Linux 2.4 Packet Filtering HOWTO

作者：Rusty Russell, mailing list netfilter@lists.samba.org
譯者：網中人 netmanforever@yahoo.com
v1.0.1 Mon May 1 18:09:31 CST 2000

此文件描述如何在 2.4 Linux kernel 上使用 iptables 對不良封包進行過濾。

目錄

1 Introduction

各位看官，歡迎到此一讀！

這裡，我假設您已經知道什麼是 IP 地址網路地址網路遮罩 (netmask)路由以及 DNS。否則，我建議您先讀一讀 Network Concepts HOWTO。

此 HOWTO 文件不止於是一個點到即止的介紹(讓您有點發熱和發毛，但又肉在砧上的感覺)，也不至於是一個抽絲剝繭式的原始大披露(讓您必有所獲，但不會神經錯亂走火入魔)。

您的網路其實一點也不安全。問題難點是在允許快速而便利通訊之同時，又想確保它只用於良好且非邪惡的意圖。其情形等同於在一個擁擠的戲院裡面，允許您高談闊論，但卻不能亂喊"著火啦！"一樣。這篇 HOWTO 不是用來解決這問題的。

所以，只有您才能決定何處才是折衷所在。我會嘗試指引您去使用一些可用工具，也會點出要留意的死穴，當然，同時希望您用於正途之上。又是(與上句)同等的問題。

2 官方的網站在哪裡？有郵件論壇嗎？

有三個官方網站非去不可:

- 感謝 *Filewatcher (http://netfilter.filewatcher.org)* `<http://netfilter.filewatcher.org>`.
- 感謝 *The Samba Team and SGI (http://www.samba.org/netfilter)* `<http://www.samba.org/netfilter>`.
- 感謝 *Jim Pick (http://netfilter.kernelnotes.org)* `<http://netfilter.kernelnotes.org>`.

至於官方的 netfilter 郵件論壇，請參閱：*Samba's Listserver (http://lists.samba.org)* <http://lists.samba.org>.

3 好了，什麼是封包過濾(Packer Filter)呢？

封包過濾就是用一個軟體查看所流經封包之表頭*(header)*，由此決定整個封包的命運。它或許會決定丟棄**(DROP)**這個封包(例如，忽略它就如根本沒收到它一樣)，或是接收**(ACCEPT)**這個封包(例如，讓這個封包通過)，或是其它更複雜的動作。

在 Linux 之下，封包過濾功能是內建於核心之內(做為一個核心模組，或者直接內建)，同時還有一些技巧我們可以運用於封包之上的，不過最慣用的依然是查看表頭以決定封包的命運。

3.1 我為什麼要封包過濾？

簡而言之：控制保安警戒。

Control:

當您用您的 Linux 主機將您的內部網路連接至其它網路(比方說，ineternet)的時候，您有機會允許特定類型的交通，而禁止其它的。例如，一個封包的表頭會包含封包的目的地地址，所以您可以防止封包流向外部網路的某一部份。再如，我用 Netscape 連線至 Dilbert archives，那網頁上有一個來自 doubleclick.net 的廣告，這樣 Netscape 會浪費我的時間去下載它們。只要讓封包過濾機制不允許任何來自doubleclick.net 的封包，我們就可以解決這個問題(當然，有更好的方法來做這件事情啦，請參考 Junkbuster)。

Security:

當您的 Linux 主機是您井然有序的內部網路和外面那個混沌無比的 internet 之間的唯一通道，而您知道可以限制哪些東西才能進入您的門戶，誠是不錯之舉吧。例如，您或許會放行任何從內部網路出去的東西，但又擔心來自外面的惡名昭章的'Ping of Death'。又如，您或許並不希望別人從外面 telnet 上您的 Linux 主機，盡管全部的帳號都有密碼保護。或許，您還想(正如大部份人一樣)在 internet 上當看客而不願當伺服器(也可能您是願意的)，最簡單莫如用封包過濾來拒絕任何意欲連線的封包，不讓任何人連進來。

Watchfulness:

有時候，一台設定差劣的機器會從本地網路向外面嘔送封包。而好消息是您可以讓封包過濾來告訴您是否有變態的事情發生。您或許會對之採取行動，又或許早已見慣不怪了。

3.2 如何在 Linux 下過濾封包？

Linux 的核心自從 1.1 版就已經有封包過濾功能。第一代是 1994 年由 Alan Cox 基於 BSD 的 ipfw 移植過來的，後來在 Linux 2.0 版本再由 Jos Vos 加强，利用'ipfwadm'這隻使用者空間(userspace *)工具來控制核心的過濾規則。在 1998 年年中，我在 Micahel Neuling 的大力幫助下，投注了相當的精力在 Linux 核心 2.2 上面，推出了'ipchains'這隻工具。終於，Linux 核心 2.4 的第四代工具'iptables'連同其它核心改寫也在 1999 年年中進行開發了。這就是目前這個 iptables 的 HOWTO 文件所致力之所在。

(* 譯者註："使用者空間"通常是用來區別系統記憶體的使用範圍，主要類型分為核心空間和使用者空間。原作者可能以為大家都是程式高手，故會用如此專門術語。然對一般讀者來說，理解上或許有困難，故此多說兩句。在往後的閱讀中也請留意。)

您需要一個核心有 netfilter 建構於其中：netfilter 是 Linux 核心中一個通用架構，可以讓其它東西(例如 iptables 模組) 插入(plug into)。換句話說，您需要核心 2.3.15 或更新的版本，同時在核心編譯時以，Y，回答 CONFIG_NETFILTER 這個選項。

`iptables` 這隻工具會和核心對講並告訴它什麼封包要過濾。除非您是一個程式人員，或異想天開，那您就是用它來控制封包怎樣過濾的了。

3.2.1 iptables

這隻 `iptables` 工具可以插入或移除核心封包過濾表格(packet filtering table) 中的一些規則(rules)。也就是說，無論您設定了什麼，要是重新啓動(reboot)系統的話，就會全部丟失；請參閱 3.2.2 (制定永久性規則(Making Rules Permanent))，看看如何確保設定在下次 Linux 啓動後可以回存。

`iptables` 是用來取代 `ipfwadm` 和 `ipchains` 的：請參閱 8 (使用 ipchains 和 ipfwadm (Using ipchains and ipfwadm))，看看如何無痛的避免使用 iptables，假如您目前正使用它們其中之一。

