Detecting packet injection: a guide to observing packet spoofing by ISPs

Introduction

Certain Internet service providers have begun to interfere with their users' communications by injecting forged or spoofed packets – data that appears to come from the other end but was actually generated by an Internet service provider (ISP) in the middle. This spoofing is one means (although not the only means) of blocking, jamming, or degrading users' ability to use particular applications, services, or protocols. One important means of holding ISPs accountable for this interference is the ability of some subscribers to detect and document it reliably. We have to learn what ISPs are doing before we can try to do something about it. Internet users can often detect interference by comparing data sent at one end with data received at the other end of a connection.

Techniques like these were used by EFF and the Associated Press to produce clear evidence that Comcast was deliberately interfering with file sharing

applications; they have also been used to document censorship by the Great Firewall of China. [1] In each of these cases, an intermediary was caught injecting TCP reset packets that caused a communication to hang up - even though the communicating parties actually wanted to continue talking to one another. In this document, we describe how to use a network analyzer like Wireshark to run an experiment with a friend and detect behavior like this. Please note that these instructions are intended for use by technically experienced individuals who are generally familiar with Internet concepts and are comfortable installing software, examining and modifying their computers' administrative settings, and running programs on a command line.

Requirements

Making use of these techniques requires some general understanding of Internet technology and some technical expertise. If you don't understand the process, you may not produce meaningful evidence about what your ISP is doing. Although we have attempted to explain most of the network concepts and principles involved, it may prove helpful to have read at least one technical book or web site about the TCP/IP protocol suite before beginning.

The test described here must be performed in conjunction with a friend who is using a different Internet connection (and therefore is probably in a different location). Both you and your friend must have a good understanding of the process described here; this test relies on comparing observations made at two different locations in order to find differences between them, so it would not be meaningful if performed by one party alone. Therefore, these instructions are primarily useful for testing peer-to-peer applications or applications for which you can run your own server. It is therefore difficult to confirm if an ISP is blocking a third-party service like Google unless the operator of that service is interested in participating directly in the tests. [2]

The tests described here are most relevant as a means of debugging a specific observed and reproducible problem (for example, an inability to connect to another party) rather than as a speculative means of investigating ISP behavior. This is primarily because of the limitations of tools to automate the process of comparing packet traces from two ends of a connection. Traditionally, this comparison had to be performed by hand, which can be a quite laborious process if one isn't looking for anything in particular. EFF has begun to develop tools to automate this process so that large packet traces can be compared automatically and packet injection can be detected even when it is not

specifically suspected.

Each party participating in the experiment must have all of the following:

- a computer capable of running Wireshark, with appropriate privileges to install and run it;
- the ability to connect this computer directly to the Internet, with a public IP address, outside of any firewalls (for example, not via a typical home wireless router);
- the ability to determine the computer's public IP address;
- the ability to disable any firewall software running on the computer itself;
- some application to test, and the ability to configure that application to communicate directly with the other party (by IP address).

A note on privacy

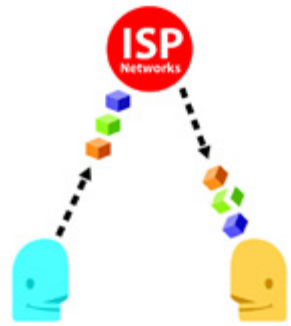
Using a packet sniffer can capture all of the traffic passing by your computer (including all of your communications, and potentially communications of other users on the same network); if your computer is connected to a wireless network, for example, the packet sniffer may record everything you do and everything everyone else on the wireless network does on-line. Please do not record other people's communications without their consent. Doing so is impolite and, under some circumstances, may be prohibited by law. One way to avoid recording third parties' communications is to avoid using promiscuous mode for your packet capture, unless you specifically need it.

If you produce a capture file (packet trace) with evidence of the results of your experiment, please be aware that the capture file will reveal your IP address, the IP address of the other person involved, and a complete record of everything you did on-line during the course of the experiment. For example, if you downloaded a file, the packet trace will typically reveal which file (and even include the full contents of that file). If you browsed the web or checked your e-mail while the packet sniffer was running, the identities and contents of web pages you visited and e-mail messages you downloaded may appear in the packet trace. In addition, any HTTP cookies sent by your browser (which might include your username and password for web sites you visited!) will be included in the trace. You should exercise caution when publishing or sharing a packet capture file to ensure that you don't reveal more information than you intended.

Theory

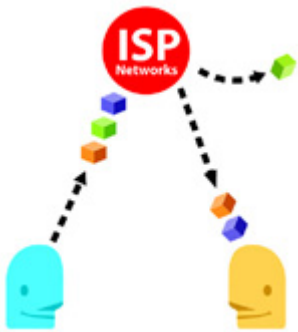
The traditional Internet architecture is characterized by an end-to-end design in which ISPs passively forward unmodified packets from one user to another. This means that, in the best case, every packet sent by one user should be received as an identical packet at the other end. There are several reasons that this ideal might not be attained even in the absence of packet injection by ISPs:

Fragmentation.



The Internet Protocol standard permits ISPs to fragment packets that are too large (for example, because a particular network technology used by an ISP has a maximum packet size). The packets are then broken up into smaller fragments which arrive separately at the destination; the destination computer is responsible for reassembling the fragments. Fragmentation has become somewhat less common in practice for reasons that may include conservative packet size defaults in operating system network code and mechanisms like path MTU discovery (to automatically select a packet size that is small enough to avoid fragmentation). In test results we've seen so far, fragmentation generally did not occur, and we will ignore this possibility here, although it should be considered as a possible cause of any observed discrepancies between packets sent and received.

Packet loss.



Under conditions of network congestion, it is normal for some packets to be discarded rather than forwarded, a phenomenon called packet loss. Packet loss is normally measured as a percentage; the ping utility measures packet loss with ICMP echo request packets, counting how many ICMP echo replies are received in response to a certain number of probes. High rates of packet loss could be caused intentionally by an ISP as a means of reducing the performance of a targeted application or protocol, but they can also occur as a result of congestion on the network or other technical problems. When a packet is lost (also called a "dropped packet", "dropped frame", or "dropped segment"), it is not received by the destination at all. Some higher-level Internet protocols include mechanisms for coping with packet loss, such as TCP's mechanism for

explicitly retransmitting data from packets that are lost.

Reordering.

Sometimes packets are not delivered in the same order in which they were originally transmitted. If packet B was transmitted after packet A, receiving packet B at the other end does not mean that packet A has been dropped; it might still be on its way. TCP can also generally correct for reordering. Like packet loss, reordering could be used intentionally by an ISP to degrade an application, but also occurs normally in the course of Internet routing.

