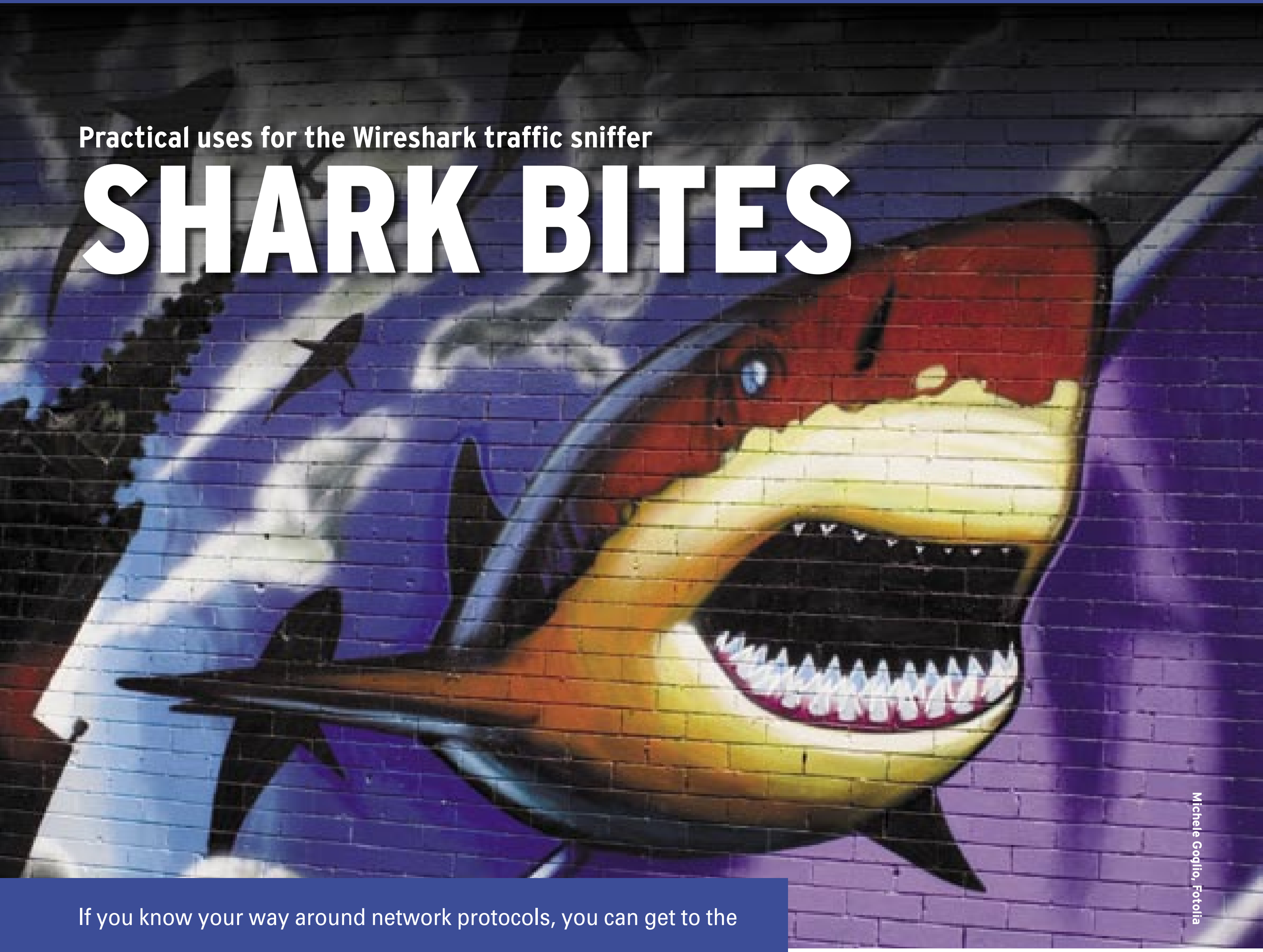


Practical uses for the Wireshark traffic sniffer

SHARK BITES

If you know your way around network protocols, you can get to the source of a problem quickly with Wireshark. **BY ARMIJN HEMEL**

A network sniffer is an indispensable tool for the troubleshooting sys admin. Linux users used to watch their networks with the popular open source sniffer known as Ethereal. Even Hollywood recognized the importance of Ethereal by featuring it in the movie *Firewall*. However, you might have noticed that no one has been talking about Ethereal recently.

