Blocking protocols at Layer 7 with the L7 patch

BEYOND THE PORT

If you need a tool for filtering protocols that doesn't depend on the port, try L7, an IPTables patch that operates through regular expressions.

BY JÖRG HARMUTH

Traditional firewalls decide whether to allow or reject packets based on IP addresses, TCP flags, MAC addresses, ports, and other criteria that reside in OSI layers two through four. Experienced admins can probably type commands like *iptables -A FORWARD -i $IF -o $OF -p tcp --dport 80 --syn -j ACCEPT* standing on their heads. But what if the web server listens on port 8500 rather than port 80? Or if a gaming server misuses this port? Peer-to-peer applications are even worse, as there is no way of predicting the ports they will use. And VoIP makes the chaos complete with Real Time Protocol (RTP), which definitely takes liberties when assigning UDP ports.

Even if you don't need a firewall, you may still be using traffic shaping for VoIP to prioritize RTP. And to do that, your router needs to be able to distinguish between protocols. The port number will not be much help, except in the most simple cases, and this leaves you with the option of inspecting data streams more closely. No big deal for an admin, maybe, but a genuine challenge to a firewall sticking closely to its set of rules. A firewall can't afford to take time off to analyze data, but at the same time, it can't afford to guess wrongly.

The L7 patch for IPtables attempts this balancing act [1]. L7 stands for OSI

Layer 7: Application	IPtables with Layer 7 patch
Layer 6: Presentation Layer	
Layer 5: Session Layer	
Layer 4: Transport	IPtables: Ports,TCP flags ...
Layer 3: Network	IPtables: IP addresses
Layer 2: Data Link	ARPtables, MAC addresses
Layer 1: Physical	

Figure 1: After installing the L7 patch, Netfilter operates in the OSI Application layer, as well as OSI Layers 2 through 4.

Listing 1a: FTP Pattern

```
01 # Pattern quality: great veryfast
02 # Matches the first two things a server should say.  Most servers
   say
03 # something after 220, even though they don't have to, and it
   usually
04 # includes the string "ftp" (l7-filter is case insensitive at the
   moment).
05 # This includes proftpd, vsftpd, wuftpd, warftpd, pureftpd,
   Bulletproof
06 # FTP Server, and whatever ftp.microsoft.com uses.  Just in case,
   the next
07 # thing the server sends is a 331.  All the above servers also send
08 # something, including "password," after this code.
09 ftp
10 # actually, let's just do the first for now, it's faster
11 ^220[\x09-\x0d -~]*ftp
12 # This will match more, but much slower
13 # ^220[\x09-\x0d -~]*ftp|331[\x09-\x0d -~]*password
```

Layer 7, the layer where application protocols such as FTP or SSH reside (Figure 1). L7 uses regular expressions to investigate the content within an individual connection. In contrast to more complex Application layer gateways, which check protocols based on the full set of RFC rules and filter dangerous content in the process, L7 uses fast regular expressions to check the data stream. L7 is able to sift quickly through incoming data using regular expression matching to identify characteristic features of the individual protocols without relying on external indicators such as port numbers.

Star Gazing

L7 expects a pattern file for each protocol it supports. The pattern file name is made up of the protocol name and the extension *.pat*; for example, *http.pat* is a regular expression that identifies HTTP.

An L7 pattern file contains one or multiple regular expressions to identify the protocol. All the options are discussed in detail in the files provided with the path, and the options are all disabled, with the exception of one expression. You will also find a qualitative evaluation (how fast, how secure), and if needed, HOWTOs on typical applications. Listings 1a and 1b show the (abridged) definitions for FTP and SMTP.

