HADMÉRNÖK

VIII. Évfolyam 2. szám - 2013. június

Dániel Nagy
nagy.daniel@operculum.hu

DETECTING SNIFFERS ON AN ETHERNET NETWORK PART II. – THE EXPERIMENTS

Abstract

Sniffing detection on an Ethernet network can be carried out by sending forged frames and then listen for the replies. The method relies on the fact that the sniffing host's NIC must be in promiscuous mode. This way it may answer irregular Ethernet frames. This seems to be a very straightforward and undertakeable approach for detecting sniffers. Practice, however, can be different from theory. This is the second and last part of two articles, where I present my experiments. I have developed a special software tool called mySnifferDetector, which is capable of sending out special purpose built frames (forged frames) to the Ethernet network, and then listens for the replies. By sending out these forged frames to a range of IP address, a sniffing host might be detected.

Ethernet hálózaton a sniffer detektálás irreguláris Ethernet keretek kiküldésével, majd az azokra érkező válaszüzenetek kiértékelésével lehetséges. A sniffelő számítógép hálózati interfésze ilyen esetben promiscuous módban kell legyen, ami azt eredményezi, hogy irreguláris Ethernet keretekre is válaszolhat. Ez egy igen egyszerű és megvalósítható sniffer detektálást valószínűsít. A gyakorlat azonban rácáfolhat erre a feltételezésre. Jelen cikk a második része egy kétrészes sorozatnak, amelyben bemutatom a témában elvégzett kísérleteimet. A cél érdekében fejlesztetem egy sniffer detektálásra alkalmas toolt, a mySnifferDetectort. A tool képes irreguláris Ethernet keretek kiküldésére, majd az ezekre érkező esetleges válaszokon alapulva a sniffelés tényét megállapítani. A módosított csomagok egy teljes IP tartományra kiküldhetők, így a hálózaton lévő sniffer elméletileg felderíthető.

Keywords: *sniffing detection, Ethernet, OSI Level2, secrecy, privacy ~ hálózati forgalom figyelés, Ethernet, OSI adatkapcsolati retag, biztonság, titkosság*

INTRODUCTION

The objective of these experiments are to see mySnifferDetector in action, and repeat some of the experiments of AbdelallahElhadj et.al. (2002) [1] and Mumatz & Yasir (2008) [2] and Sanai (2001) [3]. I was also interested in the differences between a shared medium and a switched Ethernet environment.

It is clear in order to get a close-to-complete result of the topic a more elaborate series of tests should be set up with more operating systems to be tested. A series of tests of such complexity were beyond the scope of my research. My idea was if I found only one wide-spread operating system that would slip through the ping and ARP methods, the method itself can be considered as useless.

THE TESTING ENVIRONMENT

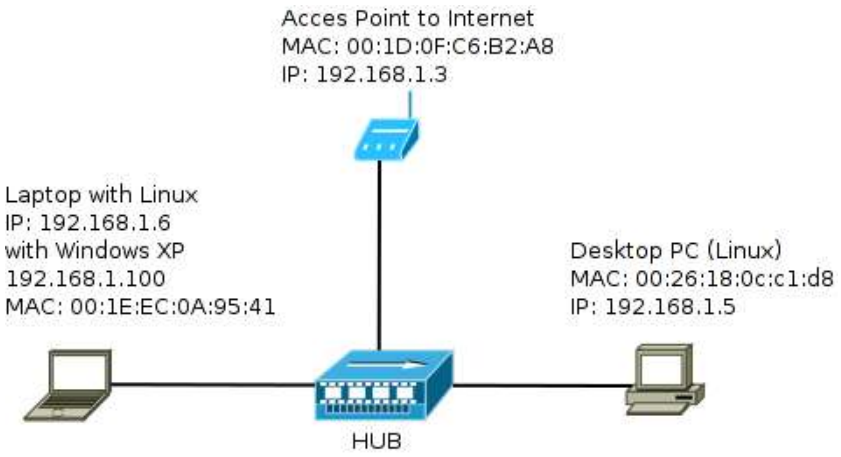
I performed the tests in the following environment.

