Netfilter Extensions HOWTO

Fabrice MARIE <fabrice@netfilter.org>, mailing list `netfilter-devel@lists.samba.org` $Revision: 3822 $ $Date: 2005-04-03 11:03:46 +0200 (Sun, 03 Apr 2005) $

This document describes how to install and use current iptables extensions for netfilter.

Contents

1 Introduction

Hello. This is a great opportunity for me to thank all the people spending a lot of time developing, testing, reporting bugs of, and using netfilter. So, thanks to you all !!

This HOWTO assumes you have read and understood Rusty's

Linux 2.4 Packet Filtering HOWTO `http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html`
. It is assumed as well that you know how to compile and install a kernel properly.

`iptables` distribution contains extensions that are not used by regular users or that are still quite experimental or finally, that are pending for kernel inclusion. These extensions are usually not compiled, unless you've asked for it.

You should find the latest version of this document on

netfilter documentation `http://www.netfilter.org/documentation/index.html#documentation-howto`
web page.

The goal of this HOWTO is to help people get started with the netfilter extensions by explaining how you can install them, and how to basically use them.

Finally, there are a few script generated complete lists of patches available in patch-o-matic:

- *Patch-o-matic pending repository summary* `http://www.netfilter.org/patch-o-matic/pom-pending.html`
  .
- *patch-o-matic base repository summary* `http://www.netfilter.org/patch-o-matic/pom-base.html`
  .
- *patch-o-matic extra repository summary* `http://www.netfilter.org/patch-o-matic/pom-extra.html`
  .

(C) 2001-2004 Fabrice MARIE. Licensed under the GNU GPL.

2 Patch-O-Matic

2.1 What is Patch-O-Matic ?

Netfilter developers distribute a set of patches that they package so that it can be used by their 'patch-o-matic-ng' (or 'p-o-m') system. p-o-m is a script that guides you through the process of choosing/selecting the patches you want to apply, and automatically patch the kernel for you.

First, you should get the latest SVN tree, to be sure that you are using the latest extensions. To do so, perform :

```
# mkdir netfilter_svn
# cd netfilter_svn
```

```
# svn co https://svn.netfilter.org/netfilter/trunk/iptables
# svn co https://svn.netfilter.org/netfilter/trunk/patch-o-matic-ng
```

This will will check out the main iptables package and the whole patch-o-matic-ng repository:

```
# ls -l netfilter_svn/
total 3
drwxr-xr-x    9 root     root          864 Nov  7 14:48 iptables/
drwxr-xr-x   13 root     root          488 Nov  7 14:54 patch-o-matic-ng/
```

Make sure your kernel source is ready in '/usr/src/linux/'. If for whatever reason the kernel you want to patch is not in '/usr/src/linux/' then you can make the variable KERNEL_DIR point to the patch where your kernel is :

```
# export KERNEL_DIR=/the/path/linux
```

Make sure the dependencies are made already. If unsure :

```
# cd /usr/src/linux/
# make dep
```

Then you can go back to the netfilter directory, in the 'patch-o-matic/' directory. You can now invoke p-o-m.

2.2 Running Patch-O-Matic

While in the 'patch-o-matic-ng/' directory, let's run p-o-m :

```
# ./runme extra

Welcome to Patch-o-matic ($Revision: 3822 $)!

Kernel:

Each patch is a new feature: many have minimal impact, some do not.
Almost every one has bugs, so don't apply what you don't need!
-------------------------------------------------------
Already applied: nf-log

Testing... not applied
The HOPLIMIT patch:
   Author: Maciej Soltysiak <solt@dns.toxicfilms.tv>
   Status: Works for me.

   This allows the user to set the IPv6 Hop Limit value on a packet or
   to increment or decrement it by a given value.

   Example:
   # ip6tables -t mangle -A OUTPUT -j HOPLIMIT --hl-inc 1
   # ip6tables -t mangle -A INPUT -j HOPLIMIT --hl-eq 64
   # ip6tables -t mangle -A OUTPUT -j HOPLIMIT --hl-dec 2
-----------------------------------------------------------------
Do you want to apply this patch [N/y/t/f/a/r/b/w/q/?]
```

p-o-m will go through most of the patches. If they are already applied, you will see so on the 'Already applied:' line. Otherwise it will prompt you to decide whether or not to patch it.

- Simply press enter if you do not want to apply it.
- Type 't' to test that the patch will apply cleanly.
- Type 'y' to apply patch.
- Type 'n' to skip this patch.
- Type 'f' to apply patch even if test fails.
- Type 'a' to restart patch-o-matic in apply mode.
- Type 'r' to restart patch-o-matic in REVERSE mode.
- Type 'b' to walk back one patch in the list.
- Type 'w' to walk forward one patch in the list.
- Finally, type 'q' to quit immediately.

A rule of thumb is to read carefully the little explanation text of each patch before actually applying it. As there are currently a LOT of official patches for patch-o-matic (and probably more unofficial ones), it is not recommended to apply them all ! You should really consider applying only the ones you need, even if it means recompiling netfilter when you need more patches later on.

Patch-o-matic's 'runme' is a nice Perl script. It optionally takes some command line arguments:

```
# ./runme --help
Usage:
    ./runme [--batch] [--reverse] [--exclude suite/patch-dir ]
    suite|suite/patch-dir

Options:
    --batch batch mode, automatically applying patches.

    --test  test mode, automatically test patches.

    --check check mode, automatically checks if patches are alreay applied.
            produces a logfile: rune.out-check

    --reverse
            back out the selected patches.

    --exclude suite/patch-dir
            excludes the named patch. can be used multiple times.

    --help  print a help message

    --man   print the whole manpage
```

Each patch has its own directory within the main directory 'patch-o-matic-ng'. Each patch is attached to a suite of patches. The three existing suites (as of the date of update of this HOWTO) are:

- pending

- base
- extra

You can instruct p-o-m to show you only patches attached to a particular suite:

```
./runme --batch pending
```

This command will show you and propose you to apply all the patches from pending suite. When you instruct './runme' to apply patches from the 'extra' patch repository it will first present you with the patches from the 'pending', and 'base' repositories.

2.3 So what's next ?

Once you have applied all the patches you wished to apply, the next step is recompile your kernel and install it. This HOWTO will not explain how to do this. Instead, you can read the *Linux Kernel HOWTO* http://en.tldp.org/HOWTO/Kernel-HOWTO.html .

While configuring your kernel, you will see new options in "Networking Options -> Netfilter Configuration". Choose the options you need, recompile & install your new kernel.

Once your new kernel is installed, you can go ahead and compile and install the "iptables" package, from the 'iptables/' directory as follows :

```
# make && make install
```

That's it ! Your new shiny iptables package is installed ! Now it's time to use these brand new functionalities.

3 New netfilter matches

In this section, we will attempt to explain the usage of new netfilter matches. The patches will appear in alphabetical order. Additionally, we will not explain patches that break other patches. But this might come later.

Generally speaking, for matches, you can get the help hints from a particular module by typing :

```
# iptables -m the_match_you_want --help
```

This would display the normal iptables help message, plus the specific "the_match_you_want" match help message at the end.

3.1 ah-esp patch

This patch by Yon Uriarte <yon@astaro.de> adds 2 new matches :

- "ah" : lets you match an AH packet based on its Security Parameter Index (SPI).
- "esp" : lets you match an ESP packet based on its SPI.

This patch can be quite useful for people using IPSEC who are willing to discriminate connections based on their SPI.

For example, we will drop all the AH packets that have a SPI equal to 500 :

```
# iptables -A INPUT -p 51 -m ah --ahspi 500 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       ipv6-auth--  anywhere             anywhere           ah spi:500
```

Supported options for the ah match are :

–ahspi [!] spi[:spi]

-> match spi (range)

The esp match works exactly the same :

```
# iptables -A INPUT -p 50 -m esp --espspi 500 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       ipv6-crypt--  anywhere             anywhere           esp spi:500
```

Supported options for the esp match are :

–espspi [!] spi[:spi]

-> match spi (range)

Do not forget to specify the proper protocol through "-p 50" or "-p 51" (for esp & ah respectively) when you use the ah or esp matches, or else the rule insertion will simply abort for obvious reasons.

