Sniffing (network wiretap, sniffer) FAQ

This document answers questions about eavesdropping on computer networks (a.k.a. "sniffing").

0. Information about this FAQ

Version 0.3.3, September 14, 2000

Official source of this document:
http://www.robertgraham.com/pubs/sniffing-faq.html (HTML)

0.7 Thanks to

Thanks to the following people for helpful info and comments (note: to avoid automated spam address collection systems, I've munged their e-mail addresses in an obvious way).

Trevor Schroeder from http://www.zweknu.org
Lachlan M. D. Cranswick <l.cranswick at dl dot ac dot uk>

0.9 Who is Robert Graham?

Among other things, between 1994-1998 I worked at Network General Corporation on the Sniffer(r) Network Analyzer. I either wrote/rewrote/ported over 300 protocol decodes for the Sniffer. Now I'm working on an intrusion detection system that similarly does protocol analysis. Also, I helped develop the "Certified Network Expert" exam, which was put together by a consortium of protocol analyzer/network analyzer vendors. In the early 1990s, I help develope the RMON standard(s) and the first RMON systems.

1. The basics

1.1 What is a "packet sniffer"?

A **packet sniffer** is a wire-tap devices that plugs into computer networks and eavesdrops on the network traffic. Like a telephone wiretap allows the FBI to listen in on other people's conversations, a "sniffing" program lets someone listen in on computer conversations.

However, computer conversations consist of apparently random binary data. Therefore, network wiretap programs also come with a feature known as "protocol analysis", which allow them to "decode" the computer traffic and make sense of it.

Sniffing also has one advantage over telephone wiretaps: many networks use "shared media". This means that you don't need to break into a wiring closet to install your wiretap, you can do it from almost any network connection to eavesdrop on your neighbors. This is called a "promiscuous mode" sniffer. However, this "shared" technology is moving quickly toward "switched" technology where this will no longer be possible, which means you will have to actually tap into the wire.

1.1.1 Is "packet sniffer" trademarked?

The word "sniffer" is a registered trademark by Network Associates referring to the "Sniffer(r) Network Analyzer". However, the term "snif" is used in many other products (some of which are listed in this document) and the term "sniffer" is more popular in everyday usage than alternatives like "protocol analyzer" or "network analyzer" (as far as my search on AltaVista reveals). I'm not sure what this means in trademark law, where brandnames like "aspirin", "escalator", and "cellophane" lose their distinctiveness over time.

1.2 What is it used for?

Sniffing programs have been around for a long time in two forms. Commercial packet sniffers are

used to help maintain networks. Underground packet sniffers are used to break into computers.

Typical uses of such wiretap programs include:

- Automatic sifting of clear-text passwords and usernames from the network. Used hackers/crackers in order to break into systems.
- Conversion of data to human readable format so that people can read the traffic
- Fault analysis to discover problems in the network, such as why computer A can't talk to computer B
- Performance analysis to discover network bottlenecks
- Network intrusion detection in order to discover hackers/crackers (see http://www.robertgraham.com/pubs/network-intrusion-detection.html
- Network traffic logging, to create logs that hackers can't break into and erase.

1.3 Is there a single point on the Internet I can plug into in order to see all the traffic?

No. The connectivity of the Internet looks much like a fisherman's net. Traffic flows through a mesh, and no single point will see it all. The Internet was built to withstand a nuclear attack -- and to survive any "single point of failure". This likewise prevents any single point of sniffing.

Think of it this: you have two machines in your own office talking to each other, and both are on the Internet. They take a direct route of communication, and the traffic never goes across the outside public portion of the Internet. Any communication anywhere in the net follows a similar "least-cost-path" principle.

1.4 How does sniffing/wiretap work?

1.4.1 How does it eavesdrop on network traffic?

Ethernet was built around a "shared" principle: all machines on a local network share the same wire.

This implies that all machines are able to "see" all the traffic on the same wire.

Thus, Ethernet hardware is built with a "filter" that ignores all traffic that doesn't belong to it. It does this by ignoring all frames whose MAC address doesn't match.

A wiretap program turns off this filter, puttin the Ethernet hardware into "promiscuous mode". Thus, Mark can see all the traffic between Alice and Bob, as long as they are on the same Ethernet wire.

1.4.2 What are the components of a packet sniffer?

The hardware
Most products work from standard network adapters, though some require special hardware. If you use special hardware, you can analyze hardware faults like CRC errors, voltage problems, cable programs, "dribbles", "jitter", negotiation errors, and so forth.

Capture driver
This is the most important part. It captures the network traffic from the wire, filters it for the particular traffic you want, then stores the data in a buffer.

Buffer
Once they frames are captured from the network, they are stored in a buffer. There are a couple captures modes: capture until the buffer fills up, or use the buffer as a "round robin" where the newest data replaces the oldest data. Some products (like the BlackICE Sentry IDS from Network ICE can maintain a full round-robin capture buffer on disk at full 100-mbps speeds. This allows have hundreds of gigabytes of buffer rather than the meager 1-gigabyte you're likely to have in a memory-based buffer.

Real-time analysis
Pioneered by the Network General Sniffer, this feature does some minor bit of analysis of the frames as they come off the wire. This is able to find network performance issues and faults while capturing. Many vendors have started to add minimal capabilities along this line to their products. Network intrusion detection systems do this, but they sift the traffic

for signs of hacker activity rather than fault/performance issues.

Decode

As discussed in section 5, this displays the contents of network traffic with descriptive text so that an analysist can figure out what is going on.

Packet editing/transmission

Some products contain features that allow you to edit your own network packets and transmit them onto the network.

1.5 What is an Ethernet MAC address?

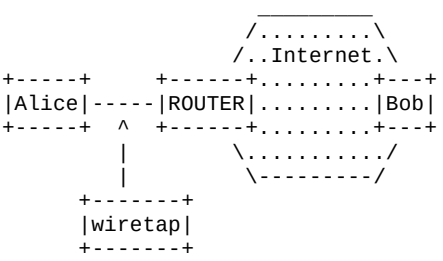
Since many machines may share a single Ethernet wire, each must have an individual identifier. This doesn't happen with dial-up modems, because it is assumed that any data you send to the modem is destinated for the other side of the phone line. But when you send data out onto an Ethernet wire, you have to be clear which machine you intend to send the data to. Sure, in many cases today there are only two machines talking to each other, but you have to remember that Ethernet was designed for thousands of machines to share the same wire.

This is accomplished by putting a unique 12-digit hex number in every piece of Ethernet hardware. Section 1.5.4 explains how to discover the Ethernet MAC address of your own machine.

To really understand why this is so important, you might want to review the information in section 5.4 below. Ethernet was designed to carry other traffic than just TCP/IP, and TCP/IP was designed to run over other wires (such as dial-up lines, which use no Ethernet). For example, many home users install "NetBEUI" for File and Print Sharing because it is unrelated to TCP/IP, and therefore hackers from across the Internet can't get at their hard-drives.

Raw transmission and reception on Ethernet is governed by the Ethernet equipment. You just can't send data raw over the wire, you must first do something to it that Ethernet understands. In much the same way, you can't stick a letter in a mailbox, you must first wrap it in an envelope with an address and stamp.

Following a is a brief explanation how this works:



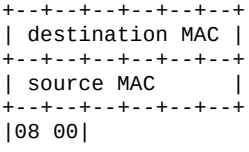
Alice has IP address: 10.0.0.23
Bob has IP address: 192.168.100.54

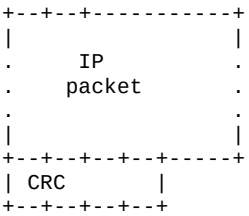
In order to talk to Bob, Alice needs to create an IP packet of the form 10.0.0.23-->192.168.100.54

As the packet traverses the Internet, it will be passed from router-to-router. Therefore, Alice must first hand off the packet to the first router. Each router along the way will examine the destination IP address (192.168.100.54) and decide the correct path it should take.

In the able diagram, we draw the Internet as a "cloud". All Alice knows about is the local connection to the first router, and Bob's eventual IP address. Alice knows nothing about the structure of the Internet and the route that packet will take.

Alice must talk to the router in order to send the packet. She uses the Ethernet to do so. An Ethernet frame looks like the following:





What this means is that the TCP/IP stack in Alice's machine might create a packet that is 100 bytes long (let's say 20 bytes for the IP info, 20 bytes for the TCP info, and 60 bytes of data). The TCP/IP stack then sends it to the Ethernet module, which puts 14 bytes on the front for the destination MAC address, source MAC address, and the ethertype 0x0800 to indicate that the other end's TCP/IP stack should process the frame. It also attaches 4-bytes on the end with a checksum/CRC (a validator to see if the frame gets corrupted as it goes across the wire).

The adapter then sends the bits out onto the wire.

All hardware adapters on the wire see the frame, including the ROUTER's adapter, the packet sniffer, and any other machines. Proper adapters, however, have a hardware chip that compares the frame's "destination MAC" with its own MAC address. If they don't match, then it discards the frame. This is done at the hardware level, so the machine the adapter is attached to is completely unaware of this process.

When the ROUTER ethernet adapeter sees this frame, it reads it off the wire and removes the leading 14-bytes and the trailing 4-bytes. It looks at the 0x0800 ethertype and decides to send it to the TCP/IP stack for processing (which will presumably forward it to the next router in the chain toward the destination).

In the above scenario, only the ROUTER machine is supposed to see the Ethernet frame, and all other machines are supposed to ignore it. The wiretap, however, breaks the rules and copies the frame off the network, too.

See Charles Spurgeon's Ethernet website at: http://wwwhost.ots.utexas.edu/ethernet/ethernet-home.html

1.5.2 What does "MAC" stand for?

MAC stands for *Media Access Control*.

The logic behind this is that the Ethernet has multiple sublayers, PHY (physical), MAC, LLC (logical). The Ethernet address is considered part of the MAC sublayer. The physical layer is responsible for the wire, the MAC is responsible for formatting the data that goes on the wire, and the logical is responisble for things such as retransmitting the data on the wire.

1.5.3 What is the format of the MAC address?

The Ethernet MAC address is a 48 bit number. This number is broken down into two halves, the first 24-bits identify the vendor of the Ethernet board, the second 24-bits is a serial number assigned by the vendor. This guarantees that no two Ethernet cards have the same MAC address (unless the vendor fouls up). Duplicate address would cause problems, so uniquess is very important. This 24-bit number is called the OUI ("Organizationally Unique Identifier").

However, the OUI is really only 22-bits long, two of the bits in that field are used for other purposes. One bit indicates if the address is a "broadcast/multicast" address, the other bit indicates if the adapter has been reassigned a "locally administered address" (where a network administrator reassigns the MAC address to fit some local policy).

For example, you will commonly see the MAC address `03 00 00 00 00 01` on the wire. The first byte contains the binary representation of `00000011` where both these special bits are set (and the rest are zero). This is a special multicast packet that is sent to all machines that run the "NetBEUI" protocol (which is commonly installed on Windows machines to share files locally without using TCP/IP as the transport).

The IEEE maintains the list of vendor/OUI codes at http://standards.ieee.org/regauth/oui/.

1.5.4 What is my own Ethernet address?

Win9x
Run the program "winipcfg.exe". It will tell you.

WinNT
Run the program "ipconfig /all" from the command-line. It will show the MAC address for your adapters. Sample results are:

```
Windows NT IP Configuration

        Host Name . . . . . . . . . . : sample.robertgraham.com
        DNS Servers . . . . . . . . . : 192.0.2.254
        Node Type . . . . . . . . . . : Hybrid
        NetBIOS Scope ID. . . . . . . :
        IP Routing Enabled. . . . . . : No
        WINS Proxy Enabled. . . . . . : No
        NetBIOS Resolution Uses DNS : No

Ethernet adapter SC12001:

        Description . . . . . . . . . : DEC DC21140 PCI Fast Ethernet Adapter
        Physical Address. . . . . . . : 00-40-05-A5-4F-9D
        DHCP Enabled. . . . . . . . . : No
        IP Address. . . . . . . . . . : 192.0.2.160
        Subnet Mask . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . : 192.0.2.1
        Primary WINS Server . . . . . : 192.0.2.253
```

Linux
Run the program "ifconfig". Sample results are:

```
eth0      Link encap:Ethernet  HWaddr 08:00:17:0A:36:3E
          inet addr:192.0.2.161  Bcast:192.0.2.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1137249 errors:0 dropped:0 overruns:0
          TX packets:994976 errors:0 dropped:0 overruns:0
          Interrupt:5 Base address:0x300
```

Solaris
Use the "arp" or "netstat -p" command, it will often list the local interface among the ARP entries.