Both expressions start with a *220* code; this is the code at the start of the banner for both services. Both FTP and SMTP are ASCII-based protocols that write messages in individual lines in a human-readable format. According to the RFCs, nothing actually has to follow the 220 code, but it can and typically does. Standard FTP command line programs output these messages, as a call to *ftp kernel.org* shows:

```
Connected to kernel.org.
220 Welcome to ftp.kernel.org.
Name (kernel.org:jha): anonymous
331 Please specify the password.
Password:
```

The *Connected* message is generated by the client and followed by the server banner, which L7 locates in the data transferred and evaluates to identify the protocol. As the banner includes an *ftp* string, L7 identifies the FTP protocol by matching this string against the regular expression (Listing 1a, line 11), and does not attempt to match against any other expressions. If the server had answered with an RFC compliant *220* code, L7 would have been unable to identify the protocol, as the SMTP expression (Listing 1b, line 7) does not return a match.

At the next step, the FTP client transmits the username, and the server responds with a code 331. This line would match the expression in the FTP listing

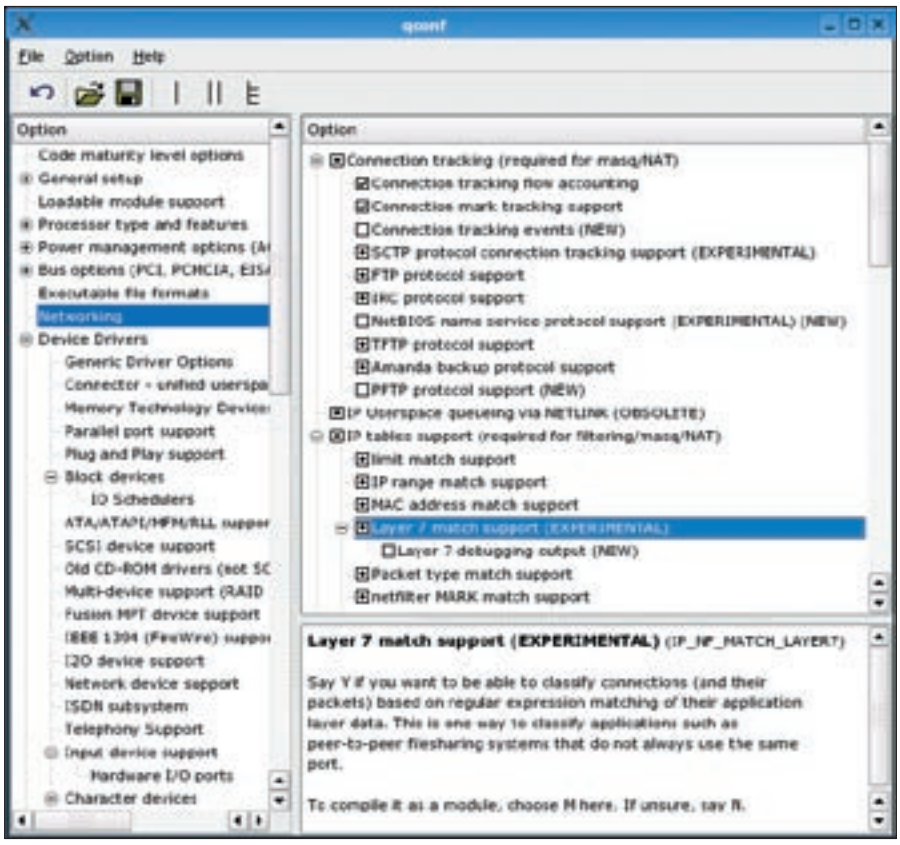

Figure 2: The configuration for L7, Netfilter, and QoS is spread across several branches of Qconf. Layer 7 support is hidden below Networking | Networking support | Networking options | Network packet filtering | IP: Netfilter Configuration | IP tables support.

(which is still disabled, see Listing 1a, line 13), as the line contains the search keys *331* and *password*. L7 would identify FTP based on this information.

L7 does not find a match until the seventh packet in this connection has been sent; the three-way handshake, the server banner, the login prompt, and the login name are transferred before this happens. This explains the comment "This will match more, but much slower." If L7 fails to identify a protocol, you might like to compare the protocol file with a session dump. Comparing the connection data with the regex should put you back on track.

