Analysis of the Internet Worm of August 2003

S. Hussain Y. Hassan K. El-Badawi K. Salah
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
Email: {saad80,yshassan,elbadawi,,salah}@ccse.kfupm.edu.sa

Abstract

In August 2003, the W32.Blaster worm spread all over the world affecting numerous systems and networks causing severe damages and disruptions. According to government and industry experts the August 14th electrical power blackout in North America was linked to the blaster worm. In this paper, an analysis of the Blaster worm is presented. The worm's methods of exploitation and propagation will be exposed. In particular, we will expose the buffer overflow vulnerability which was exploited by the worm. Buffer overflow vulnerability is accounted for 50% of the internet attacks. In addition, fundamental concepts relating to techniques, protocols, services, and applications utilized by the worm will also be explained. These include RPC, TCP, DCOM, and port scanning.

KEYWORDS: Security, Worms, Blaster Worm, Buffer Overflow, RPC, Port Scanning

1. Introduction

The analysis of the infamous MSBlast, also known as “Blaster” is of great importance because it succeeded in penetrating its way into a far greater number of computers than previously known. The worm, which began to wreak havoc in August 2003, has infected almost every Internet user in one way or the other. More than 16 million of the systems that connected to Microsoft's Windows Update service were found to be infected with MSBlast and were offered a patch [1].

On July 16, 2003 Microsoft released a security bulletin, MS03-026 [2], alerting all users of Microsoft Operating Systems of a critical buffer overrun in the RPC interface allowing attackers to execute code of their choice on a remote machine. This vulnerability was assigned the name CAN-2003-0352 by the Common Vulnerabilities and Exposures project [3]. On August 11, 2003, the Blaster worm was released that exploited the vulnerability that was addressed by Microsoft Security Bulletin MS03-026 (823980) to spread itself over networks by using open RPC ports on computers that are running most of the versions of

Windows operating systems. Users of all versions of Windows NT 4.0, Windows 2000, Windows XP, and Windows 2003 were advised to apply the published system patch released earlier on July 16, 2003 immediately.

The Blaster worm may also have contributed to the cascading effect of the August 14, 2003 blackout in which major losses of electric load occurred in the northeastern U.S., the Eastern Interconnection and Canada [4,5]. It is anticipated that on the day of the blackout, the Blaster might have degraded the performance of several communications lines which linked key data centers used by utility companies to manage the power grid. The inability of critical control data to be exchanged quickly across the grid could have hampered the operators' ability to prevent the cascading effect of the blackout and to restore power in a timely manner.

The Blaster is an example of an increasingly dangerous type of computer virus known as a "*blended threat*" [6]. Blended threats have the combined characteristics of viruses, worms, Trojan horses, and other malicious code and thus take advantage of vulnerabilities in operating systems and applications to initiate, transmit, and spread an attack. By combining multiple vectors and techniques, blended threats can spread rapidly and cause widespread damage. Effective protection from blended threats therefore requires a comprehensive security solution that contains multiple layers of defense and response mechanisms.

The rest of the paper is organized as follows. Section 2 gives adequate background on important topics and concepts to lay the ground for the advanced exploitation techniques used by the Blaster. Section 3 illustrates how the Blaster exploited the popular buffer-overflow vulnerability. Section 4 presents the propagation method. And finally, Section 5 has the conclusion.

2. Background

This section discussed the fundamental concepts and protocols. These concepts and protocols have been discussed with relation to the exploitation and propagation methods used by the blaster worm. These include TCP, RPC, and DCOM. In addition, the buffer overflow venerability exploited by the worm will also be discussed.

2.1 Transmission Control Protocol (TCP)

TCP [7] is a connection oriented transport layer protocol that requires a 3-way handshake to complete before the application layer conversation commences. The steps involved in this 3-way handshake are as follows:

1 The requesting host sends the receiving host a SYN packet. The SYN packet has the SYN, or synchronize flag set. The requesting host is asking to synchronize to communicate with the receiver.
2 Upon receipt, the receiving host will reply to the requesting host with a SYN/ACK packet. The SYN/ACK packet acknowledges (ACK's) the original SYN packet, and the SYN flag in the reply packet provides the second step in the synchronization between the two hosts.
3 The requesting host sends a final ACK (acknowledgement) packet to the receiving host to finalize the session establishment and finish coordinating the sequence numbers.

Top

Once the session is established actual data exchange begins. TCP uses sequence numbers to track receipt of packets and to be able to reorder the packets to recreate the original data. It also uses the sequence numbers to provide error checking and retransmit requests. In this regard TCP is a very reliable and robust protocol.