3.2 condition match

This patch by Stephane Ouellette <ouellettes@videotron.ca> adds a new match that is used to enable or disable a set of rules using condition variables stored in '/proc' files.

Notes:

- The condition variables are stored in the '/proc/net/ipt_condition/' directory.
- A condition variable can only be set to "0" (FALSE) or "1" (TRUE).
- One or many rules can be affected by the state of a single condition variable.
- A condition proc file is automatically created when a new condition is first referenced.
- A condition proc file is automatically deleted when the last reference to it is removed.

Supported options for the condition match are :

–condition [!] conditionfile

-> match on condition variable.

For example, if you want to prohibit access to your web server while doing maintenance, you can use the following :

```
# iptables -A FORWARD -p tcp -d 192.168.1.10 --dport http -m condition --condition webdown -j REJECT --reject-with tcp-reset

# echo 1 > /proc/net/ipt_condition/webdown
```

The following rule will match only if the “webdown” condition is set to “1”.

3.3 conntrack patch

This patch by Marc Boucher <marc+nf@mbsi.ca> adds a new general conntrack match module (a superset of the state match) that allows you to match on additional conntrack information.

For example, if you want to allow all the RELATED connections for TCP protocols only, then you can proceed as follows :

```
# iptables -A FORWARD -m conntrack --ctstate RELATED --ctproto tcp -j ACCEPT

# iptables --list
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere           ctstate RELATED
```

Supported options for the conntrack match are :

[!] –ctstate [INVALID|ESTABLISHED|NEW|RELATED|SNAT|DNAT][,...]

-> State(s) to match. The "new" ‘SNAT’ and ‘DNAT’ states are virtual ones, matching if the original source address differs from the reply destination, or if the original destination differs from the reply source.

[!] –ctproto proto

-> Protocol to match; by number or name, eg. ‘tcp’.

–ctorigsrc [!] address[/mask]

-> Original source specification.

–ctorigdst [!] address[/mask]

-> Original destination specification.

–ctreplsrc [!] address[/mask]

-> Reply source specification.

–ctrepldst [!] address[/mask]

-> Reply destination specification.

[!] –ctstatus [NONE|EXPECTED|SEEN_REPLY|ASSURED][,...]

-> Status(es) to match.

[!] –ctexpire time[:time]

-> Match remaining lifetime in seconds against value or range of values (inclusive).

3.4 fuzzy patch

This patch by Hime Aguiar e Oliveira Jr. <hime@engineer.com> adds a new module which allows you to match packets according to a dynamic profile implemented by means of a simple Fuzzy Logic Controller (FLC).

This match implements a TSK FLC (Takagi-Sugeno-Kang Fuzzy Logic Controller). The basic idea is that the match is given two parameters that tell it the desired filtering interval.

- When the packet rate is below 'lower-limit' the rule will never match.
- Between 'lower-limit' and 'upper-limit', matching will occurs according a increasing (mean) rate.
- Finally, when the packet rate comes to 'upper-limit', (mean) matching rate attains its maximum value, 99%.

Taking into account that the sampling rate is variable and is of approximately 100ms (on a busy machine), the author believes that the module presents good responsiveness, adapting fast to changing traffic patterns.

For example, if you wish to avoid Denials Of Service, you could use the following rule:

```
iptables -A INPUT -m fuzzy --lower-limit 100 --upper-limit 1000 -j REJECT
```

- Below the 100 pps (packets per second) rate, the filter is inactive.
- Between 100 and 1000 pps the mean acceptance rate drops from 100% (when we are at 100 pps) to 1% (when we are at 1000 pps).
- Above 1000 pps the acceptance rate keeps constant at 1%.

Supported options for the fuzzy patch are :

–upper-limit n

-> Desired upper bound for traffic rate matching.

–lower-limit n

-> Lower bound over which the FLC starts to match.

3.5 iplimit patch

This patch by Gerd Knorr <kraxel@bytesex.org> adds a new match that will allow you to restrict the number of parallel TCP connections from a particular host or network.

For example, let's limit the number of parallel HTTP connections made by a single IP address to 4 :

```
# iptables -A INPUT -p tcp --syn --dport http -m iplimit --iplimit-above 4 -j REJECT

# iptables --list
Chain INPUT (policy ACCEPT)
target   prot opt source    destination
REJECT   tcp  --  anywhere  anywhere     tcp dpt:http flags:SYN,RST,ACK/SYN #conn/32 > 4 reject-with icmp-port-unreachable
```

Or you might want to limit the number of parallel connections made by a whole class A for example :

```
# iptables -A INPUT -p tcp --syn --dport http -m iplimit --iplimit-mask 8 --iplimit-above 4 -j REJECT

# iptables --list
Chain INPUT (policy ACCEPT)
target   prot opt source    destination
REJECT   tcp  --  anywhere  anywhere     tcp dpt:http flags:SYN,RST,ACK/SYN #conn/8 > 4 reject-with icmp-port-unreachable
```

Supported options for the iplimit patch are :

[!] –iplimit-above n

-> match if the number of existing tcp connections is (not) above n

–iplimit-mask n

-> group hosts using mask

3.6 ipv4options patch

This patch by Fabrice MARIE <fabrice@netfilter.org> adds a news match that allows you to match packets based on the IP options they have set.

For example, let's drop all packets that have the record-route or the timestamp IP option set :

```
# iptables -A INPUT -m ipv4options --rr -j DROP
# iptables -A INPUT -m ipv4options --ts -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere            IPV4OPTS RR
DROP       all  --  anywhere             anywhere            IPV4OPTS TS
```

Supported options for the ipv4options match are :

–ssrr

-> match strict source routing flag.

–lsrr

-> match loose source routing flag.

–no-srr

-> match packets with no source routing.

[!] –rr

-> match record route flag.

[!] –ts

-> match timestamp flag.

[!] –ra

-> match router-alert option.

[!] –any-opt

-> Match a packet that has at least one IP option (or that has no IP option at all if ! is chosen).

3.7 length patch

This patch by James Morris <jmorris@intercode.com.au> adds a new match that allows you to match a packet based on its length.

For example, let's drop all the pings with a packet size greater than 85 bytes :

```
# iptables -A INPUT -p icmp --icmp-type echo-request -m length --length 86:0xffff -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             anywhere           icmp echo-request length 86:65535
```

Supported options for the length match are :

[!] –length length[:length]

-> Match packet length against value or range of values (inclusive)

Values of the range not present will be implied. The implied value for minimum is 0, and for maximum is 65535.

3.8 mport patch

This patch by Andreas Ferber <af@devcon.net> adds a new match that allows you to specify ports with a mix of port-ranges and single ports for UDP and TCP protocols.

For example, if you want to block ftp, ssh, telnet and http in one line, you can :

```
# iptables -A INPUT -p tcp -m mport --ports 20:23,80 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere           mport ports ftp-data:telnet,http
```

Supported options for the mport match are :

–source-ports port[,port:port,port...]

-> match source port(s)

–sports port[,port:port,port...]

-> match source port(s)

–destination-ports port[,port:port,port...]

-> match destination port(s)

–dports port[,port:port,port...]

-> match destination port(s)

–ports port[,port:port,port]

-> match both source and destination port(s)

3.9 nth patch

This patch by Fabrice MARIE <fabrice@netfilter.org> adds a new match that allows you to match a particular Nth packet received by the rule.

For example, if you want to drop every 2 ping packets, you can do as follows :

```
# iptables -A INPUT -p icmp --icmp-type echo-request -m nth --every 2 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       icmp --  anywhere             anywhere           icmp echo-request every 2th
```

Extensions by Richard Wagner <rwagner@cloudnet.com> allows you to create an easy and quick method to produce load-balancing for both inbound and outbound connections.

For example, if you want to balance the load to the 3 addresses 10.0.0.5, 10.0.0.6 and 10.0.0.7, then you can do as follows :

```
# iptables -t nat -A POSTROUTING -o eth0 -m nth --counter 7 --every 3 --packet 0 -j SNAT --to-source 10.0.0.5
# iptables -t nat -A POSTROUTING -o eth0 -m nth --counter 7 --every 3 --packet 1 -j SNAT --to-source 10.0.0.6
# iptables -t nat -A POSTROUTING -o eth0 -m nth --counter 7 --every 3 --packet 2 -j SNAT --to-source 10.0.0.7

# iptables -t nat --list
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
SNAT       all  --  anywhere             anywhere           every 3th packet #0 to:10.0.0.5
SNAT       all  --  anywhere             anywhere           every 3th packet #1 to:10.0.0.6
SNAT       all  --  anywhere             anywhere           every 3th packet #2 to:10.0.0.7
```

Supported options for the nth match are :

–every Nth

-> Match every Nth packet.