1.5.5 What are the Ethernet addresses of machines I'm talking to?

For WinNT and UNIX, use the command "arp -a".

1.5.6 Can I change my MAC address?

Yes. There are several places where this might be important.

First, you can spoof your address. Remember that the MAC address is just part of the frame data. Therefore, when you send an Ethernet frame on the wire, you can overwrite your own contents in those frames. Of course, you have to be running a program that does this for some reason.

Second, most adapters allow you to reconfigure the runtime MAC address. For example, some cards allow you to reconfigure the address within the Windows control panel.

Third, you can re-burn (i.e. reprogram the EEPROM) the address in the card. You need a program/hardware that knows the specifics of the chipset being used by the card. This changes the card forever to have the new address.

1.6 Can I sniff a connection between two people without having access to their wire?

In other words, you are asking about this scenario:

- Alice and Bob are in New York and Texas and are talking.
- You are located in California, nowhere near them.
- You want to eavesdrop on their communication.

The answer is of course "no", it isn't even vaguely possible. You have to have access to the wire that the communication is going across in order to eavesdrop. Same as with telephones, same as everywhere.

Remote access to the wire

However, if you are a really, really good cracker/hacker, there are ways of getting access to those lines. Typical examples are:

- Break into Alice or Bob's computer and install sniffing software that you remotely control.
- Break into the intervening ISPs, and install sniffing software.
- Find a box at the ISPs that supports sniffing, like an RMON probe or DSS (Distributed Sniffer System).
- Bribe somebody at one of those ISPs; break into the physical plant and install a packet sniffer, etc.
- ....

Close to the wire

In some situations, like cable-modems, DSL, Ethernet VLANs, etc., you can redirect traffic between two people to go through your own machine. This is because while you are not directly in the path of communication, you can sometimes move that path to flow past your own computer. It's much like the concept that you can divert a stream slightly, though not very far. See the "Redirect section under Cable-Modems.

Rootkits and Remote Admin Trojans

Another possibility is to break into a person's machine and install a sniffing program. (Exactly how to break into someone's machine is, of course, beyond the scope of this document). On UNIX, sniffing programs are part of most "rootkits". On Windows, sniffing is part of some RATs (Remote Admin Trojans, e.g. BackOrifice).

In theory, these programs can be used to sniff traffic in general, but usually they are configured to simply sniff for passwords and e-mail them back to the hacker.

Section 2.5 contains information on how to detect when this has happened to you, especially how to detect sniffers installed on a UNIX host.

2. How can I defend myself against packet sniffers?

2.1 How can I stop people from sniffing my data?

While you can configure your local network to make sniffing hard, you are pretty much powerless stopping people from out on the Internet from sniffing your traffic. The best defense in this case is to encrypt your data, so that while they can sniff it, they cannot read it. Some techniques are:

SSL

"Secure Sockets Layer", SSL is built into all popular web browsers and web servers. It allows encrypted web surfing, and is almost always used in e-commerce when users enter their credit card information.

This site for Apache SSL describes this: http://www.modssl.org/

PGP and S/MIME

E-mail can be sniffed in many alternative ways. It passes through corporate firewalls, which may monitor the traffic. It often gets logged and saved for extended periods of time. It may get accidentally misdirected, and end up in somebody else's mailbox. The best way to keep such e-mail secret is to encrypt it. The two common ways of doing this are with PGP (Pretty Good Privacy) and S/MIME (Secure MIME). PGP can be purchased as an add-on to many products.

S/MIME is built into e-mail programs by Netscape and Microsoft.

ssh

"Secure Shell", ssh has become the de facto standard for logging into UNIX machines from the Internet. You should immediately replace **telnet** with this service. Numerous other protocols can be tunneled through ssh connections (i.e. file copy). The product was originally developed by a Finish company http://www.ssh.fi/ but many open-source/freeware implementations also exist.

VPNs (Virtual Private Networks)

VPNs provide encrypted traffic across the Internet. However, if a hacker compromises the end-nodes of a VPN connection, they can still sniff the traffic. A typical scenario is an end-user who surfs the Internet normally and gets compromised with a Remote Access Trojan (RAT) that contains a sniffing plug-in. When the user establishes the VPN connection, the sniffing program is able to see not only the encrypted traffic that can be seen on the Internet, but also the unencrypted traffic before it gets sent through the stack to the VPN.

2.2 How can I stop people from sniffing my passwords?

The data-encryption solutions above also provide for secure authentication. There are other solutions that provide for secure authentication as well:

SMB/CIFS

In the Windows/SAMBA environment, make sure that you have the older LanManager authentication turned off. This requires SAMBA v2 or later, WinNT SP3 or later, and so on.

Kerberos v5

Both Windows 2000 and UNIX provide support for Kerberos authentication. This is one of the strongest generic mechanisms available. ftp://aeneas.mit.edu/pub/kerberos/doc/KERBEROS.FAQ

smart cards

There are numerous smart card implementations around providing one-time passwords. These are often used when connecting remotely, either dial-in or VPN across the Internet.

Stanford SRP (Secure Remote Password)

Enhancements to Telnet and FTP for UNIX and Windows. http://srp.stanford.edu/srp/

2.3 How can I configure my local network to make sniffing harder?

Replacing your hub with a switch will provide a simple, yet effective defense against casual sniffing.

While this solution is extremely effective in practice (and should be strongly considered), it shouldn't be relied upon as a complete defense against sniffing. A switch still creates a "broadcast domain", providing an attacker the ability to spoof ARP packets.

The easiest such exploit is the "router redirection". ARP queries contain the correct IP-to-MAC mapping for the sender. In order to reduce ARP traffic, most machines will cache this information that they read from the query broadcasts. Therefore, a malicious attacker can redirect nearby machines to forward traffic through it by sending out regular ARP packets containing the router's IP address mapped to its own MAC address. All the machines on the local wire will believe the hacker is the router, and therefore pass their traffic through him/her.

A similar attack would be to DoS a target victim and force it off the network, then begin using its IP address. If a hacker does this carefully, s/he can inherit connections already established without dropping them. Windows machines are even so polite that when they come onto the network and see someone else using their address, they will kindly shut down their own TCP/IP stacks and allow this

to continue. SMB (the Windows file sharing protocol) is also kind enough to allow predictable identifiers, allowing cr/hackers to predict enough information to keep the connection going.

Most intrusion detection systems and even network management tools like the Expert Sniffer(tm) will detect these shenanigans. For example, putting the BlackICE IDS on all the Windows end-nodes or hooked to a normal port (to receive broadcasts) will alert the security admin that such things are taking place (but, will generate false positives when DHCP reassigns addresses. Sigh.)

Most Ethernet adapters allow the MAC address to be manually configured. Thus a hacker can spoof MAC addresses by reassigning the address on the adapter, or by bypassing the built-in stack and hand-crafting frames. The hacker must maintain a a constant stream of outgoing frames in order to convince the auto-learning switch that they are the legitimate owner of the MAC address.

Many (most??) switches allow MAC addresses to be configured statically in order to prevent this sort of thing. While it may be a difficult management burden to do this for all end-nodes, it may prove useful for the router, restricting the hacker to wiretapping individual end-nodes instead of everyone all at once.

Some switches can be kicked out of "bridging" mode into "repeating" mode where all frames are broadcast on all ports all the time. This is done by overflowing the address tables with lots of false MAC addresses. This can be done with a simple traffic generation phase, or by sending a continual stream of random garbage through the switch.

2.4 Can I buy adapters that do not support sniffing?

No.

The real answer is "yes", there are some older adapters that do not support promiscuos mode. In particular, the original IBM Token Ring adapters (TROPIC chipset) were not able to run in promiscuous mode. There are also a few Ethernet where promiscuous mode is broken, either in the hardware or in the driver. Actually, there are far more adapters who simply lack the functionality in the driver in order to turn on promiscuous mode, meaning all programs that attempt to put them into promiscuous mode will fail even though the hardware supports the mode in theory. If you really must have one, then call technical support for a sniffing product vendor (such as NAI) and ask them which cards they DON'T support. For Windows, you might check with Microsoft support to see which cards do not support NetMon (I remember there are a few, but I can't find the documentation for it).

Note that in the Intel/Microsoft "PC99" guidelines, promiscuous mode is a "required" feature.

If this is a concern, it will be cheaper in the long run simply to upgrade to switching hubs, which basically does the same thing. An Ethernet switch moves the "address match" upstream, so that the switch does it rather than the adapter.

Finally, it should be noted that most new networks *are* switched in some fashion. Even though hackers cannot sniff an entire Ethernet segment, they still install sniffers on machines in order to see the incoming/outgoing traffic. A non-promiscuous adapter won't help defend against this.

2.5 How can I detect a packet sniffer?

In **theory**, it is impossible to detect sniffing programs because they are passive: they only collect packets, they don't transmit anything. However, in **practice** it is **sometimes** possible to detect sniffing programs. It is similar to how in theory it is impossible to detect radio/TV receivers, but European countries do it all the time in order to catch people avoiding the radio/TV tax.

A stand-alone packet sniffer doesn't transmit any packets, but when installed non-standalone on a normal computer, the sniffing program will often generate traffic. For example, it might send out DNS reverse lookups in order to find names associated with IP addresses.

Non-standalone packet sniffers are indeed what you *want* to detect. When crackers/hackers invade machines, they often install sniffing programs. You want to be able to detect this happening.

General Overview of Detection Methods

ping method

Most "packet sniffers" run on normal machines with a normal TCP/IP stack. This means that if you send a request to these machines, they will respond. The trick is to send a request to IP address of the machine, but not to its Ethernet adapter.

To illustrate:

1. The machine suspected of running the packet sniffer has an IP address 10.0.0.1, and an Ethernet address of 00-40-05-A4-79-3**2**.
2. You are on the same Ethernet segment as the suspect (remember, the Ethernet is used only to communicate locally on a segment, not remotely across the Internet).
3. You change the MAC address slightly, such as 00-40-05-A4-79-3**3**.
4. You transmit an "ICMP Echo Request" (ping) with the IP address and this new MAC address.
5. Remember that NOBODY should see this packet, because as the frame goes down the wire, each Ethernet adapter matches the MAC address with their own MAC address. If none matches, then they ignore the frame.
6. If you see the response, then the suspect wasn't running this "MAC address filter" on the card, and is hence sniffing on the wire.

There are ways defending against this. Now that this technique is widely publicized, newer hackers will enabled a virtual MAC address filter in their code. Many machines (notably Windows) have MAC filtering in drivers. (There is a hack for Windows: most drivers just check the first byte, so a MAC address of FF-00-00-00-00-00 looks like FF-FF-FF-FF-FF-FF (the broadcast address which all adapters accept). However, some adapters implement multicast in such as way that this address will match as a multicast, which is any address whose first byte is an odd number. Thus, this can result in false positives).

This technique will usually work on switched/bridged Ethernets. When switches see an unknown MAC address for the first time, they will "flood" the frame to all segments.

ping method, part 2

The ping method can be enhanced in a number of ways:

1. Any protocol that generates a response can be used, such as a TCP connection request or a UDP protocol such as port 7 (echo).
2. Any protocol that might generate an error on the target machine might be used. For example, bad IP header values might be used to generate an ICMP error.
3. Sometimes a broadcast address (either a "local broadcast" like 255.255.255.255 or a "directed broadcast" like 10.0.0.255) needs to be used in order to bypass software IP address filtering. This then encounters another problem in that many machines do not respond to broadcast requests (responses to broadcasts causes network problems, such as the 'smurf' hack).