2.2 Remote Procedure Call (RPC)

The Remote Procedure Call (RPC) architecture allows a program running on one computer to access data and services on other machines in its local network. One of the goals of RPC is to make it easier for computer applications to communicate with other machines, without having to provide their own code for establishing and using low level network protocols. RPC [8] is a protocol/service that allows to make calls to program code running on remote hosts and to get the results of the function calls back without having to understand or write any special networking code. The RPC model consists of a client and a server. An RPC server is some code running on a host that waits for calls to come in so it can perform its function. Another variation of RPC is ORPC. ORPC is the Object RPC protocol. This protocol represents the integration of RPC and the DCOM protocols. ORPC is built on top normal RPC but adds additional information in the calls and responses to make it particularly useful with DCOM implementations. DCOM will be briefly explained in Section 2.3.

Figure 1 illustrates briefly a typical RPC communication between two endpoints. When a program on a server starts it must inform the server's RPC run time library of the exact interfaces being used. This process is called *registering the program's interfaces*. Once this is completed the server program and the run time library create bindings on these interfaces, hence preventing any other program from using them. The server program then registers its listening channels or "*endpoints*". The *EndPoint Mapper*, also known as the Remote Procedure Call System Service (RPCSS) now knows what port the server program will be listening on.

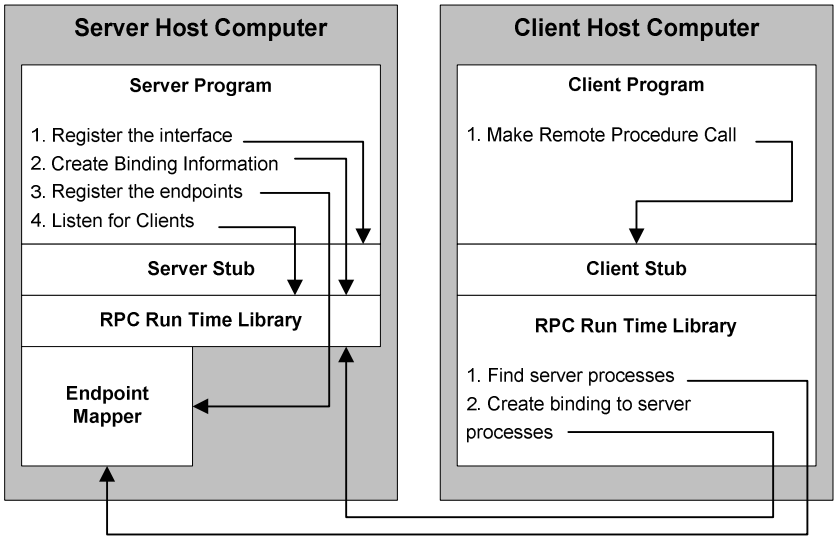

Figure 1. RPC communication between server and the client

Top

Now the program is properly initialized and ready to accept incoming client requests. Prior to obtaining services from a server, the client needs to find the correct machine that is providing the desired service. The client program makes a request for a specific service by creating a "*binding handle*". The run time library on the client will then locate the server that is providing the requested service. Once the server and service are located the clients RPC run time library will determine what port the service is listening on from the *EndPoint Mapper*. Once this is determined the clients RPC run time library directs the clients call to that port and the communication for the requested service can begin between the client and server.

Remote Procedure Calls are requests from clients needing to use a server service. The client asks the server to perform some function and return the results when done. Remote Procedure Calls are actually the client executing a function call with all the parameters the server will need to complete the operations. This function call is intercepted by the RPC service. RPC then packages or *marshals* the parameters and makes a remote call to the RPC service running on the remote host. The remote system RPC service converts the request to a call for the local function on the server. When the remote computer is done processing, it returns a value to the calling host through the RPC service. The original program on the client then continues exactly like executing a call to a local function or service. These steps are depicted in Figure 2.