3.2.2 制定永久性規則

您目前的防火牆設定是儲存於核心裡面的，也正因如此，設定會在系統重啓後丟失。iptables-save 和 iptables-restore * 的撰寫目前已經被列入 TODO 列表中了。我保證當它們問世的時候，肯定非常棒。

(* 譯者註：在 ipchains 工具中，可以使用 ipchains-save 與 ipchains-restore 來把當前的防火牆設定儲存起來，以及將之還原。如果讀者沒使用過 ipchains 那兩個功能的話，或許不知道作者在說什麼。)

目前來說，就把設定規則所需的那些命令寫進一個初始命令稿(script)中吧。要確定的是，假如有其中一個命令失敗的時候，您能提供一些智能的動作反應 (通常如，exec /sbin/sulogin')。

4 你算哪根蔥啊？為何玩我的核心？

我是 Rusty，是 Linux IP 防火牆的維護者，同時也進行其它的一些編程工作，可以算是天時地利人和的使然吧。我寫過 ipchains (請參閱前面的 3.2 (如何在 Linux 下過濾封包？(How Do I Packet Filter Under Linux?))，看看實際的工作尚得益於哪些同仁)，從其中學到足夠的東西以匡正今次的封包過濾。我希望如此。

WatchGuard <http://www.watchguard.com> 是一個非常出色防火牆公司，出售真正好用的隨插式防火牆設備(plug-in Firebox)，且向我免費提供，讓我可以全力撰寫這些東西，以及維護過往的一些東西。我原本預估 6 個月就可以了，但實際上卻花了 12 個月，不過我在最後階段覺得做得還不錯就是了。多翻重寫硬碟毀壞手提電腦遭竊數個檔案系統的損毀以及後來的熒幕壞掉，最終，還是做出來了。

在這裡，我想澄清一些朋友的錯誤觀念：我並非核心(kenrl)裡面的專家。我之所以知道這些，是由於某些核心工作讓我接觸到他們其中的一些成員： David S. Miller Alexey Kuznetsov Andi Kleen Alan Cox。不過，豬頭骨(最難的)都由他們啃了，剩下的豆腐(安全和容易的)才輪到我來收拾啦。

5 真正的 Rusty 封包過濾快速指南

大部分人都僅用單一的 PPP 撥接上網，同時並不想任何人進入他們的網路或防火牆：

```
## Insert connection-tracking modules (not needed if built into kernel).
```

```
# insmod ip_conntrack
# insmod ip_conntrack_ftp

## Create chain which blocks new connections, except if coming from inside.
# iptables -N block
# iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
# iptables -A block -j DROP

## Jump to that chain from INPUT and FORWARD chains.
# iptables -A INPUT -j block
# iptables -A FORWARD -j block
```

6 封包如何穿越(traverse)過濾

核心從 'filter' 表格的三個列表(lists) 開始；這三個列表叫做 **firewall chains(**防火牆鏈**)** 或就叫 **chains(**鏈**)**。 這三個鏈分別為**INPUTOUTPUT** 和 **FORWARD**。

這跟 **2.0** 和 **2.2** 核心有很大差別哦！

對於 ASCII 藝術迷來說 ，各鏈(chains)的佈置如下：

```
                            -----
Incoming                   /     \         Outgoing
       -->[Routing ]--->|FORWARD|------->
          [Decision]     \_____/        ^
               |                        |
               v                       ----
              ___                     /    \
             /   \                  |OUTPUT|
            |INPUT|                  \____/
             \___/                      ^
               |                        |
                ----> Local Process ----
```

其中三個圈代表著前述的三個鏈 ，當一個封包抵達上圖中的其中一個圈 ，相應的鏈就會接受檢驗(examined) ，以決定那個封包的命運。如果鏈說 DROP 掉這個封包 ，那麼它就會就地正法 ，但如果鏈說 ACCEPT 這個封包 ，那麼它就繼續在圖示中穿越。

一個鏈(chain)其實就是眾多規則(rules)中的一個檢查清單(checklist)。每一條規則都會說“如果封包表頭看起來像這樣 ，就如此這般處置這個封包”。如果規則的設定和封包並不符合(match) ，那麼就交由鏈中的下一個規則繼續處理。而最終 ，如果再沒有規則可以參考 ，那麼核心就會看鏈的**policy(**原則**)** 以決定怎麼做。在一個安全至上的系統裡 ，原則(policy)通常都會告訴核心 DROP 掉該封包。

1. 當一個封包進入的時候(假設 ，通過 Ethernet 網路卡) ，核心首先看看封包的目的地(destination)：這稱之為 ' rouging (路由)'。
2. 如果目的地址為本機 ，這個封包就按圖示下行至 INPUT 鏈。如果它能夠通過 ，那麼等待這個封包的行程(processes)就將之接管下來。

3. 否則 ，如果核心並沒啓動轉遞功能(forwarding) ，或是它不知道如何轉遞這個封包 ，那麼這個封包就會被丟棄(dropped)。如果轉遞功能已經啓動 ，同時封包指向另一個網路界面(如果您還有另外一張) ，然後這個封包就按圖示右行至 FORWARD 鏈。如果它被接受(ACCEPT) ，那麼它就會被送出去。

4. 最後一種情形 ，一個在本機運行的程式會送出網路封包。封包就直接交給 OUTPUT 鏈：如果是 ACCEPT ，然後這個封包會繼續送出至它所指向的界面。

7 使用 iptables

如果您需要特定的詳細了解 ，iptables 有一個非常詳盡的 manual page (`man iptables`)。假如您熟悉 ipchains 的話 ，或許可以直接跳到 10 (iptables 與 ipchains 的差異 (Differences Between iptables and ipchains)) 去看；它們是非常近似的。

您還可以利用 `iptables` 做許多不同的事情哦。您所開始的那三個內建(buit-in) 鏈： `INPUT` `OUTPUT` 和 `FORWARD` ，您是不能刪除的。讓我們看看整個鏈的管理運作吧：

1. 建立一個新鏈 (-N)。
2. 刪除一個空鏈 (-X)。
3. 改變一個內建鏈的原則 (-P)。
4. 列出一個鏈中的規則 (-L)。
5. 清除一個鏈中的所有規則 (-F)。
6. 歸零(zero) 一個鏈中所有規則的封包字節(byte) 記數器 (-Z)。