Two personal computers were used. A laptop computer which is referred as the 'laptop' in this document, and a desktop PC which is referred as the 'desktop'. Both computers are connected by wired Ethernet. The laptop's built-in WiFi interface was shut down.

On both the desktop and laptop computers Debian 6 (Squeeze) was running with most updated packages from the testing repositories. The kernel version when experiments were performed was 3.2.0-3-amd64 (Debian official build) in testing. The laptop had a dual boot configuration. When the tests were performed with Windows as a sniffing host, the laptop was running Windows XP SP3 with the most recent automatic updates.

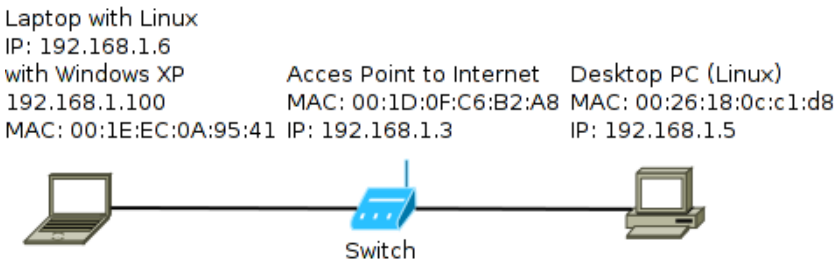
On the laptop Wireshark [4] was installed, for Linux and Windows equally with version 1.8.2. I used it to put the NIC of the sniffer into promiscuous mode, and also of course for monitoring traffic. On the desktop mySnifferDetector was run. mySnifferDetector is described in the upcoming paragraphs in more detail.

I used two different network topologies. One for the shared medium network environment (without a switch) and another for the switched environment (with a switch). For the shared medium tests I connected the laptop, the desktop and my access point to the HUB.



1. figure. The shared medium testing environment I used
(source: illustration by the author)

For the switched environment tests I connected the laptop and the desktop to the access point. (I did not use the HUB in switched Ethernet tests.) The access point has an Ethernet switch built into it, so the traffic was switched by the access point. This fact was later tested with the initial tests.



2. figure. The switched testing environment I used
(source: illustration by the author)

MYSNIFFERDETECTOR

„mySnifferDetector" is a small experimental program for detecting promiscuous hosts on a shared medium Ethernet network. It uses two promiscuous mode detection techniques: the ping method and the ARP request method. Both of these methods are based on sending out forged Ethernet frames and then listening for the replies. A forged frame is an Ethernet frame which is not assembled by the networking program of the operating system as it normally would be, but by the user space application. This way it is possible to create Ethernet frames that would not normally exist and send them out to the network.

If a host has its NIC in promiscuous mode (considered as a sniffer), the operating system of that host will receive every packet, not only the ones that are meant for it. By forging frames with a faked MAC address field, the operating system of the sniffer might answer to this packets, and this way a sniffer can be detected.

After starting, mySnifferDetector sends out the forged frames for the first specified test (normally the ping test) to every possible IP address on the subnet to which the PC is connected.

When the sending of the frames is finished (this is minimum one packet and at most the maximum number of possible IP addresses on the subnet), mySnifferDetector will examine the network buffer provided by libpcap for possible responses. The filter for this buffer is set for the current test performed. If packets can be retrieved from the buffer by the filter this means that one or more host(s) answered for the forged packets, and so their interface is in promiscuous mode. These packets are listed after each test, by their source IP and MAC addresses, the timestamp when the packet was received and the ethertype of the packet.

The program is not likely to give false positives. If a host answers to one of these packets its NIC is in promiscuous mode. However, this fact does not necessary mean that the host is a malicious sniffer. On the other hand, it is not complicated to alter a host in a way that makes it undetectable by these methods. Then again, if someone just fires up a Wireshark in promiscuous mode on a network under Linux, this program will detect it with a high probability.

mySnifferDetector needs be run from the command line, with at least one command line option which specifies the tests to perform. The program has no default tests to run just by typing mySnifferDetector. This is for safety reasons to avoid running the program by mistake, and potentially cause some problem on the network.