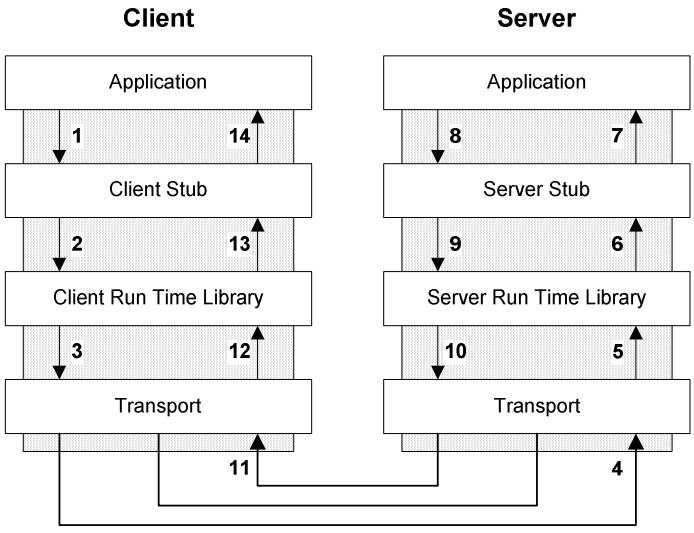

Figure 2. The flow of messages involving a remote procedure call

2.3 Distributed Component Object Model (DCOM)

The Distributed Component Object Model (DCOM) protocol uses RPC to enable software components to communicate over a network, providing an additional level of abstraction for interoperability in network environments. COM [9] is the Component Object Model. This programming model allows the creation and registration of special purpose software components that can be registered on a machine and then called by other programs when their special functions were needed. COM technology extends the concepts of software reuse and separation of functions in application design. DCOM simply takes this same concept and extends

Top

the component availability to objects that reside on remote hosts separated by a network connection. It does this by using special RPC calls to interact with COM objects residing on the remote hosts. As mentioned above, Microsoft implements DCE RPC standards for it's RPC functionality. It then extends the functionality of RPC by linking it tightly with DCOM. The Microsoft implementation of RPC uses some of the RPC fields for DCOM specific purposes.

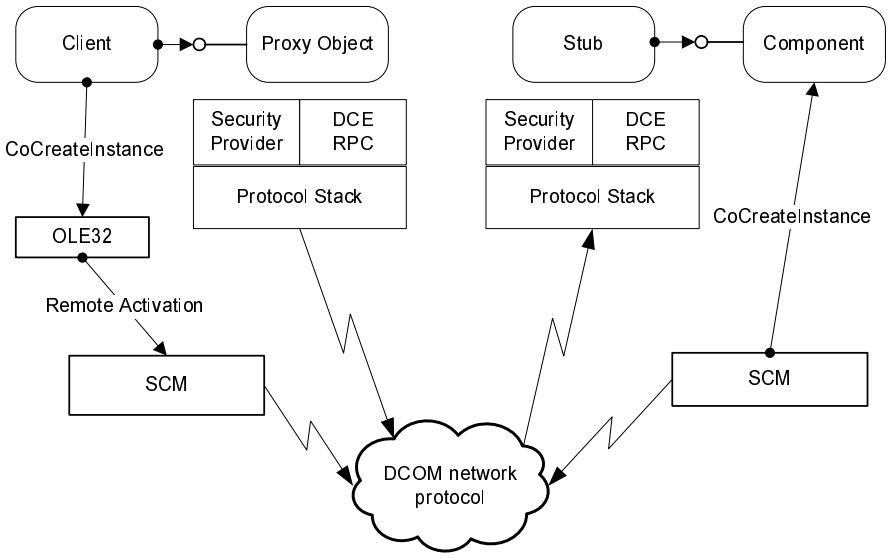

Figure 3. DCOM Architecture

DCOM, like RPC, is a client/server model. When a server component is developed, a unique identification number is assigned to it and it is registered on the system where it will reside. That same *Universally Unique ID (UUID)* number is also registered on the client computers that will use the services of the server component. When a call to a component of the type running on the server is executed, the DCOM layer makes a call via RPC to the system hosting the server component. If an instance of the object type requested already exists then a reference to it is passed back to the client and the client can work with the remote instance just as if it were running locally. If there is no instance on the server or a new instance is specifically requested, a new object is created and a reference to that newly instantiated object is passed back to the client. In both cases, it is clear that the object is remote and is logically transparent to the client using the objects services. Figure 3 shows the DCOM architecture adapted by most of the operating systems.

2.4 Buffer Overflow Attacks

This section describes the technicalities of the buffer overflow attack. The most dominant form of buffer overflow exploitation is stack smashing. A synthetic example [10] is used to explain the working of the attack. Figure 4 shows the number of security alerts reported by CERT/CC [13] between 1988 and April, 2004. Attacks that exploit buffer overflow vulnerabilities have accounted for approximately half of all the alerts after 2000.