有好些方法可以統籌一個鏈中的規則：

1. 延增(append) 一個新規則到一個鏈 (-A)。
2. 在鏈內某個位置插入(insert) 一個新規則(-I)。
3. 在鏈內某個位置替換(replace) 一條規則 (-R)。
4. 在鏈內某個位置刪除(delete) 一條規則 (-D)。
5. 刪除(delete) 鏈內第一條規則 (-D)。

7.1 當您的機器啓動時 ，您所看到的

iptables 可以做成模組(module) ，叫做 'iptable_filter.o' ，當您第一次跑 `iptables` 就會被自動載入。它也可以永久性的建置於核心裡面。

在跑任何 iptables 命令之前 (小心：有些套件(distributions) 或許會用它們的起始命令稿來跑 iptables) ，內建鏈('INPUT''FORWARD'和 'OUTPUT')將不帶任何規則 ，所有鏈都將原則設為 ACCEPT。您可以將 iptable_filter 模組選項設為 'forward=0' ，來改變預設的 FORWARD 鏈原則。

7.2 一個單一規則的運作

下面讓我們來熟練一下原則的運用吧 ，所謂熟能生巧是也。您最常用的或許會是 append (-A) 和 delete (-D) 命令。至於其它如 insert (-I) 和 replace (-R) ，只是這些概念的延伸而已。

一條規則都限定了一組條件(conditions)與特定封包比對 ，以及當它們符合時要如何處置(指一個'target')。比方說 ，您或許要丟棄所有來自127.0.0.1 這個 IP 地址的 ICMP 封包 ，因而我們這裡的條件就成為這樣：協定必須是 ICMP ，而來源地址必須是 127.0.0.1 ，而我們的 target(目標)將會是'DROP'。

我們稱 127.0.0.1 為 'loopback' 界面 ，就算您沒有真實的網路連接 ，您也會有這個界面的。您可以用 'ping' 這隻程式產生這樣的封包 (它只是送出一個 type 8(echo request)的 ICMP 封包 ，而所有樂於回應的合作端(cooperative hosts) 則送回一個 type 0(echo reply)的 ICMP 封包)。用來測試是很好用的。

```
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
# iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
#
```

這裡您可以看到第一個 ping 成功了(這裡的 '-c 1' 參數是告訴 ping 只送出一個封包)。

然後 ，我們為'INPUT' 延增(-A)一條規則 ，將來自 127.0.0.1('-s 127.0.0.1') 的 ICMP 協定 ('-p icmp') 封包送至 DROP 這個目標 (-j DROP)。

然後我們可以用第二個 ping 來測試我們的規則。在程式放棄繼續等待那些永不到來的回應之前 ，將有一段暫停。

我們有兩個方法可以移除規則。首先 ，因為我們目前制定在 input 鏈中只有唯一一條規則 ，所以我們可以指定數字來移除 ，例如：

```
# iptables -D INPUT 1
#
```

這樣就把第一條規則從 INPUT 鏈中移除掉。

第二個方法是映射(mirro)上面的 -A 命令 ，但用 -D 來代替 -A 而已。當您有一個鏈 ，裡面寫有非常複雜的規則 ，而又不想逐行數出第 37 行就是您要的那條規則 ，這時候 ，這方法就非常有用了。

```
# iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
#
```

在命令行中 ，其語法是 -D 必須和 -A (或 -I或 -R) 命令的位置一致。如果在同一個鏈中有數條相同的規則 ，那麼只有第一條會被移除掉。

7.3 過濾規格

我們已經看過用 '-p' 來指定協定 ，以及用 '-s' 來指定來源地址 ，但還有其它選項我們是可以用來指定出一個封包的特征。底下是一個完整的概述。

7.3.1 指定來源和目的地之 IP 地址

我們可以用四種方法來指定來源('-s'或'–source'或 '–src') 和目的地('-d'或'–destination'或'–dst') IP 地址。最常用的方法是使用完整名稱 ，例如 'localhost' 或 'www.linuxhq.com'。第二種方法是指定其 IP 地址 ，例如 '127.0.0.1'。

第三和第四種方法允許指定一組(group) IP地址 ，例如 '199.95.207.0/24' 或 '199.95.207.0/255.255.255.0' ，這兩個設定都指定了所有從 199.95.207.0 到 199.95.207.255 之間的 IP 地址；而在數字後面的 '/' 符號是告訴系統哪部份 IP 才有效。 '/32' 或 '/255.255.255.255' 為預設值(所有 IP 值都必須吻合)。全部用 '/0' 來指定 IP 地址也是可行的 ，例如：

```
[ NOTE: '-s 0/0' is redundant here. ]
# iptables -A INPUT -s 0/0 -j DROP
#
```

不過這非常少用 ，因為以上的效果和不指定 '-s' 毫無兩樣。

7.3.2 相反指定

許多旗標(flags) ，包括 '-s' (或 '–source')和 '-d' (或 '–destination') ，可以在它們前面放置一個 '!' 符號(發音為'not') ，來符合所有非(NOT)其賦予值的地址。比方說 ，'-s ! localhost' 符合所有非**(not)** 來自本機的封包。

7.3.3 指定協定

協定可以用 '-p' (或 '–protocol') 旗標來指定。協定可以為一個號碼(假如您知道 IP 協定數值的話) ，或是一個諸如 'TCP'或'UDP'或'ICMP' 這樣的名稱。大小寫沒關係 ，所以 'tcp' 和 'TCP' 都可以工作。

協定也可以加上一個 '!' 前置符號 ，使之相反。例如 '-p ! TCP' 則指定了所有非 TCP 的封包。

7.3.4 指定界面

我們用 '-i' (或 '–in-interface') 和 '-o' (或 '–out-interface') 選項來指定一個符合的界面**(interface)**。一個界面就是封包進入('-i') ，或傳出('-o')之物理設備。您可以用 `ifconfig` 命令列出哪些界面是跑起來('up')的。

穿越 `INPUT` 鏈的封包不會有傳出(output)界面的 ，所以 ，任何在鏈中使用 '-o' 選項的規則都不與之符合。同樣的 ，穿越 `OUTPUT` 鏈的封包也不會有傳入(input)界面 ，所以在鏈中任何帶 '-i' 選項的規則也是不符合的就是了。

