

# FRAGMENTS: SMALL PACKETS BIG PROBLEMS

Paul Rusty Russell

Welcome back, gentle reader. Last month I provided a brief introduction to packet filtering under Linux — how to get your Linux box to drop specific network packets which pass through it. This month I'm going to do something I wouldn't ordinarily, but hey, I was busy working on the next-generation packet filtering stuff when the deadline for this column hit me; just don't tell the editors and maybe we can get away with it :).

In this column I will discuss packet fragments. Fragments have historically presented us with a number of problems and we need to understand what they are

In my previous column, I wrote about messages on the Internet being broken up into "packets", each one containing a "header", with administrative details (where the packet is from, where it's going to, what type of packet it is, etc.), and a "body", containing what we actually want to send.

An example of an IP (Internet Protocol) header is shown in Figure 1. The header contains a number of fields, the most important of which are:

- 1) IP Version Number: The current Internet Protocol is version 4.
- 2) Source Address: The IP Address from which the packet is being sent.
- 3) Destination Address: The IP Address to which the packet is going.
- 4) Protocol: The type of packet being sent; the most common is TCP (Transmission Control Protocol), used for Web traffic and e-mail.

Of course, the less commonly used fields are the ones most likely to cause problems for network implementations; problems with the important fields show up immediately.

Every type of network has a Maximum Transmission Unit, or "MTU". This specifies how large a single packet can be on that type of network. For an Ethernet, it is 1500 bytes. For many dial-up PPP connections, it is configurable: often 296 bytes. A packet traversing a large network like the Internet may be too big for certain segments of the network. It is then the responsibility of the devices connecting those segments to split the packet up into "fragments".

For example, Figure 2A shows an IP packet with a total size of 1500 bytes (20-bytes for the IP header, and 1480 bytes for the body). If this packet hit a link with a MTU of 296 bytes, it would be fragmented into 296 byte packets and look like Figure 2B. Note that each fragment gets a complete IP header, plus a section of the body.

Now, instead of receiving one packet of 1500 bytes (1480 data bytes plus 20 bytes of IP header), we receive six packets of 1600 bytes total (1480 data bytes plus 120 bytes of IP headers). This means more network traffic and a greater chance of lost packets (most protocols allow for lost packets, but it slows them down significantly).

Modern TCP implementations (such as Linux's) go to great lengths to avoid sending packets that need fragmentation. They do this by using the other of the two flags in the IP header, called "Don't Fragment" (DF): if this is set, then a packet needing fragmentation will be dropped and a special "destination-unreachable" packet will be sent back to the source. This packet contains the reason why the destination was unreachable (in this case, "fragmentation needed and DF set"), and the MTU of

# FRAGMENTS: SMALL PACKETS BIG PROBLEMS

Paul Rusty Russell

the path which required fragmentation. If a “destination unreachable” packet is received, the source will then transmit a smaller packet, which will fit down that path. This is called “Path MTU discovery”.

Fragments have been the cause of many show-stopping bugs in TCP/IP network stacks. A few of those bugs are summarized below:

**Ping of Death:** Normally, a packet cannot be more than 65535 bytes long. This is the largest number that can fit in the IP header’s “Total Length” field. If we create a fragment with length 1000 bytes and set the “Fragment Offset” to 65400, some machines try to reassemble a packet of length 66400 and fall over.

Many operating systems were vulnerable to this bug (including Linux before version 2.0.24), which was discovered because the Microsoft Windows ping program allowed sending of oversized ping packets. Any fragmented packet could have the same effect, but the name “Ping of Death” stuck.

**Teardrop, Bonk, Boink and Newtear:** Normally a packet’s fragments will line up head-to-toe to create a whole packet. If we carefully craft two fragments which overlap, we can confuse the machine; in Linux the code to merge two overlapping fragments didn’t handle the case where a single fragment is completely contained within a previously received fragment. Teardrop was the name of the original exploit program as posted to the Linux-kernel mailing list, and that post noted that Windows 95 and Windows NT machines had a similar bug.

Bonk (and a modified version, called Boink) and Newtear were variants of Teardrop which affected even Microsoft Windows machines that had been patched against Teardrop. I’m happy to say that Linux fixed this correctly the first time in version 2.0.32.

**Fragment Bombs:** Normally a machine collects fragments until it has an entire packet. Of course, in case one of the fragments gets lost, you should eventually give up and drop the non-reassembled fragments. If you don’t, you will eventually run out of memory. Some MS-DOS based implementations were vulnerable to this, but it’s not a problem any more.

**Zero Length Fragments:** This was a Linux-specific problem introduced in the 2.1 series and fixed in 2.2.4. It’s a fragment bomb with a twist: if a packet claims to be a fragment of zero length, it is stored in the fragment queue like any other fragment, waiting for reassembly. Unfortunately, there was an error in the fragment clean-up code, and such fragments never got dropped, eventually causing Linux’s networking to stop.

**Filtering Out “Destination Unreachable” Packets:** Earlier, I mentioned “destination unreachable” packets. This is a type of ICMP packet: ICMP is the Internet Control Message Protocol; the message-runner of the Internet.