Top

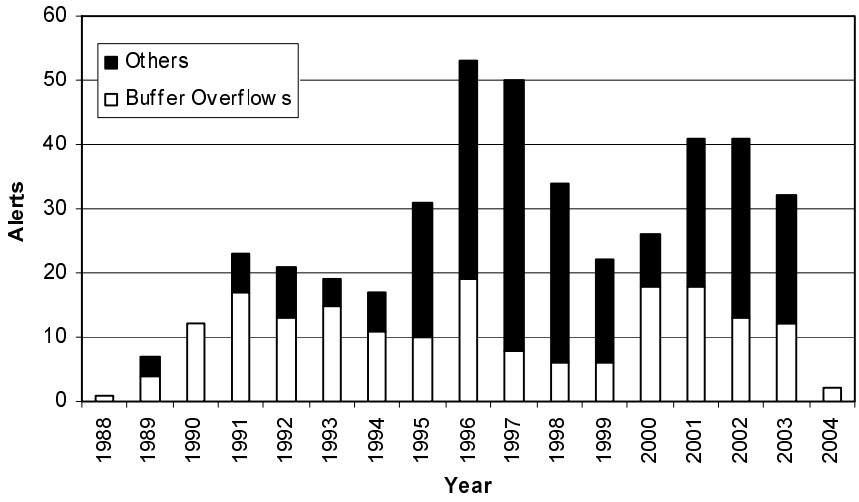

Figure 4. CERT Security alerts by years

A buffer overflow is an example of a blended attack. The origin of blended attacks began in November 1988, which was the year that the Morris worm was introduced [11]. The Morris worm exploited flaws in standard applications of BSD systems. Although the worm was not deliberately destructive, it overloaded and slowed down machines so much that it was very noticeable after repeated infections. Thirteen years later, in July, 2001, CodeRed repeated a very similar set of attacks against vulnerable versions of Microsoft Internet Information Server (IIS) systems [12]. Using a well-crafted buffer overflow technique, the worm executed copies of itself (depending upon its version) on Windows 2000 systems running vulnerable versions of Microsoft IIS. The slowdown effect was similar to that of the Morris worm.

Buffers are data storage areas, which generally hold a predefined amount of finite data. A buffer overflow occurs when a program attempts to store data into a buffer, where the data is larger than the size of the buffer. When the data exceeds the size of the buffer, the extra data overflows into adjacent memory locations, corrupting valid data and possibly changing the execution path and instructions. The ability to exploit a buffer overflow allows the injection of arbitrary code into the execution path. This arbitrary code enables remote, system-level access, thus giving unauthorized access not only to malicious hackers, but also to replicating malware. Buffer overflows are generally broken into multiple categories, based on both ease of exploitation and historical discovery of the technique. While there is no formal definition, buffer overflows are, by consensus, broken into three generations. First generation buffer overflows involve overwriting stack memory. The second generation overflows involve heaps, function pointers, and off-by-one exploits. And the third generation overflows involve format string attacks and vulnerabilities in heap structure management. The stack overflow method is discussed below as the Blaster worm uses it.

2.4.1. Overflowing the Stack

Before a program starts executing in the memory, operating system allocates memory space for it. This memory space is split into code segment (instructions that are to be executed by microprocessors), data

Top

segment (where data is kept) and stack segment (where variables and temporary data are kept). Following is an example of a C code making the buffer overflow.

```
int main()
{
      char Buffer[4];
      printf("%d",Add(5,6,7));
      return 0;
}

int Add(int a, int b, int c)
{
      a= a + b + c;
      return a;
}
```

A stack is a contiguous block of memory containing data and is used whenever a function call is made. A stack pointer (SP) points to the top of the stack. In the example above, program starts executing from main() block. When it calls Add function, following steps take place in the memory:

- Function parameters will be pushed onto memory from the right to the left (in this case 7, 6, 5 will be pushed onto stack) as shown in Figure 5.
- The return address is pushed onto the stack followed by a frame pointer (address to be executed after the function returns, in this case after executing Add function). A frame pointer is used to reference the local variables and function parameters because they are at a constant distance from the FP. In most computer architecture stacks grow from higher memory address to lower and data are copied from lower to high address. For example, if SP pointed to memory address `0x0000FFFF`, stack can hold 64-kilo bytes of data.