[–counter] num

-> Use counter 0-15 (default:0).

[–start] num

-> Initialize the counter at the number 'num' instead of 0. Must be between 0 and (Nth-1).

[–packet] num

-> Match on the 'num' packet. Must be between 0 and Nth-1. If '–packet' is used for a counter, then there must be Nth number of –packet rules, covering all values between 0 and (Nth-1) inclusively.

3.10 pkttype patch

This patch by Michal Ludvig <michal@logix.cz> adds a new match that allows you to match a packet based on its type : host/broadcast/multicast.

If For example you want to silently drop all the broadcasted packets :

```
# iptables -A INPUT -m pkttype --pkt-type broadcast -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere           PKTTYPE = broadcast
```

Supported options for this match are :

–pkt-type [!] packettype

-> match packet type where packet type is one of

host

-> to us

broadcast

-> to all

multicast

-> to group

3.11 pool patch

Patch by Patrick Schaaf <bof@bof.de>. Joakim Axelsson and Patrick are in the process of re-writing it, therefore they will replace this section with the actual explanations once its written.

3.12 psd patch

This patch by Dennis Koslowski <dkoslowski@astaro.de> adds a new match that will attempt to detect port scans.

In its simplest form, psd match can be used as follows :

```
# iptables -A INPUT -m psd -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target  prot opt source    destination
DROP    all  --  anywhere  anywhere    psd weight-threshold: 21 delay-threshold: 300 lo-ports-weight: 3 hi-ports-weight: 1
```

Supported options for psd match are :

[–psd-weight-threshold threshold]

-> Portscan detection weight threshold

[–psd-delay-threshold delay]

-> Portscan detection delay threshold

[–psd-lo-ports-weight lo]

-> Privileged ports weight

[–psd-hi-ports-weight hi]

-> High ports weight

3.13 quota patch

This patch by Sam Johnston <samj@samj.net> adds a new match that allows you to set quotas. When the quota is reached, the rule doesn't match any more.

For example, if you want to limit put a quota of 50Megs on incoming http data you can do as follows :

```
# iptables -A INPUT -p tcp --dport 80 -m quota --quota 52428800 -j ACCEPT
# iptables -A INPUT -p tcp --dport 80 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere           tcp dpt:http quota: 52428800 bytes
DROP       tcp  --  anywhere             anywhere           tcp dpt:http
```

Supported options for quota match are :

–quota quota

-> The quota you want to set.

3.14 random patch

This patch by Fabrice MARIE <fabrice@netfilter.org> adds a new match that allows you to math a packet randomly based on given probability.

For example, if you want to drop 50% of the pings randomly, you can do as follows :

```
# iptables -A INPUT -p icmp --icmp-type echo-request -m random --average 50 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source       destination
DROP       icmp --  anywhere     anywhere        icmp echo-request  random 50%
```

Supported options for random match are :

[–average percent]

-> The probability in percentage of the match. If omitted, a probability of 50% percent is set. Percentage must be within : 1 <= percent <= 99.

3.15 realm patch

This patch by Sampsa Ranta <sampsa@netsonic.fi> adds a new match that allows you to use realm key from routing as match criteria similar to the one found in the packet classifier.

For example, to log all the outgoing packet with a realm of 10, you can do the following :

```
# iptables -A OUTPUT -m realm --realm 10 -j LOG

# iptables --list
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere           REALM match 0xa LOG level warning
```

Supported options for the realm match are :

–realm [!] value[/mask]

-> Match realm

3.16 recent patch

This patch by Stephen Frost <sfrost@snowman.net> adds a new match that allows you to dynamically create a list of IP addresses and then match against that list in a few different ways.

For example, you can create a 'badguy' list out of people attempting to connect to port 139 on your firewall and then DROP all future packets from them without considering them.

```
# iptables -A FORWARD -m recent --name badguy --rcheck --seconds 60 -j DROP
# iptables -A FORWARD -p tcp -i eth0 --dport 139 -m recent --name badguy --set -j DROP

# iptables --list
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
DROP       all  --  anywhere             anywhere           recent: CHECK seconds: 60
DROP       tcp  --  anywhere             anywhere           tcp dpt:netbios-ssn recent: SET
```

Supported options for the recent match are :

–name name

-> Specify the list to use for the commands. If no name is given then 'DEFAULT' will be used.

[!] –set

-> This will add the source address of the packet to the list. If the source address is already in the list, this will update the existing entry. This will always return success or failure if '!' is passed in.

[!] –rcheck

-> This will check if the source address of the packet is currently in the list and return true if it is, and false otherwise. Opposite is returned if '!' is passed in.

[!] –update

-> This will check if the source address of the packet is currently in the list. If it is then that entry will be updated and the rule will return true. If the source address is not in the list then the rule will return false. Opposite is returned if '!' is passed in.

[!] –remove

-> This will check if the source address of the packet is currently in the list and if so that address will be removed from the list and the rule will return true. If the address is not found, false is returned. Opposite is returned if '!' is passed in.

[!] –seconds seconds

-> This option must be used in conjunction with one of 'rcheck' or 'update'. When used, this will narrow the match to only happen when the address is in the list and was seen within the last given number of seconds. Opposite is returned if '!' is passed in.

[!] –hitcount hits

-> This option must be used in conjunction with one of 'rcheck' or 'update'. When used, this will narrow the match to only happen when the address is in the list and packets had been received greater than or equal to the given value. This option may be used along with 'seconds' to create an even narrower match requiring a certain number of hits within a specific time frame. Opposite returned if '!' passed in.

–rttl

-> This option must be used in conjunction with one of 'rcheck' or 'update'. When used, this will narrow the match to only happen when the address is in the list and the TTL of the current packet matches that of the packet which hit the –set rule. This may be useful if you have problems with people faking their source address in order to DoS you via this module by disallowing others access to your site by sending bogus packets to you.

3.17 record-rpc patch

This patch by Marcelo Barbosa Lima <marcelo.lima@dcc.unicamp.br> adds a new match that allows you to match if the source of the packet has requested that port through the portmapper before, or it is a new GET request to the portmapper, allowing effective RPC filtering.

To match RPC connection tracking information, simply do the following :

```
# iptables -A INPUT -m record_rpc -j ACCEPT

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     all  --  anywhere             anywhere
```

The record_rpc match does not take any option.

Do not worry for the match information not printed, it's simply because the print() function of this match is empty :

```
/* Prints out the union ipt_matchinfo. */
static void
print(const struct ipt_ip *ip,
      const struct ipt_entry_match *match,
      int numeric)
{
}
```

3.18 string patch

This patch by Emmanuel Roger <winfield@freegates.be> adds a new match that allows you to match a string anywhere in the packet.

For example, to match packets containing the string "badstring" anywhere in the packet and queue them to a userland IDS, you could use :

```
# iptables -A INPUT -m string --string 'badstring' -j QUEUE

# iptables --list
```

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
QUEUE      all  --  anywhere             anywhere           STRING match badstring
```

Please do use this match with caution. A lot of people want to use this match to stop worms, along with the DROP target. This is a major mistake. It would be defeated by any IDS evasion method.

In a similar fashion, a lot of people have been using this match as a mean to stop particular functions in HTTP like POST or GET by dropping any HTTP packet containing the string POST. Please understand that this job is better done by a filtering proxy. Additionally, any HTML content with the word POST would get dropped with the former method. This match has been designed to be able to queue to userland interesting packets for better analysis, that's all. Dropping packet based on this would be defeated by any IDS evasion method.

Supported options for the string match are :

–string [!] string

-> Match a string in a packet

3.19 time patch

This patch by Fabrice MARIE <fabrice@netfilter.org> adds a new match that allows you to match a packet based on its arrival or departure (for locally generated packets) timestamp.

for example, to accept packets that have an arrival time from 8:00H to 18:00H from Monday to Friday you can do as follows :

```
# iptables -A INPUT -m time --timestart 8:00 --timestop 18:00 --days Mon,Tue,Wed,Thu,Fri -j ACCEPT