僅僅是穿越 `FORWARD` 鏈的封包才會同時有傳入和傳出界面。

指定一個不存在的界面是完全合法(legal)的；反正在界面還沒起來之前 ，這條規則是不會符合的。這對於 PPP 撥接(通常會是`ppp0`) 或相類連線 ，就極之有用了。

例如在一個特殊例子中 ,界面是用一個 '+' 結尾的話 ,就泛指所有以此字串開頭的界面(不管它們目前是否起來了)。例如 ,要指定一條規則來符合所有的 PPP 界面的話 ,`-i ppp+` 選項就可以用上了。

面名稱前面可以用一個'!' 符號來符合一個與指定界面 不 符合的封包。

7.3.5 指定封包碎片 (Fragments)

有時候 ,一個封包會因為太大而不能一次過塞進連線去。當這樣的事情發生了 ,封包會被切割成 碎片 **(fragments)** ,同時會以多個封包來傳送。而另一端則重組這些碎片以還原整個封包。

但碎片的問題是 ,第一個起始碎片有整個封包表頭欄位(IP+TCPUDP和 ICMP)可供檢查 ,但後繼封包卻只包含表頭的小部份(不帶額外協定欄位的 IP)。這樣的話 ,要檢查後繼碎片之協定表頭(比方由 TCPUDP和 ICMP extensions 而成) ,就不可能了。

如果您要做連線追蹤或 NAT ,那所有碎片在遞給封包過濾碼之前都會匯合回一起 ,所以您無需擔心碎片問題。

然而 ,要弄明白過濾規則如何處理碎片的 ,就變得非常重要了。任何規則要詢問的資料而我們並沒有時 ,將被視為 不 符合。也就是說 ,第一個碎片封包的處理和其它封包一樣。但第二及之後的碎片就不是這樣了。這樣的話 ,一條 `-p TCP -sport www` (指定來源埠口為'www')的規則 ,將永遠不和碎片符合(除第一個碎片外)。相反的規則如`-p TCP -sport ! www` 也一樣就是了。

不過 ,您可以用 '-f' (or '–fragment') 旗標特別為第二及以後的碎片指定一條規則。在 '-f' 前面加上一個 '!' 來指定一條規則 不 適用於第二及以後碎片 ,也是可行的。

通常 ,讓第二及以後碎片通過是被視為安全的 ,因為如果過濾會影響第一個碎片的話 ,那麼也就可以避免在目標主機進行重組;但是 ,一些已知的臭虫顯示 ,丟送碎片封包可以輕易的讓主機當掉。那是閣下要應付的事情了。

網路玩家要留意的是:當進行這樣的檢測時 ,不完整的封包(太短的 TCPUDP和 ICMP 封包會讓火牆程式讀不到埠口或 ICMP 碼和類型) 會被丟棄。因此 ,TCP 碎片都由第 8 個位置開始的 *。

(* 譯者註:我也不是很明白作者這裡所指何物 ,原文是:'So are TCP fragments starting at position 8'。因為懶得去翻資料 ,故不知道 position 8 是指 TCP 表頭位置還是其它。假如您找到答案 ,歡迎寫信給我以作澄清。)

舉例來說 ,以下的規則會丟棄任何送給 192.168.1.1 的碎片。

```
# iptables -A OUTPUT -f -d 192.168.1.1 -j DROP
#
```

7.3.6 延伸 iptables :新的比對(matches)

`iptables` 是 可延伸的**(extensible)** ,也就是說 ,核心和 iptables 工具可以進行擴展以提供新的功能。

某些延伸(Extensions)是標準的 ,但有些則可以說是派生出來的。別的朋友或許會製做出一些延伸 ,同時散播給合適的用戶。

核心的延伸通常居於核心模組目錄內 ,例如 /lib/modules/2.3.15/net。假如您的核心是用 CONFIG_KMOD 設定來編譯的話 ,它們是應需求載入的 ,所以您無需手動的插入它們。

然而 ,iptables 程式的延伸則通常是居於 /usr/local/lib/iptables/ 裡面的分享函式庫 ,或者有些散播版本會將它們放進 /lib/iptables 或 /usr/lib/iptables 裡去。

延伸有兩個種類：新目標(target) ，和新比對(match)；下面我們就講講新目標吧。有些協定會自動提供新的測試(tests)：目前有 TCPUDP和 ICMP ，如下述。

命令後使用 '-p' 選項把延伸載入進來 ，您就可以來指定一個新測試了。當延伸選項允許的時候 ，使用 '-m' 來載入延伸 ，則可以明確指示一個新測試。

如需某個延伸的求助資料 ，可以使用選項後接 '-h' 或 '–help' 將之載入('-p' '-j'或 '-m') ，例如：

```
# iptables -p tcp --help
#
```

TCP 延伸 果指定了 '-p tcp' ，TCP 之延伸會自動載入的。它提供如下選項(並不符合 fragments)。

–tcp-flags

後接一個 '!' 選項 ，則有兩個旗標的字串讓您能夠對指定的 TCP 旗標進行過濾。 第一個字串是遮罩(mask)：一個您欲檢查的旗標列表。第二個字串是要説哪些東西要設定。例如：

```
# iptables -A INPUT --protocol tcp --tcp-flags ALL SYN,ACK -j DENY
```

這表示所有旗標都要檢查 ('ALL' 就是泛指 'SYN,ACK,FIN,RST,URG,PSH') ，但只有 SNY 和 ACK 被設定而已。另外有一個參數 'NONE' 則是沒旗標的意思。

–syn

為'–tcp-flags SYN,RST,ACK SYN' 的簡寫 ，其前面可以備選一個 '!' 符號。

–source-port

其後可以備選 '!' ，然後是一個單獨的 TCP 埠口或一個埠口值域(range)。埠口可以為 /etc/services 所列島埠口名稱 ，也可以是一個數字。如果是值域的話 ，可以是一對用':' 符號分隔的埠口名字 ，或一個埠口後面帶 ':' (指大於和等於該埠口) ，又或是一個埠口前面帶 ':' (指小於和等於該埠口)。

–sport

等同於 '–source-port'。

–destination-port

和

–dport

與上同 ，只是它們是用來指定目的地而非來源埠口加以比對。

–tcp-option

其後可以備選 '!' ，然後為一個數字 ，用來比對一個 TCP 選項等於該數字的封包。假如需要檢查 TCP 選項 ，那些 TCP 表頭不完整的封包就會自動的被丟棄。

