";
    # 如果 server 本身已經用 #46 的方法設置 NAT rule，則可省略下列程序
    firewall "-t nat -A POSTROUTING -s 211.23.161.14 -j MASQUERADE";
  };

# down{} 是連線關閉時的自動執行程序
  down {
    program /sbin/arp "-d 211.23.161.14 -i %%";
    program /sbin/arp "-d 211.23.161.14 -i eth0";
    route "del -net 192.168.0.0/16 gw 211.23.161.14";
    # 如果 up{} 沒設置 NAT rule，則可省略下列程序
    firewall "-t nat -D POSTROUTING -s 211.23.161.14 -j MASQUERADE";
  };
}
```

然後用下列命令啟動 vtund 伺服器：

```
root@server:~# vtund -s
```

在 client 這邊，整個程序是啟動通道、丟棄原本預設路徑、然後以通道的另一端（211.23.161.14）為新的預設路徑，請參考下列組態內容（放在 client 機器上的 /usr/local/etc/vtund.conf 檔案）：

```
options {
  port 5000;
  ifconfig /sbin/ifconfig;
  route /sbin/route;
  firewall /sbin/iptables;
}

default {
  compress no;
```

```
  speed 0;
}

home {
  type tun;
  proto tcp;
  keepalive yes;
  pass sHHH; # 兩端密碼必須一致，而且不可省略。

# 譯註：
# 211.23.161.14 是 PPP 連線在本地端（client）的 IP 位址
# 211.23.161.13 是 PPP 連線在遠端（server）的 IP 位址
# 211.23.161.10 是 Tunnel Server 的 IP 位址
# 192.168.0.7 是本地端（client）的 NAT 閘道器位址（內側）
  up {
    # 啟動 tun0 介面
    ifconfig "%% 211.23.161.14 pointopoint 211.23.161.13 arp";
    # 透過本端的 NAT 閘道到 tunnel server
    route "add 211.23.161.10 gw 192.168.0.7";
    # 刪除原本的預設路徑
    route "del default";
    # 使用通道另一端的 IP 位址為預設路徑
    route "add default gw 211.23.161.13";
  };

  # 還原路由表
  down {
    route "del default";
    route "del 211.23.161.10 gw 192.168.0.7";
    route "add default gw 192.168.0.7";
  };
}
```

最後，在 client 機器上執行下列命令：

```
[root@client:~]# vtund -p home server
```

大功告成！現在不只是在 client 與 server 之間搭建了一條通道，就連預設路徑也設定好了（以通道的對端為閘道）。現在再看看從 client `traceroute` 到同一個網站的結果：

```
[root@client ~]# traceroute -n tw.yahoo.com
traceroute to tw.yahoo.com (202.1.237.21), 30 hops max, 38 byte packets
 1  211.23.161.13  122.112 ms  104.705 ms  112.139 ms
 2  211.23.161.9  105.952 ms  112.623 ms  111.771 ms
 3  10.23.161.254  159.177 ms  172.400 ms  185.127 ms
 4  168.95.82.194  172.147 ms  172.046 ms  172.081 ms
 5  211.22.34.6  173.140 ms  170.927 ms  185.346 ms
```

```
 6  211.22.35.169  188.332 ms  189.018 ms  196.661 ms
 7  211.22.41.89  194.883 ms  188.416 ms  196.859 ms
 8  202.1.237.21  347.439 ms  318.071 ms  473.486 ms
```

如果你仔細比對搭建通道前後的路線，你將會發現，在搭建通道之前，client 到 tw.yahoo.com 的第一站是當地的 NAT 閘道器（192.168.0.7），在搭建通道之後，第一站是 VTun 通道的遠端（211.23.161.13）。只要搭配適當的 `route` 與 `iptables` 命令，client 與 server 兩端的區域網路就可以連結成一個邏輯網路。此外，Internet 上的任何主機都可以透過該通道伺服器（211.23.161.14）來存取 client 上的任何伺服行程。在 client 網路上的 NAT 閘道器（192.168.0.7）無須任何改變。

在 client 機器上，會看到一個新的通道介面：

```
[root@client ~]# ifconfig tun0
tun0      Link encap:Point-to-Point Protocol
          inet addr:211.23.161.14  P-t-P:211.23.161.13
              Mask:255.255.255.255
          UP POINTOPOINT RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:36 errors:0 dropped:0 overruns:0 frame:0
          TX packets:36 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:10
          RX bytes:2076 (2.0 Kb)  TX bytes:1368 (1.3 Kb)
```

以下是 client 的新路由表。請注意，client 仍必須保留一筆「通過 NAT 閘道器到達通道伺服器的 IP 位址」的主機路徑，否則通道封包就送不出去了：