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source           destination
ACCEPT     all  --  anywhere         anywhere        TIME from 8:0 to 18:0 on Mon,Tue,Wed,Thu,Fri
```

Supported options for the time match are :

–timestart value

-> minimum HH:MM

–timestop value

-> maximum HH:MM

–days listofdays

-> a list of days to apply, from (case sensitive)

- Mon
- Tue
- Wed
- Thu
- Fri
- Sat
- Sun

3.20 ttl patch

This patch by Harald Welte <laforge@gnumonks.org> adds a new match that allows you to match a packet based on its TTL.

For example if you want to log any packet that have a TTL less than 5, you can do as follows :

```
# iptables -A INPUT -m ttl --ttl-lt 5 -j LOG

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
LOG        all  --  anywhere             anywhere           TTL match TTL < 5 LOG level warning
```

Options supported by the ttl match are :

–ttl-eq value

-> Match time to live value

–ttl-lt value

-> Match TTL < value

–ttl-gt value

-> Match TTL > value

3.21 u32 patch

Don Cohen was kind enough to write an iptables module that pulls any bytes you'd like out of the packet, does some manipulation, and sees if the result is in a particular range. For example, I can grab the Fragmentation information out of the IP header, throw away everything except the More Fragments flag, and see if that flag is set.

What I'll do is introduce the core concepts here, and put in hopefully enough annotated examples that you'll be able to write your own tests.

I won't be focusing on what these fields are, or why you'd want to test them; there are lots of (warning - shameless plug for my employer ahead!) *resources* http://www.sans.org/ for doing that. If you simply need a quick reference for the packet headers, see *tcpip.pdf* http://www.sans.org/resources/tcpip.pdf .

All byte positions in this article start counting at 0 as the first byte of the header. For example, in the IP header, byte "0" holds the 4 bit "Version" and 4 bit "IP Header Length", byte "1" holds the "TOS" field, etc.

Check the value of a 2 byte field In it's simplest form, u32 grabs a block of 4 bytes starting at Start, applies a mask of Mask to it, and compares the result to Range. Here's the syntax we'll use for our first examples:

```
iptables -m u32 --u32 "Start=Range"
```

We'll generally pick a "Start" value that's 3 less than the last byte in which you're interested. So, if you want bytes 4 and 5 of the IP header (the IP ID field), Start needs to be 5-3 = 2. Mask strips out all the stuff you don't want; it's a bitmask that can be as large as 0xFFFFFFFF. To get to our target of bytes 4 or 5, we have to discard bytes 2 and 3. Here's the mask we'll use: 0x0000FFFF . We'll actually use the shorter, and equivalent, 0xFFFF instead.

So, to test for IPID's from 2 to 256, the iptables command line is:

```
iptables -m u32 --u32 "2&0xFFFF=0x2:0x0100"
```

To read this off from left to right: "Load the u32 module, and perform the following u32 tests on this packet; grab the 4 bytes starting with byte 2 (bytes 2 and 3 are the Total Length field, and bytes 4 and 5 are the IPID), apply a mask of 0x0000FFFF (which sets the first two bytes to all zeroes, leaving the last two bytes untouched), and see if that value - the IPID - falls between 2 and 256 inclusive; if so, return true, otherwise false."

There is no standalone IPID check in iptables, but this is the equivalent of the "ip[2:2] >= 2 and ip[2:2] <= 256" tcpdump/bpf filter.

I leave off actions in these examples, but you can add things like:

```
-j LOG --log-prefix "ID-in-2-256 "
-j DROP
```

or any other action. You can also add other tests, as we'll do in a minute.

Don offers this test to see if the total packet length is greater than or equal to 256. The total length field is bytes 2 and 3 of the IP header, so our starting position is 3-3 = 0. Since we're pulling out two bytes again, the mask will be 0xFFFF here as well. The final test is:

```
iptables -m u32 --u32 "0&0xFFFF=0x100:0xFFFF"
```

This is the same as:

```
iptables -m length --length 256:65535
        or the bpf filter
"len >= 256"
```

Check the value of a 1 byte field Much the same, except we'll use a mask of 0x000000FF (or it's shorter equivalent 0xFF) to pull out a single byte from the 4 bytes u32 initially hands us. Let's say I want to test the TTL field for TTL's below 3 to find people tracerouting to us. Yes, there's a ttl module, but let's see how this would be done in u32.

I want to end up with byte 8 of the IP header, so my starting position is 8-3 = 5. Here's the test:

```
iptables -m u32 --u32 "5&0xFF=0:3"
```

Which is equivalent to:

```
iptables -m ttl --ttl-lt 4
        or the bpf filter
"ip[8] <= 3"
```

Looking at 4 bytes at once To check a complete destination IP address, we'll inspect bytes 16-19. Because we want all 4 bytes, we don't need a mask at all. Let's see if the destination address is 224.0.0.1:

```
iptables -m u32 --u32 "16=0xE0000001"
```

This is equivalent to:

```
iptables -d 224.0.0.1/32
```

If we only want to look at the first three bytes (to check if a source address is part of a given class C network), we'll need to use a mask again. The mask we'll use is 0xFFFFFF00 , which throws away the last octet. Let's check if the source address (from bytes 12-15, although we'll ignore byte 15 with the mask) is in the class C network 192.168.15.0 (0xC0A80F00):

```
iptables -m u32 --u32 "12&0xFFFFFF00=0xC0A80F00"
```

Which is the same as:

```
iptables -s 192.168.15.0/24
```

Inspecting early bytes in the header Obviously, if I want to look at the TOS field (byte 1 of the IP header), I can't start at byte 1-3 = -2. What we'll do instead is start at byte 0, pull out the byte we want, and then move it down to the last position for easy testing. This isn't the only way we could do this, but it helps demonstrate a technique we'll need in a minute.

To pull out the TOS field, I first ask u32 to give me bytes 0-3 by using an offset of 0. Now, I pull out byte 1 (the second byte in that block) with a mask of 0x00FF0000 . I need to shift the TOS value down to the far right position for easy comparison. To do this, I use a technique called, unsurprisingly, "right shift". The symbol for right shift is ">>"; this is followed by the number of bits right to move the data. If you're unfamiliar with right shift, take a look at this

tutorial from Harper College http://www.harpercollege.edu/bus-ss/cis/166/mmckenzi/lect19/l19n.htm .

I want to move TOS two bytes - or 16 bits - to the right. This is done with ">>16". Now that we have TOS in the correct position, we compare it to 0x08 (Maximize Throughput):

```
iptables -m u32 --u32 "0&0x00FF0000>>16=0x08"
```

which is the equivalent of:

```
iptables -m ttl --tos 8
```

Inspecting individual bits I'd like to look at the "More Fragments" flag - a flag which has no existing test in iptables (-f matches 2nd and further fragments, I want to match all fragments except the last). Byte 6 contains this, so I'll start with offset 3 and throw away bytes 3-5. Normally this would use a mask of 0x000000FF, but I also want to discard the other bits in that last byte. The only bit I want to keep is the third from the top (0010 0000), so the mask I'll use is 0x00000020 . Now I have two choices; move that bit down to the lowest position and compare, or leave it in its current position and compare.

To move it down, we'll right shift 5 bits. The final test is:

```
iptables -m u32 --u32 "3&0x20>>5=1"
```

If I take the other approach of leaving the bit where it is, I need to be careful about the compare value on the right. If that bit is turned on, the compare value needs to be 0x20 as well.