QoS rather than Drop and Reject

L7 can handle all of Iptables standard targets (*DROP*, *REJECT*, *ACCEPT*...), and this would allow you to base a firewall ruleset on L7 matches. However, the L7 project has good reason for advising admins not to do this. Experienced attackers could trivially spoof protocol information, slipping a red herring to L7 and opening up unauthorized communication paths into your network. Instead, you should view matches as a useful addition to a strict ruleset that you have applied previously.

In fact, L7 is recommended for QoS (Quality of Service). The worst thing that can happen is that QoS might assign a connection it incorrectly identifies to a band with the wrong priority. This keeps user complaints down to an acceptable level. The reward for all this effort is that L7 keeps unimportant services, such as peer-to-peer networks, from eating up your bandwidth. A well-crafted configuration will keep most of these unwanted services at bay, leaving more breathing space for important services. Blocking peer-to-peer networks completely would be just like throwing down the gauntlet to the users to find a workaround.

Listing 1b: SMTP Pattern

```
01 # Pattern quality: great
   veryfast
02 smtp
03 # As usual, no text is
   required after "220," but all
   known servers have some
04 # there.  It (almost?) always
   has the string "smtp" in it.
   The RFC examples
05 # do not, so we match those
   too, just in case anyone has
   copied them
06 # literally.
07 ^220[\x09-\x0d -~]*
   (e?smtp|simple mail)
```

How it Works

In order to identify protocols based on regular expressions, L7 needs to investigate a longer segment of the data stream, which will typically comprise multiple packets. By default, L7 will investigate the first 2048 bytes or the first ten packets in a connection at the most (eight packets for older versions), depending on which of these occurs earlier. You can change the threshold value for the number of packets to investigate via the proc interface:

```
echo number > ➥
/proc/net/layer7_numpackets
```

Starting with version 2.0, you can change the number of bytes to buffer on launching the module. *modprobe ipt_layer7 maxdatalen=no_of_bytes* handles this, whereas a rebuild was required with earlier versions. Under normal circumstances, the default of 2048 bytes should be fine; and remember that too high a value will mean a big performance hit.

When L7 identifies a protocol, it performs the action assigned for the IPtables target. The command for HTTP is *iptables options -m layer7 --l7proto http -j target*. If L7 fails to identify a protocol, as none of the known patterns matches, it will assume the *unknown* name. In other words, you can define an L7 rule to match *unknown*. While L7 is waiting for data, and investigating the packet, it will not assign a name to the protocol.

FTP and IRC-DCC are well-known special cases that have caused no end of fun with firewalls. They use additional channels to transfer data parallel to control data. L7 can identify the control chan-

Installing L7

To support QoS with the L7 patch, you will need the Iproute2 tools and a matching kernel configuration. L7 requires the following packets:

- Current L7 patch [1] (version 2.0 was released recently)
- Protocol descriptions in the form of a regular expression [1] (currently: 2005-11-20)
- Iptables source code [2] (currently: 1.3.4, at least 1.3.0)
- Kernel sources [3]. You may be able to patch distribution specific kernels. L7 v2.0 is designed for kernel versions 2.6.14 and 2.4.31.

Unpack both L7 archives and apply the patches. To do so, change to the kernel and IPtables directories and enter *patch -p1 patchfile*. You must then go on to make the new, hidden file, *iptables-1.3.4/extensions/.layer7-test* in the IPtables package executable (*chmod +x*).

make install gives you a simple way of dropping the protocol descriptions in the default directory, */etc/l7-protocols/protocols/*. If you change the path, you will then need to specify the protocol directory in every L7 rule.

Configuring the Kernel

For meaningful QoS support (and for the examples in this article), you need to enable a number of kernel options (*y* or *m*, details for kernel 2.6), some of which only become visible when you enable *Code maturity level options* / *Prompt for development and/or incomplete code/drivers*.