Not too long ago (May 2006) the original author of Ethereal went to work for another company. All the trademarks for the Ethereal program stayed with his former employer; however, Ethereal is no longer actively maintained. A new sniffer called Wireshark [1] is a fork of Ethereal that is maintained by the original Ethereal developer.

Wireshark (Figure 1) is a GPL application that is available for all major Unix (-like) operating systems as well as Microsoft Windows. By default, it uses a graphical user interface, but you will also find a text version called tethereal, a name left over from Ethereal days.

The art of network sniffing requires a thorough knowledge of protocols and an understanding of how specific protocols are used by specific applications. Although I don't have room in this article for an expansive discussion of network theory, I can recount some stories from my personal experience of solving practical network problems with Wireshark.

What It Is

Wireshark works by catching all network traffic on one or more network interfaces. When you sniff the interface or interfaces), you must first set the interface into so-called *promiscuous mode*.

In promiscuous mode, the interface will accept every packet that arrives, even if it is not intended for that interface. (Normally, the kernel driver for the network card will silently drop packages that are not intended for the machine.)

The traffic on a busy network includes huge quantities of packets for dozens of different protocols. One of the most powerful features of Wireshark is that you can create filters that limit the number of visible packets so you are not buried in noise. In the Wireshark GUI, you can construct filtering expressions in a special dialog box (Figure 2) and even combine various filters to create more powerful expressions.

An especially useful Wireshark feature is the ability to track complete TCP streams with the *Follow TCP stream* option (*Analyze* | *Follow TCP stream*). All

Tip

You can sort the various columns in the Wireshark user interface simply by clicking on them. This feature makes it easy to organize capture data by protocol or host.