```
[root@client ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
211.23.161.13   0.0.0.0         255.255.255.255 UH    0      0        0 tun0
211.23.161.10   192.168.0.7     255.255.255.255 UGH   0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         211.23.161.13 0.0.0.0         UG    0      0        0 tun0
```

要撤銷通道，只須殺掉 client 的 `vtund` 行程即可。在 `vtund` 真正結束之前，它會將網路環境還原成啟動之前的狀態（執行 `down{}` 所指定的程序）。

Tunnel Server 是黑客（cracker）的最愛，因為這是用來掩飾犯罪證據的絕佳工具，雖然 VTun server 具備基本的密碼驗證，而且也提供了加密機制，但是其效果畢竟比不上 `ssh`。幸運的是，我們可以使用 `ssh` 來包裝 VTun 封包，並享用 `ssh` 的驗證與加密能力。就本例而言，我們只要將 client 的 TCP port 5000 轉接（forward）到 server 的同一個 TCP port 即可。就像這樣：

```
[root@client ~]# ssh -f -N -c blowfish -C -L5000:localhost:5000 server
[root@client ~]# vtund -p home localhost
```

```
[root@client ~]# traceroute -n tw.yahoo.com
traceroute to tw.yahoo.com (202.1.237.21), 30 hops max, 38 byte packets
 1  211.23.161.13  483.486 ms  197.324 ms  108.274 ms
 2  211.23.161.9  112.504 ms  111.535 ms  105.938 ms
 3  10.23.161.254  158.723 ms  172.354 ms  172.437 ms
 4  168.95.82.194  171.352 ms  171.927 ms  172.229 ms
 5  211.22.34.6  172.759 ms  172.051 ms  171.299 ms
 6  211.22.35.169  172.051 ms  171.695 ms  172.402 ms
 7  211.22.41.89  172.037 ms  171.928 ms  173.045 ms
 8  202.1.237.252  172.011 ms  172.045 ms  211.067 ms
```

為了打消 client `vtund` 直接連線到 server port 5000 的念頭，我們可在 server 設置一條防火牆規則，阻止外界使用這個通訊埠：

```
root@server:~# iptables -A INPUT -t filter -i eth0 -p tcp --dport \
>   5000 -j DROP
```

這使得只有本端連線才得以穿越（因為使用 loopback，而非 eth0），進而迫使 client 必須先搭建一條 ssh 通道到 server，才可以搭建 VTun 通道。

如你所見，VTun 是超級好用的必備工具，除了能讓躲在 NAT 後方的伺服器有一個 routable IP 位址之外，還可以用一條 `ssh` 連線將兩個網路連接成一個邏輯網路（用任何一方發出的 ssh 連線都可以）。

如果你的腦袋已經被 vtund.conf 搞得七葷八素，或是你懶到無藥可救，或是你根本沒搞懂如何設定 client 的 vtund.conf ... 別灰心，或許 #53 會讓你的日子好過些。

重點提示

- 兩端（client 與 server）的「連線名稱」（本例中的 home）必須完全相同，否則你會得到莫名其妙的『server disconnected』訊息。
- 兩端 vtund.conf 中的 *password* 欄位，也必須完全相同，否則將搭不上線。
- 如果連線不順利，請確定兩端都使用相同版本的核心，而且伺服器已經啟動（試著在 client 使用 `telnet server 5000` 檢驗伺服器是否還活著）。
- 先嘗試直接連接的方法，在確定 vtund.conf 的設定值都沒有問題之後，才使用 ssh 連線。
- 如果還有問題，先從 /etc/syslog.conf 檢查「auth」訊息的去處，然後看看 client 與 server 兩端的 auth 日誌訊息。

相關參考

- VTun 的首頁（位於 http://vtun.sourceforge.net/）
- /usr/src/linux/Documentation/networking/tuntap.txt
- `man vtund`、`man vtund.conf`
- 《#53 自動產生 vtund.conf》
- 《#45 簡易防火牆》
- 《#71 使用 ssh 轉接通訊埠》

HACK #53 自動產生 vtund.conf

即時產生符合當時網路環境的 vtund.conf

如果你才剛看完《#52 使用 VTun over SSH 來避開 NAT》這裡有一個稱為 vtundconf 的 Perl 程式，可自動產生用於 clinet 的 vtund.conf 組態檔。如果你還沒看過《#52 使用 VTun over SSH 來避開 NAT》或是你不曾用過 VTun，那麼，請先回頭練完功夫再說。

基本上，這程式是依據當地主機當時的預設閘道來決定 vtund.conf 的內容，讓我們不必煩惱如何修改路由表。

在檢視實際程式碼之前，先說明它的用法。要執行此程式的第一步，你必須先修改程式開頭關於「環境設定」的參數，將通道名稱、兩端的 IP 位址、通道伺服器的 IP 位址、密碼等參數設定在 `$Config` 變數。

除了基本網路參數之外，這程式還需要一個代表「通道名稱」的參數，你可以直接在命令列指定此參數，像這樣：