一個 **TCP** 旗標的解釋 時候 ，允許單向而非雙向的 TCP 連線會很好用。例如 ，您或許想允許連線到外部 WWW 伺服器 ，但卻不想來自該伺服器的連線。

最幼稚的舉動或許會是擋掉來自該伺服器的 TCP 封包。但不幸的是 ，TCP 連線根本就要求封包是雙向傳遞的。

解決之道是把那些要求連線的封包擋掉。這些封包被稱為 **SYN** 封包(嗯 ,技術上講 ,它們是帶 SYN 設定的封包 ,而 FIN 和 ACK 標籤則是空白 ,只是我們將之簡稱為 SYN 封包而已)。要只限制這樣的封包的話 ,我們就可以制止那些外來的連線嘗試了。

'–syn' 旗標可以用於這些方面:它僅對那些指定為 TCP 協定的規則有作用。例如 ,指定來自 192.168.1.1 的 TCP 連線請求:

```
-p TCP -s 192.168.1.1 --syn
```

這旗標也可以後接一個 '!' 來反設 ,意指每一個非該類初始連線的封包。

UDP 延伸　果 '-p udp' 被指定的話 ,這些延伸就會自動載入。它提供了 '–source-port' '–sport' '–destination-port'以及 '–dport' 這些選項 ,一如前述的 TCP 設定。

ICMP 延伸　果 '-p icmp' 被指定的話 ,這個延伸就會自動載入。它只提供一個新的選項:

–icmp-type

其後可以備選 '!' ,然後是一個 icmp 名稱類型(如 'host-unreachable') ,或是一個數字類型(如 '3') ,或是一對用 '/' 分隔的數字類型和編碼(如 '3/3')。使用 '-p icmp –help' 就可以獲得一份可用 icmp 類型名稱清單。

其它比對的延伸　nerfilter 套件中的其它延伸則是展示性(demonstration)的延伸內容 ,可以用 '-m' 選項來呼叫(假如已安裝了的話)。

mac

此一模組必須要明確的用 '-m mac' 或 '–match mac' 來指定。它用於比對傳入封包的來源 Ethernet (MAC) 地址 ,因而只對那些穿越 PREROUTING 和 INPUT 鏈的封包起作用。它只提供一個選項:

–mac-source

其後可以備選 '!' ,然後是一個用冒號分隔的十六進制 ethernet 地址 ,如 '–mac-source 00:60:08:91:CC:B7'。

limit

這個模組必須明確的用 '-m limit' 或 '–match limit'來指定。它用來限制一個比對等級 ,諸如抑制記錄信息等。它只能比對一個每秒次數值(預設是每一個小時 3 個比對 ,伴隨 5 個觸發(burst))。它接受兩個備選參數:

–limit

後接一個數值;指定可允許的每秒最大平均比對數值。該數值可以用 '/second''/minute''/hour' 或 '/day'或其中部份 (故 '5/second' 和 '5/s' 是一樣的) ,來明確指定單位(unit) ,

–limit-burst

後接一個數值 ,指示出引起前述限制之前的最大觸發次數。

這個比對常用於 LOG 目標 ,以進行比率限制(rate-limited) 之記錄。為了更好了解它是如何工作的 ,讓我們看一看下面的規則 ,是以預設限制引數來記錄封包的:

```
# iptables -A FORWARD -m limit -j LOG
```

當此規則第一次引用的時候，封包就會被記錄下來；事實上，由於預設的觸發為 5，那為首的 5 個封包就會記錄下來。然後，再隔 20 分鐘此規則才會再記錄封包，而不管期間有多少個封包抵達。而且，每 20 分鐘如果沒有符合的封包通過，則會恢復 (regained) 一個觸發數值；假如 100 分鐘內再無這樣的封包觸及這規則的話，那麼觸發次數就會完全復原(recharged)；回到我們開始時的狀態。

註：您目前不能以大於 59 小時的復原時間來建立一個規則，故此，假如您設定一個平均率為每天一次，那麼，您的觸發率則一定要少於 3。

您也可以用這模組去避免以快速比率提昇服務回應的阻斷服務攻擊(DoS)。

Syn-flood protection：

```
# iptables -A FORWARD -p tcp --syn -m limit --limit 1/s -j ACCEPT
```

Furtive port scanner：

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,ACK,FIN,RST RST -m limit --limit 1/s -j ACCEPT
```

Ping of death：

```
# iptables -A FORWARD -p icmp --icmp-type echo-request -m limit --limit 1/s -j ACCEPT
```

此模組的工作原理有點像“截流閥”一樣，請參考下面的圖示。

```
          rate (pkt/s)
                ^        .---.
                |       / DoS \
                |      /       \
Edge of DoS  -|.....:.........\.......................
 = (limit *    |    /:          \
limit-burst)  |   / :           \         .-.
                |  /  :            \       /   \
                | /   :             \     /     \
End of DoS   -|/....:..............:.../.......\..../.
 = limit       |     :              :`-'         `--'
-------------+-----+--------------+------------------> time (s)
   LOGIC =>  Match | Didn't Match |    Match
```

比方說，我們以 5 個封包觸發來比對每秒一個封包，但封包從每秒四個開始傳入，持續三秒，然後等三秒再重新開始。

```
        <--Flood 1-->           <---Flood 2--->

Total  ^                  Line  __--      YNNN
Packets|              Rate  __--      YNNN
       |           mum  __--      YNNN
    10 |       Maxi __--         Y
       |         __--           Y
       |     __--              Y
       | __--    YNNN
```