You don’t need a license to set up a packet filter — and it shows! There are a number of sites which block all ICMP packets in a misguided attempt to be more secure. Of course, this breaks path MTU discovery, and there’s nothing you can do about it (short of disabling path MTU discovery altogether). The signs of this are that connections to the site will stall forever, as the side sending large TCP packets never receives the “fragmentation needed and DF set” message required to make it reduce packet size.

**Masquerading and Path MTU Discovery:** For a long time, there was a bug in Linux boxes doing IP masquerading between networks with different MTUs. Masquerading is a technique that makes it

# FRAGMENTS: SMALL PACKETS BIG PROBLEMS

Paul Rusty Russell

possible for many computers on a private network to “share” one IP address. An Internet Service Provider will often provide PPP users with a single IP address, and you must send out packets with that source address on them in order to receive replies. Masquerading allows you to make the most of that single link. One computer on a private network is attached to the Internet via the PPP link and thus has its own IP address. The machines without IP addresses forward messages bound for the Internet to the machine with the IP address. Messages going out to the Internet appear to come from that machine, and messages coming in from the Net are translated back for the private network.

In 2.2 kernels prior to 2.2.4, a packet would be translated in preparation to go out over the dial-up link, but then the Linux box would realize that the packet wouldn't fit. If the “Don't Fragment” flag was set, the Linux box would generate an ICMP “fragmentation needed and DF set” message for the too-large packet. Unfortunately, the machine would then ignore the message (you always ignore ICMP errors about packets you never sent), and the source machine wouldn't know to resize the packet to correspond with the destination MTU.

This problem went largely ignored because the symptoms are very similar to the “Filtering Out Destination Unreachable Packets” case. Eventually, Andi Kleen posted a three-line explanation of the problem to the linux-kernel list. It was fixed a few days later.

## Packet Filtering Fragments

In Linux 2.0, non-head fragments were never filtered. In Linux 2.2, you can now specify rules for filtering non-head fragments (using the ipchains -f flag as described in the ipchains HOWTO at <http://www.rustcorp.com/linux/ipchains>). Still, filtering non-head fragments is difficult because a packet filter often looks inside the IP packet header to make intelligent decisions. With non-head fragments, this cannot always be done easily. Fragmented TCP packets provide a good example of how this can become complicated.

TCP is a protocol that adds features to IP, including retransmission of lost packets, dealing with non-consecutive packets, and congestion control. If the “protocol” field in the IP header is set to TCP (whose assigned number is 6), then a TCP header immediately follows the IP header. TCP packets are often filtered based on rules that are compared to fields in the TCP header.

The TCP header contains source and destination port fields. These identify different data streams coming from the same host. For example, a Web server listens on port 80, and a mail server listens on port 25, so if your packet filter doesn't allow TCP packets to port 25, no one can talk to your mail server.

The other important TCP header field is the “SYN” (synchronize) flag; this flag is set for the very first packet in a TCP session. So if you block any TCP packets coming in to your network with this flag set, you essentially create a “one-way” firewall that only allows outgoing TCP connections.

What makes filtering TCP fragments so difficult is that once a packet becomes fragmented, the TCP header is only included in the head fragment. Since packet filters make decisions based on the header, this is a problem. However, if we wish to filter all TCP packets, we can do so by only allowing non-head fragments through. Since without the header the host does not have the information it requires to reassemble the fragments, it should just drop them. This makes it appear as though the whole packet was blocked.

## Linux As a Defragmenting Router

# FRAGMENTS: SMALL PACKETS BIG PROBLEMS

Paul Rusty Russell

When you compile a new Linux kernel, you can tell it to “always defragment”. Normally, only the final recipient of a packet will glue the fragments back together, but with this option, your Linux box will defragment any packets that pass through it. As you can imagine, this solves many of the problems discussed so far.

There are a few problems with being a defragmenting router though, the main one being that all fragments must go through it! However, for a firewall this is fine, since we want all packets to pass through the firewall anyway. But watch out — if you have two Linux firewalls connecting your network to the outside world and only half of the fragments go through each one, they will both gather some of the fragments and wait for a complete packet that will never arrive.

## Other Potential IP Problems

The other IP field which causes security-conscious people grief is the Source-Route option. The IP header can have extra things tacked on the end called options. Normally there are no options, so the IP header is 20 bytes long.

A machine which receives a packet decides where to send it next. This is called “routing”. Usually the “next hop” is decided based on the destination IP address field, but you can override this using the “Source Route” IP option. Many firewalls connect two networks but their “IP Forwarding” is switched off, so they will never route between the networks. Traditionally, however, source-routing overrides this, and so can be used to squeeze a packet past such machines.

In Linux 2.0, you can enable “IP: Drop source routed frames” when you configure the kernel (it's normally enabled). In 2.2, it's always enabled unless you explicitly set it using the `proc/sys/net/ipv4/conf/all/accept_source_route` file. This is set in the same way that Source Address Verification would be, as indicated in last month's column.