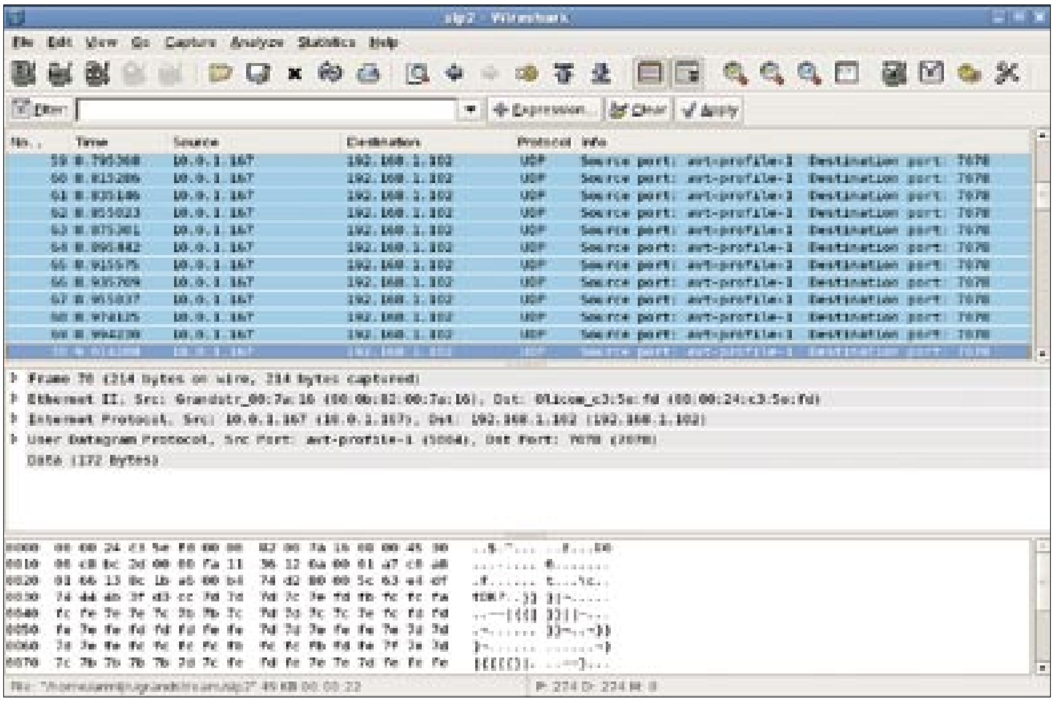


Figure 1: Wireshark monitors and filters network traffic.

packets that are part of a session (from the first *SYN* packet to the last *FIN-ACK*) are displayed. The stream-tracing feature lets you follow complete sessions, such as MSN Messenger conversations or web surfing sessions. Of course, this feature does not work for UDP because UDP is a connectionless protocol.

Getting Started

Many distributions install Wireshark by default or have precompiled packages readily available. Just look around your Linux version for a copy of Wireshark, and if you don't find it, check your Linux vendor. Wireshark is also available for download from the project website [1].

Compiling Wireshark is not difficult. The source distribution uses GNU Autotools to generate configure scripts and makefiles, so to set up the source code version of Wireshark, you just need to run *configure* with the right options, then type *make* and *make install*.

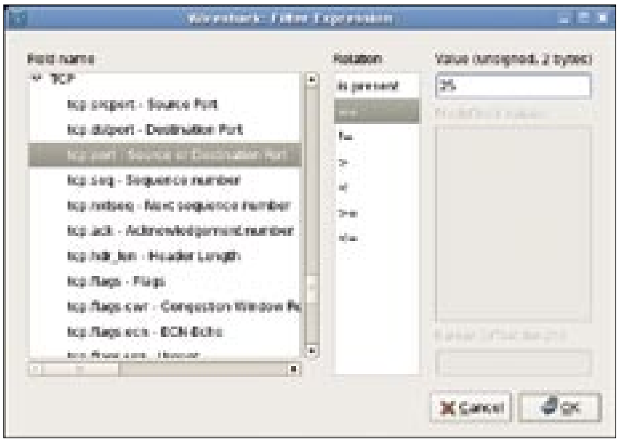

Wireshark: Filter Expression
Field name
TCP
tcp.srcport - Source Port
tcp.dstport - Destination Port
tcp.port - Source or Destination Port
tcp.seq - Sequence number
tcp.nxtseq - Next sequence number
tcp.ack - Acknowledgement number
tcp.hdr_len - Header Length
tcp.flags - Flags
tcp.flags.cwr - Congestion Window Re
tcp.flags.ecn - ECN-Echo
Relation
is present
Value (unsigned, 2 bytes)
25
Cancel
OK

Figure 2: Constructing an expression in Wireshark.