```
iptables -m u32 --u32 "3&0x20=0x20"
```

Both approaches return true if the More Fragments flag is turned on.

Combining tests If you want to inspect more than one aspect of a packet, use:

```
&&
```

between each test.

Moving on to the TCP header This is a little tricky. Let's say I'd like to look at bytes 4-7 of the TCP header (the TCP sequence number). Let's take the simple approach first, and then look at some ways to improve this.

For our first version, let's assume that the IP header is 20 bytes long - usually a good guess. Our starting point is byte 4 of the tcp header that immediately follows the IP header. Our simplistic test for whether the sequence number is 41 (hex 29) might look like this:

```
iptables -m u32 --u32 "24=0x29"
```

For packets where the IP header length is 20, this will actually work, but there are a few problems. Let's fix them one by one.

First, we never check to see if the packet is even a TCP packet. This is stored in byte 9 of the IP header, so we'll pull 4 bytes starting at byte 6, drop 6-8, and check to see if it's 6. The new rule that first checks if this is a TCP packet at all and also checks that the Sequence Number is 41 is:

```
iptables -m u32 --u32 "6&0xFF=0x6 && 24=0x29"
```

The second problem we've momentarily ignored is the IP header length. True, it usually is 20 bytes long, but it can be longer, if IP options are used.

Here are the steps. We pull the IP header length (a nibble that shows how many 4 bytes words there are in the header, usually 5) out of the IP header. We multiply it by 4 to get the number of bytes in the IP header. We use this number to say how many bytes to jump to get to the beginning of the TCP header, and jump 4 more bytes to get to the Sequence number.

To get the header length, we need the first byte:

```
"0>>24"
```

, but we need to only grab the lower nibble and we need to multiply that number by 4 to get the actual number of bytes in the header. To do the multiply, we'll right shift 22 instead of 24. With this shift, we'll need to use a mask of 0x3C instead of the 0x0F we would have used. The expression so far is:

```
"0>>22&0x3C"
```

. On an IP header with no options, that expression returns 20; just what we'd expect. Now we need to tell u32 to use that number and make a jump that many bytes into the packet, a step performed by the "@" operator.

```
iptables -m u32 --u32 "6&0xFF=0x6 && 0>>22&0x3C@4=0x29"
```

The "@" grabs the number we created on its left (20, normally) and jumps that many bytes forward (we can even do this more than once - see the TCP payload section below). The 4 to its right tells u32 to grab bytes 4-7, but u32 knows to pull them relative to the 20 bytes it skipped over. This gives us the Sequence Number, even if the IP header grows because of options. *phew*!

The last quirk to handle is fragments. When we were only working with the IP header, this wasn't an issue; IP is designed in such a way that the IP header itself can never be fragmented. The TCP header and application payload technically might be, and if we're handed the second or further fragment, we might be looking not at the Sequence Number in bytes 4-7, but perhaps some other part of the TCP header, or more likely, some application layer data.

What we'll do is check that this is the first fragment (or an unfragmented packet, the test won't care), so that we're sure we're looking at tcp header info. To do this, we test the fragment offset in most (we discard the top three flag bits) of bytes 6 and 7 of the IP header to make sure the offset is 0. The test is:

```
"4&0x1FFF=0"
```

The final expression (check for TCP, check for unfragmented packet or first fragment, and jump over the IP header, checking that bytes 4-7 of the TCP header are equal to 41) is:

```
iptables -m u32 --u32 "6&0xFF=0x6 && 4&0x1FFF=0 && 0>>22&0x3C@4=0x29"
```

If the packet is, in fact, fragmented, we have one more consideration; the fragment might be so small that the field we're testing might have been put in a future fragment! In this one case, it's not an issue because every IP link should handle packets of at least 68 bytes; even if the IP header was at its maximum of 60 bytes, the first 8 bytes of the TCP header should be included in that first fragment.

When we start testing for things further in to the packet, we'll have to depend on u32's ability to simply return false if we ever try to ask for a value that falls outside of the packet being inspected.

Checking for values in the ICMP header Let's look for ICMP Host Unreachable packets (ICMP, type 3, code 1). Just as in the above example, we need to check for the Protocol field (Protocol 1 = ICMP this time) and that we're looking at a complete packet or at least the first fragment:

```
"6&0xFF=1 &&
4&0x1FFF=0"
```

To check for the ICMP Type and Code, we skip over the IP header again (

```
"0>>22&0x3C@..."
```

). To grab the first two bytes, we'll start at offset 0 and just right shift 16 bits. The final test is:

```
iptables -m u32 --u32 "6&0xFF=1 && 4&0x1FFF=0 && 0>>22&0x3C@0>>16=0x0301"
```

Checking for values in the UDP payload Lets try going all the way into the packet payload now, and match packets that are UDP DNS queries. Here we're not only going to check for destination port 53, but we're also going to test the top bit of byte 2 of the payload; if set, this is a DNS query.

We start by checking that this is a UDP packet:

```
"6&0xFF=17"
```

. We add the now familiar check for first fragment:

```
"4&0x1FFF=0"
```

.

To test the destination port, we grab bytes 2 and 3 from the udp header (after jumping over the IP header as in the previous examples):

```
"0>>22&0x3C@0&0xFFFF=53"
```

.

If the packet has passed all of the above, we go back to check the payload (remember we have to jump over the variable-length IP and 8 byte UDP headers

```
"0>>22&0x3C@8 ..."
```

) to make sure this is a DNS query rather than a response. To grab the high bit from byte 2, I'll use offset 8 to grab the first 4 payload bytes, right shift 15 bits to deposit the Query bit in the lowest position, and throw away all the rest of the bits with a mask of 0x01:

```
"0>>22&0x3C@8>>15&0x01=1"
```

The final test is:

```
iptables -m u32 --u32 "6&0xFF=17 && 4&0x1FFF=0 && 0>>22&0x3C@0&0xFFFF=53 && 0>>22&0x3C@8>>15&0x01=1"
```

Ugh. I've seen stellar noise that had less entropy :-) Note that we're doing the whole thing with u32 checks; we could pull out the "udp", "first/no fragment" and "port 53" checks into other modules, and end up with this slightly more readable version:

```
iptables -p udp --dport 53 \! -f -m u32 --u32 "0>>22&0x3C@8>>15&0x01=1"
```

Tests First, a recap of the above, then some additional tests.

```
"2&0xFFFF=0x2:0x0100"
```

Test for IPID's between 2 and 256

```
"0&0xFFFF=0x100:0xFFFF"
```

Check for packets with 256 or more bytes.

```
"5&0xFF=0:3"
```

Match packets with a TTL of 3 or less.

```
"16=0xE0000001"
```

Destination IP address is 224.0.0.1

```
"12&0xFFFFFF00=0xC0A80F00"
```

Source IP is in the 192.168.15.X class C network.

```
0&0x00FF0000>>16=0x08
```

Is the TOS field 8 (Maximize Throughput)?

```
"3&0x20>>5=1"
```

Is the More Fragments flag set?

```
"6&0xFF=0x6"
```

Is the packet a TCP packet?

```
"4&0x1FFF=0"
```

Is the fragment offset 0? (If so, this is either an unfragmented packet or the first fragment).

```
"0>>22&0x3C@4=0x29"
```

Is the TCP Sequence number 41? (This requires the previous two checks for TCP and First Fragment as well)

```
"0>>22&0x3C@0>>16=0x0301"
```

Check for ICMP type=3 and code=1 (needs UDP and first fragment tests too)

```
"0>>22&0x3C@0&0xFFFF=53"
```

Is the UDP destination port 53? (Check for udp and first/no fragment first)

```
"0>>22&0x3C@8>>15&0x01=1"
```

Check that the UDP DNS Query bit is set (again, check for UDP, first/no fragment, and dest port 53 first).

And now, some new tests:

```
"6&0xFF=1"
```

Is this an ICMP packet? (From Don Cohen's documentation)

```
"6&0xFF=17"
```

Is this a UDP packet?

```
"4&0x3FFF=0"
```

Is the fragment offset 0 andMF cleared? (If so, this is an unfragmented packet).

```
"4&0x3FFF=1:0x3FFF"
```

Is the fragment offset greater than 0 orMF set? (If so, this is a fragment).