```
# vtundconf home
```

或者，你也可以預先設定 `$TunnelName` 環境變數，而不指定任何命令列參數（如果設定了環境變數又同時指定命令列參數，則 `vtundconf` 會以命令列參數為準）。

如果忘了設定 `$TunnelName` 環境變數，也沒指定命令列參數，則它會以自己的程式名稱為通道名稱。這表示你可以這樣做：

```
# ln -s vtundconf home
# ln -s vtundconf tunnel2
```

如此一來，當你執行 `home` 時，它就產生 home 通道的 vtund.conf 組態檔；當你執行 `tunnel2` 時，則產生 tunnel2 通道的組態檔：

```
# ./home > /usr/local/etc/vtund.conf
```

你或許會好奇，為何要這麼麻煩，去寫一個能在現場產生 vtund.conf 的程式？畢竟一旦設定完畢之後，就不會再改變了，不是嗎？

沒錯，實情確實如此，但是想像一下這種情況：如果有一台時常需要連接不同網路的 Linux 筆記型電腦（比方說，在家裡連接 DSL，在辦公室接 Ethernet，在咖啡店接無線網路 ...）。我們只要在不同地點分別跑一次 `vtundconf`，就可以得到適用於不同地點的組態檔，即時當時的 IP 與閘道器是由 DHCP 指派的也無所謂。這讓我們可不管當地的網路環境，而得到一個可用的 routable IP 位址。

順便一提，`vtund` 與 `vtundconf` 都可在目前的 Linux、FreeBSD、Mac OS X、Solaris 上順利運作。

程式：vtundconf

```
#!/usr/bin/perl -w

# vtund wrapper in need of a better name.
#
# (c)2002 Schuyler Erle & Rob Flickenger
#

################ 環境設定

# 如果沒設定 TunnelName，則使用 @ARGV 或 $0。
#
# Config 的意義：
# TunnelName、LocalIP、RemoteIP、TunnelHost、TunnelPort、Secret
# 請依據你的網路環境來修改這些參數

my $TunnelName = "";
my $Config = q{
    home 211.23.161.14 211.23.161.13 211.23.161.10 5000 passWord
    tunnel2 10.0.1.100 10.0.1.1 192.168.1.4 6001 foobar
};
# 「tunnel2」只是為了示範如何用此程式多應付另一種環境環境而已。


################ 主程式開始

use POSIX 'tmpnam';
use IO::File;
use File::Basename;
use strict;

# 工具程式的位置，以及資訊來源
#
```

```
$ENV{PATH}   = "/bin:/usr/bin:/usr/local/bin:/sbin:/usr/sbin:/usr
                /local/sbin";
my $IP_Match = '((?:\d{1,3}\.){3}\d{1,3})';
               # IP 位址(xxx.xxx.xxx.xxx)的正規樣式
my $Ifconfig = "ifconfig -a";
my $Netstat  = "netstat -rn";
my $Vtund    = "/bin/echo";
my $Debug    = 1;

# 從 DATA 區載入樣板（template）
#
my $template = join( "", <DATA> );

# 開一個暫存檔（移植自《Perl Cookbook》，第一版，7.5 節
#
my ( $file, $name ) = ("", "");
$name = tmpnam() until $file = IO::File->new( $name,
                               O_RDWR|O_CREAT|O_EXCL );
END { unlink( $name ) or warn "Can't remove temporary file $name!\n"; }

# 如果沒指定 TunnelName，則使用命令列的第一的引數，
# 如果兩者都沒有，則使用本程式的 basename。
# 這讓使用者可以將不同的通道名稱 symlink 到同一個程式
#
$TunnelName ||= shift(@ARGV) || basename($0);
die "Can't determine tunnel config to use!\n" unless $TunnelName;