```
    |-     YNNN
  5 |    Y
    |   Y                              Key:  Y -> Matched Rule
    |  Y                                     N -> Didn't Match Rule
    | Y
    |Y
  0 +-------------------------------------------------->  Time (seconds)
     0   1   2   3   4   5   6   7   8   9  10  11  12
```

您會發現頭五個封包被允許超過每秒一個封包，然後就引起限制了，如果有一個停歇，其它的觸發也將被允許，但就不能通過規則設定的最高比率(在該觸發使用後為每秒一個封包)。

owner

此模組為本機產生的封包比對不同特征的封包建立者(creator)。它僅對 OUTPUT 鏈有用，而且，甚至某些封包(如 ICMP ping responses)或許沒有 owner，將被視為不符合哦。

–uid-owner userid
如果封包由一個行程以有效(數字式) user id 建立的，則為符合。

–uid-owner groupid
如果封包由一個行程以有效(數字式) group id 建立的，則為符合。

–pid-owner processid
如果封包由一個行程以 process id 建立的，則為符合。

–sid-owner processid
如果封包由一個行程以 session group 建立的，則為符合。

unclean

此一實驗性模組必須以 '-m unclean' 或 '–match unclean' 來明確指定。它會對封包進行不同的隨機判斷檢測。這模組尚未被稽查過，所以不應該用於安全設備上(它或許會把事情搞砸，因為它本身或許有臭虫的)。它並沒提供選項設定。

The State Match 最有用的比對判斷標準由 'state' 延伸所提供，以詮釋 'ip_conntrack' 模組的連線追蹤分析。這是非常值得鼓勵使用的。

指定 '-m state' 則允許另一個額外的 '–state' 選項，可以為一個豆點分隔的比對陳述列表('!' 旗標指示不**(not)** 符合那些陳述)。這些陳述是：

NEW

一個建立新連線的封包。

ESTABLISHED

一個屬於現有連線(如：已經回應封包了)之封包。

RELATED

一個與現有連線相關，但卻並不限於其中部份的封包，諸如 ICMP 錯誤，或是建立 FTP 數據連線的封包(FTP 模組已插入)。

INVALID

一個因某些原因不能被鑒別的封包：這包括記憶體不足和不能回應任何已知連線的 ICMP 錯誤。通常，這樣的封包都會被丟棄掉。

7.4 目標(Target)規格

現在，我們知道可以對封包做什麼樣的檢查了，我們還需要一個方法來說出對一個符合我們測試的封包要做什麼樣動作。這就是所謂的一條規則之目標(**target**) 啦。

有兩個非常相類的內建目標：DROP 和 ACCEPT，我們已經接觸過了。如果一條規則符合一個封包，同時目標是其中之一，那麼就再沒有規則需要咨詢：封包的命運已經定下來了。

除了內建外，也有兩種類型的目標：延伸和用戶自定鏈。

7.4.1 用戶自定鏈

`iptables` 承襲了 `ipchains` 一個非常厲害的功能，就是讓使用者可以創建出新鏈，附加於三個內建鏈(INPUTFORWARD和 OUTPUT)之外。按慣例，用戶自定鏈用小寫以示區別(待會我們會在後面的 7.5 (在整鏈上運作(Operations on an Entire Chain)) 那裡解釋如何去建立新的用戶自定連)

當一個封包符合一條目標為用戶自定鏈之規則時，封包就會開始穿越用戶自定鏈中的規則。假如該鏈未能決定出封包的命運，則一旦結束穿越該鏈後，就會接著當前鏈中的下一個規則繼續穿越下去。

繼續玩玩 ASCII 藝術好了。假設有這麼兩條(怪)鏈：`INPUT` (內建鏈)，和 `test` (用戶自定鏈)。

```
 `INPUT'                         `test'
----------------------------    ----------------------------
| Rule1: -p ICMP -j DROP   |    | Rule1: -s 192.168.1.1    |
|--------------------------|    |--------------------------|
| Rule2: -p TCP -j test    |    | Rule2: -d 192.168.1.1    |
|--------------------------|    ----------------------------
| Rule3: -p UDP -j DROP    |
----------------------------
```

假設一個來自192.168.1.1 的 TCP 封包，要到 1.2.3.4 那裡去。它進入`INPUT` 鏈，並受到 Rule1 的測試 - 但不符合。但是符合 Rule2，且它的目標是 `test`，所以下一個要檢驗的規則將從 `test` 開始。在 `test` 中的 Rule1 符合，但並沒有指定目標，所以再檢驗下一條規則，也就是 Rule2。不過它並不符合，所以我們已經抵達這條鏈的末端了。然後我們回到 `INPUT` 鏈中，也就是我們剛才檢驗 Rule2 那裡，所以我們現在就要檢查 Rule3，依然不符合。

這樣，該封包的路徑是這樣子的：

```
                        v    __________________________
 `INPUT'                |   /    `test'                v
------------------------|--/    -----------------------|----
| Rule1                 | /|    | Rule1                |   |
|-----------------------|/-|    |----------------------|---|
| Rule2                 /  |    | Rule2                |   |
|--------------------------|    -----------------------v----
| Rule3                 /--+___________________________/
------------------------|---
                        v
```

用戶自定鏈也可以再跳到另一個用戶自定鏈去(但不要做成迴圈：您的封包如果被發現處於迴圈中就會被丟棄)。

7.4.2 iptables 之延伸：新目標

另一類型的目標是一個延伸。一個目標的延伸由核心模組和可選的 `iptables` 延伸組成，以提供新的命令行選項。在預設的 netfilter 散播版本中有好幾個延伸：

LOG

此模組提供核心記錄符合的封包。它提供這些額外選項：

–log-level

後接一個層次(level)號碼或名稱。合法的名稱有(大小寫有別)：'debug''info''notice''warning''err''crit''alert'以及 'emerg'，相對的號碼由 7 到 0。各層次號碼的解釋請參考 syslog.conf 的 man page。

–log-prefix

後接一個最多 30 個字母的字串。此一信息由記錄信息開始時送出，令其可以個別的被鑒別出來。

此模組常用於一個限制目標後，所以，您不要灌爆您的記錄檔哦。

REJECT

此模組除了向發送端送出一個 'port unreachable' 這樣的 ICMP 錯誤外，和 'DROP' 是一樣的。註：在下列條件中，ICMP 錯誤信息將不會送出(請參考 RFC 1122)：

- 被過濾的封包一開始就是一個 ICMP 錯誤信息，或是其它不明的 ICMP 類型。
- 被過濾的封包為一個無頭 (non-head) 碎片。
- 我們目前已經送出太多至該目的地的 ICMP 錯誤信息了。

REJECT 另外還接受一個 '–reject-with' 選項來更改其回應封包：請參考說明文件。

7.4.3 特殊的內建目標

有兩種特殊的內建目標：`RETURN` 和 `QUEUE`。

`RETURN` 和掉到一個鏈的末端有相同的效果：對一條內建鏈的規則而言，則啓用該鏈的原則。對一條用戶自定規則而言，則會回到前一個鏈中繼續穿越，就接在跳到這個鏈的那條規則之後。

`QUEUE` 也是一個特殊目標，可以替使用者空間(userspace)行程儲列封包。要運用它，兩個功能組件是必需的：

- 其一為 "queue handler"，處理使用者空間與核心之間傳送封包的實質機制；
- 另一個為一使用者空間的應用程式，去接收，或說操控，以及對封包做出裁決。

IPv4 iptables 的標準 queue handler 為 ip_queue 模組，它目前是以實驗性質與核心一起發佈的。

如下是一個如何用 iptables 為使用者空間行程進行儲列封包的簡單例子：