The program can be started with different command line parameters. The parameters make it possible to set the method which the application will use, setting output verbosity, logfile path and many other options which will determine the generated network traffic and ultimately the behavior of the application. The used command line settings will be commented in the sections later in the article where they were actually used.

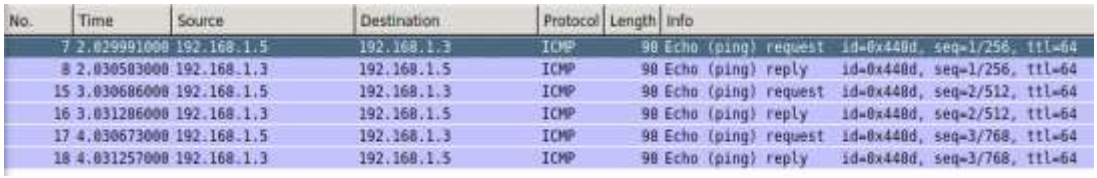
THE TESTS

First test: Checking the setup

The first test aimed to make sure, that the network setup worked and the laptop really was able to sniff the traffic on the segment. To achieve this I sent pings from the desktop to the access point. I put Wireshark on the laptop into promiscuous mode, and filtered for ICMP traffic.

```
root@bubugep:/home/bubu/Documents/mySnifferDetector# ping -c 3 192.168.1.3
PING 192.168.1.3 (192.168.1.3) 56(84) bytes of data.
64 bytes from 192.168.1.3: icmp_req=1 ttl=64 time=0.749 ms
64 bytes from 192.168.1.3: icmp_req=2 ttl=64 time=0.748 ms
64 bytes from 192.168.1.3: icmp_req=3 ttl=64 time=0.749 ms
```

As I expected, Wireshark on the laptop registered the traffic between desktop and the access point.



No.	Time	Source	Destination	Protocol	Length	Info
7	2.029991000	192.168.1.5	192.168.1.3	ICMP	98	Echo (ping) request id=0x440d, seq=1/256, ttl=64
8	2.030503000	192.168.1.3	192.168.1.5	ICMP	98	Echo (ping) reply id=0x440d, seq=1/256, ttl=64
15	3.030686000	192.168.1.5	192.168.1.3	ICMP	98	Echo (ping) request id=0x440d, seq=2/512, ttl=64
16	3.031206000	192.168.1.3	192.168.1.5	ICMP	98	Echo (ping) reply id=0x440d, seq=2/512, ttl=64
17	4.030673000	192.168.1.5	192.168.1.3	ICMP	98	Echo (ping) request id=0x440d, seq=3/768, ttl=64
18	4.031257000	192.168.1.3	192.168.1.5	ICMP	98	Echo (ping) reply id=0x440d, seq=3/768, ttl=64

3. figure. Wireshark [4] screenshot
(source: taken by the author)

Then I arranged for the switched environment to make sure that my access point is really a switch. I did the same pinging as above and watched Wireshark's output on the laptop. There was no traffic detected, so I concluded that my access point really did work as a switch as I expected.

In the first case the ping packages were physically transmitted on my whole network because a HUB just repeats every frame it receives to every port it has. That is why the laptop on the other segment was able to receive the ping traffic between the AP and the desktop. When using the AP only, there were no HUB. The AP was pinged by the desktop, and the AP switched these packets to the destination (itself) only.

Second test: Sending normal pings and ARP requests with mySnifferDetector

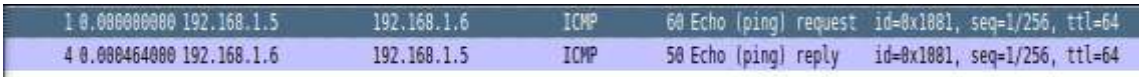
I set up mySnifferDetector to send out a ping (ICMP echo request) and an ARP request packet. I expected the laptop to answer as normal. I checked it in Wireshark (which was not in promiscuous mode this time) and by mySnifferDetector readings as well.