```
0>>22&0x3C@12>>26&0x3C@-3&0xFF=0:255
```

Is there any payload on this tcp packet (check for tcp and not fragmented first)? This elegant test was contributed by Don Cohen as I fumbled for a way to look for payload on a SYN packet. By simply testing to see if payload byte 0 has a value between 0 and 255, we get true if payload byte 0 exists (read: "if there is any payload at all"), and false if we've gone beyond the end of the packet (read: "if there is no payload").

Don Cohen wrote the u32 module, and also wrote some (if you'll forgive me) somewhat cryptic documentation inside the source code for the module. William Stearns wrote this text, which borrows some examples and concepts from Don's documentation. Many thanks to Don for reviewing an early draft of this article. Thanks also to Gary Kessler and SANS for making the TCP/IP pocket reference guide freely available.

4 New netfilter targets

In this section, we will attempt to explain the usage of new netfilter targets. The patches will appear in alphabetical order. Additionally, we will not explain patches that break other patches. But this might come later.

Generally speaking, for targets, you can get the help hints from a particular module by typing :

```
# iptables -j THE_TARGET_YOU_WANT --help
```

This would display the normal iptables help message, plus the specific "THE_TARGET_YOU_WANT" target help message at the end.

4.1 ftos patch

This patch by Matthew G. Marsh <mgm@paktronix.com> adds a new target that allows you to set the TOS of packets to an arbitrary value.

For example, if you want to set the TOS of all the outgoing packets to be 15, you can do as follows :

```
# iptables -t mangle -A OUTPUT -j FTOS --set-ftos 15

# iptables -t mangle --list
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
FTOS       all  --  anywhere             anywhere           TOS set 0x0f
```

Supported options for the FTOS target are :

–set-ftos value

-> Set TOS field in packet header to value. This value can be in decimal (ex: `32`) or in hex (ex: `0x20`)

4.2 IPV4OPTSSTRIP patch

This patch by Fabrice MARIE <fabrice@netfilter.org> adds a new target that allows you to strip all the IP options from an IPv4 packet.

It's simpled loaded as follows :

```
# iptables -t mangle -A PREROUTING -j IPV4OPTSSTRIP

# iptables -t mangle --list
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
IPV4OPTSSTRIP  all  --  anywhere             anywhere
```

This target doesn't support any option.

4.3 NETLINK patch

This patch by Gianni Tedesco <gianni@ecsc.co.uk> adds a new target that allows you to send dropped packets to userspace via a netlink socket.

For example, if you want to drop all pings and send them to a userland netlink socket instead, you can do as follows :

```
# iptables -A INPUT -p icmp --icmp-type echo-request -j NETLINK --nldrop

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
NETLINK    icmp --  anywhere             anywhere           icmp echo-request nldrop
```

Supported options for the NETLINK target are :

–nldrop

-> Drop the packet too

–nlmark <number>

-> Mark the packet

–nlsize <bytes>

-> Limit packet size

For more information on netlink sockets, you can refer to the

Netlink Sockets Tour http://www.skyfree.org/linux/kernel_network/netlink.html .

4.4 NETMAP patch

This patch by Svenning Soerensen <svenning@post5.tele.dk> adds a new target that allows you create a static 1:1 mapping of the network address, while keeping host addresses intact.

For example, if you want to alter the destination of incoming connections from 1.2.3.0/24 to 5.6.7.0/24, you can do as follows :

```
# iptables -t nat -A PREROUTING -d 1.2.3.0/24 -j NETMAP --to 5.6.7.0/24

# iptables -t nat --list
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
NETMAP     all  --  anywhere             1.2.3.0/24         5.6.7.0/24
```

Supported options for NETMAP target are :

–to address[/mask]

-> Network address to map to.

4.5 ROUTE patch

This patch by Cédric de Launois <delaunois@info.ucl.ac.be> adds a new target which enables you to setup unusual routes. For example, the ROUTE target lets you route a received packet through an interface or towards a host, even if the regular destination of the packet is the router itself. The ROUTE target is also able to change the incoming interface of a packet.

The target does not modify the packets, and can be or not a final target. It has to be used inside the mangle table.

Whenever possible, you should use the MARK target together with iproute2 instead of this ROUTE target. However, this target is useful to force the use of an interface or a next hop and to change the incoming interface of a packet. People also use it for easiness and to simplify their rules (one rule to route a packet is easier that one MARK rule + one iproute2 rule).

Options supported by the ROUTE target are :

–oif ifname

Send the packet out using ‘ifname’ network interface. The destination host must be on the same link or the interface must be a tunnel. Otherwise, arp resolution cannot be performed and the packet is dropped.

–iif ifname

Change the packet's incoming interface to 'ifname'.

–gw ip

Route the packet via this gateway. The packet is routed as if its destination IP address was 'ip'.

–continue

Route the packet and continue traversing the rules.

Note that –iif and –continue options can't be used together.

Example for –oif option For example, assume that you want to redirect ssh packets towards a server inside your network, without modifying those packets in any way (this excludes the use of the standard port forwarding mechanism). A solution is to use an ipip tunnel and the ROUTE target to reroute ssh packets to the real ssh server, which has the same IP address as the router. It is not possible to reroute those packets using the standard routing mechanisms, because the kernel locally delivers a packet having a destination address belonging to the router itself.

Time for ASCII art :

```
          eth0  +------+ 192.168.0.1        192.168.0.2 +----+
----------------|router|-------------------------------|host|
IP: 150.150.0.1 +------+                                +----+
                   | | tunl1             IP: 150.150.0.1 | |
                   | +-----------------------------------+ |
                   +---------------------------------------+
                                 IPIP tunnel
```

For the example above, you can do as follows :

```
# iptables -A PREROUTING -t mangle -i eth0 -p tcp --dport 22 -j ROUTE --oif tunl1
# iptables -A PREROUTING -t mangle -i tunl1 -j ROUTE --oif eth0

# iptables -L PREROUTING -t mangle
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
ROUTE      tcp  --  anywhere             anywhere           tcp dpt:ssh ROUTE oif tunl1
ROUTE      all  --  anywhere             anywhere           ROUTE oif eth0
```

Example for –gw option If you want to quickly and easily balance the load between two gateways 10.0.0.1 and 10.0.0.2, then you can do as follows :

```
# iptables -A PREROUTING -t mangle -m random --average 50 -j ROUTE --gw 10.0.0.1
# iptables -A PREROUTING -t mangle -j ROUTE --gw 10.0.0.2

# iptables -L PREROUTING -t mangle
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
ROUTE      all  --  anywhere             anywhere           random 50% ROUTE gw 10.0.0.1
ROUTE      all  --  anywhere             anywhere           ROUTE gw 10.0.0.2
```

Example for –iif option The teql per-packet round-robin scheduler is used to aggregate the sending bandwidth on several slave interfaces; see the top of linux/net/sched/sch_teql.c for usage and caveats. [For Ethernet, the bonding driver is preferred.]

Per-packet scheduling is most useful when a single flow might exceed the bandwidth of the individual interfaces, as is commonly the case with T1 links.

Using WAN interfaces hdlc[0-3] on a 4 T1/E1 WAN card, one might do the following:

```
modprobe sch_teql
for i in 0 1 2 3
        ip link set dev hdlc${i} up
        sysctl -w net.ipv4.conf.hdlc${i}.rp_filter=0
        tc qdisc add dev hdlc${i} root teql0
done
ip link set dev teql0 up
sysctl -w net.ipv4.conf.teql0.rp_filter=0
ip address add $IPADDR peer $REMIP dev teql0
ip route add default via $REMIP dev teql0
```

Since teql0 is a send-only interface, incoming packets appear on the slave hdlc* interfaces, and the usual caveats regarding asymmetric routing apply. E.g., Quagga OSPF implements its own reverse-path filtering neighbor logic; replies from peers that are received on the wrong interface are dropped.

The following rule causes all incoming packets on the hdlc* interfaces to appear to have been received on teql0:

```
iptables -t mangle -A PREROUTING -i hdlc\+ -j TTL --ttl-inc 1
iptables -t mangle -A PREROUTING -i hdlc\+ -j ROUTE --iif teql0
```

The TTL increment is required to avoid dropping packets, as multicast and broadcast packets are typically sent with TTL = 1. In this example, Quagga also sets TTL = 1 on packets that it sends to its peers.

4.6 SAME patch

This patch by Martin Josefsson <gandalf@wlug.westbo.se> adds a new target which is similar to SNAT and will gives a client the same address for each connection.

For example, if you want to modify the source address of the connections to be 1.2.3.4-1.2.3.7 you can do as follows :

```
# iptables -t nat -A POSTROUTING -j SAME --to 1.2.3.4-1.2.3.7

# iptables -t nat --list
Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
SAME       all  --  anywhere             anywhere           same:1.2.3.4-1.2.3.7
```

Options supported by the SAME target are :

–to <ipaddr>-<ipaddr>

-> Addresses to map source to. May be specified more than once for multiple ranges.

–nodst

-> Don't use destination-ip in source selection

4.7 tcp-MSS patch

This patch by Marc Boucher <marc+nf@mbsi.ca> adds a new target that allows you to examine and alter the MSS value of TCP SYN packets, to control the maximum size for that connection.

As explained by Marc himself, THIS IS A HACK, used to overcome criminally brain-dead ISPs or servers which block ICMP Fragmentation Needed packets.

Typical usage would be :

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --clamp-mss-to-pmtu

# iptables --list
Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
TCPMSS     tcp  --  anywhere             anywhere           tcp flags:SYN,RST/SYN TCPMSS clamp to PMTU
```

Options supported by the tcp-MSS target are (mutually-exclusive) :

–set-mss value

explicitly set MSS option to specified value

–clamp-mss-to-pmtu

automatically clamp MSS value to (path_MTU - 40)

4.8 TTL patch

This patch by Harald Welte <laforge@gnumonks.org> adds a new target that enables the user to set the TTL value of an IP packet or to increment/decrement it by a given value.