Spoofing.

Spoofing or packet injection occurs when an entity other than one of the endpoints generates traffic using the source address of an endpoint. Spoofing is the most straightforwardly detectable means of interference with Internet traffic because it produces concrete evidence in the form of the anomalous spoofed packets, and because it does not occur normally in traditional Internet routing.

Spoofing can be detected by looking for packets that were received by one end but never sent by the other end. If user B receives a packet apparently from user A that user A has no record of having sent, user B can conclude that someone in between the two has spoofed this packet. The remainder of this article describes the means of collecting packet traces to allow the packets actually transmitted between two users to be compared in this way.

Spoofing need not involve preventing or blocking communications; it could also involve changing their content, as with a transparent proxy. [3] Some ISPs have been experimenting with modifying HTML in third-party web pages on the fly in order to inject advertising. Here is a recent real-world example:

In this image supplied by Chris Palmer, a wireless ISP has modified the source

code of Google's home page to add its own advertising, presumably by spoofing packets with a transparent HTTP proxy. Although similar behavior often results from adware, in this instance Palmer verified that the ISP was at fault by installing a fresh instance of Windows inside a virtual machine. [4]

Setup

Step 1. Install Wireshark

Download a copy of Wireshark for your platform from the Wireshark home page at http://www.wireshark.org/. (Wireshark is also prepackaged for most Unix-like operating systems and may be available from your distributor's package repository. In older operating system releases, it may still be packaged under its former name, Ethereal.) Install Wireshark and make sure that you can run the program. [5]

Step 2. Connect directly to the Internet

In order to obtain the most valid and conclusive results, we strongly recommend that your computer be directly connected to the Internet, with a globally-valid public IP address, without any firewalls or network address translation (NAT) routers. Performing these tests from behind NAT could produce valid results but creates some uncertainty about whether unexpected network behavior is due to an ISP or a local NAT router. A public IP address is one that another party can use directly to communicate with you without the need to configure a tunnel or firewall rule or use a proxy to connect. [6]

If you are using an institutional network connection, such as at a school or business, that has a firewall that you are not permitted to disable, you may still be able to perform these tests, but any packet spoofing you detect may be a result of your institution's firewall rather than its upstream ISP connection. As we describe below, observing packet spoofing shows that someone is doing it, but does not directly reveal who. We want to reduce the number of possible responsible parties.

Therefore, before beginning the test, disable any firewalls and NATs located between your computer and the Internet – including both software or "personal" firewalls running on your computer and firewall appliances or firewall functionality in routers. As an alternative, connect your computer to a

point on the network located outside of any firewalls or NAT routers (for example, by directly connecting it to a cable modem or DSL modem). Some network designs and documentation refer to this location as a DMZ.

Step 3. If possible, disable TCP and UDP checksum offloading and TCP segmentation offloading

Checksum offloading (sometimes called "TCP checksum offloading", although UDP checksums may also be offloaded) is a feature of some recent Ethernet cards, particularly Gigabit Ethernet–capable cards, that allows the Ethernet card to construct portions of some network packets in hardware, saving load on the CPU. However, the use of checksum offloading makes packet captures inaccurate because it prevents the local operating system from seeing what was actually transmitted. This may cause a discrepancy since one end mistakenly thinks it sent something slightly different from what the other end correctly received; the resulting mismatch of TCP or UDP checksum values could be misinterpreted as tampering by an ISP, since ISPs are not supposed to alter these checksums. Checksum offloading should, if possible, be disabled at both ends before beginning the experiment. If you know that your Ethernet card or network driver does not perform checksum offloading, you do not need to disable it. It may also be possible to get valid results when checksum offloading is enabled; workarounds for this purpose are described in a later section. Here are typical means of disabling checksum offloading on several popular operating systems:

On Linux (as root):

ethtool -K eth0 rx off tx off *(choose correct network interface if not eth0)*

On FreeBSD (as root):

ifconfig em0 -rcxsum -tcxsum *(choose correct network interface if not em0)*

On MacOS (as root):

sysctl -w net.link.ether.inet.apple_hwcksum_tx=0

sysctl -w net.link.ether.inet.apple_hwcksum_rx=0

(Note that this may cause some local applications to work incorrectly!)

On Windows

Right-click My Computer, then select Device Manager / Network Adapters / (select device) / Properties / Advanced; then disable checksum offloading, if the option is available.

For general information about checksum offloading and why it can cause errors when capturing packets, see http://www.wireshark.org /docs/wsug_html_chunked/ChAdvChecksums.html and http://www.wireshark.org/faq.html#q11.1. Note that the approach suggested there of disabling TCP checksum verification in Wireshark does not help for our purposes, because we want to compare packets; having TCP checksums that are different across capture files will still appear as a discrepancy between those capture files even if the checksums' values are never verified.

If your system performs checksum offloading and you are unable to disable it, other options are available. The pcapdiff program described below allows you to ignore TCP and UDP checksum values entirely, in case you have reason to believe that checksum offloading is in use. You can also perform the capture on a separate machine distinct from the computer that is generating the test traffic - as long as it is connected to the same local area network and is able to see the traffic passing by. In this case, the capture should be performed in promiscuous mode (see footnote 3 above) and, on a network used by multiple people, extra care should be taken to avoid capturing other users' communications without their knowledge.

Another form of offloading that can cause packet capture accuracy problems is TCP segmentation offloading, also known as large segment offload. In TCP segmentation offloading, an Ethernet card, rather than operating system software, splits a large TCP packet into multiple TCP packets. This can cause a serious discrepancy in the number of packets a host believes it transmitted as against the number of packets it actually transmitted; since packets are split up by the network card in a way invisible to the sender's operating system, every TCP packet large enough to be split may appear to be "forged", since the sender will have no record of having sent any of the received packets in the form in which they were received. TCP segmentation offloading should also be disabled if your system uses it. pcapdiff, for example, is not able to ignore TCP segmentation discrepancies in the same way that it can ignore TCP and UDP checksum mismatches. See http://www.inliniac.net/blog/2007/04/20 /snort_inline-and-tcp-segmentation-offloading.html for a discussion of TCP segmentation offloading's consequences for packet sniffing, and information

about disabling it on Linux. In general, the validity of the results of packet capture experiments will be improved by disabling all available offloading features.