ARP method

The ARP method is similar to the ping method, but an ARP packet is used instead. An explanation (in Spanish) is given at http://www.apostols.org/projectz/neped/ which includes a program called **neped** to do this detection.

The simplest ARP method transmits an ARP to a non-broadcast address. If a machine responds to such an ARP of its IP address, then it must be in promiscuous mode.

A variation of this technique takes advantage of the fact that machines "cache" ARPs. Each ARP contains the complete information of both the sender as well as the desired target information. In other words, when I send out a single ARP to the broadcast address, I include my own IP-to-Ethernet address mapping. Everyone else on the wire remembers this information for the next few minutes. Therefore, you could do something like sending out a non-broadcast ARP, then a broadcast ping. Anybody who responds to your ping without ARPing you could only have gotten the MAC address from a sniffed ARP frame. (To make double-sure, use a different source MAC address in the ping).

DNS method

Many sniffing programs do automatic reverse-DNS lookups on the IP addresses they see. Therefore, a promiscuous mode can be detected by watching for the DNS traffic that it generates.

This method can detect dual-homed machines and can work remotely. You need to monitor incoming inverse-DNS lookups on the DNS server in your organization. Simply do a ping sweep throughout the company against machines that are known not to exist. Anybody doing reverse DNS lookups on those addresses are attempting to lookup the IP addresses seen in ARP packets, which only sniffing programs do.

This same technique works locally. Configure the detector in promiscuous mode itself, then send out IP datagrams to bad addresses and watch for the DNS lookups.

One interesting issue with this technique is that hacker-based sniffing programs tend to resolve IP addresses as soon as they are found, whereas commercial programs tend to delay resolution until the point where the packet sniffer user views the protocol decodes.

source-route method

Another technique involves configuring the source-route information inside the IP header. This can be used to detect packet sniffers on other, nearby segments.

1. Create a ping packet, but put a loose-source route to force it by another machine on the same segment. This machine should have routing disabled, so that it will not in fact forward it to the target.
2. If you get a response, then it is likely the target sniffed the packet off the wire.
3. In the response, doublecheck the TTL field to find out if it' came back due to sniffing (rather than being routed correctly)

Details:

In loose source-routing, an option is added to the IP header. Routers will ignore the destination IP address and instead forward to the next IP address in the source-route option. This means when you send the packet, you can say "please send packet to Bob, but route it through Anne first".

In this scenario, both "Anne" and "Bob" are on the segment. Anne does not route, and therefore will drop the packet when received. Therefore, "Bob" will only respond if he has sniffed the packet from the wire.

On the off chance that Anne does indeed route (in which case Bob will respond), then the TTL field can be used to verify that Bob responded from routing through Anne, or answering directly.

The decoy method

Whereas the ping and ARP methods only work on the local network, the decoy method works everywhere.

Since so many protocols allow "plain text" passwords, and hackers run sifters looking for those passwords, the decoy method simply satisfies that need. It consists simply of setting up a client and a serve on either side of the network, which the client runs a script to logon to the server using Telnet, POP, IMAP, or some other plain-text protocol. The server is configured with special accounts that have no real rights, or the server is completely virtual (in which case, the accounts don't really exist).

Once a hacker sifts the usernames/passwords from the wire, he/she will then attempt to log on using this information. Standard intrusion detection systems or audit trails can be configured to log this occurance, alerting the fact that a sniffing hacker has found the traffic and attempted to use the information.

http://www.zurich.ibm.com/~dac/Prog_RAID98/Full_Papers/sniffer_detector.html/index.htm

host method

When hackers break into your systems, they will often leave behind wiretap programs running in the background in order to sniff passwords and user accounts off the wire. These are often

imbedded (as a trojan) in other programs, so the only way to find if something like this is running is to query the interfaces to see if they are running in promiscuous mode.

The most technique is to run the program "ifconfig -a". On my computer (Solaris 2.6) the output looks like:

```
# ifconfig -a
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
                    inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,PROMISC,MULTICAST> mtu 1500
                    inet 192.0.2.99 netmask ffffff00 broadcast 192.0.2.255
                    ether 8:0:20:9c:a2:98
```

Of course, the first thing a hacker will do is replace the 'ifconfig' program to hide this. There are other utilities you can download from the net that will query the hardware directly in order to discover this information, or you could run the 'ifconfig' program directly from a CD-ROM distribution.

latency method

This is a more evil method. On one hand, it can significantly degrade network performance. On the other hand, it can 'blind' packet sniffers by sending too much traffic.

This method functions by sending huge quantities of network traffic on the wire. This has no effect on non-promiscuous machines, but has a huge effect on sniffing machines, especially those parsing application layer protocols for passwords. Simply ping the machine before the load and during the load and testing the difference in response time can indicate if the machine is under load.

One problem with this technique is that packets can be delayed simply because of the load on the wire, which may case timeouts and therefore false positives. On the other hand, many sniffing programs are "user mode" whereas pings are responded to in "kernel mode", and are therefore independent of CPU load on a machine, thereby causing false negatives.

TDR (Time-Domain Reflectometers)

A TDR is basically RADAR for the wire. It sends a pulse down the wire, then graphs the reflections that come back. An expert can look at the graph of the response and figure out if any devices are attached to the wire that shouldn't be. They also roughly tell where, in terms of distance along the wire, the tap is located.

This can detect hardware packet sniffers that might be attached to the wire, but which are completely silent otherwise.

TDRs used to be used a lot in the old days of coax Ethernet in order to detect vampire taps, but these days with star topologies, they are used very rarely.

There also exist OTDR equipment, but this is really only for the truely paranoid.

hub lights

You can manually check hub-lights to see if there are any connections you don't expect. It helps to have labeled cables to figure out where (physically) a packet sniffer might be located.

SNMP monitoring

Smart hubs with SNMP management can provide automated monitroning of Ethernet (and other) hubs. Some management consoles will even let you log connections/disconnections to all your ports. If you've configured the system with the information where all the cables terminate, you can sometimes track down where a packet sniffer might be hiding.

Tools to detect packet sniffers

AntiSniff

http://www.l0pht.com/antisniff/

The most comprehensive sniffer-detection tool.

CPM (Check Promiscuous Mode)
ftp://coast.cs.purdue.edu/pub/tools/unix/cpm/

A tool from Carnegie-Mellon that checks to see if promiscuous mode is enabled on a UNIX machine.

neped
http://www.apostols.org/projectz/neped/

A tool from The Apostols that detects packet sniffers running on the local segment.

sentinel
http://www.packetfactory.net/Projects/sentinel/

cpm (Check Promiscuous Mode)
ftp://ftp.cerias.purdue.edu/pub/tools/unix/sysutils/cpm/

A UNIX tool for checking the promiscuous-mode status of adapters.

ifstatus
ifstatus

Another UNIX utility that should be run from `crontab` in order to monitor when adapters are put into promiscuous mode.

Other Sniffing Detection Resources

http://www.securiteam.com/unixfocus/Detecting_sniffers_on_your_network.html

3. How to

Wiretaps are extremely useful programs for security managers. They are the only mechanism that can log absolutely everything on the wire in such a way that hackers can't erase the logs.

3.1 Where can I get a sniffing program for my computer?

Windows

Ethereal
Ethereal is a UNIX-based program that also runs on Windows (which means installation is more difficult than you would expect and it looks strange). However, it is probably the best freeware solution available for sniffing on Windows.

It comes in both a read-only (protocol analyzer) version as well as a capture (sniffing) version. The read-only version is great for decoding existing packet captures (such as the traces that BlackICE generates). It avoids the hassle of installing the packet capture driver.

ftp://ethereal.zing.org/pub/ethereal/win32/

Installation is a little difficult; you'll have to hunt around on the website in order to figure out how to do it.

WinDump

A version of tcpdump for Windows.

http://netgroup-serv.polito.it/windump/

Network Associates Sniffer (for Windows)
You can probably go to NAI's website and download an eval copy of their Sniffer(r) Network Analyzer. I'm not quite sure of the details. They still sell their older, DOS-based version, which is still in many ways a superior product. Go to http://www.nai.com/mktg/survey.asp?type=d&code=2483

WinNT Server
Microsoft's WinNT Server comes with a built-in program called "Network Monitor". Go to the Networking control panel, select the "Services" tab, hit "Add..." and select "Network Monitor Tools and Agent". Once installed, you can run it from the program menu under "Administrative Tools".

BlackICE Pro
http://www.networkice.com/ BlackICE is an intrusion detection system that can also log evidence files to disk in a format that can be read by other protocol analyzers. This may be more useful than a generic sniffing program when used in a security environment. However, it is non-promiscuous, and only snifs the packets going into/out-of the machine.

CiAll
This is a program that can decode-only. This is great for such programs like "BlackICE" which can only log packets, but which do not contain their own built-in decoders. It is shareware for 90 days. http://members.tripod.com/~radhikau/ciall/ciall.htm. (However, since this is hosted on "Tripod", it is much less reputable than a real company, and as far as I know, it could contain a trojan designed to compromise the system: user beware).

EtherPeek
http://www.aggroup.com/ I think you can download a trial-ware version.

Intellimax LanExplorer
http://www.intellimax.com/

Triticom LANdecoder32
http://www.triticom.com/TRITICOM/LANdecoder32/ You can sign up for a free demo versino.

SpyNet/PeepNet
Doesn't decode frames, but reassembles sessions. This is the product that I want to write, but I haven't gotten around to it yet. http://members.xoom.com/Laurentiu2

Analyzer: a public domain protocol analyzer
A toolkit for doing various kinds of analyses. http://netgroup-serv.polito.it/analyzer/

Other Windows...
There are a larger number of Windows-based sniffing programs, many of which can be downloaded and installed like any other application.

Macintosh

EtherPeek
http://www.aggroup.com/ EtherPeek has been around for years in a Macintosh version and has also ported their software to Windows.

UNIX

UNIX solutions are generally based upon libpcap and/or BPF (Berkeley Packet Filters).

If you have a UNIX computer, then you should be using both **tcpdump** and **Ethereal**.

tcpdump
The oldest and most common wiretap program. In its simplest mode, it will dump a single-line decode of the packets to the commandline, one line per packet. It is *the* standard for UNIX packet capture.

The version that seems to have the best on-going maintainance is at http://www.tcpdump.org/.

The original version from LBL is at ftp://ftp.ee.lbl.gov/

A port for Windows has been done at http://netgroup-serv.polito.it/analyzer/

Ethereal
It currently looks like this is the **best** GUI-based sniffing program for UNIX. It is actively maintained. It is available at: http://ethereal.zing.org

snoop
An old standby for Sun Solaris machines. It is much less capable than tcpdump, but it is better at Sun-specific protocols like NFS/RPC. Snoop's tracefile has been specified in RFC 1761. It can be converted to tcpdump/libpcap format via many utilities, including 'tcptrace'.

sniffit
http://reptile.rug.ac.be/~coder/sniffit/sniffit.html Useful when trying to analyze application-layer data.

snort
A libpcap based packet-sniffer/logger with extensive filtering. http://www.clark.net/~roesch/security.html

trinux
Contains tcpdump and sniffit, among numerous other security utilities on a floppy bootable disk. http://www.trinux.org/

karpski
http://niteowl.userfriendly.net/linux/RPM/karpski.html A GUI Linux packet sniffer including a GTK interface.