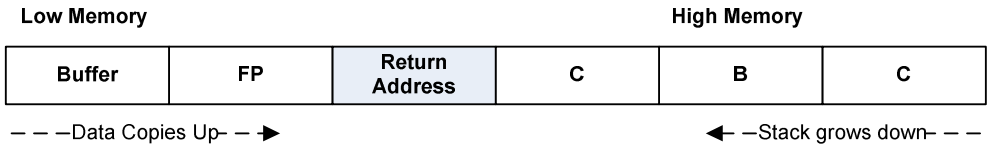

Figure 5. Memory view of the System

Skillful attackers can overwrite the return address with the address of their own program, so that it points back to buffer and executes the intended code and thus altering the processor's execution path. The root cause of buffer overflow problem is that C/C++ is inherently unsafe. There are no bound checks on array and pointer references, that means, developer has to check the bounds, moreover, a number of unsafe string operations also exist in the standard C library, i.e., `strcpy()`, `strcat()`, `sprintf()`, `gets()`. For example, the following program declares a buffer that is 256 bytes long. However, the program attempts to fill it with 512 bytes of the letter "A" (0x41).

Top

```
int i;
void function(void)
{
      char buffer[256]; // create a buffer
      for(i=0;i<512;i++) // iterate 512 times
      buffer[i]='A'; // copy the letter A
}
```

In an attempt to copy the character 'A' to the small buffer, the data exceeds the size of the buffer and overwrites adjacent areas of data stored on the stack including the return address. By overwriting the return address, an attacker can change what the program should execute next. Most commonly, attackers exploit buffer overflow to get an interactive session on the machine by specifying their own program to run next.

2.5 Port Scanning

Blaster propagates itself to other hosts by scanning the network for hosts that have port 135 open. Microsoft used port 135 as the RPC *EndPoint Mapper*. This port scanning [14] is implemented by using the `connect()` system call. Then a `select()` call can be issued to determine which hosts have responded. This is the most basic form of TCP scanning. The `connect()` system call provided by the operating system is used to open a connection to every interesting port on the machine. If the port is listening, `connect()` will succeed, otherwise the port isn't reachable. One strong advantage to this technique is that special privileges are not needed. Any user on most UNIX boxes can use this call. Another advantage is speed. While making a separate `connect()` call for every targeted port in a linear fashion would take very long over a slow connection, the scan can be speeded up by using many sockets in parallel. Using non-blocking I/O permits to set a low time-out period and watch all the sockets at once. The drawback of using this approach is that this sort of scan is easily detectable and filterable. The target hosts logs will show a bunch of connection and error messages for the services which take the connection and then have it immediately shutdown.

3. Blaster's Exploitation of Buffer Overflow Vulnerability

The Blaster worm [15-18] exploits the vulnerability in the part of RPC that deals with message exchange over TCP/IP. The exploit is successful because of incorrect handling of malformed messages. This particular vulnerability affects a Distributed Component Object Model (DCOM) interface with RPC, which listens on RPC-enabled ports. This interface handles DCOM object activation requests that are sent by client machines to the server. An attacker who successfully exploits this vulnerability would be able to run code with Local System privileges on an affected system. The attacker would be able to take any action on the system, including installing programs, viewing data, changing data, deleting data, or creating new accounts with full privileges.

To exploit this vulnerability, the attacker must be able to send a specially crafted request to port 135, port 139, port 445, or any other specifically configured RPC port on the remote computer. For intranet environments, these ports are typically accessible, but for Internet-connected computers, these ports are

typically blocked by a firewall. If these ports are not blocked, or if the target machines are in an intranet environment, the attacker does not have to have any additional privileges.

The Blaster worm takes advantage of a flaw in a low level function that extracts the server name from a UNC path that is passed to it. (A UNC path looks like \\computername\sharename\path\filename.ext) The function that is used as a pathway to this lower level function is CoGetInstanceFromFile. A specific buffer overflow is possible due to an unchecked parameter within a DCOM function CoGetInstanceFromFile.

```
HRESULT CoGetInstanceFromFile(
      COSERVERINFO * pServerInfo,
      CLSID * pclsid,
      IUnknown * punkOuter,
      DWORD dwClsCtx,
      DWORD grfMode,
      OLECHAR * szName,
      ULONG cmq,
      MULTI_QI * rgmqResults);
```

The `CoGetInstanceFromFile` function above is used to create a new object and initialize it from a file. This function contains a parameter of `szName` which is used to specify the file to be initialized. This parameter is of type `OLECHAR*` which is just another name for a `char*` (a pointer to a string). Normally, this would contain a legitimate path to a file that the caller would like a COM object instantiated from.