Step 4. Determine local IP address

Next, determine the IP address of your computer. You can obtain this locally from your computer's network configuration tools, and you can also obtain it from a web site such as http://whatismyipaddress.com/ or http://www.whatismyip.com/, which displays the IP address from which it is being accessed. Use both methods to ensure that you are really directly connected to the Internet and not using a proxy server or NAT connection. (If you are not using NAT, your computer's locally-configured IP address should be identical to the IP address seen by web sites and other Internet users. If they still disagree, it's possible that your ISP or institutional network is forcing all users to use NAT or a proxy, rather than providing direct access to the Internet.)

From here on, we will suppose that your IP address is 12.13.14.15 and that the IP address of the person at the other end is 4.8.16.32. You should replace these example addresses with the real IP addresses involved. [7]

Step 5. Confirm IP addresses and test connectivity

To ensure that both you and the person at the other end have correctly determined your IP addresses, try to perform some operation that allows you to communicate with one another by specifying each other's IP addresses. You could use the ping command or try the application that you eventually plan to test out, such as a P2P file-sharing client or VoIP application. If you can't establish some kind of end-to-end communication using the IP addresses you've determined, you'll need to debug this problem before proceeding any further. Possible causes could include the presence of a firewall, including a software firewall, that has not yet been disabled at one end.

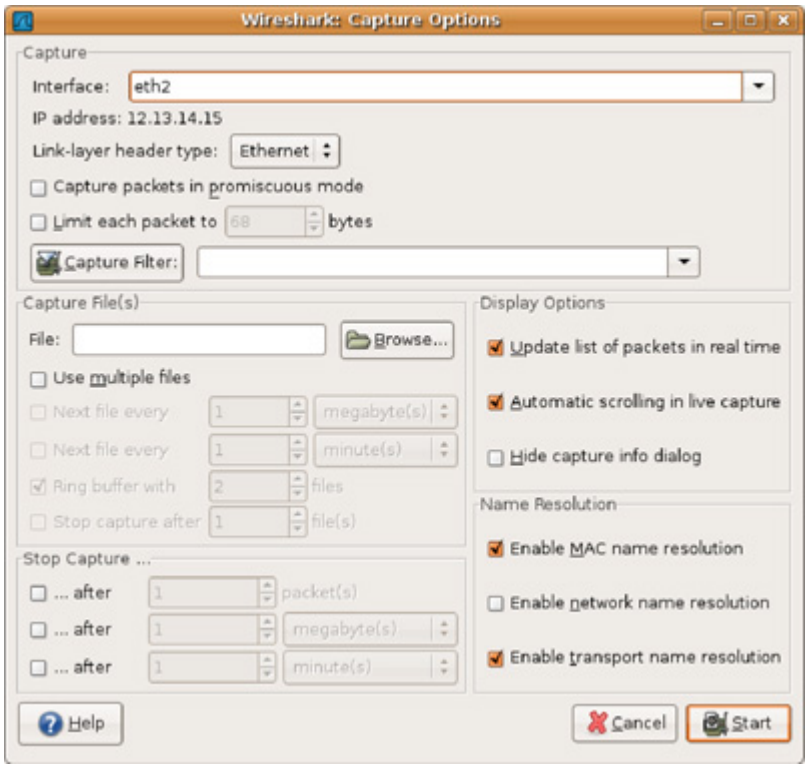
Step 6. Synchronize computer clocks

If your operating system supports it, make sure that your computer's clock is synchronized to an authoritative Network Time Protocol (NTP) network time server, so that the dates and times recorded in your packet capture will be accurate and will correspond to those recorded by the other computer. This will help make results or log entries from multiple computers easier to compare.

Running the test

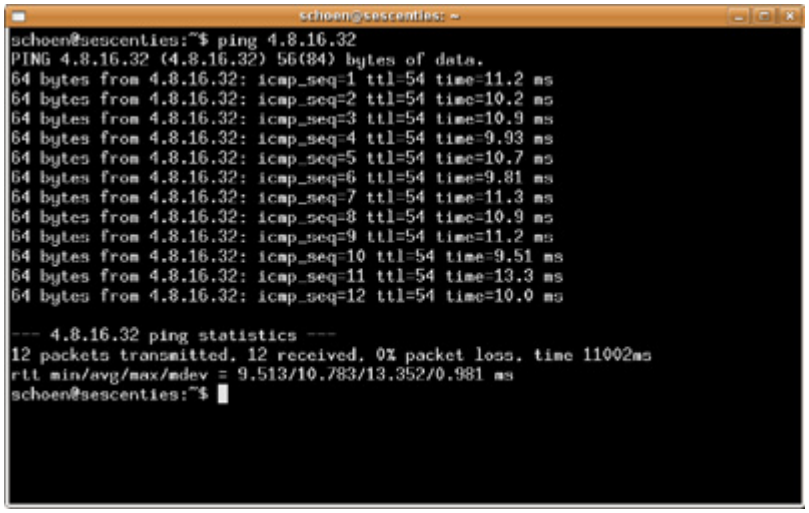
Start Wireshark (with administrative privileges, e.g. root privileges, sufficient to perform a raw packet capture [8]). Select "Interfaces" from the "Capture" menu. Choose the interface corresponding to the network device you will use to capture packets; the IP addresses bound to all available network interfaces are displayed, which may help you distinguish them if you are unsure which network interface is which. Click the Options button for the correct interface.

In the Capture Options dialogue, ensure that the IP address you expected your computer to be using is displayed in the "IP address" field. We recommend setting a capture filter to ensure that only packets directly to or from the other computer will be captured. If the other computer's IP address is 4.8.16.32, a suitable capture filter string is "host 4.8.16.32". We also recommend setting "Update list of packets in real time" and "Automatic scrolling in live capture", which help you watch the capture process while it's underway, unless your computer is too slow.