SuperSniffer v1.3
http://www.mobis.com/~ajax/projects/ To quote the site: *enhanced libpcap based packet sniffer with many modifications like DES encryption of log file, traffic can be logged by regular expression pattern matching, POP and FTP connections are logged on one line, telnet negotiation garbage is discarded, duplicate connections are discarded, tcp packet reassembly, parellel tcp connection logging. Daemon mode where logs are dumped to specified port with authentication. Duplicate POP/FTP connections are not logged. Compiles under most operating systems, uses GNU autoconf. September 6, 1999.*

Thanks to < cripto at datasurge dot net > for the link

esniff
A small packet sniffer that helps sift passwords and usernames, published in Phrack issue 45-5. See http://www.phrack.com/search.phtml?view&article=p45-5. Esniff only runs on older SunOS platforms, so it really isn't relavent today.

exdump
http://exscan.netpedia.net/exdump.html Lightweight packet sniffer for Linux?

DOS

Because DOS isn't a true OS, it is in some ways more flexible as platform for running tricking things like wiretaps.

Sniffer(r) Network Analyzer
An oldy but a goody. The Sniffer defines all products in this genre. It consists of a 3-megabyte executable, which, since DOS has a max memory size of 640-kilobytes, is a pretty impressive achievement. This is a commercial product that has since been obsoleted by the Windows Sniffer, but I think it is still available. In many ways, it is better than the GUI version. http://www.nai.com

The Gobbler and Beholder
The Gobbler is a simple DOS-based packet sniffer with advanced packet-filtering capabilities. It is very old, but still in use. It is from the Delft University in the Netherlands. Beholder is an RMON probe based upon the same technology. http://nmrc.org/files/msdos/gobbler.zip

Klos PacketView
A low-end DOS product. http://www.klos.com

Other

Here are some other utilities that aren't classified above.

snmpsniff

A sniffing program dedicated to SNMP. http://www.idt.ipp.pt/~rff-ribe/snmpsniff/

3.2 Where can I get a utility to inject strangely formatted packets onto the wire?

libnet
http://www.packetfactory.net/libnet/

libnet is a public library written for C that can be used to format not only raw packets, but also a number of higher level protocols.

rootshell
http://www.rootshell.com/

www.rootshell.com contains a number of "exploit" scripts (such as bonk, teardrop, etc.) that demonstrate injecting raw packets onto the wire.

ipsend
http://coombs.anu.edu.au/~avalon/

ipsend is a utility that comes with the ipfilter package. It contains a scripting language for creating IP packets.

Sun Packet Shell (psh) Protocol Testing Tool
http://playground.sun.com/psh/

A Tcl/Tk based software toolset for protocol developement.

Net::RawIP
http://www.ic.al.lg.ua/~ksv/index.shtml
http://quake.skif.net/RawIP/

A PERL-based library for manipulating raw IP using libpcap.

CyberCop Scanner's CASL
http://www.nai.com

A scripting language that comes as part of CyberCop scanner.

3.2.1 Where can such a utility for Windows?

The CASL scripting is available for Windows, and you can use the NAI Sniffer product to generate hand-crafted packets. However, I'm not aware of any really good solutions to this for Windows, especially Win9x.

3.4 How can I sniff cable-modem segments?

Channels

The first problem is that cable-modems split upstream and downstream into two asymmetric channels. The cable-modem can "receive-only" on one high-speed channel (between 30-mpbs and 50-mbps), and "transmit-only" on the low-speed channel (typically around 1-mbps). Thus, your own cable-modem box cannot receive the transmit channel data.

Most cable-modem boxes have only a 10-mbps Ethernet output, which is far less than the 30-mbps they are reading from the cable. Furthermore, most cable-modem plants are designing their systems to fill that channel as much as possible. Most cable-modem users are seeing slower download speeds due to this congestion. It would be like drinking from a firehose -- you would miss lots of the data.

Vendor Diversity

There are many different cable-modem equipment vendors. What may work on one segment may not work on a different one.

The Cable Box Itself

Most cable-modem boxes are either bridges or routers themselves, and have separate MAC addresses and IP addresses. This means that putting your own Ethernet adapter into "promiscuous mode" has zero effect on the cable-modem itself.

It should be remembered that the cable-segment is a very "virtual" object: it looks like Ethernet to your computer, but it is very different on the other end.

When the box is a bridge, it can sometimes be reconfigured to pass all traffic.

These days, cable-modems frequently also support encryption, though not very good encryption. I'm not sure whether this raises the difficulty at all.

Broadcasts

While traditional sniffing is out of the question, there are some other ways to sniff the wire. First of all, you can sniff broadcasts, multi-casts, and semi-directed packets. For example, whenever anybody starts up PCAnywhere, they send out a PCAnywhere packet to all their neighbors. This will advertise to you the presence of somebody who may be running PCAnywhere, and who may be hacked.

The average cable-modem segment is filled with other broadcasts, such as NetBIOS packets (advertising user names), SNMP broadcasts (advertising network equipment such as routers and printers), bootp/DHCP broadcasts (advertising devices that you can probably configure and take over), etc.

Redirect

There are many ways of redirecting traffic through your own computer in order to sniff into connections:

ARP
: Many games can be played with ARP. First of all, you can send out an ARP packet that claims that you are the local router. This is likely to flood your own connection because EVERYONE will think you are the router. Conversally, you can send out an ARP where you claim to be a particular person. This is convenient with Windows because it will shut down its own connection -- you can often masquarade as somebody else this way.f

ICMP Redirects
: Many opperating systems support ICMP Redirects, where you can tell a machine to forward packets through you rather than the local router.

ICMP Router Advertisements
: A different variation of ICMP that does much the same as redirects: convinces a machine that you are the correct router.

In all of these cases, you have to configure your machine to further pass along the traffic to the real destination.

3.5 How can I sniff DSL segments?

See the cable-modem section above.

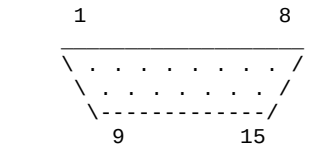
3.6 How can I create a receive-only Ethernet adapter?

By clipping the "transmit" wires.

Clipping the AUI

The easiest system to clip is 10-mbps using the old AUI transceiver. Note that in today's networking, such equipment is hard to find and expensive to buy "new", but littered all over the place as used equipment. You should consider stockpiling such equipment now, especially PCI Ethernet cards with transceivers.

Other forms of Ethernet are not easily altered. Thin-coax Ethernet uses the same wire for both transmitting/receiving. 10-BASE-T Ethernet requires a link heartbeat on the transmit line so that the hub knows it is connected. In short, only the AUI port contains pins that can be clipped without affecting the operation of the device.



Pin	Sgnl	Description
1	GND	
2	CI-A	Control In Circuit A
3	DO-A	Data Out Circuit A
4	GND	
5	DI-A	Data In Circuit A
6	VC	Voltage Common
7	-	
8	GND	
9	CI-B	Control In Circuit B
10	DO-B	Data Out Circuit B
11	GND	
12	DI-B	Data In Circuit B
13	VP	+12 Volts DC
14	GND	
15	GND	

Cutting pings 3 and 10 will stop transmission. The easiest way is to clip the pins on the connector rather than the wire itself.

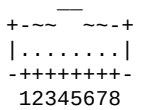
Clipping the UTP (10-BASE-T, 100-BASE-TX)

Creating UTP receive-only adapters is harder. Ethernet hubs check the "link integrity" with the adapter, which is done by sending a regular voltage pulse down the wire. On 100BaseTX, the problem is even more difficult because both sides send a steady stream of empty traffic down the wire. Therefore, you can't simply cut the transmit wires, because the hub will think the adapter is no longer there, and disconnect it.

However, you can "denature" the wire, introducing CRC errors that disrupt outgoing frames, but which still allows the stream of symbols from the card and/or link integrity pulses. This is actually a problem you may have encountered before: the link lights on the hub and adapter show a connection, but you cannot communicate on the network. Ethernet is actually very robust, so creating such a cable is difficult.

The high-speed integrity is maintained by using two wires for each signal, and twisting the wires around each other. You can think of it as one wire shielding the other, or that any electrical disturbance affects both equally, and since the *difference* in voltage is measured, it all evens out (Geek note: differential SCSI or Ultra-DMA/66 cables are based on the same principle).

As a consequence, simply "untwisting" the transmit pair will degrade the signal to the point that outgoing traffic will be corrupted with CRC erros, yet the hub will still get enough of a signal to know the adapter is still there.



```
-----------------------------------------------
  Pin   MDI signal        MDI-X signal
-----------------------------------------------
   1    TD+ xmit to UTP     RD+ rcv from UTP
   2    TD- xmit            RD- rcv from UTP
   3    RD+ rcv from UTP    TD+ xmit to UTP
   4    Not used            Not used
   5    Not used            Not used
   6    RD- rcv from UTP    TD- xmit to UTP
   7    Not used            Not used
   8    Not used            Not used
```

In order to denature it, you must use a 4-pair cable where normally two of the pairs are unused. Each twisted-pair is color coded, with COLOR-stripes on white for one wire, and white-stripes on COLOR for the other wire. In the above diagram, the transmit pair is normally orange/white, whereas the receive pair is normally green/white (for the AT&T or EIA/TIA 568B standard). However, the EIA/TIA 568A standard reverses these colors (aargg). Therefore, all you are worried about is that pins 3 and 6 *must* share the same color, but in order to denature the transmit pair, pins 1 and 2 must NOT share a color. Pick any of the others, it doesn't matter. *Both ends of the cable must be wired the same.*

You should make this a *long* cable, as long as the standard will allow. The problem is that it doesn't always degrade the signal enough, so the longer the better (I've used 40-feet, but still about 5% of the transmitted frames still get through). Furthermore, you should use 100-mbps (100-BASE-TX) for this, as the signal degrades easier. I've never tried with 10-mbps.

However, this approach still leaves the possibility that somebody might "fix" this by replacing the bad cable with a good one. An alternate technique involves altering the NIC itself, inside the box where nobody can accidentally fix it. In this case, instead of clipping the wire you should *add* something to it. Most adapters leave the traces exposed (though some are beginning to shield the RJ45 connector, but you can easily peel back the shielding). Solder a paperclip or wire onto the transmit pin #1 on the backside of the connector. Adjust the length as needed until the transmission is screwed up. I've never tried this, so if you get it to work, please drop me e-mail.

Using either of these techniques may cause the FCC to come after you because of radio wave transmission. The computer case *should* shield the altered adapter, but I make no guarantees.

Another proposal is to create a circuit to generate the symbols. This may be easy for hardware engineers, but I'm a software engineer and have never done it.

Why would I want to do this?

For security reasons. Networking is full of accidents waiting to happen, which crackers/hackers exploit in order to break into systems. Clipping the transmit wires on an Ethernet adapters generates a "one-way-drop" that can receive data, but cannot be compromised by a hacker through some accident.

Examples:

- Receiving syslog messages and storing them to a non-compromisable system. The 'syslog' protocol is used by numerous UNIX services to log security events, and is based upon UDP so one-way datagrams can be used forward to a device that cannot respond. ARP and route tables need to be manually configured to ensure this operation.
- Similarly receiving SNMP traps, which also use UDP. Many systems generate SNMP traps in response to security related events.
- Promiscuous packet capture. Many systems will dump ethernet traffic to disk files on the fly. By setting up a receive-only system on a potentially insecure network, you don't risk adding another security hole to the system.

Some people have suggested this as a way to conceal a wiretap from sniffer-detection programs. This is overkill: in order to conceal yourself, you simply unbind the TCP/IP stack or install firewall filters on the machine. This approach is not adequate for the one-way-drops mentioned above because of the ease at which such functionality can be accidentally re-enabled.

3.7 What about eavesdropping on wireless like IEEE 802.11 a.k.a. AirPort?

In late 1999, Apple released their iBooks and at the same time their AirPort wireless networking. This is actually an implementation of the IEEE 802.11 wireless standard, which in theory means that both Apple computers and Windows-based PCs (as well as a host of other equipment) should be able to use the same wireless infrastructure. For example, a person with a Nokia wireless PCMCIA adapter in their notebook should be able to connect via TCP/IP to an AirPort base station (and be configured automatically via DHCP).

There are two IEEE 802.11 standards, one for 2-mbps and one for 11-mbps. This discussion focuses on the 2-mbps standard.