First the ping. With the command line options shown below, mySnifferDetector sent out one IMCP echo request (ping) to 192.168.1.6. (Note that mySnifferDetector by default forges the Ethernet destination address to FF:FF:FF:FF:FF:FE. With the --mac option, I changed this value to the laptop MAC address. This way, from the laptop's perspective these were legitimate ping and ARP packets.)

```
root@bubugep:/home/bubu/Documents/mySnifferDetector#  ./mySnifferDetector  --tests=ping  --
target=192.168.1.6 --mac=00:1E:EC:0A:95:41

--- Our host ---
Interface: eth1
IP: 192.168.1.5
MAC:00:26:18:0C:C1:D8
Netmask: 255.255.255.0
Max possible number of IP addresses on this network: 256
Packet sendout delay: 10 ms
Response collecting timeout: 500ms
Maximum timeouts: 2
Tests to be performed: ping
Sending pings: 1 / 1
--- Pingtest results ---
Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: IP at: 2012-09-07 14:22:03.457
-- The program has reached its end. --
```

As I expected, the laptop answered with an echo reply as it can be seen in the output of mySnifferDetector (after the row with 'Pingtest results' in it) and in Wireshark as well.



4. figure. Wireshark [4] screenshot
(source: taken by the author)

Please note, that from now on, I stripped down the header of mySnifferDetector's output and substituted with the "(...)" sign.

Second the ARP request. With this setting mySnifferDetector sent a normal ARP request to the broadcast Ethernet address to get the MAC address of 192.168.1.6.

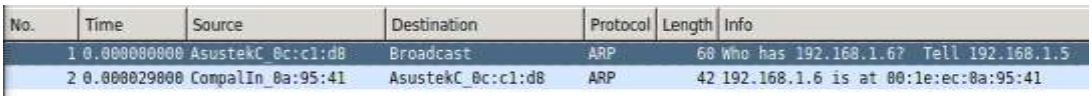
```
   root@bubugep:/home/bubu/Documents/mySnifferDetector#   ./mySnifferDetector   --tests=arp   --
target=192.168.1.6 -mac=FF:FF:FF:FF:FF:FF
   (...)
   Tests to be performed: arp

   Sending ARP requests: 1 / 1

   --- ARP Test results ---
   Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: ARP at: 2012-09-07 14:24:45.1

   -- The program has reached its end. --
```

The laptop sent back an ARP reply message as it can be seen in mySnifferDetector's output, and in Wireshark as well.



5. figure. Wireshark [4] screenshot
(source: taken by the author)

By the tests above I assured that mySnifferDetector works as expected, the network behavior is as expected and Wireshark displays what is expected.

Third test: Sending requests with different fake MAC addresses

In this test-sequence I repeated some of the experiments of AbdelallahElhadj et.al. (2002) [1] and Mumatz & Yasir (2008) [2] and Sanai (2001) [3] with mySnifferDetector. The objective of these tests is that what the results are with different fake MAC addresses. This time I omitted the '--target' option of mySnifferDetector which caused mySnifferDetector to attempt every IP address on the subnet. (This is the default procedure, when mySnifferDetector actually searches for sniffers.)

The idea was to find a MAC address that is rejected by the NIC in non promiscuous mode, but processed by the networking program of the kernel when the NIC is not in promiscuous mode. The difference between the two can be used to detect if the target host sniffs or not.

With the laptop in not promiscuous mode:

```
   root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --test=ping,arp --
mac=FF:FF:FF:FF:FF:FF
   (...)
   Sending pings: 256 / 256
   --- Pingtest results ---
   Reply from: IP:192.168.1.3 MAC: 00:1D:0F:C6:B2:A8 type: IP at: 2012-09-08 18:48:03.898
   Reply from: IP:192.168.1.3 MAC: 00:1D:0F:C6:B2:A8 type: IP at: 2012-09-08 18:48:03.912
   Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: IP at: 2012-09-08 18:48:03.927
   Reply from: IP:192.168.1.3 MAC: 00:1D:0F:C6:B2:A8 type: IP at: 2012-09-08 18:48:05.223

   Sending ARP requests: 256 / 256

   --- ARP Test results ---
   Reply from: IP:192.168.1.3 MAC: 00:1D:0F:C6:B2:A8 type: ARP at: 2012-09-08 18:48:06.245
   Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: ARP at: 2012-09-08 18:48:06.261
   -- The program has reached its end. --
```

The above is normal, since I used a legitimate broadcast address.