```
# modprobe iptable_filter
# modprobe ip_queue
# iptables -A OUTPUT -p icmp -j QUEUE
```

用此規則，本機產生的對外 ICMP 封包(如用 ping 建立) 就會被送至 ip_queue 模組去，然後嘗試將封包傳給使用者空間應用程式。如果沒有使用者空間應用程式在等待的話，該封包就會被丟棄。

要寫一個使用者空間應用程式，需使用 libipq API。它也是和 iptables 一起發佈的。程式碼範例可以在 CVS 中的 testsuite 工具(如 redirect.c) 找到。

ip_queue 的狀態可以用如下方法來檢查：

```
/proc/net/ip_queue
```

儲列的最大長度(如傳遞給使用者空間且無需送回裁決封包之數量)可以通過這樣的方式來控制：

```
/proc/sys/net/ipv4/ip_queue_maxlen
```

最大儲列長度的預設值為 1024。一旦達到此限制，新的封包就會被丟棄，直到儲列長度跌回低於限制之數為止。好的協定，如 TCP，會將丟棄的封包解釋為擁擠(congestion)，同時理想地，當儲列填起來後會將之擋回去。然而，如果預設值在所舉情形下覺得太小的話，或許需要一些實驗來決定其理想的最高儲列長度。

7.5 在整鏈上運作

`iptables` 的一個非常有用的功能是，它能夠組合(group)相關的規則於鏈中。只要您喜歡，您可以隨便為鏈起一個名字，但我建議您使用小寫字母以避免和內建鏈及目標搞混了。鏈名最長可以去到 31 個字母。

7.5.1 建立一個新鏈

現在就讓我們一起建一個新鏈吧。因為我實在是一個愛幻想的傢伙，所以我稱之為`test` (哈，有點諷刺)。這裡，我們用 '-N' 或 '–new-chain' 選項：

```
# iptables -N test
#
```

就是這麼簡單。好了，現在您可以將一些規則加入其中，一如前面說的那樣。

7.5.2 刪除一條鏈

要刪除一條鏈也是一樣簡單，用 '-X' 或 '–delete-chain' 即可。為什麼用 '-X' 呢？嗯，好用的字母都一早給用光了啦。

```
# iptables -X test
#
```

要刪除一條鏈的話，會有好些限制：它們必需是空的 (請參考後面的 7.5.3 (清空一條鏈(Flushing a Chain)))，同時它們必需不能作為任何規則的目標。任何三條內建鏈您都不能刪除就是了。

假如您不指定一條鏈，那麼如果可能的話，全部 用戶自定點鏈都會被刪除。

7.5.3 清空一條鏈

有一個簡單的方法可以清空一條鏈中的所有規則，就是使用 '-F' (或 '–flush') 命令。

```
# iptables -F forward
#
```

如果您不指定是哪一條鏈，那麼 全部 鏈都會被清空。

7.5.4 列示一條鏈

您可以使用 '-L' (或 '–list') 命令列示一條鏈中的所有規則。

每一個用戶自定鏈所列的 'refcnt'，是說有多少數目的規則是以該鏈為目標的。在該鏈被刪除之前，這數目必需為零(同時鏈是空的)。

如果沒提供鏈名稱的話，所有鏈都會被列示出來，就算空鏈也一樣。

有三個選項可以伴隨 '-L' 一起使用的。首先是 '-n' (numeric) 選項，它很有用，因為它可以避免 `iptables` 去嘗試查找 IP 地址，假如您的 DNS 沒有設定正確的話，或是您已經過濾掉 DNS 請求了，這或許會造成嚴重的延遲(假設您和大多數人一樣都是使用 DNS)。它同時也會將 TCP 與 UDP 埠口顯示為數字而非名稱。

第二個是 '-v' 選項，它會顯示出您全部規則的細節，諸如封包的 byte 流量統計TOS 比較以及界面等。否則這些數值是被略掉的。

註：封包的 byte 流量統計可以分別使用 'K', 'M' 或 'G' 這些字尾，分別代表 1000 1,000,000 以及 1,000,000,000，來顯示。使用 '-x' (expand numbers) 旗標同樣也可以顯示出完整的數字，根本不理會它們有多長。

7.5.5 重設(歸零)流量記數器(counter)

能夠重設流量記數器當然是有用的。您可以用 '-Z' (或 '–zero') 選項來做。

唯一麻煩是，有時候在進行重設之前，您必需立即記住流量統計值。在前面的例子中，當您下 '-L' 然後 '-Z' 命令，某些封包可能會在這期間通過。因此，您可以把 '-L' 和 '-Z' 一起 使用，在讀取的同時進行記數器重設。

7.5.6 設定原則(policy)

我們在前面探討封包如何通過一個鏈的時候，已詮釋過當封包抵達內建鏈末端時將會發生什麼事情。此時，就由該鏈的原則來決定封包的命運。只有內建鏈(`INPUT``OUTPUT`以及 `FORWARD`) 才有原則設定，因為，如果一個封包掉至一個用戶自定鏈的時候，則會回到上一個鏈中繼續穿越。

原則可以為 `ACCEPT` 或 `DROP`。

8 使用 ipchains 與 ipfwadm

在 netfilter 套件中，有兩個模組分別叫做 ipchains.o 和 ipfwadm.o。您只要將其中一個插入進核心裡面(註：它們和 iptables.oip_conntrack.o 及 ip_nat.o 是不兼容的！)。然後您就可以如往常一般使用

ipchains 或 ipfwadm 了。

在一定時期內這仍會被支持的。我認為合理的計算公式是：2 * [替代產品發佈 - 初始穩定發行] ，再加上替代產品可以真正穩定發行的日子。

換而言之，對 ipfwadm 的最後支持將會延至：

```
2 * [October 1997 (2.1.102 release) - March 1995 (ipfwadm 1.0)]
        + January 1999 (2.2.0 release)
    = November 2003.
```

而對 ipchains 的最後支持則為：