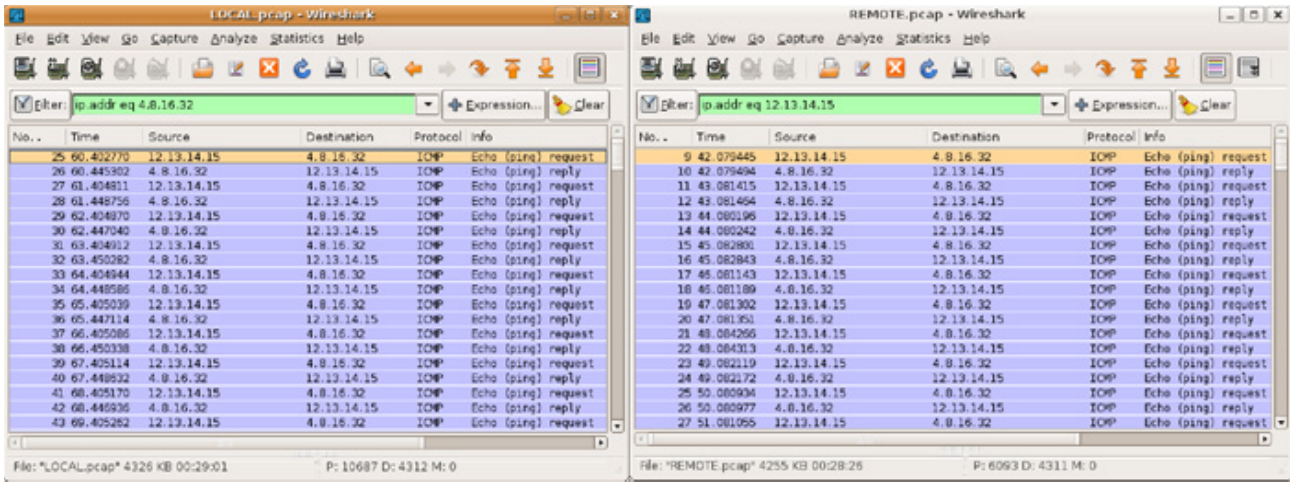


When you're ready to begin capturing packets, click the Start button. If you've set a capture filter to limit capturing to traffic to or from the other computer, you will probably not see any packets appear in the capture until you deliberately generate some traffic between the two computers. To ensure that

the traffic is showing up, you should ensure that Wireshark packet captures are running at both ends, and then have one computer ping the other computer by IP address. This generates a steady stream of ICMP echo request and echo reply packets. Current Unix, Windows, and MacOS operating systems all allow you to start the ping process by typing ping 4.8.16.32 at a terminal (command-line) prompt. (On some systems, the ping will stop automatically after a predetermined number of pings; on others, you can interrupt it by pressing Ctrl+C.)



If both ends of the connection are capturing data, the ICMP packets that represent ping requests and replies should appear in the Wireshark window at each end. These packets should be identical at the IP layer and - unless the ping utility itself reported packet loss - there should be an identical number of packets seen from both ends. This test will show if the capture process is set up and working properly.



Here, we see how the capture process appears at both ends following a large number of ping requests from 12.13.14.15 to 4.8.16.32. The packets displayed agree one-to-one (as could be verified by looking more closely at the packets' contents than we do here); there are no dropped packets and no spoofed packets. (Note that you can't see this view from both ends in real-time; this screen shot was created after the fact by copying a saved capture file from one computer onto another. While the capture is running, each party sees only one of the two windows displayed here; in order to verify that the packet captures correspond properly while they're in progress, you'll need to use some other means of communication to talk to the person at the other end, such as a telephone or instant messaging application. You'll use this channel to coordinate your activities and to compare notes.)

Once you're confident that both computers are talking to each other over the Internet correctly and are saving a valid packet trace, you can begin to gather experimental data about whatever application is of interest to you - or try to reproduce any reported or conjectured errors or problems. For example, for our tests with Comcast, we configured one of the computers as a BitTorrent tracker and seeder and gave the other computer a BitTorrent file instructing it to attempt to download the file hosted by the first computer. The details of what you'll test depend on what you're interested in and will require you to be familiar with the application you're testing in some detail. (In the BitTorrent example, it's not sufficient simply to have both parties start running BitTorrent at this point; rather, one computer needs to be configured explicitly to offer a download to the other computer, which, in turn, needs to be configured to request a download from that computer.)

When your experiment is complete, you should stop the capture and save the resulting capture files to disk. You can then exchange these files with the person at the other end - by e-mail, for instance. Wireshark can be used to open and display a saved capture file generated on another computer.

There are many other network analyzers or packet sniffers that could be used instead of Wireshark. We have chosen to describe Wireshark because it is powerful, user friendly, open source, and available for several platforms. As long as the network analyzer you use can save its packet traces in pcap format, they can be read by a wide variety of software, including Wireshark; thus, meaningful and comparable results could be obtained using other software. If you're capturing packets on a device that can't run Wireshark - such as a remotely-accessible server in a colocation facility with no graphical user interface - you should consider the tcpdump program, which is a standard part

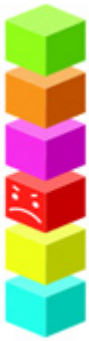
of many Unix-like operating systems and is available at http://www.tcpdump.org/.

It is important to be aware that some network analyzers default to capturing only a portion of each packet. Wireshark's option to "Limit each packet to 68 bytes" in the screen capture above is (correctly) disabled by default, but the corresponding option in some other programs could be (wrongly) enabled by default. To meaningfully compare the resulting packet traces, the packet size limit should be set to the largest possible value (usually 65535 bytes) or disabled entirely. For example, using the tcpdump utility, an appropriate command line would be

```
tcpdump -v -s 0 -w packet-trace.pcap
```

Setting `-s 0` sets the packet size limit to "unlimited" instead of tcpdump's default of 68 bytes. (When running tcpdump on a computer with multiple network interfaces, it may also be necessary to specify a network interface with the `-i` option.)