For example, if you want to set the TTL of all outgoing connections to 126, you can do as follows :

```
# iptables -t mangle -A OUTPUT -j TTL --ttl-set 126

# iptables -t mangle --list
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
TTL        all  --  anywhere             anywhere           TTL set to 126
```

Supported options for the TTL target are :

–ttl-set value

-> Set TTL to <value>

–ttl-dec value

-> Decrement TTL by <value>

–ttl-inc value

-> Increment TTL by <value>

4.9 ulog patch

This patch by Harald Welte <laforge@gnumonks.org> adds a new target which supplies a more advanced packet logging mechanism than the standard LOG target. The 'libipulog/' contains a library for receiving the ULOG messages.

Harald maintains a

web page http://www.gnumonks.org/projects/ulogd containing the proper documentation for ULOG, so there is no point for me to explain this here..

4.10 XOR patch

This patch by Tim Vandermeersch <Tim.Vandermeersch@pandora.be> adds a new target that enables the user to encrypt TCP and UDP traffic using a simple xor encryption.

For example, if you want to encrypt all TCP and UDP traffic between host A and host B, you can do as follows :

```
(on host A, 1.2.3.4)
# iptables -t mangle -A OUTPUT -d 1.2.3.5 -j XOR --key somekey --block-size 3
# iptables -t mangle -A INPUT -s 1.2.3.4 -j XOR --key somekey --block-size 3

# iptables -t mangle -L
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
XOR        all  --  anywhere             1.2.3.5            key: somekey block-size: 3
XOR        all  --  1.2.3.5              anywhere           key: somekey block-size: 3

(on host B, 1.2.3.5)
# iptables -t mangle -A OUTPUT -d 1.2.3.4 -j XOR --key somekey --block-size 3
# iptables -t mangle -A INPUT -s 1.2.3.5 -j XOR --key somekey --block-size 3

# iptables -t mangle -L
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
XOR        all  --  anywhere             1.2.3.4            key: somekey block-size: 3
XOR        all  --  1.2.3.4              anywhere           key: somekey block-size: 3
```

Supported options for the TTL target are :

–key string

Set the encryption key

–block-size value

Specify the block size

5 New connection tracking patches

In this sections, we will show the available connection tracking/nat patches. To use them, simply load the corresponding modules (with options if needed) for them to be in effect.

5.1 amanda-conntrack-nat patch

This patch by Brian J. Murrell <netfilter@interlinx.bc.ca> adds support for connection tracking and nat of the Amanda backup tool protocol.

5.2 eggdrop-conntrack patch

This patch by Magnus Sandin <magnus@sandin.cx> adds support for connection tracking for eggdrop bot networks.

5.3 h323-conntrack-nat patch

This patch by Jozsef Kadlecsik <kadlec@blackhole.kfki.hu> adds H.323/netmeeting support module for netfilter connection tracking and NAT.

H.323 uses/relies on the following data streams :

- port 389 -> Internet Locator Server (TCP).
- port 522 -> User Location Server (TCP).
- port 1503 -> T.120 Protocol (TCP).
- port 1720 -> H.323 (H.225 call setup, TCP)
- port 1731 -> Audio call control (TCP)
- Dynamic port -> H.245 call control (TCP)
- Dynamic port -> RTCP/RTP streaming (UDP)

The H.323 conntrack/NAT modules support the connection tracking/NATing of the data streams requested on the dynamic ports. The helpers use the search/replace hack from the ip_masq_h323.c module for the 2.2 kernel series.

At the very minimum, H.323/netmeeting (video/audio) is functional by letting trough the 1720 port and loading these H.323 module(s).

The H.323 conntrack/NAT modules do not support :

- H.245 tunnelling
- H.225 RAS (gatekeepers)

5.4 irc-conntrack-nat patch

This patch by Harald Welte <laforge@gnumonks.org> allows DCC to work though NAT and connection tracking. By default, this module will track IRC connection on port 6667. But you can change this for another port with the 'ports=xx' argument.

5.5 mms-conntrack-nat patch

This patch by Filip Sneppe <filip.sneppe@cronos.be> adds support for connection tracking of Microsoft Streaming Media Services protocol.

This allows client (Windows Media Player) and server to negotiate protocol (UDP, TCP) and port for the media stream. A partially reverse engineered protocol analysis is available from *here* http://get.to/sdp , together with a link to a Linux client.

It is recommended to open UDP port 1755 to the server, as this port is used for retransmission requests.

This helper has been tested in SNAT and DNAT setups.

5.6 pptp patch

This patch by Harald Welte <laforge@gnumonks.org> allows netfilter to track pptp connection as well as to NAT them.

5.7 quake3-conntrack patch

This patch by Filip Sneppe <filip.sneppe@cronos.be> adds support for Quake III Arena connection tracking and nat.

5.8 rsh patch

This patch by Ian Larry Latter <Ian.Latter@mq.edu.au> adds support for RSH connection tracking.

An RSH connection tracker is required if the dynamic stderr "Server to Client" connection is to occur during a normal RSH session. This typically operates as follows :

```
Client 0:1023 --> Server 514    (stream 1 - stdin/stdout)
Client 0:1023 <-- Server 0:1023 (stream 2 - stderr)
```

The author of this patch is warning you that this module could be dangerous, and that it is not "best practice" to use RSH, and you should use SSH in all instances.

5.9 snmp-nat patch

This patch by James Morris <jmorris@intercode.com.au> allows netfilter to NAT basic SNMP This is the “basic” form of SNMP-ALG, as described in

RFC 2962 http://www.faqs.org/rfcs/rfc2962.html , it works by modifying IP addresses inside SNMP payloads to match IP-layer NAT mapping.

5.10 talk-conntrack-nat patch

This patch by Jozsef Kadlecsik <kadlec@blackhole.kfki.hu> allows netfilter to track talk connections, as well as to NAT them. By default both otalk (UDP port 517) and talk (UDP port 518) are supported. otalk/talk supports can selectively be enabled/disabled by the module parameters of the ip_conntrack_talk and ip_nat_talk modules. The options are :

- otalk = 0 | 1

- talk = 0 | 1

where '0' means 'do not support' while '1' means 'do support' the given protocol flavor.

5.11 tcp-window-tracking patch

This patch by Jozsef Kadlecsik <kadlec@blackhole.kfki.hu> allows netfilter do TCP connection tracking according to the article

Real Stateful TCP Packet Filtering in IP Filter http://www.iae.nl/users/guido/papers/tcp_filtering.ps.gz by Guido van Rooij. It supports window scaling, and can now handle already established connections.

5.12 tftp patch

This patch by Magnus Boden <mb@ozaba.mine.nu> allows netfilter to track tftp connections as well as to NAT them. By default, this module will track tftp connections on port 69. But you can change this for another port with the 'ports=xx' argument.

6 New IPv6 netfilter matches

In this section, we will attempt to explain the usage of new netfilter matches. The patches will appear in alphabetical order. Additionally, we will not explain patches that break other patches. But this might come later.

Generally speaking, for matches, you can get the help hints from a particular module by typing :

```
# ip6tables -m the_match_you_want --help
```

This would display the normal ip6tables help message, plus the specific "the_match_you_want" match help message at the end.

6.1 agr patch

This patch by Andras Kis-Szabo <kisza@sch.bme.hu> adds 1 new match :

- "eui64" : lets you match the IPv6 packet based on it's addressing parameters.