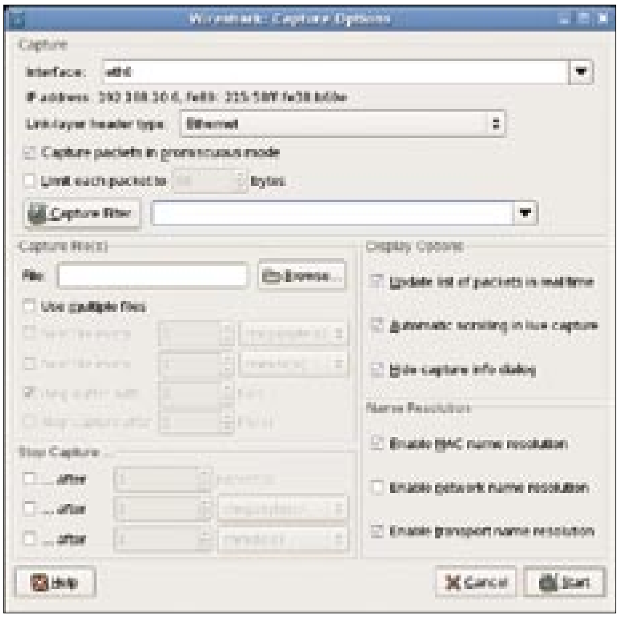


Figure 3: Use the Capture Options dialog box to configure a capture interface.

If you plan to use Wireshark to sniff network traffic, you'll need to put the network interface into promiscuous mode, which requires root privileges. If you are just using Wireshark to analyze network traffic data from a dump file, root status is not required.

Before you can sniff network traffic, you need to configure Wireshark. Choose the *Capture* menu and select *Options* to set configuration options, such as the network interface you want to use for the capture (Figure 3). Another option is live scrolling of captured packets. By default, Wireshark only shows all packets after capturing has been stopped. When debugging an application, I often prefer the live scrolling option, which lets me view the packet flow and stop the capture once the packets that interest me have been captured.

Wireshark can export and import in various formats. One important format option is the *libpcap* format. The *libpcap* library is used by various tools, including Wireshark and tcpdump. For both tools, *libpcap* is the native format, and network dumps from either one of these programs can be read by the other. You can also export to formats such as CSV, plain text, and PostScript.

Sniff Histories

A tool like Wireshark isn't just intended for large enterprise configurations. Even admins of small-scale operations can use Wireshark for practical problems that occur every day on computer networks.

The following scenarios should give you a better taste for the tricks you can play with Wireshark.

Scenario 1: SSL Sites

I once faced a situation in which some websites suddenly did not work while other sites continued to work just fine. The sites that didn't work were located in different domains, on different servers, and at different ISPs. I started up Wireshark to investigate.

A closer inspection revealed that the websites I couldn't reach all used SSL encryption (HTTPS). Wireshark helped me determine that the initial packets to set up the connections were sent and acknowledged, but subsequent packets went unanswered.

The problem turned out to be a misconfigured router at the ISP. Setting back the maximum transmission unit (MTU) value (the maximum size of a packet that can be sent on an interface) from 1500 bytes (the default) to a value below 1492 bytes fixed the problem.

When the IEEE 802.3 standard was developed, the existing Ethernet protocol was taken as a basis, but some of the features were adapted slightly. One of the parameters that changed was the MTU size. In 10Mbps Ethernet (and faster), the MTU size is 1500 bytes, but in IEEE 802.3, the size is 1492 bytes. When a router is configured to use 802.3 and packets arrive from a network that is configured to use Ethernet, the packets are too big. The IP protocol then resorts to fragmenting: packets are split into smaller packets, which are then sent individually. On the receiving end, they are combined into the original IP packet.

However, HTTPS packets use the *don't fragment* IP option, so they are too big; no solution is possible, so the packets are discarded. The end result is that the website appears broken. Wireshark helped me discover this problem by showing that the HTTPS sessions were lost after the initial request.

Scenario 2: VoIP

Internet telephony services were behaving strangely on my network. My configuration consisted of the Linphone VoIP softphone application connecting to a