Interpreting the results



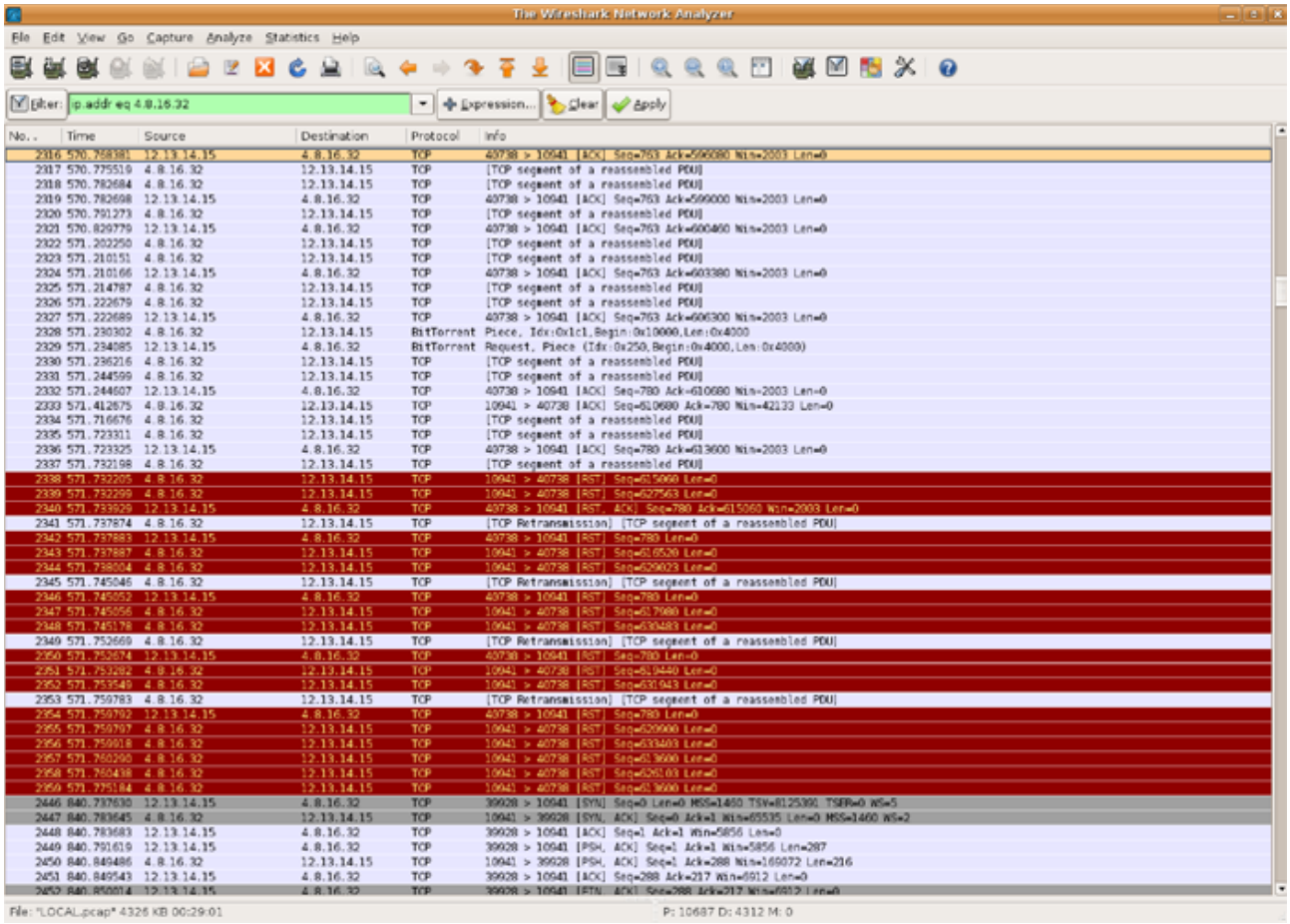
As we described above, packet trace files generated on separate machines that were communicating directly with one another can be compared to see how packets sent by each computer correspond to packets received by the other computer. Since a simple file transfer or VoIP conversation could generate thousands of packets, the lack of automated tools to perform this comparison can make the process tedious. We intend to collaborate with other interested parties to produce such comparison tools in the near future, enabling much larger data sets to be analyzed quickly for spoofed packets, even where specific sorts of spoofing are not suspected. Here are a few tips for comparing packet traces by hand:

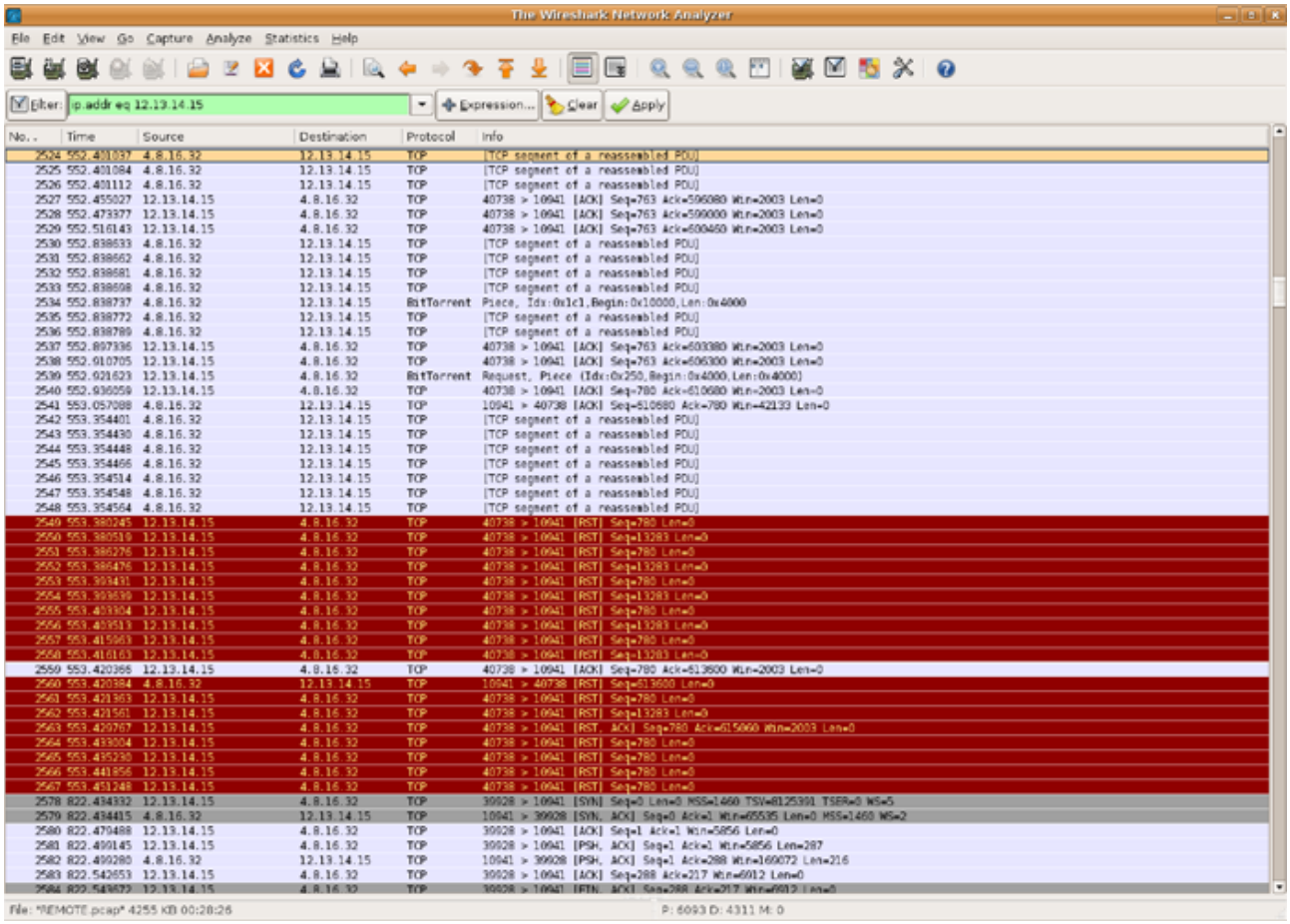
- Spoofed packets may correspond to protocol errors or extreme delays reported or observed in application software; for example, if a client program gives an error like "Connection reset by peer", "Connection closed by foreign host", "Lost connection", etc., or data rates suddenly

drop, packet spoofing may have occurred. Noting the timing of suspicious errors may provide guidance for where to look for spoofed packets in a packet trace file.