Spread Spectrum

The first question dealing with sniffing is the signaling. This wireless standard uses "spread spectrum" technology.

- Rather than transmitting data at a single frequency, data is transmitted over a range of frequencies.
- This makes it more immune to electronic noise.
- It allows many users to share the same spectrum like cellular. In fact, CDMA, a cellular technology, uses spread spectrum, where each "code" (code division multiplexing) determines the sequence used to "spread" the signal.
- In theory, spread-spectrum makes it impossible to eavesdrop. The eavesdropper would need to know the "spreading" function used.

Spread-spectrum technology came out of the cold war as a way of sending signals that were near impossible to eavesdrop on. The theory is that an eavesdroppper only hears whitenoise, and that even proving there is a signal could be difficult.

However, this assumed that you could securily communicate the "spreading function" to both the transmitter and receiver. This isn't reasonable in consumer-grade products that we'll be buying. The keys will be distributed manually. Moreover, there aren't that many keys. The upshot is that spread-spectrum has little impact as an anti-sniffing countermeasure. To be fair, it isn't intended to be -- it's used in wireless communication for its anti-noise, bandwidth-sharing capabilities. Security will be accomplished via standard digital encryption techniques (see below).

The spectrum used by the standard is 2.400 GHz to 2.4835 GHz, though in theory different frequencies are defined for Japan and Europe.

Signal range

Roughly 100-meters (300-feet) indoors and 300-meters (1000-feet) outdoors. Estimates indicate that a base station will be able to communicate one floor in each direction. In extreme office conditions, vendors are quoting about 20-meters. However, with parabolic antennas, eavesdroppers can receive signals from much further away.

In theory, any IEEE 802.11 compliant device can eavesdrop on all the packets that are within range. (Though, of course, the may need to decrypt it as well).

Encryption

A method called "Wired Equivalent Privacy (WEP)" is used. This is based upon the RC4 encryption protocol with 40-bit keys. This is essentially the same protocol used by web-browsers for secure connections. RC4 supports up to 128-bit encryption, but the 802.11 standard is at 40-bit encryption for export purposes. Some vendors are providing more bits in the security key, for example Lucent is selling their "WaveLAN Gold" cards supporting 128-bit encryption (though it appears that 128-bit is available outside the US, not inside, as it was developed outside the US and Lucent is protesting US export restrictions by not selling the stronger version inside the US).

WEP only protects the data portion. This means that a sniffing machine can decode the physical layer transmissions, but must decrypt the data portion.

WEP uses a "shared-key" architecture. This is actually a very bad design, because it requires users to enter in their keys in order to use the network. These keys can will likely be based upon passwords, which usually result in effective keys even less than 40-bits. In contrast, web-browsers

using SSL are able to encrypt data with no predetermined shared key.

WEP is optional, by default it is turned off. We will have to see in the future how often it really is used. For example, WEP is not practical for "ad-hoc" networks (peer-to-peer networks); it is more useful with the use of Access Points (APs). From the look of it, vendors are selling the encryption feature as an "add-on" as well. This bodes well for malicious packet sniffers.

To summarize encryption: it will make sniffing difficult, but not impossible. Most people won't use encryption, but even then it will be easy to decrypt the 40-bit encrypted messages. Specialized hardware could be built to recover the key in near real time, but also distributed computing power (like http://www.distributed.net) can also be used to recover keys. In particular, because the data portion is very well known (IP headers), it is susceptible to a "known plaintext" attack.

Access

A security concern related to sniffing is simple access. Someone can walk into a building with a notebook computer, which can connect to the network behind the firewall. Internal networks tend to be insecure, so this is a real danger. The WEP protocol has a number of features to prevent this, such as the ability to hard-code MAC addresses into the base-stations specifying who may connect to the network.

Roving

Users can rove around a network, and be handed off from area-to-area much like cell-phones. A unit maintains the same MAC address and IP address, but changes who it's talking to. This means that the backbone to handle this has to be switched Ethernet or ATM with VLANs. In theory, you could walk around a company and have your notebook beep at you as soon as it finds an area where computers are talking unencrypted.

Airports

Several companies are equiping places like airports with access stations that allow you to surf online. The WEP protocol as no support for this type of authentication (shared secrets suck). These companies plan on charging connect time, but you could equally have fun by sitting down with your notebook and sniffing everyone else connected to the web. Have fun reading their e-mail, eavesdropping on their chatrooms, and the like.

3.8 How can I sniff a switched network?

In theory, you cannot sniff a switched network. All that the sniffing would do would be to see your own traffic anyway. In practice, there are numerous ways.

3.8.1 Switch Jamming

Some switches can be kicked out of "bridging" mode into "repeating" mode where all frames are broadcast on all ports all the time. This is done by overflowing the address tables with lots of false MAC addresses. This can be done with a simple traffic generation phase, or by sending a continual stream of random garbage through the switch. In security terms, this is known as "fail open" behavior rather than "fail close", meaning that when the device fails, security provisions are removed. Of course, switches aren't really designed with security in mind.

See the HUNT Project at http://www.cri.cz/kra/index.html for more info.

3.8.2 ARP Redirect

ARP packets contain both the local binding as well as the desired binding. For example, let's say that Alice wants to find Bob's Ethernet MAC address. Bob has an IP address of "192.0.2.1". Alice would send out an ARP request with the following information.

Operation:	Request	
Alice:	192.0.2.173	00-40-05-A4-79-32
Bob:	192.0.2.1	?? ?? ?? ?? ?? ??

The entire exchange might looke like the diagram below. Alice has an IP packet of some sort

(let's say an ICMP ping) to send to Bob. In order to find Bob's MAC address, Alice ARPs it. Bob responds to Alice, telling her his MAC address. Now Alice sends her IP packet to that Ethernet MAC address.

```
Alice ----> ARP broadcast request ----> Bob
Alice <----  ARP unicast response <---- Bob
Alice ----> ICMP ping request     ----> Bob

Alice <----  ICMP ping response   <---- Bob
Alice <----  ICMP ping request    <---- Charles
```

Now Bob has an IP packet to send to Alice. In theory, Bob would need to ARP Alice in order to find her MAC address. But he doesn't. This is because he has remembered her MAC address information that was sent in the original ARP request.

In fact, everyone on the local Ethernet saw that request since it was broadcasted. So if Charles at this point wants to ping Alice, he doesn't need to ARP her either, even though he wasn't involved with the original exchange.

Broadcasts are sent to everyone on an Ethernet switch. Therefore, you can subvert the switch by sending out ARPs claiming to be somebody else as the source address. You can broadcast out an ARP claiming to be the router, in which case everyone will try to route through you. Or you can send an ARP request just to the victim's MAC address, claiming to be the router, at which point just the victim will forward packets to you. Conversely, you can send an ARP to the router's MAC address claiming to be the victim.

In all these cases, you must be prepared to forward packets in the real direction, otherwise you cut off communication.

See http://www.zweknu.org/src/MiM/ for some sample programs.

3.8.3 ICMP Redirect

An ICMP redirect tells a machine to send its packets in a different direction. A typical example is when there are two logical subnets on the same physical segment. Alice is on one subnet talking to Bob on another subnet. Neither knows they are on the same physical segment, but the local router knows. When Alice sends the router a packet destined for Bob, the router sends an ICMP Redirect to Alice informing here of the fact that she can send the packet to Bob directly.

A hacker (Mark) can subvert this by sending a redirect to Alice claiming that she should send him Bob's packets.

3.8.4 ICMP Router Advertisements

ICMP Router Advertisements inform people who the router is. A hacker can send these packets out claiming to be a router; people will believe them and start forwarding their traffic through that person.

Luckily, many machines don't support this feature because it is relatively new. Now that the security implications are well known, many new systems still won't support it.

See http://www.l0pht.com/advisories/rdp.txt for more information.

3.8.5 Fake the MAC address of the victim

The idea is to cause the switch to start fowarding you the frames destined to the victim. You can do this simply by sending out frames with the source address of the victim. The "auto-learning" feature of the switch will now believe you are the victim, and send frames your way.

The obvious problem is that victim itself will still send out frames with its MAC address (cuasing the switch to revert). Another problem is that if the victim doesn't receive the frame, then its communications breaks, and there won't be anything more to sniff.

There are a few solutions to this problem, depending upon what you want to do. You may want to subvert an authenticated connection, in which case you DoS the victim (taking it offline), redirect the switch, and continue on with the connection as if nothing happened. For example,

let's say that Alice has a Telnet connection to the BOB server. You DoS Alice's machine, taking her off line. You then send out packets with Alice's MAC address, causing the switch to send you all packets destined for her. In order to pick up her connection, you cause the server to send you a TCP packet (i.e. use the talk service to prompt her for a connection). At this point, you simply start reversing the sequence and acknowledgement numbers to continue the Telnet connection.

Another solution to this problem is to "sample" traffic. Send out packets with the victim's MAC on occasional intervals. You'll likely get the next few packets destined for the victim, but the victim will timeout and recover the connection. The victimized user will notice occasional network delays.

A similar solution is that when you receive an incoming packet, turn around and *broadcast* it back out. This way the victim still receives the packet. A steady stream of outgong traffic and broadcasts of the incoming traffic will allow you to recover a good percentage of the original traffic.

3.8.6 Reconfigure span/monitor port on switch

Most switches allow ports to be configured as "monitor" or "span" ports that will copy some or all of the traffic going across the switch. In fact, these ports are designed for packet sniffers when the network administrator needs to solve a problem.

In many cases, a hacker can telnet to the switch or reconfigure it with SNMP. Most switches are installed with default or backdoor passwords.

SNMP is particularly fun because the hacker can 'grind' through all the passwords until he/she finds the correct one (though most come default with "public" or "private"). In some cases, the network admin configures the switch to only allow SNMP from the IP address of the network management console. However, this same network management console will likely be sending other SNMP packets about: sniffing broadcasts or just incoming SNMP directed at the host will likely reveal who the SNMP management console is, at which point IP spoofing can be used to manage the switch. Similarly DNS Zone Transfers might reveal suggestive names such as "hpov.example.com" (hpov = HP OpenView, a popular SNMP console).

3.8.7 Cable-taps

You can tap into full-duplex Ethernet cable. A couple vendors make products that do this for you. Some features of these products are:

- Your packet sniffer needs two Ethernet adapters to receive the send/receive channels. Most products support sniffing from only a single adapter at a time, which means you can only sniff from one channel at a time.
- All the products I know are fault tolerant, which means that if the power fails on the box, they will not interupt the flow of traffic.

Of course, you caa always tap into cables between switches or between an important host and a switch. Vendors of cable-taps are:

www.Shomiti.com

The "Century Tap" family is used for this purpose. The basic tap works as described above. They also have a fiber-optic tap. Finally, they have a 12-port tap that allows roving analysis on twelve full-duplex lines.

www.NetOptics.com

They not only have the basic tap, but also one that support fiber-optics as well. Since they don't make their own protocol analyzer, they resell their products for other packet sniffer companies.

www.Sniffer.com

Rather than a passive tap described above, they have a full "pod" that does packet capture and filtering on the unit itself. It contains several Ethernet chips, CPUs, and memory.

3.9 How can I put an adapter into promiscuous mode?

Normal Ethernet adapters reject all incoming traffic that isn't sent to that adapter. In order to sniff on the wire, the adapter must be reconfigured to accept all traffic on the wire. This is called **promiscuous mode**.

In order to sniff on the wire, a driver must be written that both puts the adapter into promiscuous mode, as well as buffers the incoming frames.

3.9.1 How can I configure promiscuous mode in the user interface?

You can't.

Many people are looking for a configuration option called "promiscuous mode". They don't exist. Putting an adapter in this mode is useless unless you've got a program setup to do something with the received frames. Therefore, it is pointless to simply configure it in the user interface because nothing is around that will process these frames.