There are two important options in the *Networking* / *Networking support* / *Networking options* / *TCP/IP networking* configuration area:

- *IP: advanced router*
- *IP: policy routing* and *IP: use netfilter MARK value as routing key* lower down.

The following settings are required in the Netfilter configuration below *Networking* / *Networking support* / *Networking options* / *Network packet filtering* / *IP: Netfilter Configuration*:

- *Connection tracking*, and more specifically *Connection tracking flow accounting* (required for L7). Also, *FTP protocol support* and *IRC protocol support*, to identify FTP and IRC sessions.
- *IP tables support*, and *Layer 7 match support* below this (Figure 2), as well as *netfilter MARK match support*, *Packet filtering*, *Packet mangling* and again *MARK target support* below this.

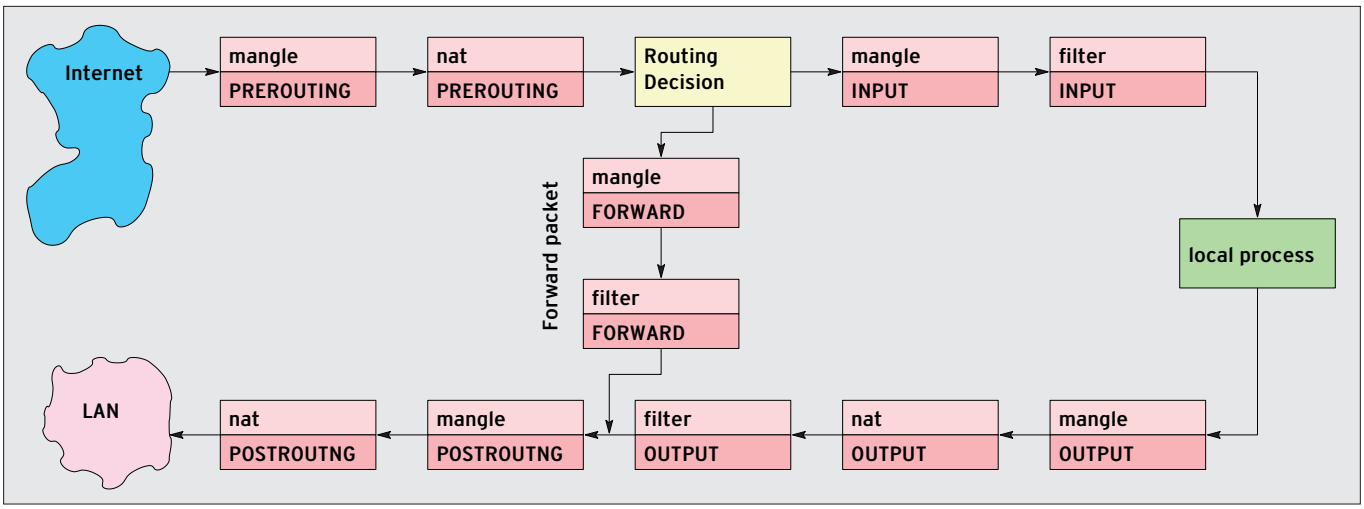

Figure 3: With Iptables, incoming packages first pass through the mangle table in the PREROUTING chain; the NAT table then follows. If the packet is destined for another host, the packet is simply passed to the mangle table in the POSTROUTING chain.

nel, but it will not find the data channels. The *ip_conntrack_ftp* and *ip_conntrack_irc* kernel modules solve this problem; both are part of the IPtables distribution (*FTP protocol support* and *IRC protocol support* in the *Connection tracking* configuration area).

Matches

The best place to put L7 matches is in the *mangle* table. L7 rules will only work in the third table, *filter*, if the chain policy is *ACCEPT*. The reasons become apparent when you consider how a TCP connection is set up:

- a SYN packet arrives to initiate the connection.
- The SYN packet does not possess any Layer 7 protocol information, so L7 fails to identify the protocol, and the L7 rule can't be applied (for the time being).
- The packet passes through all the rules in this chain and ends up at the policy. If this is not an *ACCEPT* policy, the packet is dropped, and the connection is not established.
- In this case, L7 rules are never applied.