- Spoofed packets used to disrupt connections are often TCP segments with the FIN or RST flags set (also known as "FIN packets" and "RST packets"); each of these flags indicates that a computer does not want to continue a TCP conversation. Wireshark can be configured to color these packets differently so that they stand out in a packet display. Keep in mind that there are legitimate uses for FIN and RST packets within the TCP standards and that the presence of forged FIN or RST packets, not the presence of such packets generally, is suspicious. (A client software or firewall bug, for example, could cause one end of a connection to disconnect prematurely – but that isn't the ISP's fault!)
- If a problem you're investigating is widespread, someone may already have published claims about precisely when or under what circumstances spoofed packets may appear; you can consult the details of other people's allegations to see whether you can reproduce these claims for yourself.

Setting aside the possibility of fragmentation, we have explained that packets are spoofed when they are received by one computer but were not transmitted by the other computer. Below, we give examples of summary views showing a BitTorrent transfer disrupted by TCP RST spoofing:


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter: ip.addr eq 4.8.16.32
Expression... Clear Apply
No.. Time Source Destination Protocol Info
2316 570.758381 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=763 Ack=596080 Win=2003 Len=0
2317 570.775519 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2318 570.782684 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2319 570.782698 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=763 Ack=599000 Win=2003 Len=0
2320 570.791273 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2321 570.829779 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=763 Ack=600480 Win=2003 Len=0
2322 571.202250 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2323 571.210151 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2324 571.210166 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=763 Ack=603380 Win=2003 Len=0
2325 571.214787 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2326 571.222679 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2327 571.222689 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=763 Ack=606300 Win=2003 Len=0
2328 571.230302 4.8.16.32 12.13.14.15 BitTorrent Piece, Idx:0x1c1,Begin:0x10000,Len:0x4000
2329 571.234085 12.13.14.15 4.8.16.32 BitTorrent Request, Piece (Idx:0x250,Begin:0x4000,Len:0x4000)
2330 571.236216 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2331 571.244599 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2332 571.244607 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=780 Ack=610680 Win=2003 Len=0
2333 571.412875 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [ACK] Seq=610680 Ack=780 Win=42133 Len=0
2334 571.716676 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2335 571.723311 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2336 571.723325 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [ACK] Seq=780 Ack=613600 Win=2003 Len=0
2337 571.732198 4.8.16.32 12.13.14.15 TCP [TCP segment of a reassembled PDU]
2338 571.732205 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=615060 Len=0
2339 571.732299 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=627563 Len=0
2340 571.733929 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [RST, ACK] Seq=780 Ack=615060 Win=2003 Len=0
2341 571.737874 4.8.16.32 12.13.14.15 TCP [TCP Retransmission] [TCP segment of a reassembled PDU]
2342 571.737883 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [RST] Seq=780 Len=0
2343 571.737887 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=616520 Len=0
2344 571.738004 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=629023 Len=0
2345 571.745046 4.8.16.32 12.13.14.15 TCP [TCP Retransmission] [TCP segment of a reassembled PDU]
2346 571.745052 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [RST] Seq=780 Len=0
2347 571.745056 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=617980 Len=0
2348 571.745178 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=630483 Len=0
2349 571.752669 4.8.16.32 12.13.14.15 TCP [TCP Retransmission] [TCP segment of a reassembled PDU]
2350 571.752674 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [RST] Seq=780 Len=0
2351 571.752782 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=619440 Len=0
2352 571.753549 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=631943 Len=0
2353 571.759783 4.8.16.32 12.13.14.15 TCP [TCP Retransmission] [TCP segment of a reassembled PDU]
2354 571.759792 12.13.14.15 4.8.16.32 TCP 40738 > 10941 [RST] Seq=780 Len=0
2355 571.759797 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=620900 Len=0
2356 571.759818 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=633403 Len=0
2357 571.760290 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=613600 Len=0
2358 571.760438 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=626103 Len=0
2359 571.775184 4.8.16.32 12.13.14.15 TCP 10941 > 40738 [RST] Seq=613600 Len=0
2446 840.737630 12.13.14.15 4.8.16.32 TCP 39928 > 10941 [SYN] Seq=0 Len=0 MSS=1460 TSV=8125390 TSER=0 WS=5
2447 840.783645 4.8.16.32 12.13.14.15 TCP 10941 > 39928 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 WS=2
2448 840.783683 12.13.14.15 4.8.16.32 TCP 39928 > 10941 [ACK] Seq=1 Ack=1 Win=5856 Len=0
2449 840.791619 12.13.14.15 4.8.16.32 TCP 39928 > 10941 [PSH, ACK] Seq=1 Ack=1 Win=5856 Len=287
2450 840.849486 4.8.16.32 12.13.14.15 TCP 10941 > 39928 [PSH, ACK] Seq=1 Ack=288 Win=169072 Len=216
2451 840.849543 12.13.14.15 4.8.16.32 TCP 39928 > 10941 [ACK] Seq=288 Ack=217 Win=6912 Len=0
File: "LOCAL.pcap" 4326 KB 00:29:01
P: 10687 D: 4312 M: 0



Before the RST packets begin (packet no. 2338 in the local capture and packet no. 2549 in the remote capture [9]), the packets transmitted and received correspond directly to one another (although they appear in rather different orders). [10] Once the RST packets begin, a large number of packets are received at each end that do not correspond to packets transmitted at the other end. We can verify this by looking at the detailed contents of these packets (for example, their sequence numbers, which are displayed by Wireshark as Seq=nnnnn), although simply counting them tells the tale in this case. The local machine, with IP address 12.13.14.15, reported transmitting a total of five RST packets to the remote machine at 4.8.16.32 (packet nos. 2340, 2342, 2346, 2350, and 2354), while it reported receiving 13 such packets from 4.8.16.32 (packet nos. 2338, 2339, 2343, 2344, 2347, 2348, 2351, 2352, 2355, 2356, 2357, 2358, and 2359). The remote machine, with IP address 4.8.16.32, reported transmitting only a single RST packet (packet no. 2560) while it believes it received 17 RST packets from 12.13.14.15 (2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2561, 2562, 2563, 2564, 2565, 2566, 2567). Evidently, many of the RST packets received by each machine did not actually originate from the other. [11] The result of these RST packets in this case was that the BitTorrent session stopped and did not resume for around four and a half minutes. (A new TCP session is established with the characteristic SYN and SYN/ACK packets at local

capture packet nos. 2446 and 2447, which correspond to the remote capture's packet nos. 2578 and 2579.)