Indeed, some sniffing programs will leave the adapter in promiscuous mode after you exit them. This causes problems because now rather than rejecting incoming traffic in hardware, the traffic is now rejected in software. Either way, without a sniffing program, the traffic will be rejected.

3.10 How can I write a sniffing program on Windows (using VisualBasic, Delphi, C++, etc.)?

It is not quite that simple.

Sniffing Driver

The first step would be to figure which packet capture driver to use. These drivers cannot be written in high-level languages like VB, so you will have to either write one in a different language (C or assembler) or download a special sniffing driver from the net. The two most commonly used drivers these days are the commercial sniffing driver from PCAUSA (which is used in Carnivore) and the freeware driver part of the WinDump package. Note Microsoft includes a sample "packet capture" driver in its NT DDK (Device Driver Kit). It is in the directory `\DDK\src\network\packet`. This driver is useful reference information, but it really doesn't work in practice.

Interface

TODO

4. Protocols

4.1 What protocols are vulnerable to sniffing?

Following is a sampling of typical protocols that are sniffed, especially for passwords.

Telnet and rlogin

Sniffing can capture the keystrokes as the user types them, including the user name and password. A long time ago I wrote a commercial product that would capture all the text and dump it to a terminal emulator, which reconstructed exactly what the end-user was seeing. This basically produced a realtime viewer of the remote users screen.

http

The default version of HTTP has numerous holes. Many web sites use "Basic" authentication, which sends passwords across the wire in plain-text. Many web sites use another technique which prompts the user for a username and password, which are also sent across the network in plain-text. Data sent in clear-text.

SNMP

Alomost all SNMP traffic is SNMPv1, which has no good security. SNMP passwords (called community-strings) are sent across the wire in the clear.

NNTP

Passwords sent in the clear. Data sent in clear

POP

Passwords sent in the clear. Data sent in clear

FTP

Passwords sent in the clear. Data sent in clear

IMAP

Passwords sent in the clear. Data sent in clear

Note that all of these systems have secure alternatives. When entering things like credit card information, most web sites use SSL encryption rather than normal HTTP. Similarly, S/MIME and PGP can encrypt e-mail at a level higher than e-mail protocols like POP/IMAP/SMTP.

5. Protocol Analysis

5.1 What is protocol analysis?

Protocol analysis is the process of capturing network traffic (with sniffing programs) and looking at it closely in order to figure out what is going on.

As data is sent across a wire, it is "packetized", meaning broken down into multiple packets that are each sent individually across the network, then reassembled back on the other side. For example, you probably downloaded this document from the network. Since this document is around 45,000 bytes and the typical packet size is 1,500 bytes, it took about 30 packets to deliver this document to you.

Below is a sample packet. This packet was taken from a packet sniffer that watch my workstation download this FAQ from my website. This packet was originally 1514 bytes long, but only the first 512 bytes are shown here:

```
000  00 00 BA 5E BA 11 00 A0 C9 B0 5E BD 08 00 45 00 ...^......^...E.
010  05 DC 1D E4 40 00 7F 06 C2 6D 0A 00 00 02 0A 00 ....@....m......
020  01 C9 00 50 07 75 05 D0 00 C0 04 AE 7D F5 50 10 ...P.u......}.P.
030  70 79 8F 27 00 00 48 54 54 50 2F 31 2E 31 20 32 py.'..HTTP/1.1.2
040  30 30 20 4F 4B 0D 0A 56 69 61 3A 20 31 2E 30 20 00.OK..Via:.1.0.
050  53 54 52 49 44 45 52 0D 0A 50 72 6F 78 79 2D 43 STRIDER..Proxy-C
060  6F 6E 6E 65 63 74 69 6F 6E 3A 20 4B 65 65 70 2D onnection:.Keep-
070  41 6C 69 76 65 0D 0A 43 6F 6E 74 65 6E 74 2D 4C Alive..Content-L
080  65 6E 67 74 68 3A 20 32 39 36 37 34 0D 0A 43 6F ength:.29674..Co
090  6E 74 65 6E 74 2D 54 79 70 65 3A 20 74 65 78 74 ntent-Type:.text
0A0  2F 68 74 6D 6C 0D 0A 53 65 72 76 65 72 3A 20 4D /html..Server:.M
0B0  69 63 72 6F 73 6F 66 74 2D 49 49 53 2F 34 2E 30 icrosoft-IIS/4.0
0C0  0D 0A 44 61 74 65 3A 20 53 75 6E 2C 20 32 35 20 ..Date:.Sun,.25.
0D0  4A 75 6C 20 31 39 39 39 20 32 31 3A 34 35 3A 35 Jul.1999.21:45:5
0E0  31 20 47 4D 54 0D 0A 41 63 63 65 70 74 2D 52 61 1.GMT..Accept-Ra
0F0  6E 67 65 73 3A 20 62 79 74 65 73 0D 0A 4C 61 73 nges:.bytes..Las
100  74 2D 4D 6F 64 69 66 69 65 64 3A 20 4D 6F 6E 2C t-Modified:.Mon,
110  20 31 39 20 4A 75 6C 20 31 39 39 39 20 30 37 3A .19.Jul.1999.07:
120  33 39 3A 32 36 20 47 4D 54 0D 0A 45 54 61 67 3A 39:26.GMT..ETag:
130  20 22 30 38 62 37 38 64 33 62 39 64 31 62 65 31 ."08b78d3b9d1be1
140  3A 61 34 61 22 0D 0A 0D 0A 3C 74 69 74 6C 65 3E :a4a"....<title>
150  53 6E 69 66 66 69 6E 67 20 28 6E 65 74 77 6F 72 Sniffing.(networ
160  6B 20 77 69 72 65 74 61 70 2C 20 73 6E 69 66 66 k.wiretap,.sniff
170  65 72 29 20 46 41 51 3C 2F 74 69 74 6C 65 3E 0D er).FAQ</title>.
180  0A 0D 0A 3C 68 31 3E 53 6E 69 66 66 69 6E 67 20 ...<h1>Sniffing.
190  28 6E 65 74 77 6F 72 6B 20 77 69 72 65 74 61 70 (network.wiretap
1A0  2C 20 73 6E 69 66 66 65 72 29 20 46 41 51 3C 2F ,.sniffer).FAQ</
1B0  68 31 3E 0D 0A 0D 0A 54 68 69 73 20 64 6F 63 75 h1>....This.docu
1C0  6D 65 6E 74 20 61 6E 73 77 65 72 73 20 71 75 65 ment.answers.que
1D0  73 74 69 6F 6E 73 20 61 62 6F 75 74 20 74 61 70 stions.about.tap
1E0  70 69 6E 67 20 69 6E 74 6F 20 0D 0A 63 6F 6D 70 ping.into...comp
1F0  75 74 65 72 20 6E 65 74 77 6F 72 6B 73 20 61 6E uter.networks.an
```

```
....
```

This is the standard "hexdump" representation of a network packet, before being decoded. A hexdump has three columns: the offset of each line, the hexadecimal data, and the ASCII equivalent. This packet contains a 14-byte Ethernet header, a 20-byte IP header, a 20-byte TCP header, an HTTP header ending in two line-feeds (0D 0A 0D 0A) and then the data.

The reason both hex and ASCII are shown is that sometimes ones is easier to read than the other. For example, at the top of the packet, the ASCII looks like garbage, but the hex is readable, from which you can tell, for example, that my MAC address is 00-00-BA-5E-BA-11 (i.e. "BASEBAll").

A "protocol analyzer" will then take this hexdump and interpret the individual fields:

```
ETHER: Destination address : 0000BA5EBA11
ETHER: Source address : 00A0C9B05EBD
ETHER: Frame Length : 1514 (0x05EA)
ETHER: Ethernet Type : 0x0800 (IP)
IP: Version = 4 (0x4)
IP: Header Length = 20 (0x14)
IP: Service Type = 0 (0x0)
IP: Precedence = Routine
IP: ...0.... = Normal Delay
IP: ....0... = Normal Throughput
IP: .....0.. = Normal Reliability
IP: Total Length = 1500 (0x5DC)
IP: Identification = 7652 (0x1DE4)
IP: Flags Summary = 2 (0x2)
IP: .......0 = Last fragment in datagram
IP: ......1. = Cannot fragment datagram
IP: Fragment Offset = 0 (0x0) bytes
IP: Time to Live = 127 (0x7F)
IP: Protocol = TCP - Transmission Control
IP: Checksum = 0xC26D
IP: Source Address = 10.0.0.2
IP: Destination Address = 10.0.1.201
TCP: Source Port = Hypertext Transfer Protocol
TCP: Destination Port = 0x0775
TCP: Sequence Number = 97517760 (0x5D000C0)
TCP: Acknowledgement Number = 78544373 (0x4AE7DF5)
TCP: Data Offset = 20 (0x14)
TCP: Reserved = 0 (0x0000)
TCP: Flags = 0x10 : .A....
TCP: ..0..... = No urgent data
TCP: ...1.... = Acknowledgement field significant
TCP: ....0... = No Push function
TCP: .....0.. = No Reset
TCP: ......0. = No Synchronize
TCP: .......0 = No Fin
TCP: Window = 28793 (0x7079)
TCP: Checksum = 0x8F27
TCP: Urgent Pointer = 0 (0x0)
HTTP: Response (to client using port 1909)
HTTP: Protocol Version = HTTP/1.1
HTTP: Status Code = OK
HTTP: Reason = OK
....
```

In the above hexdump and decode, I've underlined the "Time to Live" field of 0x7F. This is how a protocol decode works: it pulls each of the fields out of the packet and attempts to explain what the numbers mean. Some fields are as small as a single bit, other span many bytes.

Protocol analysis really is a difficult art, and requires *a lot* of knowledge about protocols in order to do it well. However, the rewards are that a lot of information can be easily gleaned from protocols. This info can be useful to network managers trying to debug problems, or hackers who are trying to break into computers.

5.2 What is hexadecimal?

All data within a computer is represented as numbers. Hexadecimal (or simply "hex") is a better numbering system for viewing this data than the "decimal" numbers everyone is already familiar with. Hexadecimal is one of those computer science concepts that is difficult to understand, until the "aha"

moment when you finally understand it. After that point, it becomes second nature. Everybody has a different path to that "aha" moment, so don't feel bad if you don't understand the following discussion. However, you *must* eventually understand hexadecimal, so you really should look it up on the web.