This patch can be quite useful for people using EUI-64 IPv6 addressing scheme who are willing to check the packets based on the delivered address on a LAN.

For example, we will redirect the packets that have a correct EUI-64 address:

```
# ip6tables -N ipv6ok
# ip6tables -A INPUT -m eui64 -j ipv6ok
# ip6tables -A INPUT -s ! 3FFE:2F00:A0::/64 -j ipv6ok
# ip6tables -A INPUT -j LOG
# ip6tables -A ipv6ok -j ACCEPT

# ip6tables --list
Chain INPUT (policy ACCEPT)
```

```
target     prot opt source               destination
ipv6ok     all      anywhere             anywhere            eui64
ipv6ok     all     !3ffe:2f00:a0::/64    anywhere
LOG        all      anywhere             anywhere            LOG level warning

Chain ipv6ok (2 references)
target     prot opt source               destination
ACCEPT     all      anywhere             anywhere
```

This match hasn't got any option.

6.2 ahesp6 patch

This patch by Andras Kis-Szabo <kisza@sch.bme.hu> adds a new match that allows you to match a packet based on its ah and esp headers' content. The name of the matches:

- "ah" : lets you match the IPv6 packet based on its ah header.
- "esp" : lets you match the IPv6 packet based on its esp header.

For example, we will drop all the AH packets that have a SPI equal to 500, and check the contents of the restricted area in the header :

```
# ip6tables -A INPUT -m ah --ahspi 500 --ahres -j DROP

# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere            ah spi:500 reserved
```

Supported options for the ah match are :

–ahspi [!] spi[:spi]

-> match spi (range)

–ahlen [!] length

-> length ot this header

–ahres

-> checks the contents of the reserved field

The esp match works exactly the same as in IPv4 :

```
# ip6tables -A INPUT -m esp --espspi 500 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere            esp spi:500
```

Supported options for the esp match are :

–espspi [!] spi[:spi]

-> match spi (range)

In IPv6 these matches can be concatenated:

```
# ip6tables -A INPUT -m ah --ahspi 500 --ahres --ahlen ! 40 -m esp --espspi 500 -j DROP

# iptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere           ah spi:500 length:!40 reserved esp spi:500
```

6.3 frag6 patch

This patch by Andras Kis-Szabo <kisza@sch.bme.hu> adds a new match that allows you to match a packet based on the content of its fragmentation header. The name of the match:

- "frag" : lets you match the IPv6 packet based on its fragmentation header.

For example, we will drop all the packets that have an ID between 100 and 200, and the packet is the first fragment :

```
# ip6tables -A INPUT -m frag --fragid 100:200 --fragfirst -j DROP

# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere           frag ids:100:200 first
```

Supported options for the frag match are :

–fragid [!] id[:id]

-> match the id (range) of the fragmenation

–fraglen [!] length

-> match total length of this header

–fragres

-> checks the contents of the reserved field

–fragfirst

-> matches on the first fragment

–fragmore

-> there are more fragments

–fraglast

-> this is the last fragment

6.4 ipv6header patch

This patch by Andras Kis-Szabo <kisza@sch.bme.hu> adds a new match that allows you to match a packet based on its extension headers. The name of the match:

- "ipv6header" : lets you match the IPv6 packet based on its headers.

For example, let's drop the packets which have got hop-by-hop, ipv6-route headers and a protocol payload:

```
# ip6tables -A INPUT -m ipv6header --header hop-by-hop,ipv6-route,protocol -j DROP

# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere           ipv6header flags:hop-by-hop,ipv6-route,protocol
```

And now, let's drop the packets which have got an ipv6-route extension header:

```
# ip6tables -A INPUT -m ipv6header --header ipv6-route --soft -j DROP

# ip6ptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere           ipv6header flags:ipv6-route soft
```

Supported options for the ipv6header match are :

[!] –header headers

-> You can specify the interested headers with this option. Accepted formats:

- hop,dst,route,frag,auth,esp,none,proto
- hop-by-hop,ipv6-opts,ipv6-route,ipv6-frag,ah,esp,ipv6-nonxt,protocol
- 0,60,43,44,51,50,59

–soft

-> You can specify the soft mode: in this mode the match checks the existence of the header, not the full match!

6.5 ipv6-ports patch

This patch by Jan Rekorajski <baggins@pld.org.pl> adds 4 new matches :

- "limit" : lets you to restrict the number of parallel TCP connections from a particular host or network.
- "mac" : lets you match a packet based on its MAC address.
- "multiport" : lets you to specify ports with a mix of port-ranges and single ports for UDP and TCP protocols.
- "owner" : lets you match a packet based on its originator process' owner id.

These matches are the ports of the IPv4 versions. See the main documentation for the details!

6.6 length patch

This patch by Imran Patel <ipatel@crosswinds.net> adds a new match that allows you to match a packet based on its length. (This patch is shameless adaption from the IPv4 match written by James Morris <jmorris@intercode.com.au>)

For example, let's drop all the pings with a packet size greater than 85 bytes :

```
# ip6tables -A INPUT -p ipv6-icmp --icmpv6-type echo-request -m length --length 85:0xffff -j DROP

# ip6ptables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       ipv6-icmp --  anywhere             anywhere           ipv6-icmp echo-request length 85:65535
```

Supported options for the length match are :

[!] –length length[:length]

-> Match packet length against value or range of values (inclusive)

Values of the range not present will be implied. The implied value for minimum is 0, and for maximum is 65535.

6.7 route6 patch

This patch by Andras Kis-Szabo <kisza@sch.bme.hu> adds a new match that allows you to match a packet based on the content of its routing header. The name of the match:

- "rt" : lets you match the IPv6 packet based on its routing header.

For example, we will drop all the packets that have 0 routing type, the packet is near the last hop (max 2 hops far), the routing path contains ::1 and ::2 (but not exactly):

```
# ip6tables -A INPUT -m rt --rt-type 0 --rt-segsleft :2 --rt-0-addrs ::1,::2 --rt-0-not-strict -j DROP

# ip6tables --list
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
DROP       all      anywhere             anywhere           rt type:0 segslefts:0:2 0-addrs ::1,::2 0-not-strict
```

Supported options for the rt match are :

–rt-type [!] type

-> matches the type

–rt-segsleft [!] num[:num]

-> matches the Segments Left field (range)

–rt-len [!] length

-> total length of this header

–rt-0-res

-> checks the contents of the reserved field

–rt-0-addrs ADDR[,ADDR...]

-> Type=0 addresses (list, max: 16)

–rt-0-not-strict

-> List of Type=0 addresses not a strict list

7 New IPv6 netfilter targets

In this section, we will attempt to explain the usage of new netfilter targets. The patches will appear in alphabetical order. Additionally, we will not explain patches that break other patches. But this might come later.

Generally speaking, for targets, you can get the help hints from a particular module by typing :

```
# ip6tables -j THE_TARGET_YOU_WANT --help
```

This would display the normal iptables help message, plus the specific "THE_TARGET_YOU_WANT" target help message at the end.

7.1 LOG patch

This patch by Jan Rekorajski <baggins@pld.org.pl> adds a new target that allows you to LOG the packets as in the IPv4 version of iptables.

The examples are the same as in iptables. See the man page for details!

7.2 REJECT patch

This patch by Harald Welte <laforge@gnumonks.org> adds a new target that allows you to REJECT the packets as in the IPv4 version of iptables.

The examples are the same as in iptables. See the man page for details!

8 New IPv6 connection tracking patches

The connection tracking hasn't supported, yet.

9 Contributing

9.1 Contributing a new extension

Netfilter core-team always welcome new extensions/bug-fixes. In this section we will not focus on how to package a new extension to ease its inclusion into patch-o-matic yet. But this might come in a future version of this HOWTO.

First of all, you should be familiar with the

Netfilter Hacking HOWTO http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html
.

It's a good idea to subscribe to netfilter-devel mailing list. More info on how to subscribe can be found on the netfilter homepage.

9.2 Contributing to this HOWTO

You are mostly welcome to update this HOWTO. To do so, the preferred way is to send a patch of the SGML master of this document to the netfilter-devel mailing list.

You are also welcome to translate this HOWTO into your favorite language, or to update an outdated translation.

Thanks for your help! Thanks to the developers who contributed the netfilter-extensions-HOWTO parts related to their patches.