Wireshark and Security

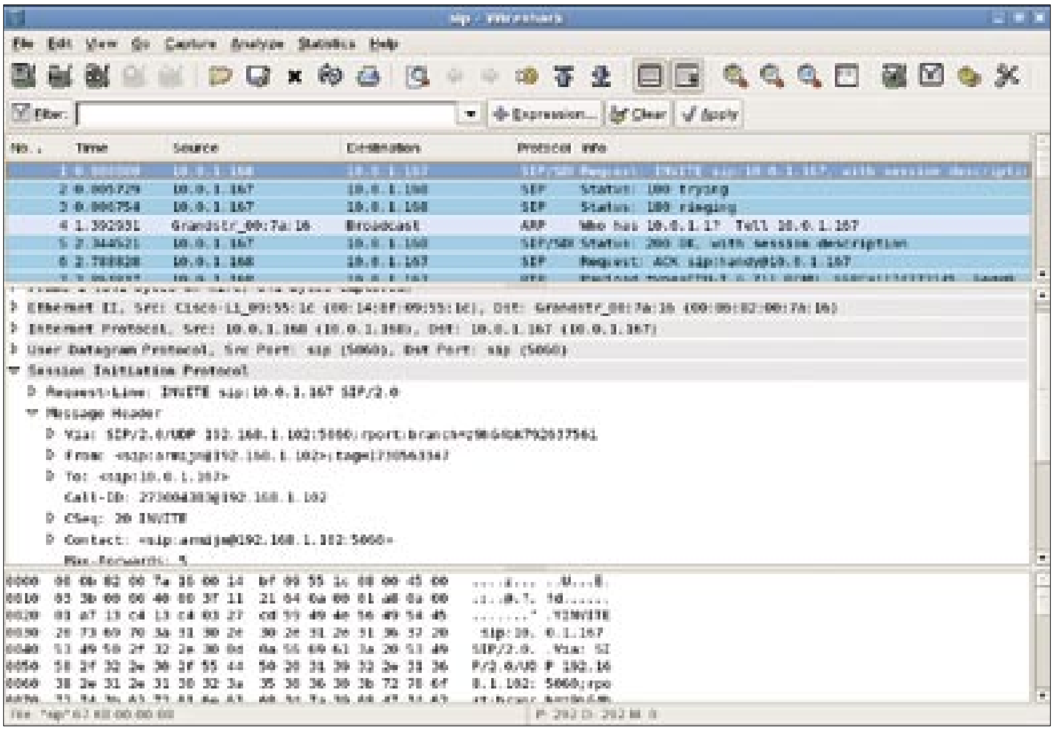
For a tool that is often used for security purposes, Wireshark seems to have a lot of security-related bugs. The security issues have a lot to do with the way Wireshark is designed. For analyzing network packets, special plugins called dissectors parse the network packets. By far, most of the vulnerabilities in Wireshark are found in these dissector plugins. To capture network packets, Wireshark often has to run with root privileges. Analyzers are fed the data that Wireshark gets from the network interface. This data is not guaranteed to be well formed. If the code of the analyzer that parses the data contains an error, an attacker could send specially crafted packets that crash the application or execute code with root privileges.

An excellent and thorough description of this situation is found at the LWN.net website [2].

A good practice is to use command line tools, like tcpdump, to capture traffic and then use the GUI tools later as a non-root user to analyze the data.

Grandstream HT-286 analog telephone adapter. Between Linphone and the telephone adapter was a Linksys WRT54G router performing network address translation (NAT). Linphone was on the private network behind the NAT device (with the IP address 192.168.1.102), and the telephone adapter resided on the WAN side (with the address 10.0.1.167).