Wireshark provides the ability to view not only a packet summary list like the lists displayed above but also the sequence of bytes that comprise each individual packet, as well as a dissector view which shows what the packet's contents actually mean from the point of view of various Internet protocol layers. A careful examination of this evidence could involve a byte-for-byte, packet-for-packet comparison to determine exactly which packets were spoofed. If one is performing such a comparison, it is important to appreciate that Wireshark captures more than just the IP packets that get transmitted over the Internet; each IP packet is typically captured wrapped inside a link-layer header (often referred to as an Ethernet frame header or Ethernet packet header on an Ethernet-style network, including a wired Ethernet or wifi network). Link-layer headers are used to communicate between computers on the same local Ethernet network and are discarded and regenerated whenever a packet is forwarded through any router. Thus, there is no reason to expect link-layer headers to correspond between two packet traces unless those packet traces were captured on the same physical local-area network. When manually performing a detailed packet comparison, packets should be considered "the same" if the Internet Protocol headers and payload match up; the link-layer headers can simply be ignored because they were never forwarded over the Internet.

Even some discrepancies between IP headers are to be expected; for example, the Time-To-Live (TTL) field is supposed to be decremented by each router that forwards a packet, so the TTL value when a packet is received should always be smaller than the original TTL value when it was transmitted. Similarly, the IP checksum field that indicates whether a packet's IP headers were transmitted without error has to be recalculated each time the packet is forwarded, because the TTL value has changed. There are also other circumstances in which an ISP's routers may, consistent with Internet protocol standards, legitimately alter other fields in the IP headers. [12]

On the modern Internet, there are usually several ISPs involved in forwarding packets from their source to their destination. As a technical matter, any of these ISPs has the ability to spoof (or drop) packets. Detecting the presence of spoofed packets, then, does not directly reveal or conclusively establish which ISP was responsible for injecting them. To find out which ISPs were involved in the process of forwarding packets from one computer to another, one can use the traceroute tool or any of its descendants or variants. (On Unix and MacOS,

the command-line version of this tool is generally called traceroute; on Windows, it is known as tracert.) This tool is run with a host name or IP address as its argument and experimentally determines the path probably followed by probe packets, displaying a list of the routers in between the local computer and target computer. In order to establish which ISP is responsible for packet spoofing, one could try experiments with a wide variety of ISPs in order to determine if users of one ISP routinely or disproportionately experience that sort of spoofing; one could also try to enable a virtual private network (VPN) to hide traffic from a local ISP, although the details of this process are beyond the scope of this document.

Using pcap files

As we've described above, packet capture files produced by Wireshark and many other network analyzer programs are normally in the pcap format (also known as libpcap format or tcpdump format). This format is an open standard that is widely understood by network analysis software. If you're certain that a pcap file or set of pcap files you've made does not contain sensitive personal information - recalling our earlier warning that it may, by default, contain a record of all Internet activity your computer performed while the capture was active - you can c onsider sharing pcap files with an ISP as part of a problem report or support request, or publishing them on a blog or web site when documenting a problem you've experienced. These files are concrete, useful evidence that can help technically knowledgeable people diagnose network problems and confirm that a problem such as packet spoofing by an ISP is really occurring.

EFF is developing a tool called pcapdiff to help automate the process of comparing large pcap trace files that were made simultaneously at each end of a connection. This automation makes it easier to find packet spoofing or tampering when one doesn't know what to look for ahead of time and could make the comparison process less tedious (considering that many communications involve thousands of packets or more, and not all tampering will have results as obvious as TCP RST injection). The pcapdiff program is a command-line utility written in Python which requires exactly two pcap packet traces; it automatically compares them to find discrepancies indicating dropped and spoofed packets. You can download pcapdiff from <http://www.eff.org/testyourisp/pcapdiff>. Your computer must already have the Python interpreter installed; if not, you can obtain it from http://www.python.org/.

Currently, pcapdiff also requires the pcapy module from http://oss.coresecurity.com/projects/pcapy.html in order to read pcap files;

binary versions of this module are available for Windows and Linux, but it must be compiled from source code for MacOS. Future versions of pcapdiff may be able to run without pcapy.

pcapdiff is run from the command line; you must specify two pcap trace files and the local IP address of the computer where each was captured. In addition to producing statistics about overall rates of packet dropping and spoofing, pcapdiff will produce a list of the IP identification field values of each such packet. This can help you locate packets of interest in a program like Wireshark much more quickly. For example, if pcapdiff says that a packet with IP identification value 4321 was spoofed in the outbound direction, you can enter the display filter ip.id eq 4321 into Wireshark's display filter field and see only the packet (or packets) with this particular IP identification value. Typically, IP identification values uniquely identify IP packets transmitted by a particular computer, although these values will be repeated at least every 65536 (2^{16}) packets.

Observing a very large number of forged packets may suggest that something is systematically wrong – for example, you may have started or ended the captures at different times (so that one machine has no record of having sent many of the packets that it did, in fact, send), you may have limited the number of bytes captured per packet at one end but not at the other end, you may have some kind of protocol offloading active on one end (so that one computer's operating system is wrong about the contents of the packets that the network card actually sent over the wire), you might still have NAT or a firewall enabled at one end of the connection, or your ISP might be forging or altering packets routinely, perhaps with a technique like transparent HTTP proxying that is invisible to most application software. It is important to think critically about the significance of evidence and whether results are reproducible or attributable to confounding factors. pcapdiff and the techniques described here are meant to help users find anomalous discrepancies between packets sent and packets received, not to definitively assign blame for the source of those anomalies.

Learning more about TCP/IP and interpreting network packet traces

For a deeper understanding of network protocols and packet traces like those described here, you can consult the Internet standards documents that specify the TCP/IP protocol suite. Most of these documents have been published in the RFC document series available at http://www.rfc-editor.org/ and http://www.faqs.org/rfcs/. For example, the Internet Protocol (IP) is described

in RFC 791, the Transmission Control Protocol (TCP) in RFC 793, and the User Datagram Protocol (UDP) in RFC 768.