The word "decimal" has the root "dec", meaning "10". This means that there are 10 digits in this numbering system:

```
0 1 2 3 4 5 6 7 8 9
```

The word "hexadecimal" has the roots "hex" meaning 6 and "dec" meaning 10; add them together and you get 16. This means there are sixteen digits in this numbering system:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
```

The is useful because all data is stored in a computer as "bits" (binary-digits, meaning two digits: 0 1), but all bits are grouped into 8-bit units known as "bytes" or "octects", which in theory have 256 digits. Bits are two small to view data, because all we would see is a stream like 00101010101000010101010110101101101011110110, which is unreadable. Similarly, using 256 digits would be impossible: who can memorize that many different digits? Hexadecimal breaks a "byte" down into a 4-bit "nibble", which has 16-combinations (256 = 16*16). This allows us to represent each bytes as two hexadecimal digits.

Hexadecimal allows technical people to visualize the underlying binary data. A technical person has the following table memorized:

```
0000 = 0      0001 = 1      0010 = 2      0011 = 3
0100 = 4      0101 = 5      0110 = 6      0111 = 7
1000 = 8      1001 = 9      1010 = A      1011 = B
1100 = C      1101 = D      1110 = E      1111 = F
```

In other words, when you encounter the hexadecimal digit "B", you should immediately visualize the bit pattern "1011" in your head. It is much like memorizing multiplication tables as a kid, memorizing this table will serve much the same purpose.

Hexadecimal is often preceded by a special character(s). For example, when you see the number "12", is this "twelve" (decimal) or "eighteen" (hexadecimal)? If it is hex, it is often written as either "0x12", "x12", or "$12". The former is the preferred version, since that is how many programming languages represent it. Naturally, this isn't needed for hex dumps because the fact we are showing hex is pretty much assumed.

5.3 What is ASCII?

Computers represent everything as numbers. This means the text your are reading right now is represented as numbers within the computer. ASCII is one just representation. In ASCII, the letter 'A' is represented by the number 65, or in hex, 0x41. The letter 'B" is represented by the number 66/0x42. And the process continues for all characters, numbers, punctuation, and so forth.

If you look at the normal (U.S. English) keyboard you will count 32 punctuation characters, 10 decimal digits, 26 letters, and 26 *more* letters when you take into account UPPER/lower case. This comes to 94 different characters. In binary, you need 7-bits to represent that number of combinations. This maps nicely onto the standard 8-bit bytes used in computers, with room left over.

In hex dumps, note that the ASCII columns contains lots of periods. A byte has 256 combinations, but we can only view 94 of them. Any character that is not one of these 94 visible characters is shown as a period.

5.4 What is the "OSI 7-Layer Model"?

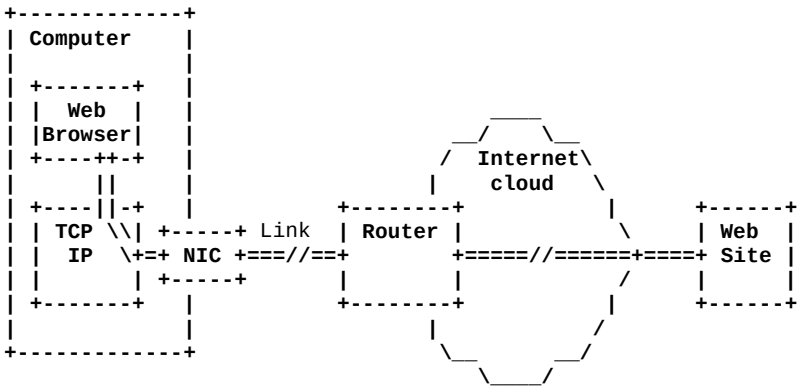
Any discussion of protocol analysis usually starts with the OSI model.

Previous to the OSI Model, networking was generally "monolithic". In other words, the application that displayed the data on your screen was also responsible for the hardware that moved the bits across the wire. You couldn't change either the software or the hardware with upgrading the entire system. Imagine having to buy a new computer simply to upgrade the software!

The concept behind the OSI model is to separate the functionality into different conceptual modules. As a quick introduction to this, consider the following 3-layer model that most consumers are familiar with:

Application	Web browser, e-mail, RealAudio
Transfer	TCP/IP
Link	Dial-up modem, Cable modem, DSL, Ethernet

Conceptually, this can be viewed in the following diagram:



In this conceptual representation, the user's "Web Browser" application is trying to view a web-page located on a "Web Site" located out on the web somewhere. The "Web Browser" passes it down to the "TCP/IP" stack, which sends it out the "NIC" across the local "Link" to the nearest "Router" gateway. At this point, the client doesn't really know what is going to happen to the data. Presumably, it is passed from router-to-router through the Internet "cloud" until it reaches the destination "Server" hosting the "Web-Site".

The important point to learn from all this is the concept of **abstraction**. Each component of this diagram does not know anything about the other components. For example, consider the postman that delivers your mail (physical mail, not e-mail). The postman has no knowledge of the contents of your letters. S/he simply moves the mail between the local post-office and your mail box. In much the same way, the IP layer within the machine has no knowledge of the contents of the packets. Its only responsibility is to accept packets from the TCP layer, and send them out NIC toward the local Router. The IP layer is even fuzzy on the exact details of how the NIC transports the packets to the local router, and is completely clueless as to what happens to the packets after that point.

In other words, *each layer has a single job to do, and doesn't know anything about what is going on in the other layers.*

This is difficult for humans to understand, because we can see what the entire process is trying to accomplish. It is difficult for us to constrain our view to just a single component.

Note that in the above diagram, there is lots of stuff toward the left of the diagram, but not much detail toward the right. This is because the user really has no idea how packets are really routed on the Internet, nor does the user really know much about the web site. In fact, the web site may consist of multiple computers for load-balancing purposes, or conversely may consist of a single computer hosting many web sites.

In order to understand the OSI Model, you must first understand the political backdrop behind its creation. In the late 1970s, computer networking was dominated by large, proprietary systems. Once you bought product from a single vendor, you could never buy products from other vendors that would work with it. You were "locked-in", and the vendor was free to charge whatever they wanted. Therefore, the OSI working group (OSI = Open Systems Interconnect) as part of the ISO (International Organization for Standardization) was created in order to standardize network protocols. In theory, if everyone conformed to standards, then consumers could buy products from different vendors at lower prices and save lots of money.

However, the OSI/ISO standardization process is fundamentally dysfunctional. For example, what does the acronym I-S-O stand for? In English, this stands for "International Organization for Standardization" or IOS. In French, it stands for "*Organisation Internationale de Normalization*", or OIN. (This can be seen on their homepage at http://www.iso.ch). English and French are the two

official languages of ISO, and acronyms are chosen so that they match neither the English or French terms they refer to. Generally, standards that start within the ISO follow the same logic: in an effort to appease everybody, they end up satisfying nobody.

The OSI Model was a blueprint for an entire protocol suite that would implement the individual layers. They actually succeeded in generating this standard, but it never achieved popular use and has largely been supplanted by TCP/IP. While large organizations (government, industrial), mostly in Europe, have attempted to use it, it has largely become a boat anchor.

The following is a description of the 7-Layers within the model, and how they map onto the TCP/IP suite that we are familiar with.

Layer	***Unit***	***As far as...***	***Explanation***
Application 7	File transaction command	user	This layer doesn't mean the application itself, but the protocols that do the work for the application. Examples: HTTP for web browsers, SMTP/POP/IMAP for e-mail,
Presentation 6			This layer is an example of the political processes mentioned above. The theory was that the application layer didn't need to format the data for transmission across the wire; that would be the job of an underlying layer. Furthermore, the idea was that a client and server would negotiate which format they wanted to use. This actually made sense back in the late 1970s, because most networking involved dumb, character-mode terminals controlled by mainframes. An application wanted to deal with abstract concepts like database forms that a user filled out. Different terminals have different control codes to display this type of information. However, it makes virtually no sense nowadays. As a result, OSI protocols always negotiate a Presentation encoding, even though only one option is available. In TCP/IP, the only protocol that really does his negotiation is Telnet. Character-mode applications such as '`vi`' go through a package called "curses". When you Telnet to a host, you exchange your terminal type with with the host. When an application such as 'vi' wishes to clear the screen, the 'curses' packages tells it how to do so for your particular terminal. However, the concept of how data is formatted on the wire is extremely important to applications. So even though it doesn't exist as a real layer outside of OSI protocols, it is still an important component of all applications.
Session 5			Like the Presentation layer, the Session layer is an artifact of the ISO political processes, but more so. In fact, as it turns out, the Session layer is completely useless. If you pick up a book on OSI and read up on the Session layer, you will end up reading lots of technical content about it but you still won't be able to answer the question: What is the session layer for? If you ask somebody what the Session layer does, they will tell you something like "It establishes a session between two entities". If you ask what a "session" is, the answer would be "It's what the Session layer establishes". If they give you a more detailed answer, they are probably confused and will really be describing a Transport layer "connection", which isn't the same thing. There is a Session layer protocol defined by OSI, but the OSI hasn't defined any uses for it (that I know of). In other words, all the OSI applications establish both Transport layer

			"connections" and Session layer "sessions" when they start talking to each other, then tear down the connections/sessions together when they stop talking. Luckily, the OSI protocol has been defined in a fairly efficient manner such that both can be established in the same packet, and the Session layer only adds a couple bytes on average to every packet sent. Now I could actually tell you what the Session layer really is for, but I'm not going to. (Hint: it has to do with terminals talking to mainframes.) You should simply remember the concept that it really doesn't do anything. The 7-Layer model really just contains 6-layers (actually, 5-layers, because the Presentation layer should be thrown out as well).
Transport 4	connection	other program	The Transport layer is TCP (and UDP). Like the discussions above, the official OSI Transport layer has lots of worthless features attached to it. However, I'm going to pretend it just describes how TCP implements this layer. Furthermore, in order to understand this layer, you need to understand the layer below (IP), so I explain both in the next section.
Network 3	packet	other machine	The Network layer is IP. Everything on the Internet pretty much focuses on the IP protocol. Even though there are 7 layers in the OSI model, the "first" layer is layer #3, the Network layer. Like the discussions above, the OSI has specified lots of useless features for the Network layer. Each layer was given to its own group to design, so each group designed into their layers pretty much all the features of the rest of the stack. Therefore, it is difficult to point to any particular networking feature and say "it belongs to layer #X". Luckily, TCP/IP is much more clean that way. Therefore, I will follow the standard practice of describing only the features that the TCP/IP protocol implements rather all possible features. The IP protocol is designed around the concept of an "unreliable datagram". Its one purpose is to get a packet of data from one machine to the destination machine all the way across the Internet. In order to do this, each machine is given an IP address: e.g. 192.0.2.14. The originating machine puts that address into a packet, then sends it to the nearest router. The router then looks at the IP address, and decides which direction it goes, and forwards it to the next router in that direction. The packet travels from hop-to-hop through the Internet until it reaches its destination. The whole process takes, on average, about a tenth of a second. You can visualize this as being just like normal mail. You write a letter, wrap it in an envelope, then address it. You stick the envelope in the mailbox and somehow it disappears, gets routed from hop-to-hop through the postal network, and eventually ends up at its destination. The important concept to remember is "unreliable": letters get lost in the mail. Roughly 1 out of every 100 IP datagrams gets lost on the Internet, sometimes more, sometimes less. This is where TCP (from the Transport layer above) steps in: it keeps track of all the datagrams going back and forth, and if one is lost, it automatically retransmits it. In order for TCP to provide this reliability, it must create a "connection". In other words, before two programs can talk to

			each other across the Internet, they must first establish a TCP connection through a process known as a "handshake". Whenever one machine sends data to the other, the receiver must send back an acknowledgement so that the sender knew it arrived. Likewise, IP itself requires no connection; it is "connectionless". In fact, many protocols who don't want the complexity of TCP choose bypass it by using UDP on top of IP: UDP is essentially the same as TCP, but doesn't acknowledge packets and doesn't require a connection. A good example of this usage is an application like IP Phone: it doesn't matter if a bit of data is lost here and there (cell phones do it all the time). Most of the time you won't notice a single lost packet, and if multiple packets are lost in succession, you can simply ask the other person to repeat what they said. In fact, TCP is very bad for this application: whenever a packet is lost, transmission halts until both sides are caught up again. In an IP Phone conversation, this means you might hear a pause, then the other person's delayed comments that need to be replayed very fast in order to catch up. For this reason, "real time" applications like audio, video, and games prefer UDP over TCP. Finally, the IP address will get your data as far as the destination machine, but many programs on that machine may be waiting for data to arrive. How does that machine know to which application that belongs? The Transport layer (TCP and UDP) contain their own unique addresses for each program on that machine. Each program is bound to a different "port" number. For example, imagine a machine with two web services running on it. You get to the different web services by specifying different ports in the URL. the URL http://www.example.com:80/index.html will get to the web service running at port 80, and http://www.example.com:81/index.html will get to the web service running at port 81. Thus, these URLs will return different web pages, depending on which service responded.