The mechanism of parsing the UNC string on the remote server determines the way the function is exploited. This parameter is allocated a value of 0x20 (32 bytes) for the filename but the input is not checked. As reported in [17] a potential bug is found in `GetMachineName` function. The logic of this code is as follows:

1. Allocate 0x20 (32 bytes) on the stack as a local buffer to hold the machine name in the UNC path. This should fit under normal circumstances since maximum machine name length is 16 characters and this function uses a Unicode encoding (2 bytes per character).
2. Start in the string where the server name should be and compare each character to 0x5c (the backslash character \). If the character is not a backslash, write it to the buffer allocated above, move the buffer pointer by one byte and move on to the next character in the UNC path string.
3. Repeat step 2 until the end of the string or a backslash character is reached.

The problem occurs when we pass a UNC path string that doesn't contain a "\" character within the first 32 bytes of the area where the server name should be. Since the logic of the program does not check that the machine name is the proper size, we can overflow the buffer and overwrite the stack with malicious string terminated by a backslash (\x5c) or with a null character as the function would not be able to parse properly. The service on the target system can be crashed by passing invalid values to the parameter. A string with the following characteristics is needed for this.

1. The string fills the buffer / stack space up to the function return pointer

Top

2. It overwrites the legitimate return pointer with a new one that points to the instructions inserted in step 3.
3. It inserts assembly byte code that when executed, causes the machine to open an instance of *cmd.exe* and bind it to a shell on port 4444/tcp.

If successful, the bound command shell will be running under "Local system"; the same security context as *rpcss.exe*. Thus, upon connection to the listening shell, the attacker will have complete control over the local system. This is the critical flaw in the DCOM RPC interface which allows the exploit to succeed. By inserting instructions into the data which is overflowed the exploit can cause the operating system to spawn a command shell listening on a specific port.

4. Propagation Method

In this section the method of propagation used by the Blaster worm is exposed. It consists of port scanning, client-server handshake, and the launch of the worm's executable code. Some common behavior and symptoms of the Blaster are also reported.

4.1. Port Scanning

The Blaster worm starts by scanning the network for systems listening on TCP port 135. The scanning is done by generating an IP address according to the following algorithm:

- The worm uses a 60/40 split to determine the target subnet. For 40% of the time, the generated IP address is of the form A.B.C.0, where A and B are the first two parts of the infected computer's IP address.
- C is also calculated by the third part of the infected system's IP address; however, 40% of the time the worm checks whether C is greater than 20. If so, a random value less than 20 is subtracted from C. Once this semi random IP address is calculated, the worm will attempt to find and exploit a computer with the IP address A.B.C.0.
- The worm will then increment the 0 part of the IP address by 1, attempting to find and exploit other computers based on the new IP address, until it reaches 254.
- Conversely if the remainder is less than 12, then the high 3 octets of the IP address are randomized. With a probability of 60%, the generated IP address is completely random.

4.2. Client-Server Handshake

The three-way Client-Server handshake takes place in the following steps:

1. After discovering a vulnerable machine that is listening on port 135, it connects to the TCP port 135 of that machine. Before the call to the DCOM instantiation function on the remote server can take place, the client makes an RPC bind to the remote server. This RPC protocol handshake is much an application layer version of the TCP handshake that takes place at the Transport layer of the network stack. One of the reasons this is done is because RPC has the ability to run over many different protocols and is independent of the services being run by the protocols below it. The string sent to

the attacking machine contains the beginning of a normal ORPC call to instantiate a remote object on the target server using a UNC specified file. The code takes this normal request and modifies it so that the server name is filled in with buffer filler and the shell code.

2. The first section of the shell code contains characters to fill the allocated server name buffer. Once this is completely full, the rest of this string is written over the contents of the stack starting with:
    a. The new return address (written over the real return address that should execute when the function exits)
    b. Pointers to space in the primary thread data block in memory
    c. A NOP sled
    d. The assembly byte code that binds the shell to port 4444/tcp
3. Hence the victim machine opens a command shell listening on TCP port 4444.
4. The attacking machine then connects to shell via port 4444 on the victim's machine. At this point the attacker has full system level privileges of the victim machine via a remote command line shell on port 4444. The entire process of victimizing a machine is depicted in Figure 6.