Wireshark quickly confirmed my suspicion that the problem was related to network address translation. The Grandstream HT-286 was not smart enough to see that the connection was coming from behind NAT. Audio from Linphone was received correctly by the HT-286, but the HT-286 wanted to send its audio back to the IP address of the the Linphone PC, which it can't find (Figure 4).



```
No. .  Time        Source              Destination    Protocol Info
     1 0.000000    10.0.1.168          10.0.1.167     SIP/SDP  Request: INVITE sip:10.0.1.167, with session descripti
     2 0.005729    10.0.1.167          10.0.1.168     SIP      Status: 100 trying
     3 0.005754    10.0.1.167          10.0.1.168     SIP      Status: 180 ringing
     4 1.392931    Grandstr_00:7a:16   Broadcast      ARP      Who has 10.0.1.1? Tell 10.0.1.167
     5 2.344521    10.0.1.167          10.0.1.168     SIP/SDP  Status: 200 OK, with session description
     6 2.788828    10.0.1.168          10.0.1.167     SIP      Request: ACK sip:sandy@10.0.1.167

▷ Ethernet II, Src: Cisco-Li_09:55:1c (00:14:bf:09:55:1c), Dst: Grandstr_00:7a:16 (00:0b:82:00:7a:16)
▷ Internet Protocol, Src: 10.0.1.168 (10.0.1.168), Dst: 10.0.1.167 (10.0.1.167)
▷ User Datagram Protocol, Src Port: sip (5060), Dst Port: sip (5060)
▽ Session Initiation Protocol
  ▷ Request-Line: INVITE sip:10.0.1.167 SIP/2.0
  ▽ Message Header
    ▷ Via: SIP/2.0/UDP 192.168.1.102:5060;rport;branch=z9hG4bK792637561
    ▷ From: <sip:armijo@192.168.1.102>;tag=173056334?
    ▷ To: <sip:10.0.1.167>
      Call-ID: 273004383@192.168.1.102
    ▷ CSeq: 20 INVITE
    ▷ Contact: <sip:armijo@192.168.1.102:5060>
      Max-Forwards: 5
```

Figure 4: NAT causes problems for VoIP protocols.

To get SIP (session initiation protocol) working correctly is difficult if one of the calling parties is behind NAT because SIP encodes the IP address of the sender in the UDP payload. The payload is not rewritten by normal NAT applications because it is application specific and therefore goes against the "layered" principles of TCP/IP. To get this data through NAT correctly, you need NATing software that can handle SIP correctly, like a proxy, or you need to resort to other tricks. If you don't get it configured correctly, traffic will only flow from the device behind NAT to the device that is not behind NAT. The device that is not behind NAT will try to send packets back, but it won't find the right machine, and the packets will simply get lost.

Wireshark helped me uncover this problem quickly. Other telephony devices might not have this particular problem, but they could have other NAT-related issues that you can troubleshoot easily with Wireshark.

Scenario 3: Misconfigured Routers

At the university lab where I was employed as the student admin, we experienced a huge performance loss (around 95%) for NFS connections between the lab and the server room. With Wireshark, we quickly determined that NFS packets were present, but clearly some packets were getting lost. This problem only happened with NFS, not when surfing the web.

We were able to narrow the problem to something between the lab and the NFS server. It turned out that a router had been reconfigured by the university's central IT department and was talking in half-duplex mode to the switches instead of full-duplex mode.

The hardest part of fixing this problem was convincing the central IT department to take the Wireshark logs seriously. Once we showed them the evidence, they quickly fixed the misconfigured router.

Conclusion

I have used Wireshark to solve many networking problems with relatively little effort.

The Wireshark traffic sniffer is also a great tool for learning how various network protocols behave. ■

Tips for Using Wireshark

If you keep the following tips in mind, you'll have an easier time with Wireshark.

- When you want to sniff traffic meant for other machines, make sure you have a network with a hub and not a switch. A switch knows which packet to send to which port, whereas a hub simply sends packets over all interfaces (except for the incoming interface) and lets the hosts filter the packets. Traffic that is intended for other computers will not reach you if you use a switch.
- When you sniff on a server that is running headless, use tethereal or tcpdump to sniff packets. When you use tcpdump, configure it to capture the whole packet, because by default, tcpdump only captures the first so many bytes of each packet.
- Network sniffing is a very powerful technique, but when you are sniffing traffic to other machines or tracking communication of other people, you are more or less eavesdropping, which might be illegal. A good system administrator knows where to draw the line between debugging and violating a user's privacy.

INFO

[1] Wireshark: *http://www.wireshark.org/*

[2] Ethereal and security: *http://lwn.net/Articles/175527/*