Data Link 2	frame	next hop	The most important concept to remember about the Data Link layer is "next hop". It's only purpose is to connect two machines together: you machine and the nearest router, or two routers. This is the "Link" component in the diagram above. In other words, on an Ethernet wire, your machine wraps the IP packet with Ethernet information, sends it to the first router. That router than strips off the Ethernet header and forgets about it. The router then decides which direction to foward the packet, the wraps it with the Data Link framing information to go across that wire to the next router. A machine might have both an Ethernet "MAC" address and an IP address. The Ethernet is the Data Link layer, and the MAC address is only visible locally, and is used by the local router in order to figure out how to send incoming traffic to you (vs. anybody else sharing the same Ethernet wire). Conversely, the IP address is global. If somebody in Siberia sends you traffic, they will use your IP address.
Physical 1	bit	wire	The Physical layer simply gets the bits out onto the wire. Different wires require different ways of encoding the bits. A telephone modem, for example, converts the bits into sounds patterns that go across the telephone wire. Ethernet, on the other hand, converts the bits into a series of high/low voltage

			levels.

People often have trouble understanding all these concepts. I like to summarize as follows:

- The Physical layer (1) sends bits onto the wire.
- The Data Link layer (2/Ethernet/PPP) sends frames as far as the next hop.
- The Network layer (3/IP) sends packets as far as the destination machine across the Internet.
- The Transport layer (4/TCP) creates connections to the program on that destination machine.
- The Application layer (7/HTTP/SMTP/POP/IMAP) communicates the received information (such as files) to the user.
- Forget about the Session layer (5) and Presentation layer (6).

A more laid-back approach can be found at: http://www.europa.com/~dogman/osi/

5.5 What is a packet?

To truely understand this answer, you must read sections section 5.1 and section 5.4 above. However, this is a common question, so I'll introduce the concept briefly here.

All down that is transfered on the Internet is packaged in individual units known as "packets". It takes between 30 and 50 packets for this document to be transfered to your computer, for example. Each packet is labeled with an "IP address" that specifies its destination.

The trick is that everything that is sent by the computer needs to be broken down into these packets. For example, if you listen to Internet "radio" to a streaming broadcast, it appears to you as one continuous stream, but in reality the transmitter is breaking the data down into individual packets, then your machine is reassembling them back into a stream.

The entire effort of sniffing consists of looking at either the individual packets, the reassembled data, or the sifted information (like passwords) out of the reassembled data.

5.6 What is a the TCP three-way handshake?

TCP is a "connection-oriented" protocol. This means that before you send data across it, you must first establish the connection. In English terms, the TCP Three-Way Handshake (TWHS) is the following:

1. I would like to talk to you
2. Sure, let's talk
3. Thanks

TCP is a "reliable" service, meaning that everthing has to be acknowledged in order to verify that it was received correctly. Similar protocols to TCP use 2, 3, or 4 packet handshakes in order to setup the connection. An amazing amount of work went into choosing the optimal 3 packet exchange.

In TCP-speak, we view this exchange as :

1. SYN
2. SYN-ACK
3. ACK

Where "SYN" is a flag in the TCP header that means "let's start talking", and which only occurs in the first two packets in the exchange. The "ACK" field means that the "acknowledgement" field is valid.

TCP has this interesting concept that every packet acknowledges receipt of data in the other direction. So if I send you one packet and you send me five in response, then each of your five responses acknowledge that you have received my one packet. This means that *all* TCP packets have the ACK bit set, except for the first one (because there's nothing to acknowledge). Note: most firewalls that block incoming TCP connections really just block packets without the ACK bit set; this means TCP traffic with that bit can still make it through the firewall to "ping" yo

All connection oriented protocols have "sequence numbers" which help order the data in the correct sequence, and acknowledge how far along in the sequence you've received the data. Most connection-oriented protocols beging their sequence numbers at zero, but TCP has this weird concept of starting them at a random number. Therefore, the TWHS looks something like:

Flags	*seq*	*ack*
SYN	102723769	0
SYN-ACK	1527857206	102723770
ACK	102723770	1527857207

Both sides tell the other side what sequence numbers will be used in the connection. The first data sent in either direction will use these sequence numbers.

Another thing to remember is that a TCP connection occurs from the port on one machine to the port on another. For example, a web server typically runs at port 80, and client ports are allocated starting at port 1024. Therefore, we might expand our example to show this:

Flags	*src*	*dst*	*seq*	*ack*
SYN	1037	80	102723769	0
SYN-ACK	80	1037	1527857206	102723770
ACK	1037	80	102723770	1527857207

This explanation is just an overview provided as an alternative way of looking at the subject. I strongly recommend you grab a book on TCP/IP, or lookup "three way handshake" on the web.

6. What is RMON?

RMON (Remote MONitoring) is an SNMP-based standard that allows management of network traffic.

If you'll recall, SNMP is the standard way of remotely managing devices. In a typical network, you have routers, hubs, switches, backup power supplies, servers, mail gateways, and so forth. In a modern network, all these devices can be remotely managed with a centralized console. The console sends SNMP commands to monitor their status, to reconfigure them, and receive alerts from them.

Of course, each of these devices accepts different commands and support different parameters that can be monitored. For a router, you might be concerned with the rate that packets are being forwarded. For a hub, you might want to monitor for any cabling faults on the ports. For backup power supply, you will want to monitor the voltage of the power being supplied. The collection of monitorable/changeable parameters are stored in a virtual database called a MIB (Management Information Base).

RMON is just another SNMP MIB. The item that this MIB manages is the traffic on the wire. In other words, we aren't talking about managing a real thing, but a virtual device.

6.1 What does "9-group RMON" refer to?

The original RMON standard specified 9 groups or sub-MIBs. A device could claim to support RMON if it supported only one of those groups. Many products have come out that support only the least resource intensive RMON groups. In order to differentiate themselves, full RMON products have taken to calling themselves "full 9-group RMON".

The 9 groups defined by the original RMON specification are:

Statistics
: This contains your basic Ethernet statistics. For the most part, these statistics are only relevant for promiscuous sniffing devices. In the old days when RMON was developed, most Ethernet consisted of a single coax wire. This group was essentially a MIB for the wire itself: it monitored the load on the wire (bytes/second, frames/second), health of the wire (CRC-errors/cable-faults), and simple traffic profiling.

History
: The golden rule of the SNMP standard is that no MIB should contain any complexity that could otherwise be accomplished by a management console. The idea was that an SNMP should be a small addition to any device. This means that other SNMP MIBs do not contain history/trending/logging mechanisms because they can take up lots of memory in order to store all that data. Such activity can be better done by the console simply by polling the MIB on a regular basis, then storing the results on the console. RMON broke the mold because it was the

only MIB designed for itself. In all other cases, SNMP MIBs were added to existing equipment. In the case of RMON, equipment was built expressly to host the RMON MIB. For this reason, RMON added resource intensive items to its MIB.

The "History" group is simply the "Statistics" group, but with the ability to keep track of the variables over time. One typical thing you might do is tell the RMON box to copy the Statistics group every hour and store the samples away.

Host/HostTopN/Matrix

These three groups are better discussed together. They all refer to the MAC addresses seen on the wire. Every Ethernet frame starts with a destination and source MAC address identifying to whom the frame was sent, and who sent it.

The Host group monitors the traffic that each MAC address sends/receives on the wire (bytes sent/received, frames sent/received). The HostTopN group serves as a sort of History mechanism to the Host MIB. Keeping a history of all the data would take too much memory, so instead it would allow you to do historical summaries of only the most active hosts. For example, every day you might store only the 10 most active hosts.

The Matrix MIB keeps a table of all the source/destination pairs. You can think of this in a number of ways. For any host, you can identify who that host is talking to. Likewise, you can identify who is talking to that host. This MIB is extremely resource intensive.

Note that in this discussion, the word "host" means essentially "MAC address". In a normal routed TCP/IP environment, this means that you will generally see that everyone is talking to the router. Later versions of RMON extended the host concept to the IP address layer; described below.

Filter/Capture

These two groups provide the capabilities necessary to remotely sniff traffic from the wire.

The Filter group allows you to specify filters. The most common use for this is when you want to sniff a single host's traffic, or if you are sniffing for a certain protocol. The Capture group is where the sniffed traffic is placed. The management console tells the capture group to create a buffer, then turns on sniffing, then downloads the frames to the console for decoding.

Alarm/Event

An Alarm could be triggered according to other values in the MIB. For example, you could tell it to alarm whenever traffic exceeded a certain threshold. Once an alarm triggered, either a Trap could be sent to the console, or the alarm could be logged in the Event group for later retrieval.

Alarming within RMON was extraordinarily sophisticated. For example, you could set a Filter to scan for a certain pattern within network traffic and trigger an Alarm.

6.2 What kinds of products support RMON?

Originally, RMON was conceived as a remote sniffer. The idea was to create a stand-alone box (called a "probe") that you would attach to an Ethernet segment. While you can still purchase such probes, you are more likely to experience RMON as an add-on product to hubs, switches, and routers.

6.3 What is RMONv2?

Version 2 of RMON extended the traffic-management capabilities up the protocol stack. For example, the Host/HostTopN/Matrix groups were extended to the OSI Network Layer (i.e. IP) and the OSI Application Layer. This required the ability to track the Network and Application Layer protocols as well.

6.4 What is the difference between RMON and SNMP?

This question comes up rather often. The difference should be rather obvious - RMON is just one of the 100 standard MIBs defined by the IETF, and management consoles use SNMP to talk to the MIB.

However, the source of this confusion is that RMON wasn't your average MIB. It broke the mold of how people viewed SNMP. Before RMON, SNMP was designed to be a simple/lightweight protocol. The idea was that you would add this tiny little SNMP agent to all your networking equipment, and that this agent would have minimal impact on the equipment. RMON introduced new ways of thinking where the agent could be much more resource intensive. This came from the fact that RMON wasn't added to existing equipment, but instead existing equipment was designed to run RMON.

Over time, people found many of the RMON concepts useful. They extended into later MIBs and versions of SNMP. At the same time, people imbedded RMON into existing equipment like hobs, switches, and routers. In fact, the original coax Ethernet that RMON was designed for doesn't exist today; yet RMON is still a thriving/evolving standard.

6.5 Does RMON function as a remote sniffer?

In a pinch, RMON can be used to remotely sniff traffic. The problem is that it isn't very good at it. For example, with today's sniffing products, you can easily do a packet capture of all traffic and save it real time to gigabyte disk drives. If you try to do that with RMON, you'll find difficulties in trying to transfer all that data back to your management console. One way to reduce this data is to set packet capture filters. However, filtering is extremely weak in RMON. If you want to do sniffing, don't rely upon RMON to do it for you.

On the flip side, there is a big security concern that RMON is so pervasive on equipment throughout the Internet that you can often find open RMON probes that will allow remote sniffing. Again, it won't provide very good sniffing capabilities, but it will be good enough.

6.6 What are the limitations of RMON?

First of all, RMON suffers from the basic problems of SNMP. SNMP has always been a "checkbox" item that customers always want in their products, but which vendors don't spend a lot of time on. Thus, basic activities work within SNMP, but it never quite reaches the hype. Likewise, while there are a ton of products out there that support RMON, they don't always work correctly. Indeed, since RMON is so resource intensive, you will often find that heavy use of RMON crashes the remote device.

Second of all, RMON is extremely resource intensive. In RMONv2, the network-layer matrix table can quickly fill up after a few moments attached to the network.

A. More info

Here are some useful references:

Douglas E. Comer's NETBOOK
: A good introduction to protocols, a book that is available on the web. http://www.netbook.cs.purdue.edu/index.htm

Laura Chappell's Introduction to Network Analysis
: http://www.podbooks.com

Mark Miller's publications
: Especially the "LAN Protocol Handbook". http://www.diginet.com/publications.html

X. Glossary

Berkeley Packet Filter (BPF)
: See BPF

BPF (Berkeley Packet Filter)
: A generic system, originally for BSD-based UNIX, that provides network packet capture capabilities. See also DLPI, NIT.

Data Link Provider Interface (DLPI)
: See DLPI

DLPI (Data Link Provider Interface)
: A built-in subsystem within Solaris (and other System V UNIX systems) for packet capture. See also See also BPF, NIT.

Network Interface Tap (NIT)
: See NIT

NIT (Network Interface Tap)
The system for SunOS 4 that allows monitoring of packets on the wire. Replaced by DLPI on Solaris/SuOS 5.

packet
See packet.

[fin]