# 分析組態
#
my ($LocalIP, $RemoteIP, $TunnelHost, $TunnelPort, $Secret);
for (split(/\r*\n+/, $Config)) {
    my ($conf, @vars) = grep( $_ ne "", split( /\s+/ ));
    next if not $conf or $conf =~ /^\s*#/o;  # 略過空行、註解
    if ($conf eq $TunnelName) {
  ($LocalIP, $RemoteIP, $TunnelHost, $TunnelPort, $Secret) = @vars;
  last;
    }
}

die "Can't determine configuration for TunnelName '$TunnelName'!\n"
    unless $RemoteIP and $TunnelHost and $TunnelPort;

# 找出預設閘道
#
my ( $GatewayIP, $ExternalDevice );

for (qx{ $Netstat }) {
```

```
    # 在 Linux 與 BSD 的 netstat -r 輸出中，
    #「Gateway」在第二欄，而「Interface」在最後一欄
    #
    if ( /^(?:0.0.0.0|default)\s+(\S+)\s+.*?(\S+)\s*$/o ) {
       $GatewayIP = $1;
       $ExternalDevice = $2;
       last;
    }
}

die "Can't determine default gateway!\n" unless
    $GatewayIP and $ExternalDevice;

# 算出 LocalIP 和 LocalNetwork。
#
my ( $LocalNetwork );
my ( $iface, $addr, $up, $network, $mask ) = "";

sub compute_netmask {
    ($addr, $mask) = @_;
    # 我們必須用 $mask 遮掉 $addr，
    # 因為 linux 的 /sbin/route 會檢查網路位址是否與遮罩長度相符
    #
    my @ip = split( /\./, $addr );
    my @mask = split( /\./, $mask );
    $ip[$_] = ($ip[$_] + 0) & ($mask[$_] + 0) for (0..$#ip);
    $addr = join(".", @ip);
    return $addr;
}

for (qx{ $Ifconfig }) {
    last unless defined $_;

    # 如果遇到新裝置，則儲存先前的（如果有的話）
    if ( /^([^\s:]+)/o ) {
        if ( $iface eq $ExternalDevice and $network and $up ) {
           $LocalNetwork = $network;
           last;
        }
      $iface = $1;
      $up = 0;
    }

    # 算出目前介面的遮罩
    if ( /addr:$IP_Match.*?mask:$IP_Match/io ) {
        # Linux 風格的 ifconfig.
          compute_netmask($1, $2);
```

```
        $network = "$addr netmask $mask";
    } elsif ( /inet $IP_Match.*?mask 0x([a-f0-9]{8})/io ) {
        # BSD 風格的 ifconfig.
          ($addr, $mask) = ($1, $2);
          $mask = join(".", map( hex $_, $mask =~ /(..)/gs ));
          compute_netmask($addr, $mask);
        $network = "$addr/$mask";
    }

    # 忽略 loopback 或沒啟動的介面
    $iface = "" if /\bLOOPBACK\b/o;
    $up++       if /\bUP\b/o;
}

die "Can't determine local IP address!\n" unless $LocalIP and
    $LocalNetwork;

# 設定 OS 相關的變數
#
my ( $GW, $NET, $PTP );
if ( $^O eq "linux" ) {
    $GW = "gw"; $PTP = "pointopoint"; $NET = "-net";
} else {
    $GW = $PTP = $NET = "";
}

# 分析 config 樣版
#
$template =~ s/(\$\w+)/$1/gee;

# 寫出暫存檔，並執行 vtund
#
if ($Debug) {
    print $template;
} else {
    print $file $template;
    close $file;
    system("$Vtund $name");
}

__DATA__

options {
   port $TunnelPort;
   ifconfig /sbin/ifconfig;
   route /sbin/route;
}
```

```
default {
   compress no;
   speed 0;
}

$TunnelName {
   type tun;
   proto tcp;
   keepalive yes;

   pass $Secret;

   up {
      ifconfig "%% $LocalIP $PTP $RemoteIP arp";
      route "add $TunnelHost $GW $GatewayIP";
      route "delete default";
      route "add default $GW $RemoteIP";
      route "add $NET $LocalNetwork $GW $GatewayIP";
   };

   down {
      ifconfig "%% down";
      route "delete default";
      route "delete $TunnelHost $GW $GatewayIP";
      route "delete $NET $LocalNetwork";
      route "add default $GW $GatewayIP";
   };
}
```

#53 自動產生 vtund.conf