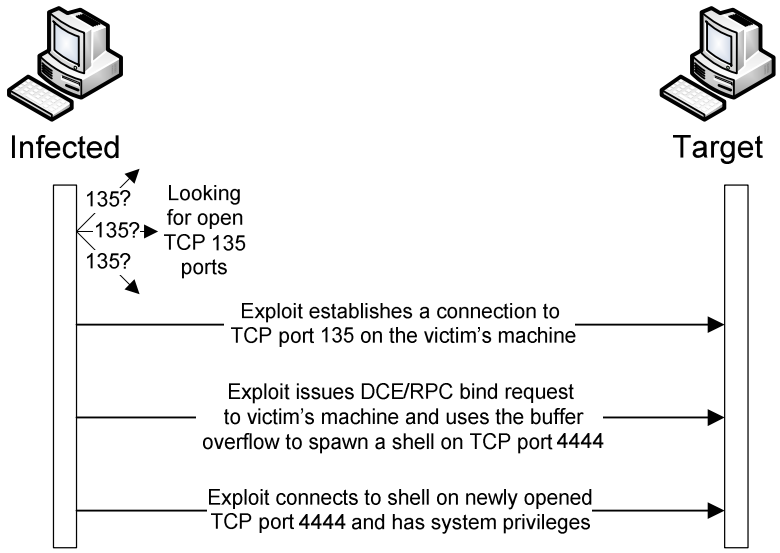

Figure 6. Looking for a victim machine

4.3. Launching of MSBLAST.EXE

Prior to the launching of the msblast.exe on the target machine, a particular file needs to be downloaded on the victim machine. That file is msblast.exe; compressed with UPX compression utility. The file is self-extracting and has a size of 11Kbyte once unpacked. Figure 7 describes the steps involved in launching the msblast.exe file, which can be summarized as follows:

1. The attacking machine then starts a TFTP server on UDP port 69 and issues a command to get the msblast.exe file through TFTP.
2. The victim machines then runs the worm code and closes the 4444 port.

Top

3. The worm code checks to see whether a computer is already infected and whether the worm is running. If so, the worm will not infect the computer a second time. Otherwise the victim machine gets infected and hence becomes an attacking node.
4. At this point both machines are infected and performing port scanning.

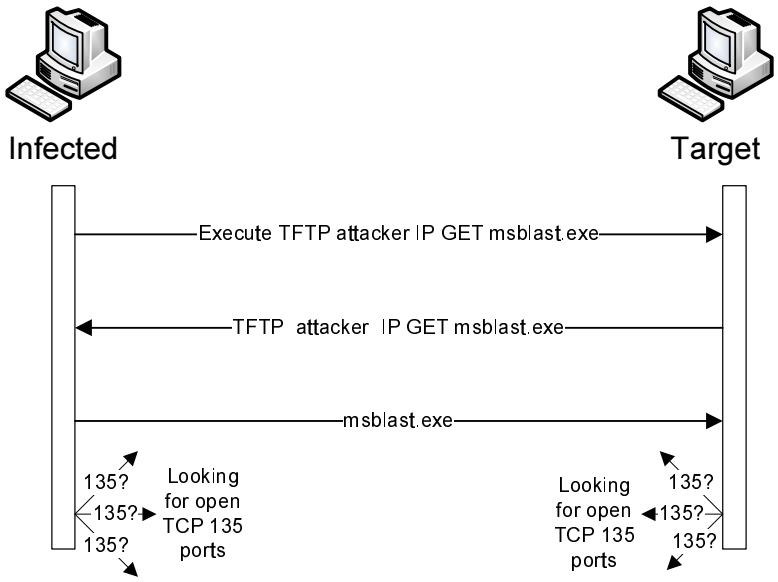

Figure 7. Launching MSBLAST.EXE on target maching

4.4. Other Reported Behavior and Symptoms

- Due to the random nature of the way that exploit data is constructed, the blaster worm kills the svchost.exe host process on the target system, whether or not a shell is obtained (e.g., if the operating system version parameter is incorrect). This causes Windows NT and Windows 2000 systems to become unstable and they hang or crash. The default behaviour of Windows XP and Windows 2003 is to generate an error message indicating an unexpected termination of the RPC service followed by a system restart.
- The worm has a number of variants [18]. The first variant of the worm code that was released creates a mutex named *"BILLY"* to avoid running multiple copies of itself and adds the following value to the registry so that the worm runs when windows is started.

  HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ windows auto update = msblast.exe
- If the current date is the 16th through the end of the month for the months of January to August, or if the current month is September through December, the worm will attempt to perform a DoS on Windows Update. The DoS traffic is a SYN flood on port 80 of windowsupdate.com and tries to send 50 HTTP packets every second. If the worm cannot find a DNS entry for windowsupdate.com, it uses a destination address of 255.255.255.255.