Adding your own rules at the back of the chain to wave the first few packets through is an approach that will only work for known ports. But the idea of L7 is to match unknown ports, so additional rules wouldn't be much help. The policies in the other two tables, *mangle* and *nat*, are mainly set to *ACCEPT* for good reasons.

UDP does not have a three-way handshake, as it is connectionless. In the case of a *DROP* policy, response packets will reach the target assuming that the first UDP packet matches the regular expression; this is handled by Netfilter connection tracking.

In Control

L7 will give you best results if it can see both sides of a connection, that is, both incoming and outgoing packets. This is not the case with the *INPUT* and *OUTPUT* chains (or with the *raw* table). Imagine you have an L7 rule in *INPUT*, but the search key that matches your regular expression is in the response. The second packet will not pass through *INPUT* but through *OUTPUT*, and the rule will never get to see it.

The *mangle* table gives you a simple answer. All packets that reach the firewall have to pass through *mangle* in *PREROUTING*, and all outgoing packets pass through *mangle* and *POSTROUTING*. The recommended approach is:

Hunting for QoS Support

The QoS options are located below *Networking* / *Networking support* / *Networking options* / *QoS and/or fair queueing*; after enabling this area, check the following:

- *HTB packet scheduler*
- *SFQ queue*
- *QoS support* and *Rate estimator* below this
- *Packet classifier API*

Surprisingly, the Qconf configuration tool stores two QoS Classifier options one level above, rather than in the QoS area:

- *Firewall based classifier*
- *Traffic policing (needed for in/egress)*

The latter option only appears when *Packet ACTION* is disabled. After completing the configuration, go on to compile the kernel and then build and install the IPtables userspace program. Caution: *iptables* will be placed in */usr/local/sbin* by default, whereas the version without L7 will be in */sbin* and thus higher up your path.

L7 and the NAT Table

A quick glance at the (greatly simplified) diagram showing how a packet flows through IPtables (Figure 3) might lead you to assume that all incoming packets pass through the *mangle* and *nat* tables in the *PREROUTING* chain, and that outgoing packets pass through the same tables but in the *POSTROUTING* chain. If this assumption were correct, you could place L7 in both chains.

Unfortunately, this only applies to the chains in the *mangle* table. The *nat* table only gets to see the first packet in a connection if you are using DNAT or SNAT. Connection tracking code cuts in as of the second packet and takes care of everything else, routing packets around the *nat* table – after all, the NAT decision has already been taken.

Attempts to use L7 at this point are doomed to failure, as the earliest point that L7 can identify a protocol is packet four (following the three-way handshake), or later in most cases.

- In case of forwarding, put the rule either in *PREROUTING* or in *POSTROUTING*.
- For local packets, put the rule both in *PREROUTING* and in *POSTROUTING*.

Apart from this, L7 matches obey the normal syntax. The following rule adds a mark 10 (*--set-mark 10*) to all outgoing packets (*-A POSTROUTING*) belonging to the HTTP protocol (*--l7-proto http*) :

```
iptables -t mangle -A ➥
POSTROUTING -m layer7 ➥
--l7proto http -j MARK ➥
--set-mark 10
```

Note that the protocol name has to match the pattern file name exactly, without the extension, and that the name is case sensitive.

Filters

Contrary to the recommendation of the developers, you can use L7 to completely block some protocols. However, if you are going to try this, you should proceed with caution, check the results, and take additional steps, if needed, to avoid opening up gaping security holes. This simple rule blocks direct access to newsgroups:

```
iptables -A FORWARD -p ➥
tcp -m layer7  --l7proto ➥
nntp -j DROP
```

A news proxy or server running on your firewall would not be affected by this forwarding rule.