```
2 * [August 1999 (2.3.15 release) - October 1997 (2.2.0 release)]
        + July 2000 (2.4.0 release?)
    = March 2004.
```

所以，在 2004 年之前都可以高枕無憂啦。

9 整合 NAT 與 Packet Filtering

要做 Network Address Translation (請參閱 NAT HOWTO) 以及封包過濾，已是很平常之事了。好消息是，將它們混合起來使用實是完全沒問題的。

當你設計封包過濾的時候，可以完全不用理會您要做怎樣的 NAT。於封包過濾中看到的來源與目的地，只會是 'real' 來源和目的地。舉例來說，如果您做 NAT，要將所有連到 1.2.3.4 port 80 的連線送到 10.1.1.1 port 8080 去，這樣封包過濾會看那些送到 10.1.1.1 port 8080 (真正的目的地)，而不是 1.2.3.4 port 80。類似的，您也可以忽略封包偽裝：封包會看起來是來自真正的內部 IP 地址(比方 10.1.1.1)，回應也看起來送回那裡。

您可以運用 'state' 比對延伸(match extension)而無需讓封包過濾做額外的工作，因為無論如何，NAT 都會要求連線追蹤。為了增强在 NAT HOWTO 裡面那個簡單的封包偽裝例子，去擋掉來自 ppp0 界面的任何新連接，您可以這樣做：

```
# Masquerade out ppp0
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE

# Disallow NEW and INVALID incoming or forwarded packets from ppp0.
iptables -A INPUT -i ppp0 -m state --state NEW,INVALID -j DROP
iptables -A FORWARD -i ppp0 0 -m state --state NEW,INVALID -j DROP

# Turn on IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward
```

10 iptables 與 ipchains 的差異

- 首先，內建鏈名稱從小寫還換成大寫，因為 INPUT 與 OUTPUT 鏈目前只會抓目標為本機以及從本機產生的封包。它們分別用來查看傳入與傳出的封包。

- 現在有一個 '-i' 旗標來代表傳入界面 ，並且只工作於 INPUT 和 FORWARD 鏈中。在 FORWARD 與 OUTPUT 鏈中就要將 '-i' 改成 '-o' 了。
- TCP 與 UDP 埠口現在都要用 –source-port 或 –sport 選項來拼寫出來(或是掉過來寫 –destination-port 或 –dport) ，同時 ，必需置於 '-p tcp' 或 '-p udp' 選項之後 ，因為 TCP 或 UDP 延伸是分開載入的。
- 以前 TCP 那個 -y 現在變成 –syn ，並且必需置於 '-p tcp' 之後。
- 原來的 DENY 目標現在終於變成 DROP 了。
- 在列示其工作的同時可以將該鏈歸零(zeroing)。
- 歸零內建鏈也可以清掉原則記數器(policy counters)。
- 列示鏈可以讓您把記數器變成微小快照(atomic snapshot)。
- REJECT 與 LOG 現在變成延伸目標了 ，意味著它們已經和核心模組分開。
- 鏈名稱最長可達 31 個字母。
- MASQ 現在變成 MASQUERADE ，而且使用不同的語法。REDIRECT 在保留相同名稱的同時 ，也經歷了語法的變遷。至於如何設定它們的詳細資料 ，請參閱 NAT-HOWTO。
- 而 -o 選項則不再用來將封包傳遞給使用者空間設備了(參考前面的 -i)。現在則用 QUEUE 目標將封包送給使用者空間。
- 哦 ，我可能已記不得那麼多了。

11 關於設計封包過濾的建議

在電腦安全戰場上最明智之舉莫過於先擋掉一切 ，然後開放必需的。有一句至理名言是：'非請勿進'。我建議您牢記於心 ，假如您最注重安全的話。

不要跑那些您用不到的服務 ，不管您是否以為已經將之擋下來了。

如果您要建立一個指定式防火牆(dedicated firewall) ，開始不要跑任何東西 ，同時擋掉所有封包 ，然後增加服務以及讓所需的封包通過。

我特別强調安全性：結合 tcp-wrappers(對於封包過濾本身的連接)服務代理(對於通過封包過濾的連接)路由驗證以及封包過濾等手段。路由驗證是指 ，那些來自未預期界面的封包就會被丟棄：舉例說 ，如果您的內部網路有一段 10.1.1.0/24 的地址 ，同時有一個來自該地址的封包卻從外部界面進入 ，那它就會被丟棄掉。它可以為一個界面(如 ppp0) 設起來 ，如：

```
# echo 1 > /proc/sys/net/ipv4/conf/ppp0/rp_filter
#
```

或是全部現有及將有的界面 ，如：

```
# for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
#     echo 1 > $f
# done
#
```

Debian 在可能之下預設就會如此了。如果您有不對稱路由(例如，您預期封包會從其它方向進入)，您應該在那些界面上關閉此一過濾。

定防火牆的時候，假如有某些東西不工作的話，記錄功能就顯得很有用了；但在一個實際運作的防火牆上，任何時候都要將它結合 'limit' 比對來一起使用，以避免有人灌爆您的記錄檔。

我強烈建議對安全系統做連線追蹤：它雖然會引致一些負擔(因為所有連線都要追蹤)，但對於貴網路的連接控制卻很有用。如果您的核心不會自動載入模組的話，您或許需要載入'ip_conntrack.o' 模組。假如您要精確追蹤複雜的協定，您還需要載入合適的 helper 模組(如，'ip_conntrack_ftp.o')。

```
# iptables -N no-conns-from-ppp0
# iptables -A no-conns-from-ppp0 -m state --state ESTABLISHED,RELATED -j ACCEPT
# iptables -A no-conns-from-ppp0 -m state --state NEW -i ! ppp0 -j ACCEPT
# iptables -A no-conns-from-ppp0 -i ppp0 -m limit -j LOG --log-prefix "Bad packet from ppp0:"
# iptables -A no-conns-from-ppp0 -i ! ppp0 -m limit -j LOG --log-prefix "Bad packet not from ppp0:"
# iptables -A no-conns-from-ppp0 -j DROP

# iptables -A INPUT -j no-conns-from-ppp0
# iptables -A FORWARD -j no-conns-from-ppp0
```

建置一個良好的防火牆已經超出這個 HOWTO 的範圍了，但我的建議是：'一切從嚴(always be minimalist)'。對於在您機器上進行測試與探索的更多資料，就要參考 Security HOWTO 了。