For the other tests I ran mySnifferDetector with the following command line arguments. This configuration made mySnifferDetector to send out the forged frames with different destination addresses as can be seen after the '--mac=' parameter.

I did not copy the results here for each test, I created a summary table of the tests and the results are below.

```
   root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --test=ping,arp --
mac=FF:FF:FF:FF:FF:FE
   root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --test=ping,arp --
mac=FF:FF:00:00:00:00
   root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --test=ping,arp --
mac=01:00:00:00:00:00
   root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --test=ping,arp --
mac=01:00:5E:00:00:00
```

Then I repeated the same as above, the only change was that I put the laptop into promiscuous mode by starting Wireshark on it.

I put the results into the table below.

P – reply for the ping test

A – reply fo rhe arp test

X – no reply

	Laptop		**Access Point**
MAC address	**not in promisc**	**in promisc**	**n.a.**
FF:FF:FF:FF:FF:FF	PA	PA	PA
FF:FF:FF:FF:FF:FE	X	PA	X
FF:FF:00:00:00:00	X	PA	X
01:00:00:00:00:00	X	PA	PA
01:00:5E:00:00:00	X	PA	PA

I rearranged my network topology again to a switched environment, and repeated all the tests:

	Laptop		**Access Point**
MAC address	**not in promisc**	**in promisc**	**n.a.**
FF:FF:FF:FF:FF:FF	PA	PA	PA
FF:FF:FF:FF:FF:FE	X	PA	X
FF:FF:00:00:00:00	X	PA	X
01:00:00:00:00:00	X	PA	PA
01:00:5E:00:00:00	X	PA	PA

Conclusions of the tests above

With Linux on the target machine the fake broadcast address FF:FF:FF:FF:FF:FE worked as a good value for the destination address, since the laptop did not answer to this when in not promiscuous mode, and answered in promiscuous mode. The same is true for all the other attempted MAC addresses, as long as it had the group bit set to 1.

We can conclude this version of the Linux kernel after receiving an Ethernet frame from the buffer of the NIC driver will not perform further processing of the frame. Simply extracts the IP packet encapsulated in it, and hands over to the upper layer of networking program. These upper layer takes the IP packet only, processes it, and acts accordingly.

The tests gave identical results in switched environment as well. This meant that the access point broadcasted all the above destination addresses, otherwise the frames would have not reached the laptop, since all the used Ethernet destination addresses were not existent on the network.

Fourth test: Sniffer detecting with Windows XP as a target system

I repeated the test-sequence on the same laptop, but with Windows XP operating system on it in shared medium environment (connected through the HUB).

	Laptop	
MAC address	**not in promisc**	**in promisc**
FF:FF:FF:FF:FF:FF	A	A
FF:FF:FF:FF:FF:FE	A	A
FF:FF:00:00:00:00	A	A
01:00:00:00:00:00	X	X
01:00:5E:00:00:00	X	X

Surprisingly the results were completely different to those with Linux. Nothing that was true for Linux was true for Windows XP. My results even oppose the ones found in my source materials of Sanai (2001) [3].

The reason must be that since the time when Sanai [3] performed these tests, Windows kernel was different. (Sanai [3] did not test with XP though.) I can not come up with an explanation for this behavior any further than this, since it would require the source code of Windows or carefully designed test-sequences to reverse engineer the behavior of Windows networking. This is out of the scope of this paper.

Nevertheless this clearly shows that although ping and ARP methods work well when detecting a sniffer running on Linux machine, they will not work on Windows XP. Windows will do 'some' further processing of the Ethernet frame.

The ultimate test: Does mySnifferDetector find the sniffer?

Lastly, I performed two tests to prove that mySnifferDetector does indeed detect a host with its interface in promiscuous mode on a shared medium network. The same result can be seen in the former tests, but as a final conclusion I inserted the full readings here. According to the differences between Linux and Windows, these tests work only on Linux.