Listing 2: Tagging Groups

```
01 IPT_PRE="iptables  -t mangle
   -A PREROUTING  -m layer7
   --l7proto"
02 IPT_POST="iptables -t mangle
   -A POSTROUTING -m layer7
   --l7proto"
03 $IPT_PRE  fasttrack -j MARK
   --set-mark 13
04 $IPT_POST fasttrack -j MARK
   --set-mark 13
05 <I>[...]<I>
06 iptables -A FORWARD -m mark
   --mark 13 -j DROP
07 iptables -A INPUT   -m mark
   --mark 13 -j DROP
```

File sharing is something that most admins want to avoid. If you are one of those admins who are constantly battling to keep users from accessing file sharing services, L7 can make things a lot easier. Port-based filters fail if an E-Mule server, for example, is listening on port 80. Some peer-to-peer software tools just use HTTP. L7 solves this problem quite reliably in most cases using excellent patterns for E-Donkey/E-Mule, Bittorrent, and Fasttrack.

Outlawing File Sharing

The following variant tags file sharing connections in the *mangle* table using a *MARK* target and leaves the actual filtering to the *filter* table. This approach allows you to form groups of protocols that are governed by a just a few filter rules. The group feature is a good option for many environments, as it allows you to tweak one knob to catch all the protocols in a group. Listing 2 shows an excerpt from an IPtables script that does this.

The first two lines in Listing 2 initialize two variables to save typing later. Lines 3 and 4 of the listing tag packets belonging to the Fasttrack protocol with a 13 mark – in *PREROUTING* and *POSTROUTING* – just to be on the safe side. You need to repeat these two lines for other protocols. The last two lines in Listing 2 send the packets off to digitial Nirvana.

Note: The *fasttrack.pat* pattern file states that the pattern can detect downloads, but not searches. This limitation is not really a major problem in most scenarios. The file also says that Fasttrack uses normal HTTP requests, and this is why you need to insert this rule into the chain before your HTTP rule. Fasttrack is used by Kazaa, Morpheus, E-Mesh, and Grokster.

VoIP is becoming increasingly widespread. The quality of the L7 pattern for matching VoIP is mediocre. Additionally, IPtables helper modules are required for H.323 and SIP, as both protocols operate on multiple ports at the same time. These modules require Kernel 2.6.11 or later. No additional modules are required for Skype.

Better Patterns

How well the Layer 7 patch performs in productive use depends on the quality of the pattern files. Better quality VoIP protocol patterns would be useful. The amazingly simple patterns are already quite capable of identifying standard protocols, even trickier candidates such as FTP or IRC-DCC. And the pattern file quality for the major peer-to-peer protocols is also fairly good.

L7 can make a system administrator's life easier, especially in combination with traffic shaping: it adds a few more weapons to the admin's armory in the constant battle against peer-to-peer network users. Instead of blocking file sharing completely, you can simply slow down the transfer rate. Most users will not blame the slow connections on their local networks – and even if they did blame the network, they couldn't really complain to anyone, because most enterprise policies have rules that outlaw file sharing. ■

SSL and L7

L7 can't identify protocols in SSL tunnels (HTTPS, IMAPS…). The only clear text packet following the TCP/IP handshake is the SSL server certificate. The key exchange follows immediately after this, and all other exchanges between the client and server are encrypted. This forces admins to treat all SSL tunneled protocols in the same way.

One thing you might try is to look for criteria in the certificate, that is, you might decide not to trust individual certification authorities. This is why the pattern file for SSL is not titled *ssl.pat* but *validcertssl.pat*. The pattern only permits known CAs and will therefore not work with self-signed certificates, although you could write your own pattern to handle this.

INFO

[1] L7 source code and protocol definitions: *http://l7-filter.sourceforge.net*

[2] IPtables source code: *http://www.netfilter.org*

[3] Kernel source code: *http://www.kernel.org*

THE AUTHOR

Jörg Harmuth is a freelance IT security consultant and network specialist. He spends most of his leisure time with his family, but he also enjoys the occasional expedition into the murky depths of OSI Layers three and four.