Analysis of the Internet Worm of August 2003

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

[**] Corresponding Author

Abstract

In August 2003, the W32.Blaster worm spread all over the world affecting numerous systems and networks causing severe damages and disruptions. 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 has 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 began to wreak havoc in August 2003 and 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 that allowed attackers to execute code of their choice on a remote machine. This vulnerability was assigned the name CAN-2003-0352 [3]. On August 11, 2003, the Blaster worm was released that exploited the buffer overrun vulnerability to spread itself over networks by using open RPC ports on computers that are running most of the versions of Windows operating systems. The Blaster is a type of computer virus known as "*blended threat*" [4]. 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. Blended threats can spread rapidly and cause widespread damage, thus requiring a comprehensive security solution that contains multiple layers of defense and response mechanisms.

2. Background

This section discusses buffer overflow attacks and the stack overflow mechanism employed by the blaster to exploit the buffer overrun. We also discuss the propagation mechanism used by the worm. It is assumed that the reader has rudimentary knowledge pertaining to the TCP, RPC, and DCOM.

2.1 Buffer Overflow Attacks

The most dominant form of buffer overflow exploitation is stack smashing. A buffer overflow is an example of a blended attack. The Morris worm [5] 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 [6].

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. Of the various categories into which buffer overflows are broken into; the stack overflow method is used by the Blaster worm.

2.2 Overflowing the Stack

```
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; }
```

The memory space allocated to a program is split into code segment, data segment and the stack segment.

Above is an example of a C code making the buffer overflow. The changes in the memory are as follows:

1. Function parameters will be pushed onto memory from the high memory location to low memory location.
2. The return address is pushed onto the stack followed by a frame pointer. A frame pointer is used to reference the local variables and function parameters.

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. Several unsafe string operations also exist in the standard C library, like `strcpy(), strcat(), sprintf(), gets()`.

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 uses a flaw in a low level function that extracts the server name from a UNC path that is passed to it. "\\computername\sharename\path\filename.ext" is an example of a UNC path. 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

The CoGetInstanceFromFile function below 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).

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

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 [9] 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.
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 in 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
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

The Blaster propagates by port scanning, performing the client-server handshake, and then launching 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:

1. 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.
2. 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.
3. 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.
4. 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. The client makes an RPC bind to the remote server before the call to the DCOM instantiation function on the remote server can take place. This RPC protocol handshake is an application layer version of the TCP handshake. 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)
    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. 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.

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 1 describes the steps involved in launching the msblast.exe file, which can be summarized as follows:

1. The attacking machine 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.
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.

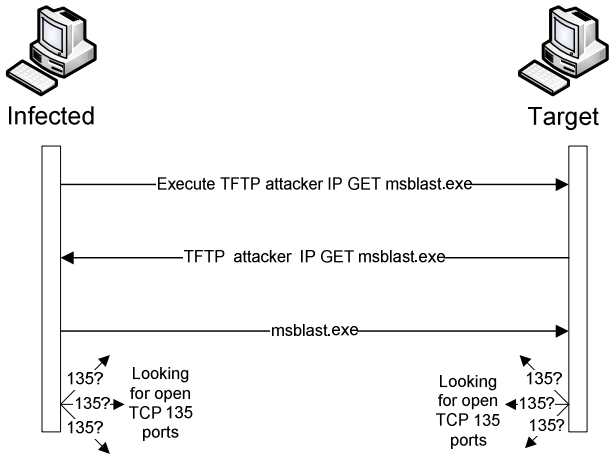

Figure 1. Launching MSBLAST.EXE on target machine

4.4. Other Reported Behavior and Symptoms

1. 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.

2. The worm has a number of variants [10]. 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.
3. HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ windows auto update = msblast.exe
4. 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.
5. 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] E. Chien and P. Ször, "Blended Attacks Exploits, Vulnerabilities and Buffer-Overflow Techniques in Computer Viruses," *Virus Bulletin*, pp. 3-7.

[5] 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.

[6] 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.

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

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

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

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