Top

- The systems affected by the Blaster Worm are mostly all versions of Windows 2000 and Windows XP. However, if the worm is executed from Windows NT or Windows 2003 machines, it can run and spread. The most plausible reason can be that the default installations of Windows 2000, Windows XP, and Windows 2003 Server have the DCOM interface to RPC accessible only through port TCP/135. However, Windows NT 4.0's interface is also available on port UDP/135. Windows ME, 95, and 98 are not included on the list of targets because DCOM is not a default service on these operating systems. There are, however, DCOM add on packages available for these platforms. Systems who have installed these types of add-ons could have this vulnerability and are potentially susceptible to this attack or a variant of it.

5. Conclusion

This paper presented analysis of the Blaster worm and the worm's methods of exploitation and propagation. The paper gave adequate details about the popular buffer overflow vulnerability. It is perceived in the research community that buffer overflow attacks will continue to dominate, although various software solutions have been proposed to tackle the problem. This is mainly due to the following reasons: (1) thousands of legacy applications are still being used and many of them are vulnerable to buffer overflow attacks; (2) many proposed software solution incur undesirable performance overheads and/or cannot protect from all attacks. As a result, users are reluctant to patch their software; (3) new software products, due to their inherent complexity and lack of thorough testing due to time-to-market pressure, can leave serious vulnerabilities concealed. The best line of protection against such vulnerabilities, for the time being, is vigilance in applying critical patches. The process of updates must be controlled and automated. Host and network-based vulnerability assessment tools can help to identify outdated systems with security holes inside the internal networks quicker, and thus security patches can be delivered faster.

Acknowledgement

The authors acknowledge the support of King Fahd University of Petroleum and Minerals in the development of this work.

References

[1] CNET NEWS.COM, "Worm Exploits a Widespread Windows Vulnerability," August 2003, http://news.com.com/2009-1002-5063226.html?tag=nl

[2] Microsoft Security Bulletin MS03-026, "Buffer Overrun In RPC Interface Could Allow Code Execution," July 16, 2003. http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx

[3] Common Vulnerabilities and Exposures, "CAN-2003-0352," http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CAN-2003-0352

[4] Dan Verton, "Blaster Worm Linked to Severity of Blackout", *Computer World Magazine*, August 29, 2003.

Top

[4] Rachel Ross, “Hackers threaten power grid, Toronto Star Newspaper, September 12, 2003.

[6] E. Chien and P. Ször, “Blended Attacks Exploits, Vulnerabilities and Buffer-Overflow Techniques in Computer Viruses,” *Virus Bulletin*, pp. 3-7.

[7] Microsoft TechNet, “TCP and UDP Port Assignments” http://www.microsoft.com/technet/treeview/default.asp?url=/technet/prodtechnol/windows2000serv/reskit/tcpip/part4/tcpappc.asp

[8] Microsoft MSDN, “The RPC Model” http://msdn.microsoft.com/library/default.asp?url=library/enus/rpc/rpc/microsoft_rpc_model.asp

[9] Microsoft MSDN, “DCOM Technical Overview”, http://msdn.microsoft.com/library/default.asp?url=/library/enus/dndcom/html/msdn_dcomtec.asp?frame=true&hidetoc=true

[10] Aleph One, “Smashing the Stack For Fun And Profit”, *Phrack Magazine*, vol. 7, no. 49, article 14 of 16. November 8, 1996.

[11] M. W. Eichin and J. A. Rochlis. “With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988,” *In Proceedings of IEEE Computer Society Symposium on Security and Privacy* (SSP ’89), pp. 326–343, 1989.

[12] CERT/CC. CERT Advisory CA-2001-19 Code Red Worm Exploiting Buffer Overflow In IIS Indexing Service DLL. http://www.cert.org/advisories/CA-2001-19.html, July 2001.

[13] CERT/CC, “CERT Advisories,” http://www.cert.org/advisories/

[14] Fyodor, "The Art of Port Scanning", *Phrack Magazine*. vol. 7, no. 51, article 11 of 17. September, 1997.

[15] Aaron Hackworth, “Windows RPC DCOM Buffer Overflow Exploit,” September, 2003, http://www.giac.org/GCIH.php

[16] Wayne J. Freeman, “An Analysis of the Microsoft RPC/DCOM Vulnerability MS03-026”, September 22, 2003. http://www.giac.org/GCIH.php

[17] Brian K. Porter, “RPC-DCOM Vulnerability & Exploit,” November 2, 2003, pp. 3-18. http://www.giac.org/GSEC.php

[18] Symantec Corporation, “The Blaster Worm and its Variants,” http://securityresponse.symantec.com/avcenter/venc/data/w32.blaster.worm.removal.tool.html

Top