First I started mySnifferDetector with the default settings while laptop was in not promiscuous mode:

```
root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --tests=arp,ping

--- Our host ---
Interface: eth1
IP: 192.168.1.5
MAC:00:26:18:0C:C1:D8
Netmask: 255.255.255.0
Max possible number of IP addresses on this network: 256
Packet sendout delay: 10 ms
Response collecting timeout: 500ms
Maximum timeouts: 2
Tests to be performed: ping arp
Sending pings: 256 / 256
--- Pingtest results ---
No replies.
Sending ARP requests: 256 / 256
--- ARP Test results ---
No replies.
-- The program has reached its end. --
```

The program did not list any replies which meant that there were no answers for the ping and ARP tests.

In the second test the only change is that the sniffer's had Wireshark running on it, with the NIC put into promiscuous mode.

```
root@bubugep:/home/bubu/Documents/mySnifferDetector# ./mySnifferDetector --tests=arp,ping
--- Our host ---
Interface: eth1
IP: 192.168.1.5
MAC:00:26:18:0C:C1:D8
Netmask: 255.255.255.0
Max possible number of IP addresses on this network: 256
Packet sendout delay: 10 ms
Response collecting timeout: 500ms
Maximum timeouts: 2
Tests to be performed: ping arp
Sending pings: 256 / 256
--- Pingtest results ---
Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: IP at: 2012-09-07 15:09:29.874
Sending ARP requests: 256 / 256
--- ARP Test results ---
Reply from: IP:192.168.1.6 MAC: 00:1E:EC:0A:95:41 type: ARP at: 2012-09-07 15:09:33.484
-- The program has reached its end. --
```

Both pingtest and ARP tests gave result. The laptop's IP address and MAC address are displayed. mySnifferDetector successfully detected a sniffer on the network!

CONCLUSIONS

Ethernet networks are widespread and can be easily eavesdropped a.k.a. sniffed. Some of these sniffers can be detected by sending out forged frames to the network and then waiting for the replies. These OSI Level 2 manipulations can force the sniffing host to send replies for the invalid frames, and thus reveal itself. The theory is only true, when the sniffer runs on an ordinary PC with a stock kernel. With modified kernels and/or networking cards a sniffer can be made totally unnoticeable on the network. Nevertheless, the research focused on sniffing host with a normally working network subsystem.

However in practice it turned out that the behavior of systems on receiving such frames varies. Promiscuous mode NIC (which is essential for sniffing) can not be reliably detected, because the networking subsystem reacts different from OS to OS, even from kernel version to kernel version. Some of my tests contradicted the results of the referenced source materials with different kernel versions. My results also showed that the ping and ARP methods are unreliable since the results with Linux and Windows were different.

I concluded that these sniffer detection techniques are very unreliable if not useless in practice. They depend on network environment, operating system and driver. It is even a bigger problem that negative cases can not be distinguished from uncertain cases. In practice we do not know, what operating system a sniffer uses. The bottom line conclusion is that forged frame based sniffer detection is rather a theory, no use for them in the practice.

If sniffing is a concern on a network (and it always should be a concern), it should be counteracted with encryption of the network traffic, rather than detecting sniffers on the network, and then being in a false feel of security if such tools would not report any.

References

[1] H. AbdelallahElhadj , H. M. Khelalfa & H. M. Kortebi (2002) 'An Experimental Sniffer Detector: SnifferWall' SEcurite des Communications sur Internet pp.69-80

[2] Mumtaz AL-Mukhtar, Yasir Ahmed Abdullah (2008) 'Developing a Sniffer Detector for Windows Operating Systems' The 1stRegional Conference of Eng. Sci. NUCEJ Spatial ISSUE 11 (1) pp84-90

[3] Sanai, D. (2001) Detection of Promiscuous Nodes Using ARP Packets. [Online] Available at: www.securityfriday.com/promiscuous_detection_01.pdf [Accessed: 10.09.2012.]

[4] Wireshark Wiki (n.d.) Ethernet capture setup. [Online] Available at: http://wiki.wireshark.org/FrontPage [Accessed: 10.09.2012.]