An excellent guide to the TCP/IP protocol suite in book form is W. Richard Stevens, TCP/IP Illustrated: Volume 1, The Protocols (Reading, MA: Addison-Wesley, 1994), ISBN 0201633469. Stevens uses the tcpdump tool to produce packet traces on 1990s-era Internet-connected networks, and carefully explains the theory and practice of Internet networking with reference to these packet traces. Doing the same thing with Wireshark might be clearer today because Wireshark has a friendlier interface and more extensive protocol dissection than tcpdump, but Stevens's explanations are clear, thorough, and generally valid for the Internet of today.

Acknowledgments

Thanks to Chris Palmer and Karl Fogel for their comments.

[1] See Clayton, Murdoch, and Watson, "Ignoring the Great Firewall of China" (available at http://www.cl.cam.ac.uk/~rnc1/ignoring.pdf).

[2] If your ISP were blocking Google, you might suspect this when you encountered difficulty connecting to Google's services - and you might see apparently anomalous packets in a packet trace from your end. However, only the addition of a recording of the same interaction from Google's end would definitively establish whether the problem lay with Google or with an ISP in between. Otherwise, it is difficult to tell whether the problem is a result of ordinary packet loss, a misbehaving computer at Google's end, or even misbehaving software on your own computer.

We urge readers to interpret the results of their tests cautiously and avoid jumping to conclusions or making spurious accusations. For example, RST packets are a legitimate part of the TCP protocol, and receiving RST packets does not normally mean that they were spoofed by an intermediary. RST packet spoofing can only be proven definitively by making simultaneous measurements at the endpoints of a connection. (Some forms of packet spoofing could produce suggestive evidence at one end because the spoofed packets have anomalous properties that make it very unlikely that they were really transmitted by the other end. But observing these anomalies will probably not be truly conclusive unless evidence is also gathered at the other end.)

It would clearly be useful to have a reliable automated means of testing non-P2P services to detect interference or degradation by ISPs. In some cases this would just be a matter of writing software, while in other cases it would require co-ordination among multiple parties (such as Google in the example just given). For example, it would be helpful to know whether ISPs give users lower-latency connections to some web sites than to others. Because the web site operators' co-operation is required for this test, it is beyond the scope of this article. One test that could be performed readily by two end-users is seeing whether exchanging the same volume of data with different protocols (for example, sending a single 1-megabyte file with HTTP, FTP, BitTorrent, Gnutella, and disguised as a SIP VoIP telephone call data stream) takes appreciably different amounts of time, and whether the round-trip latency for each of these protocols is the same or not. This article does not discuss tools that would help automate such a test. We do discuss our pcapdiff tool below and also mention the University of Washington research project using JavaScript to detect some modification of web page contents by ISPs.

3 One Internet user configured his open wireless gateway to modify images seen by users as they browsed the web, either by mirror-reversing them or blurring them. See http://www.ex-parrot.com/~pete/upside-down-ternet.html. It can be easy to forget that Internet intermediaries are able to make arbitrary changes to their users' view of the Internet.

4 This particular advertising appears to have been added by AnchorFree, one of several firms experimenting with this means of ad placement; see http://anchorfree.com/advertisers-agencies/how-it-works/. See also http://vancouver.cs.washington.edu/ for a research project investigating the prevalence of this phenomenon.

5 If you're running Wireshark on the same machine that's generating or receiving the test traffic, as we recommend here, you don't need to follow the additional directions at http://www.wireshark.org/faq.html#promiscsniff because you won't need to capture traffic in promiscuous mode. Running Wireshark in promiscuous mode on a different machine on the same local area network segment (note: not on an Ethernet switch) could, however, help mitigate problems with excessive CPU load, with the unavailability of Wireshark or a suitable packet capture driver for a particular operating system or device, with TCP or UDP checksum offloading or large segment offloading (described below), or when logging into a remote server by means such as SSH in order to run tests on that server's communications with your client machine. In this scenario, the machine capturing packets is not the same machine that generates them; however, the resulting packet trace can generally be used in the same way as a packet trace that was captured directly on the machine generating the packets, as long as the LAN to which the machines are connected broadcasts all packets to the packet-capturing machine. There are also techniques for sniffing traffic on some non-broadcast switched LANs, which are beyond the scope of this document.

Running Wireshark on MacOS X requires X11; there are other pcap-compatible native packet sniffers for MacOS X.

6 NAT devices and firewalls create uncertainty because they routinely rewrite, drop, or block traffic; if they are not disabled, it will be difficult to prove that communications that blocked or altered packets were blocked or altered by an ISP rather than by a firewall device. NATs and proxies also prevent direct packet-by-packet comparison of packet traces because the end points do not have a consistent view of the source and destination addresses in use, and there may not even be a one-to-one relationship between packets entering and exiting a NAT or proxy. Evidence of third-party packet tampering gathered in the presence of any of these devices is better than no evidence, but must be interpreted with extreme caution.

7 Note that if your IP address appears to be in a private address range defined by RFC 1918 (10.0.0.0 - 10.255.255.255, 172.16.0.0 - 172.31.255.255, or 192.168.0.0 - 192.168.255.255), you have not properly followed the instructions to disable any firewalls and NAT devices (or your ISP is forcing everyone to use its own ISP-operated NAT service).

8 Windows users may also be able to use the NPF driver to perform a packet

capture as an unprivileged user.

9 In this context, "packet number" refers to the ordinal number of the packet within a particular capture file, which is displayed by Wireshark in the left-hand column; the first packet sent or received is number 1, the next number 2, and so on. There are also other forms of serial numbering contained within the packets themselves, such as the IP identification field and TCP sequence number. pcapdiff uses the IP identification field (which Wireshark refers to as ip.id for display filter purposes) to refer to packets, but this paragraph is merely discussing Wireshark packet ordinal numbers. "Packet no. 2340" in this sense need not have IP identification value 2340.

10 This reordering is not necessarily only due to ISP packet reordering, but also to the existence of network latency; if both computers transmit a packet at noon over a network with a 1-second latency, each computer will receive the other computer's transmissions at 12:00:01 and consider its own transmission to have taken place "first".

11 To compound the suspiciousness of this situation, each machine believed that it was not the first to send a RST packet in this transaction - the local and remote machines each believed that the other party had initiated the process of disconnecting the communication. It can be legitimate in the TCP protocol to send a RST in response to a RST, and this trace suggests that each machine believed that that is what it was doing - disconnecting only after the other end had already disconnected.

12 We might compare this, albeit imprecisely, to the behavior of the post office in delivering a letter: the post office may apply a postmark or print a bar code on the outside of the envelope, stamp delivery-related notations on it, or even correct an erroneous postal code. However, the post office is not supposed to alter the contents of letters.

Downloads

packet_injection.pdf